Oracle Database 11*g*: SQL Tuning Workshop

Activity Guide

D52163GC20
Edition 2.0
October 2010
D69162

Author

James Spiller, Tulika Srivastava

Technical Contributors and Reviewers

Abhinav Gupta, Branislav Valny, Clinton Shaffer, Donna Keesling, Ira Singer, Howard Bradley, Sean Kim, Sue Harper, Teria Kidd

This book was published using: Oracle Tutor

Table of Contents

Practices for Lesson 1

Chapter 1

Overview of Practices for Lesson 1

Practices Overview

In these practices, you will test your understanding of the Database Architecture.

Practice 1-1: Exploring the Database Architecture

Fill in the blanks with the correct answers.

1. The two main components of Oracle RDBMS are
   ___________________________ and _________________________.
2. An instance consists of ______________________and ______________________ processes.
3. A session is a connection between the ________________ process and either the _______________ process or the _________________ process.
4. Name some components of the System Global Area (SGA).
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
5. List some background processes:
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
6. The _________________________ process writes the dirty buffers to the data files.
7. The _________________________ process writes the redo logs to the log files.
8. Name some files associated with an Oracle database.
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
9. Some of the logical storage structures of an Oracle database are:
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________
   - _____________________________

10. The ___________________ process copies the redo log files to an archive destination.
11. The _____________________ contains data and control information for a server or a background process.
12. The logical tablespace structure is associated with the physical ____________________ files on disk.
13. State whether the following statements are true or false:
    a. The SGA includes the Database buffer cache and the Redo log buffer. ____
    b. Each server process and background process has its own Program Global Area (PGA). ____
    c. User processes run the application or Oracle tool code. ____
    d. Oracle Database processes include server processes and background processes. ____.
14. From a terminal session, connected as the `oracle` user, execute the `processes.sh` script located in your `$HOME/solutions/Database_Architecture` directory. What does this script show you?
    a. It shows you all the database instance processes currently running on your machine. This includes both background processes and foreground processes.

```
$ cd $HOME/solutions/Database_Architecture
$ ./processes.sh
oracle    2891  3129  0 Jun14 ?        00:16:15
/u01/app/oracle/product/11.2.0/dbhome_1/jdk/bin/java -server -Xmx192M
-XX:MaxPermSize=200M -XX:MinHeapFreeRatio=20 -XX:MaxHeapFreeRatio=40 -
DORACLE_HOME=/u01/app/oracle/product/11.2.0/dbhome_1 -
Doracle.home=/u01/app/oracle/product/11.2.0/dbhome_1/oc4j -
Doracle.oc4j.localhome=/u01/app/oracle/product/11.2.0/dbhome_1/edrsr41
p1.us.oracle.com_orcl/sysman -
DEMSTATE=/u01/app/oracle/product/11.2.0/dbhome_1/edrsr41p1.us.oracle.c
om_orcl -Doracle.j2ee.dont.use.memory.archive=true -
Djava.protocol.handler.pkgs=HTTPClient -
Doracle.security.jazn.config=/u01/app/oracle/product/11.2.0/dbhome_1/o
c4j/j2ee/OC4J_DBConsole_edrsr41p1.us.oracle.com_orcl/config/jazn.xml -
Djava.security.policy=/u01/app/oracle/product/11.2.0/dbhome_1/oc4j/j2e
e/OC4J_DBConsole_edrsr41p1.us.oracle.com_orcl/config/java2.policy -
Djavax.net.ssl.KeyStore=/u01/app/oracle/product/11.2.0/dbhome_1/sysman
/config/OCMTrustedCerts.txt-
Djava.security.properties=/u01/app/oracle/product/11.2.0/dbhome_1/oc4j
/j2ee/home/config/jazn.security.props -
DEMDROOT=/u01/app/oracle/product/11.2.0/dbhome_1/edrsr41p1.us.oracle.c
om_orcl -Dsysman.md5password=true -
Drepapi.oracle.home=/u01/app/oracle/product/11.2.0/dbhome_1 -
Ddisable.checkForUpdate=true -
Doracle.sysman.ccr.ocmSDK.websvc.keystore=/u01/app/oracle/product/11.2
.0/dbhome_1/jlib/emocmclnt.ks -
Dice.pilots.html4.ignoreNonGenericFonts=true -Djava.awt.headless=true
-jar /u01/app/oracle/product/11.2.0/dbhome_1/oc4j/j2ee/home/oc4j.jar -
config
/u01/app/oracle/product/11.2.0/dbhome_1/oc4j/j2ee/OC4J_DBConsole_edrsr
41p1.us.oracle.com_orcl/config/server.xml
oracle    3129     1  0 Jun08 ?        00:01:03
/u01/app/oracle/product/11.2.0/dbhome_1/perl/bin/perl
/u01/app/oracle/product/11.2.0/dbhome_1/bin/emwd.pl dbconsole
```

```
/u01/app/oracle/product/11.2.0/dbhome_1/edrsr41p1.us.oracle.com_orcl/s
ysman/log/emdb.nohup
oracle    3253     1  0 Jun14 ?        00:00:43 ora_pmon_orcl
oracle    3255     1  0 Jun14 ?        00:00:00 ora_vktm_orcl
oracle    3259     1  0 Jun14 ?        00:00:00 ora_gen0_orcl
oracle    3261     1  0 Jun14 ?        00:00:00 ora_diag_orcl
oracle    3263     1  0 Jun14 ?        00:00:00 ora_dbrm_orcl
oracle    3265     1  0 Jun14 ?        00:00:32 ora_psp0_orcl
oracle    3267     1  0 Jun14 ?        00:19:33 ora_dia0_orcl
oracle    3269     1  0 Jun14 ?        00:00:00 ora_mman_orcl
oracle    3271     1  0 Jun14 ?        00:02:12 ora_dbw0_orcl
oracle    3273     1  0 Jun14 ?        00:03:00 ora_lgwr_orcl
oracle    3275     1  0 Jun14 ?        00:00:03 ora_ckpt_orcl
oracle    3277     1  0 Jun14 ?        00:01:44 ora_smon_orcl
oracle    3279     1  0 Jun14 ?        00:00:00 ora_reco_orcl
oracle    3281     1  0 Jun14 ?        00:01:50 ora_mmon_orcl
oracle    3283     1  0 Jun14 ?        00:00:48 ora_mmnl_orcl
oracle    3285     1  0 Jun14 ?        00:00:00 ora_d000_orcl
oracle    3287     1  0 Jun14 ?        00:00:00 ora_s000_orcl
oracle    3340     1  0 Jun14 ?        00:00:00 ora_qmnc_orcl
oracle    3344     1  0 Jun14 ?        00:01:10 oracleorcl (LOCAL=NO)
oracle    3361     1  0 Jun14 ?        00:02:07 ora_cjq0_orcl
oracle    3379     1  0 Jun14 ?        00:00:08 ora_q000_orcl
oracle    3381     1  0 Jun14 ?        00:00:01 ora_q001_orcl
oracle    3532     1  0 Jun14 ?        00:00:00 oracleorcl (LOCAL=NO)
oracle    3534     1  0 Jun14 ?        00:05:40 oracleorcl (LOCAL=NO)
oracle    3536     1  0 Jun14 ?        00:04:42 oracleorcl (LOCAL=NO)
oracle    3538     1  0 Jun14 ?        00:13:05 oracleorcl (LOCAL=NO)
oracle    3599     1  0 Jun14 ?        00:12:10 oracleorcl (LOCAL=NO)
oracle    3614     1  0 Jun14 ?        00:00:01 oracleorcl (LOCAL=NO)
oracle    3617     1  0 Jun14 ?        00:05:40 oracleorcl (LOCAL=NO)
oracle    3855     1  0 Jun14 ?        00:00:00 ora_smco_orcl
oracle    7139     1  0 Jul04 ?        00:00:02 oracleorcl (LOCAL=NO)
oracle    9861     1  0 Jun15 ?        00:01:03 oracleorcl (LOCAL=NO)
oracle   11366     1  0 15:14 ?        00:00:00 ora_w000_orcl
```

15. From a terminal session, connected as the `oracle` user, execute the `files.sh` script located in your `$HOME/solutions/Database_Architecture` directory. What does this script show you?
    a. This script shows you the location and names of all database files, initialization file, password file, and trace files.

```
$ cd $HOME/solutions/Database_Architecture
$ ./files.sh
…
SQL>
SQL> col name format a45
SQL>
```

```
SQL> select name from v$controlfile;

NAME
---------------------------------------------
/u01/app/oracle/oradata/orcl/control01.ctl
/u01/app/oracle/oradata/orcl/control02.ctl
/u01/app/oracle/oradata/orcl/control03.ctl

SQL>
SQL>
SQL> col member format a45
SQL>
SQL> select group#,member from v$logfile;

    GROUP# MEMBER
---------- ---------------------------------------------
         3 /u01/app/oracle/oradata/orcl/redo03.log
         2 /u01/app/oracle/oradata/orcl/redo02.log
         1 /u01/app/oracle/oradata/orcl/redo01.log

SQL>
SQL>
SQL> col tablespace_name format a20
SQL> col file_name format a45
SQL>
SQL> select tablespace_name, file_name from dba_data_files;

TABLESPACE_NAME      FILE_NAME
-------------------- ---------------------------------------------
USERS                /u01/app/oracle/oradata/orcl/users01.dbf
UNDOTBS1             /u01/app/oracle/oradata/orcl/undotbs01.dbf
SYSAUX               /u01/app/oracle/oradata/orcl/sysaux01.dbf
SYSTEM               /u01/app/oracle/oradata/orcl/system01.dbf
EXAMPLE              /u01/app/oracle/oradata/orcl/example01.dbf
TRACETBS             /u01/app/oracle/oradata/orcl/tracetbs.dbf
TRACETBS3            /u01/app/oracle/oradata/orcl/tracetbs3.dbf

SQL>
SQL> select tablespace_name, file_name from dba_temp_files;

TABLESPACE_NAME      FILE_NAME
-------------------- ---------------------------------------------
TEMP                 /u01/app/oracle/oradata/orcl/temp01.dbf

SQL>
SQL> exit;
```

```
…

-rw-rw----  1 oracle oinstall 1544 Aug 22  2007
/u01/app/oracle/product/11.1.0/db_1/dbs/hc_orcl.dat
-rw-r-----  1 oracle oinstall 1536 Mar 26 22:03
/u01/app/oracle/product/11.1.0/db_1/dbs/orapworcl
-rw-r-----  1 oracle oinstall 2560 Mar 27 03:13
/u01/app/oracle/product/11.1.0/db_1/dbs/spfileorcl.ora
alert  cdump  hm  incident  incpkg  ir  lck  metadata  stage  sweep
trace
-rw-r--r--  1 oracle oinstall 557386 Mar 27 13:00
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/alert_orcl.log
$

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

#!/bin/bash

cd /home/oracle/solutions/Database_Architecture


sqlplus / as sysdba @files.sql

ls -l $ORACLE_HOME/dbs/*orcl*

ls /u01/app/oracle/diag/rdbms/orcl/orcl

ls -l /u01/app/oracle/diag/rdbms/orcl/orcl/trace/alert*
--------------------------------------------------------------

col name format a45

select name from v$controlfile;


col member format a45

select group#,member from v$logfile;


col tablespace_name format a20
col file_name format a45

select tablespace_name, file_name from dba_data_files;

select tablespace_name, file_name from dba_temp_files;
```

```
exit;
```

16. From a terminal session, connected as the `oracle` user, execute the `sga.sh` script located in your `$HOME/solutions/Database_Architecture` directory. What does this script show you? (This script prints the various pools held in your SGA.)

```
$ cd $HOME/solutions/Database_Architecture
$ ./sga.sh

SQL*Plus: Release 11.2.0.1.0 Production on Mon Jul 5 15:24:52 2010
Copyright (c) 1982, 2009, Oracle.  All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing
options
SQL>
SQL> select * from v$sgainfo;
NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                      1339796 No
Redo Buffers                        5144576 No
Buffer Cache Size                 343932928 Yes
Shared Pool Size                  192937984 Yes
Large Pool Size                     4194304 Yes
Java Pool Size                      4194304 Yes
Streams Pool Size                         0 Yes
Shared IO Pool Size                       0 Yes
Granule Size                        4194304 No
Maximum SGA Size                  845348864 No
Startup overhead in Shared Pool    58720256 No

NAME                                  BYTES RES
-------------------------------- ---------- ---
Free SGA Memory Available         293601280

12 rows selected.

SQL>
SQL> col component format a30
SQL>
SQL> select component,current_size,min_size,max_size from
v$memory_dynamic_components;

COMPONENT                      CURRENT_SIZE   MIN_SIZE   MAX_SIZE
------------------------------ ------------ ---------- ----------
shared pool                       192937984  167772160  192937984
```

```
large pool                              4194304    4194304    4194304
java pool                               4194304    4194304    4194304
streams pool                                  0          0          0
SGA Target                            553648128  532676608  553648128
DEFAULT buffer cache                  343932928  331350016  352321536
KEEP buffer cache                             0          0          0
RECYCLE buffer cache                          0          0          0
DEFAULT 2K buffer cache                       0          0          0
DEFAULT 4K buffer cache                       0          0          0
DEFAULT 8K buffer cache                       0          0          0

COMPONENT                            CURRENT_SIZE   MIN_SIZE   MAX_SIZE
------------------------------ ------------ ---------- ----------
DEFAULT 16K buffer cache                      0          0          0
DEFAULT 32K buffer cache                      0          0          0
Shared IO Pool                                0          0          0
PGA Target                            293601280  293601280  314572800
ASM Buffer Cache                              0          0          0

16 rows selected.
SQL>
SQL> exit;
Disconnected from Oracle Database 11g Enterprise Edition Release
11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing
options
```

Practices for Lesson 2

Chapter 2

Overview of Practices for Lesson 2

Practices Overview

In these practices, you will get acquainted with SQL Developer functions, explore some common mistakes and how to correct them.

Practice 2-1: Using SQL Developer

1. Start Oracle SQL Developer.

   Click the Oracle SQL Developer icon on your desktop.

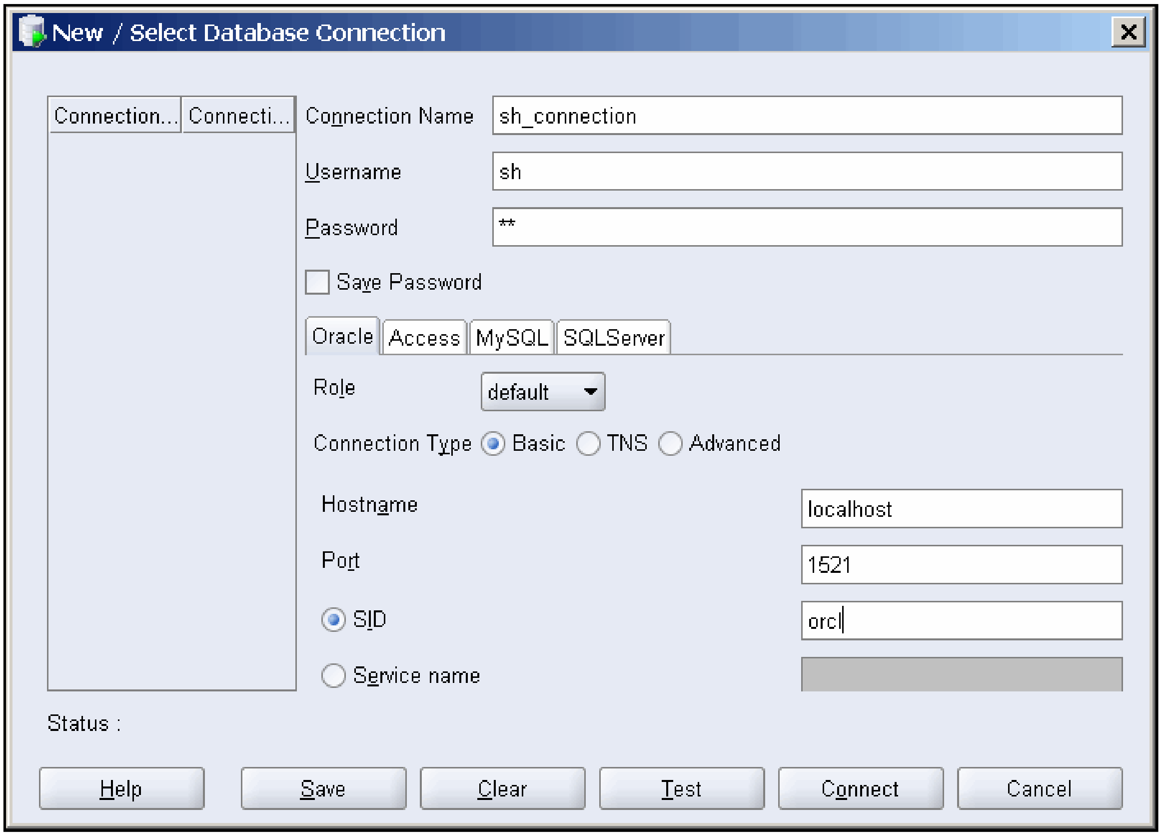
   

2. Create a database connection to the SH schema using the following information:

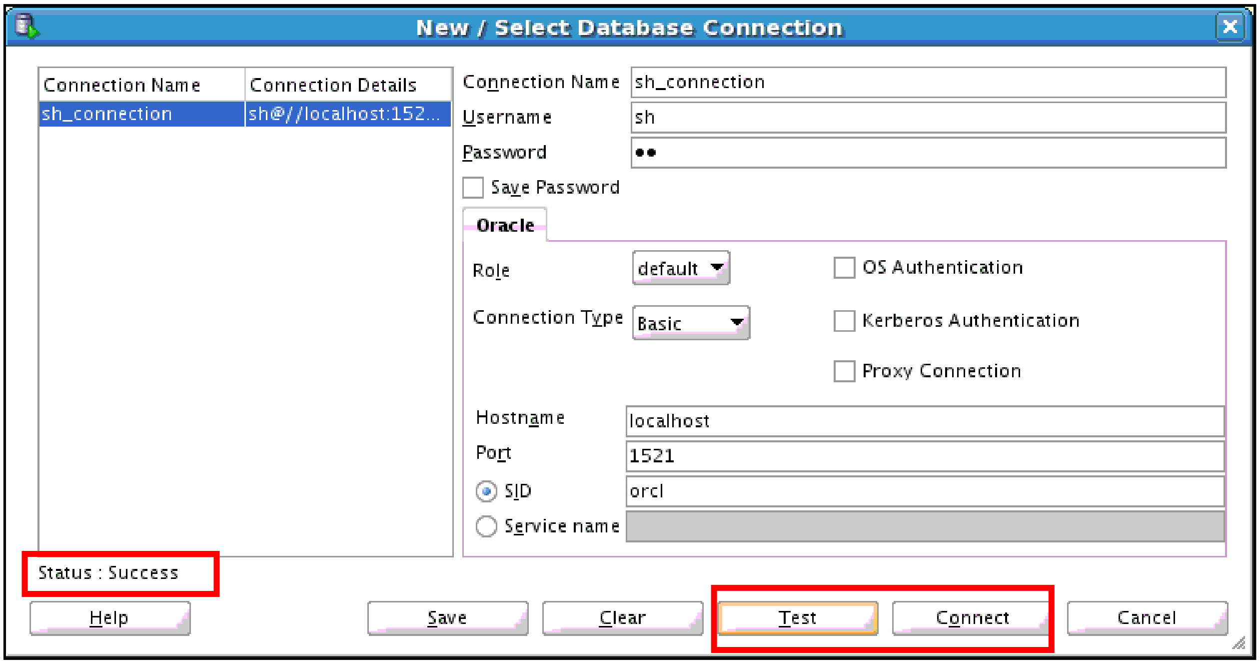
Connection Name	sh_connection
Username:	Sh
Password:	Sh
Hostname:	localhost
Port:	1521
SID:	orcl

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.

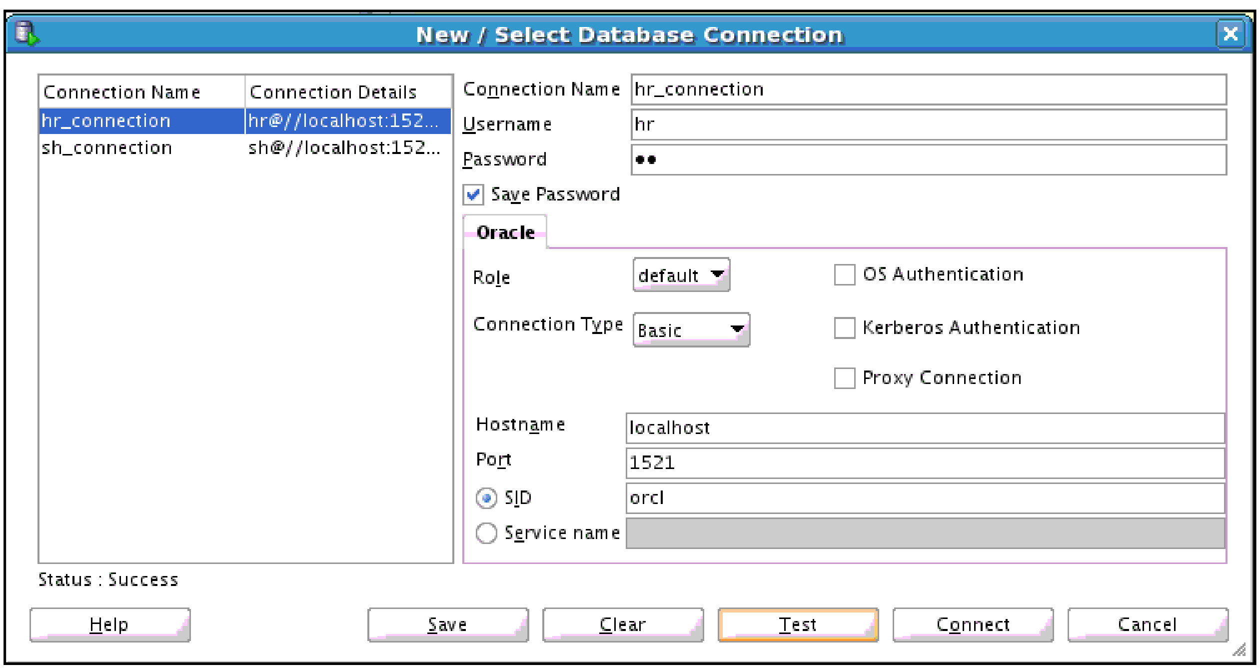
Note: To display the properties of the newly created connection, right-click the connection name, and then select Properties from the shortcut menu. Enter the username, password, host name, and service name with the appropriate information, as provided in the table. The following is a sample of the newly created database connection for the SH schema using a local connection:



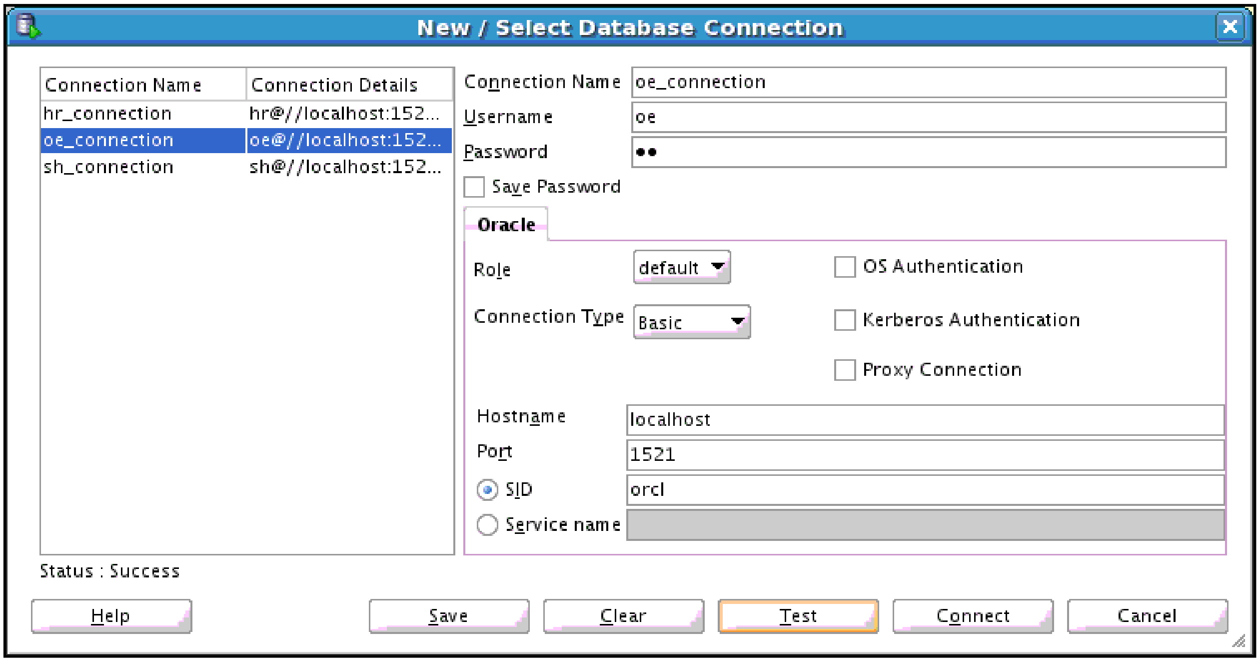
3. Test the new connection. If the Status is Success, save the connection, and then connect to the database using this new connection.
   a. Double-click `sh_connection` on the Connections tabbed page.
   b. Click the Test button in the New / Select Database Connection window. If the status is Success, click the Connect button.



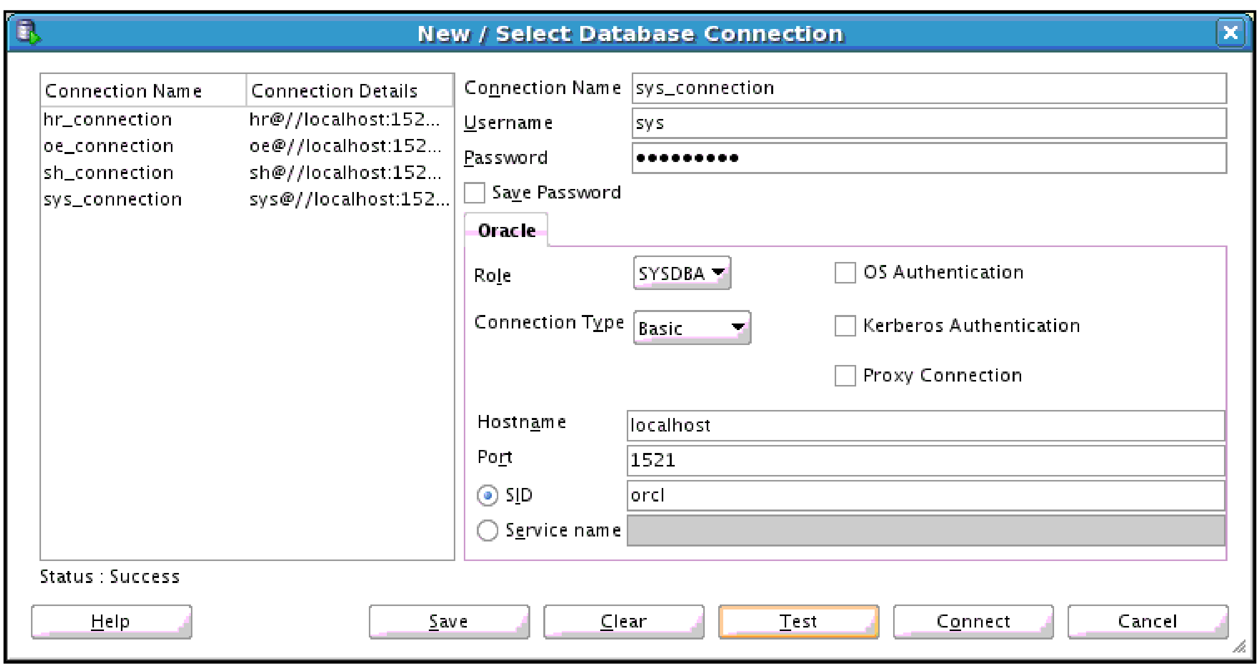
4. Create a new database connection named `hr_connection`.
   a. Right-click `sh_connection` in the Object Navigation tree, and select the Properties menu option.
   b. Enter `hr_connection` as the connection name and `hr` as the username and password, select Save Password, and then click Save to create the new connection.
   c. Test the new `hr_connection` connection.



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

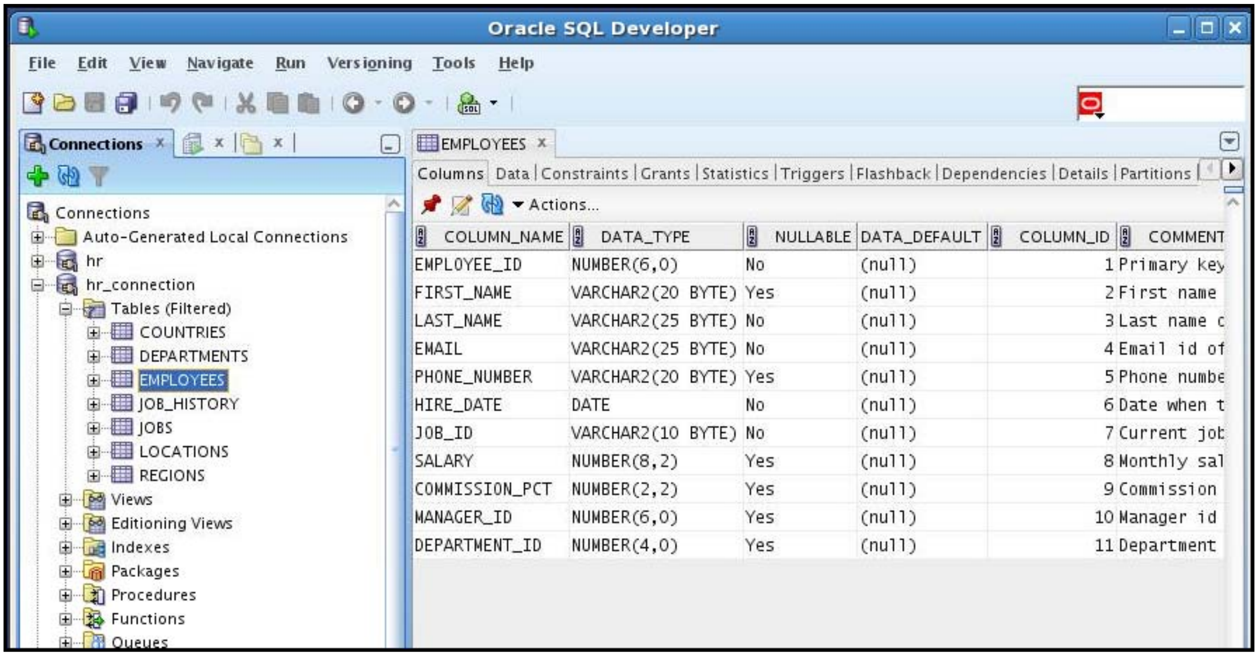


6. Repeat step 3 to create and test a new database connection named `sys_connection`. Enter `sys` in the Username field, `oracle_4U` in the Password field, and `SYSDBA` as the role. From the Role drop-down menu, select `SYSDBA` as the role.

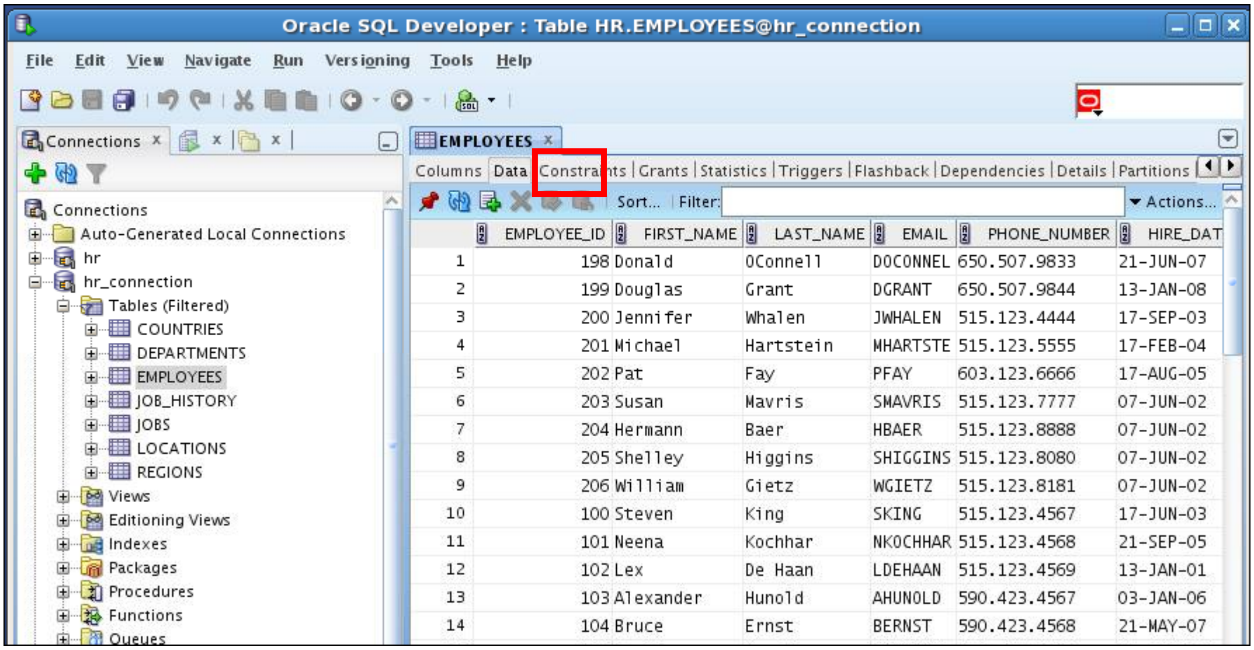


Browsing the HR, SH, and OE Schema Tables

7. 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. Double-click the EMPLOYEES table. The Columns tab displays the columns in the EMPLOYEES table as follows:



8. 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:

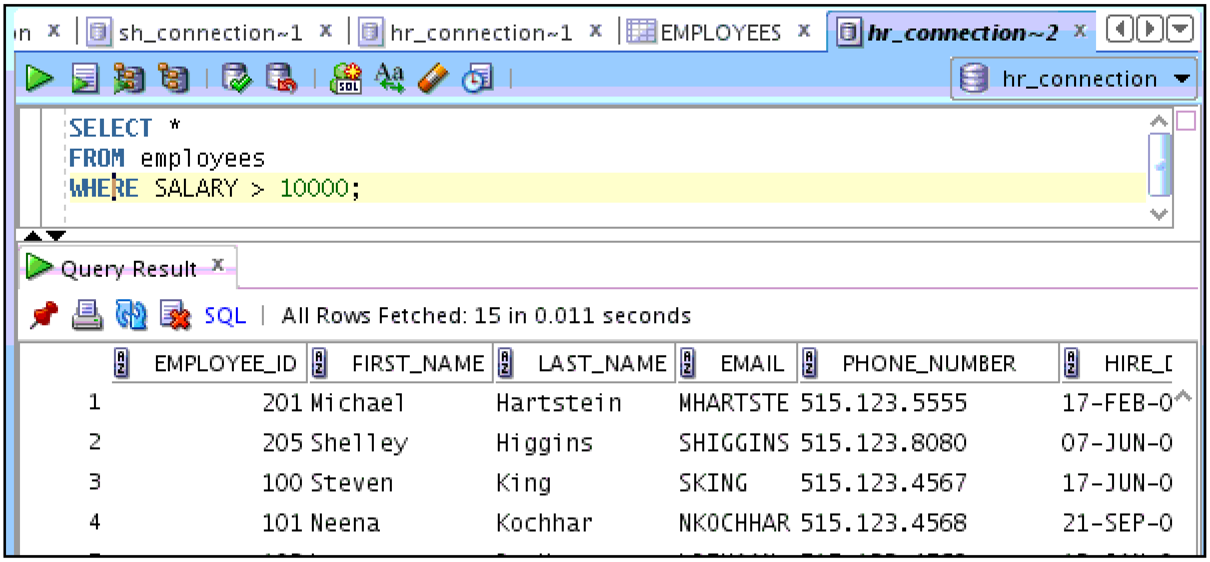


9. Use the SQL Worksheet to select the last names and salaries of all employees whose annual salary is greater than $10,000. Use both the Execute Statement (F9) and the Run Script (F5) icons to execute the `SELECT` statement. Review the results of both methods of executing the `SELECT` statements on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

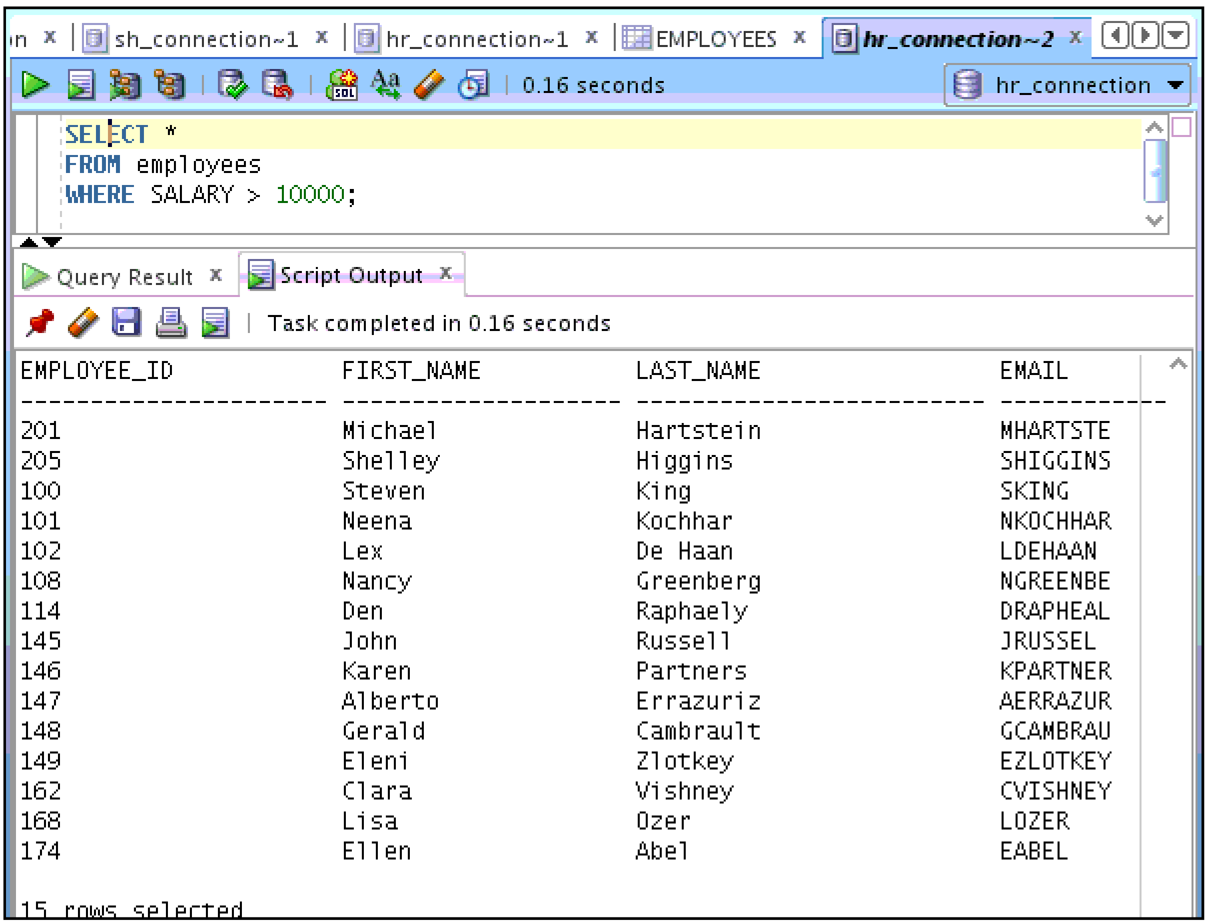
   a. Display the SQL Worksheet by using any of the following two methods: Select Tools > SQL Worksheet, or click the Open SQL Worksheet icon. The Select Connection window appears. Select `hr_connection`. Enter the following statement in the SQL Worksheet:

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

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

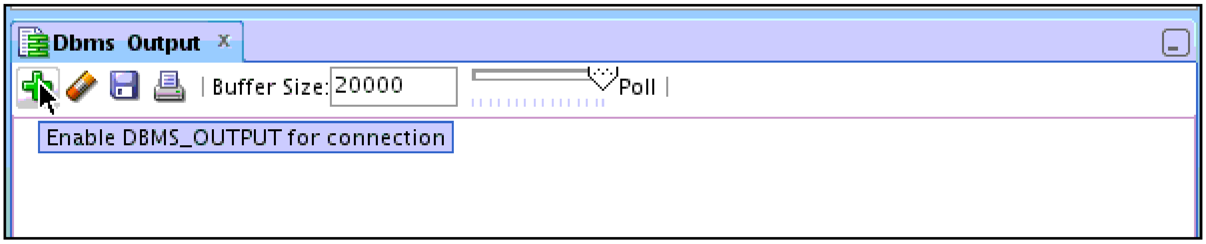


2) Execute by pressing the F5 key or by clicking the Execute Script icon 

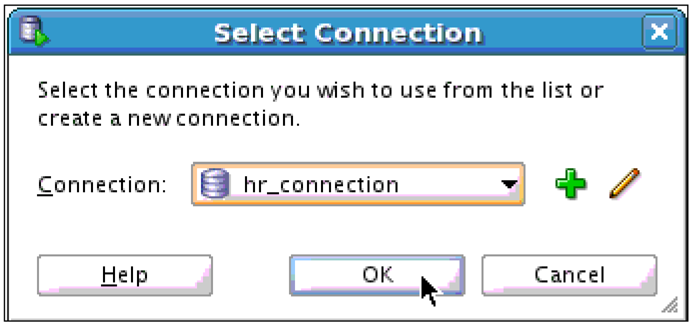


10. In the same SQL Worksheet, create and execute a simple anonymous block that outputs "Hello World."

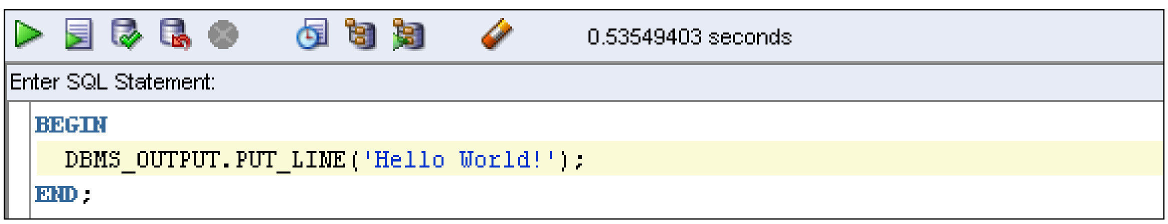
a. Enable `SET SERVEROUTPUT ON` to display the output of the `DBMS_OUTPUT` package statements. Select View > Dbms Output. Click the green plus sign as shown to enable `DBMS_OUTPUT`.



b. In the dialog box that appears, select `hr_connection` and click OK.



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



d. Click the Execute icon (Ctrl + Enter) to run the anonymous block.

The Script Output tab displays the output of the anonymous block as follows:

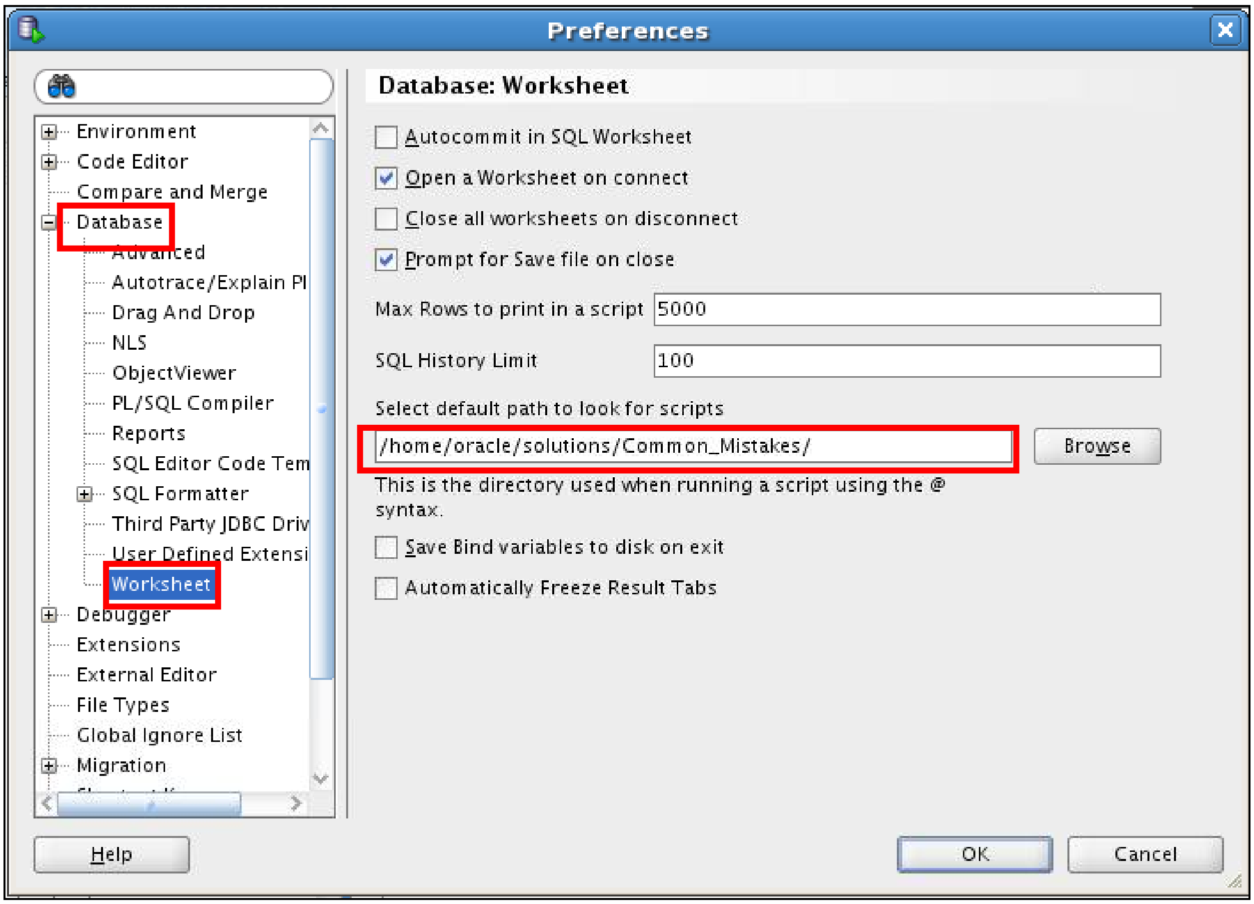

Start Page
hr_connection
hr_connection
Begin
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
Statement Output
anonymous block completed
Dbms Output
Buffer Size: 20000
Poll
hr_connection
set serveroutput on
Hello World

Practice 2-2: Avoiding Common Mistakes

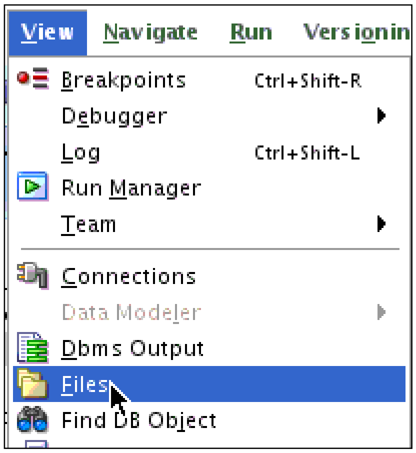
In this practice, you examine some common mistakes in writing SQL statements. You have to find a workaround to enhance performance.

Case 1: Correlated Subquery

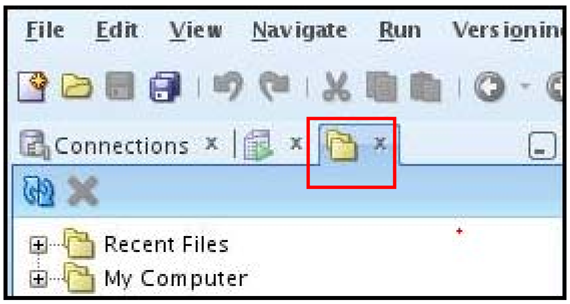
1. Execute the `correlation_setup.sh` script from the `$HOME/solutions/Common_Mistakes` directory. Make sure that you run the script connected as the `sysdba` user. You can find the scripts for all the following cases in your `$HOME/solutions/Common_Mistakes` directory.
2. Analyze a correlated subquery,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



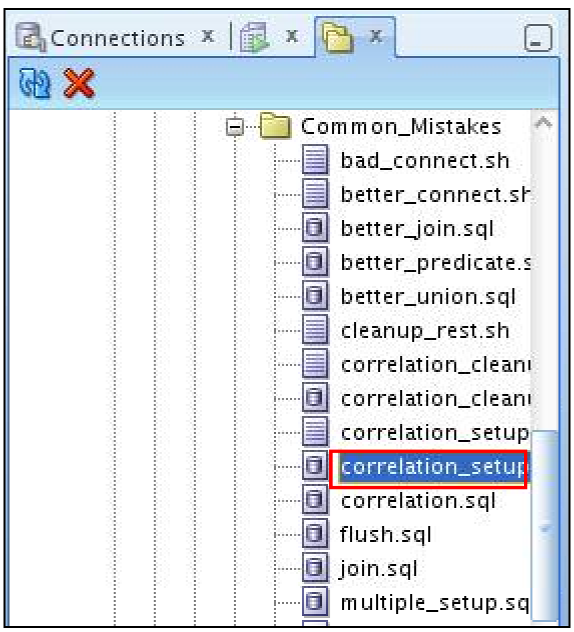
   d. Click OK.
   e. Add the files tab to the navigator pane. Select View > Files.



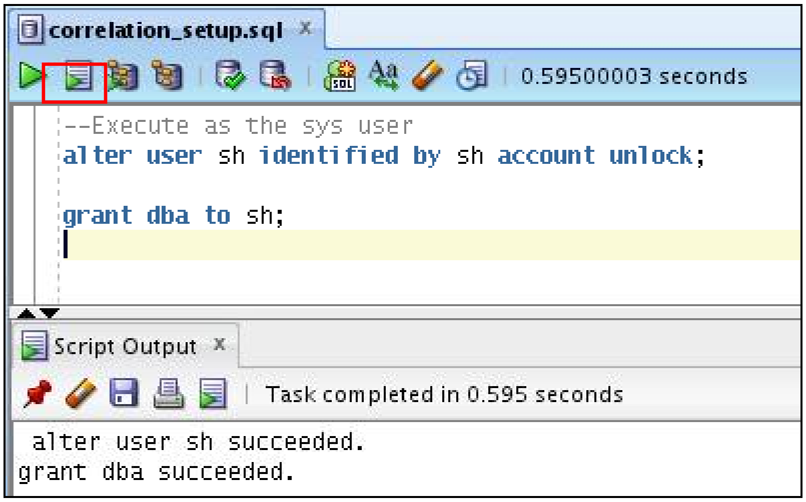
f. Click the files tab.



g. In the file explorer, open the `$HOME/solutions/Common_Mistakes/correlation_setup.sql` file.

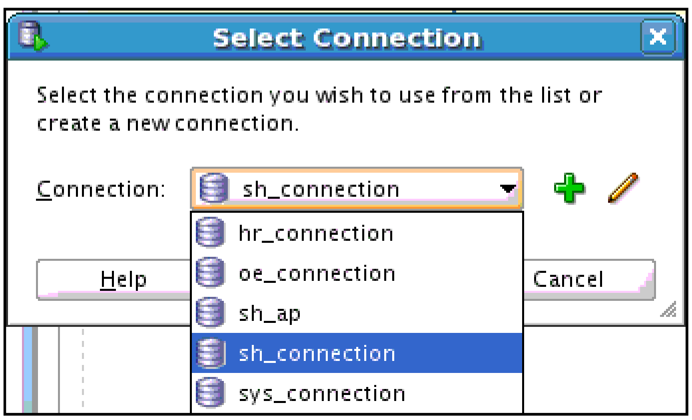


h. Run the script by clicking the Run Script icon. (Alternatively, press F5.) When you execute the script, you must select a connection. Choose `sys_connection` from the drop-down menu, and then enter the sys password `oracle_4U`.



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

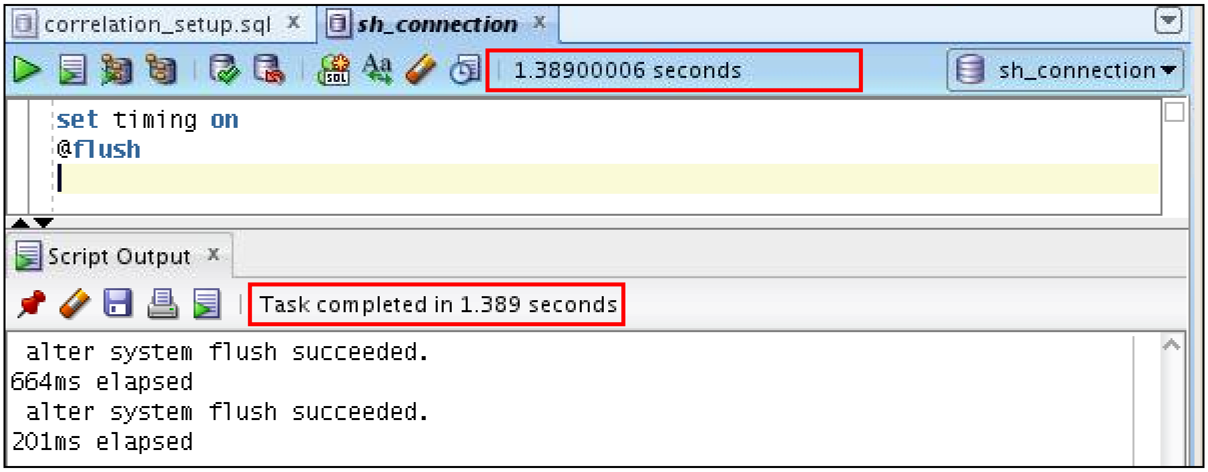
   b. Select `sh_connection` from the drop-down list.



4. Execute the following command:

```
set timing on
@flush
```

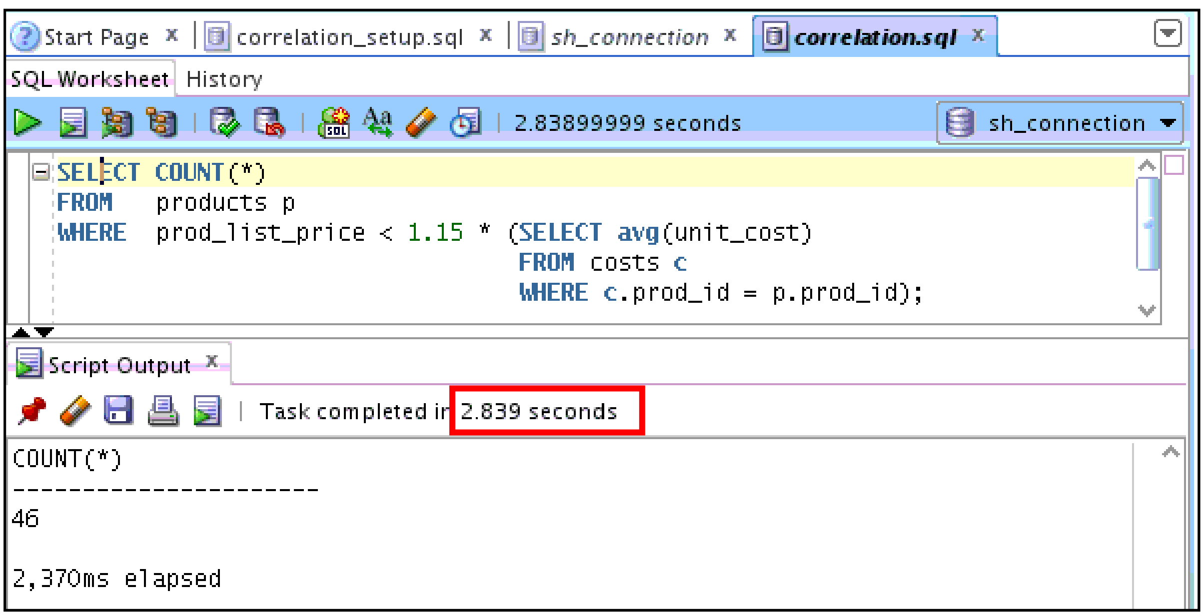
The goal of the first command is to tell you how long any command takes to execute. The `flush.sql` script flushes both the shared pool and the buffer cache to avoid most caching effects so that you can have good comparisons between two executions. **Note:** You should *not* use commands found in this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



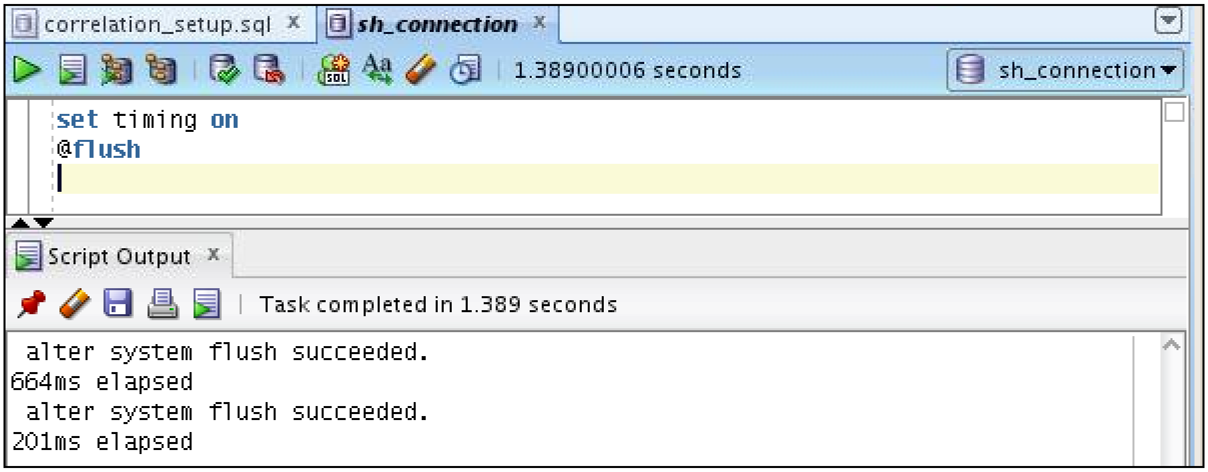
Note: SQL Developer displays the timing information in the worksheet by default and in the output pane as a result of the `SET TIMING ON` command.

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

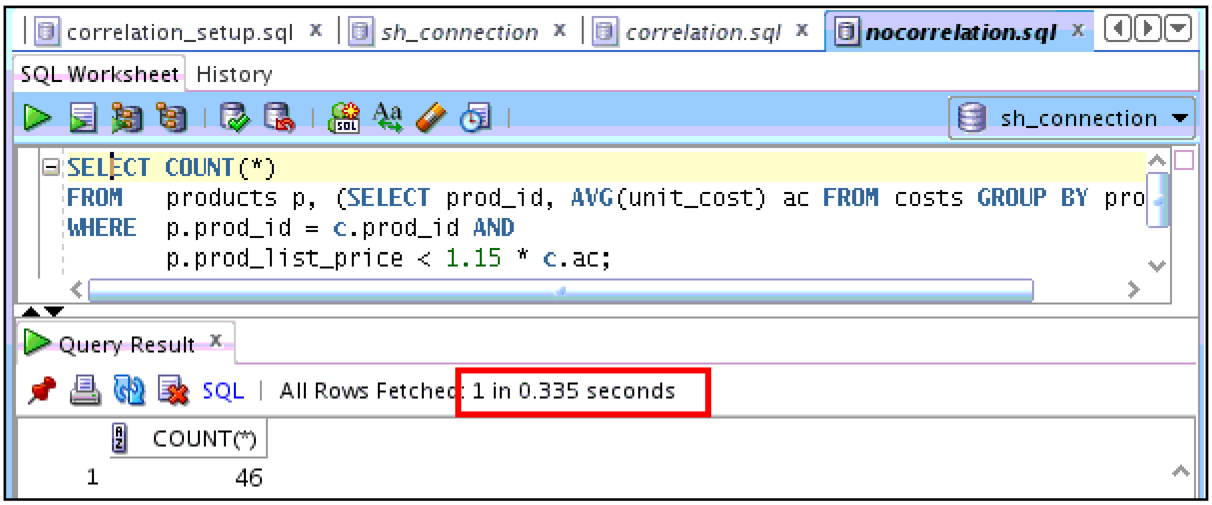
```
SELECT COUNT(*)
FROM   products p
WHERE  prod_list_price < 1.15 * (SELECT avg(unit_cost)
                                 FROM costs c
                                 WHERE c.prod_id = p.prod_id);
```



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



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



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

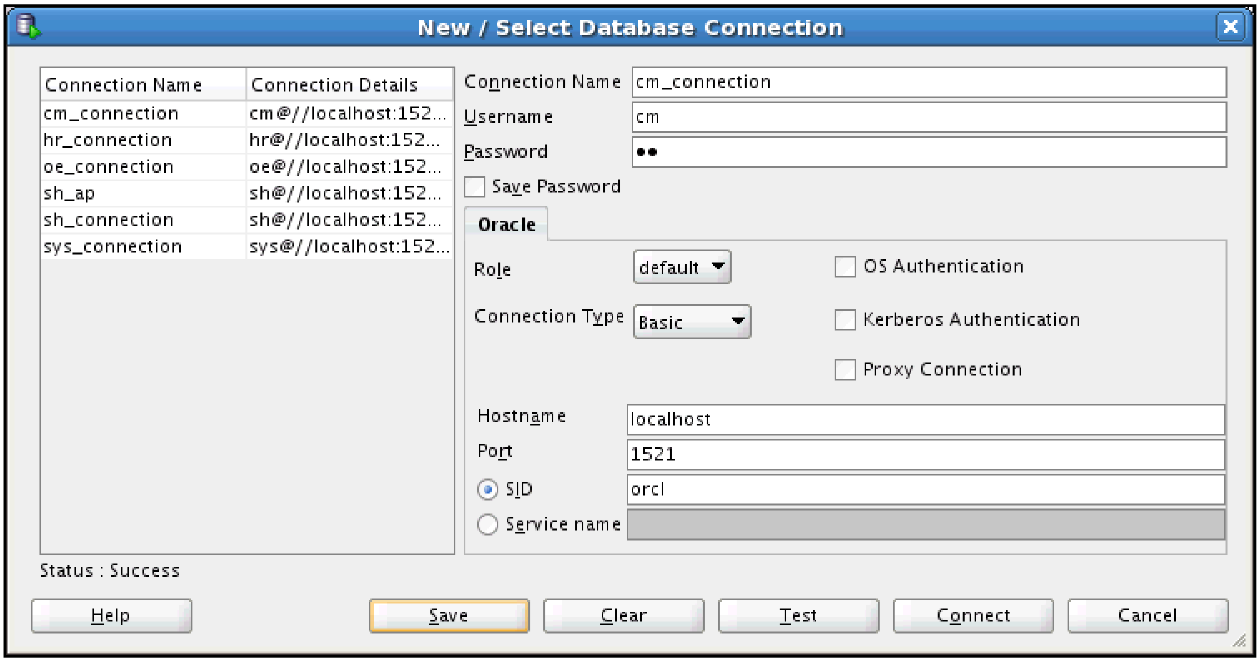
```
set echo on

drop user cm cascade;

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

grant connect, resource, dba to cm;

connect cm/cm

drop table orders purge;
```

```
create table orders (order_id_char varchar2(50) primary key,
order_total number, customer_name varchar2(300));

begin
for i in 1..500000 loop
insert into orders
values(i,100,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/

begin
for i in 1..500000 loop
insert into orders
values(500000+i,100,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/

drop table employees purge;
drop table job_history purge;

create table employees (employee_id number primary key, name
varchar2(500));

begin
for i in 1..500000 loop
insert into employees
values(i,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/
```

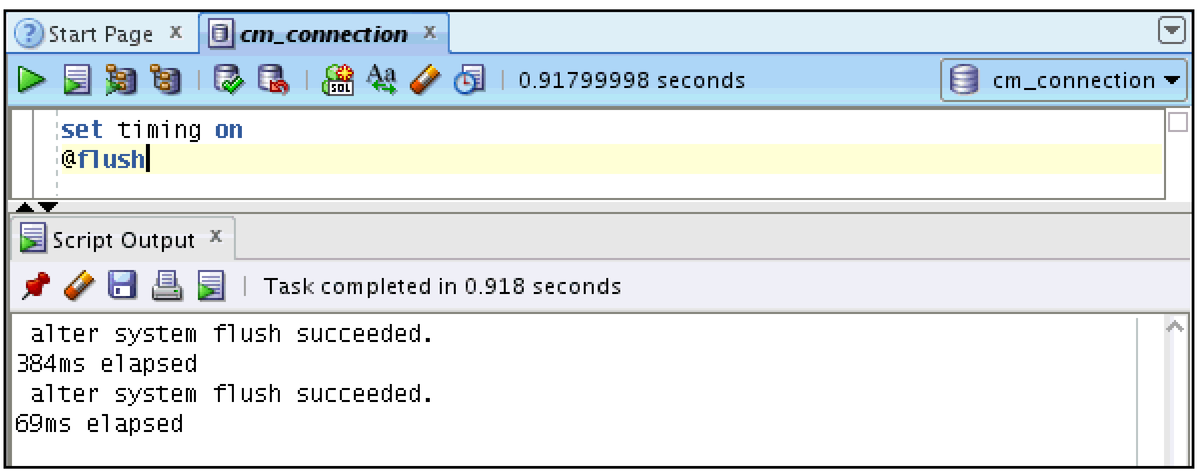
```
create table job_history (employee_id number, job
varchar2(500));

begin
for i in 1..500000 loop
insert into job_history
values(mod(i,1000),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/

create index job_history_empid_indx on job_history(employee_id);


drop table old purge;
drop table new purge;

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

begin
for i in 1..500000 loop
insert into old
values(i,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;
/

begin
for i in 1..500000 loop
insert into new
values(500000+i,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
```

```
end;
/
```

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



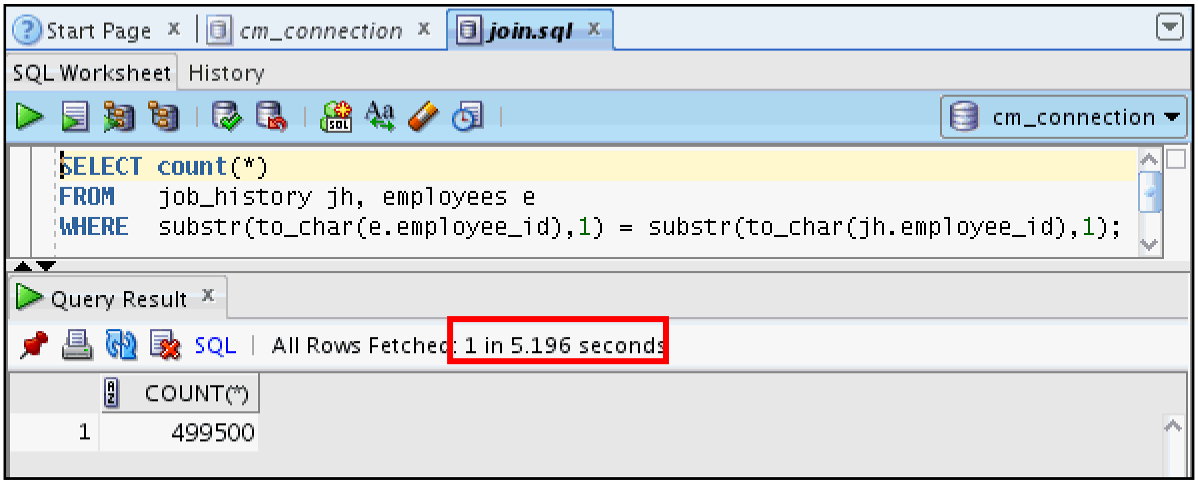
10. Set timings on and flush your environment again before starting the second case. You can use the `set timing on` command and the `flush.sql` script for this.



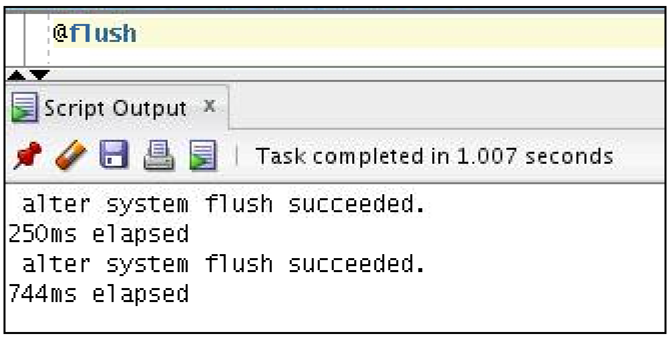
Case 2: Joins Conditions

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

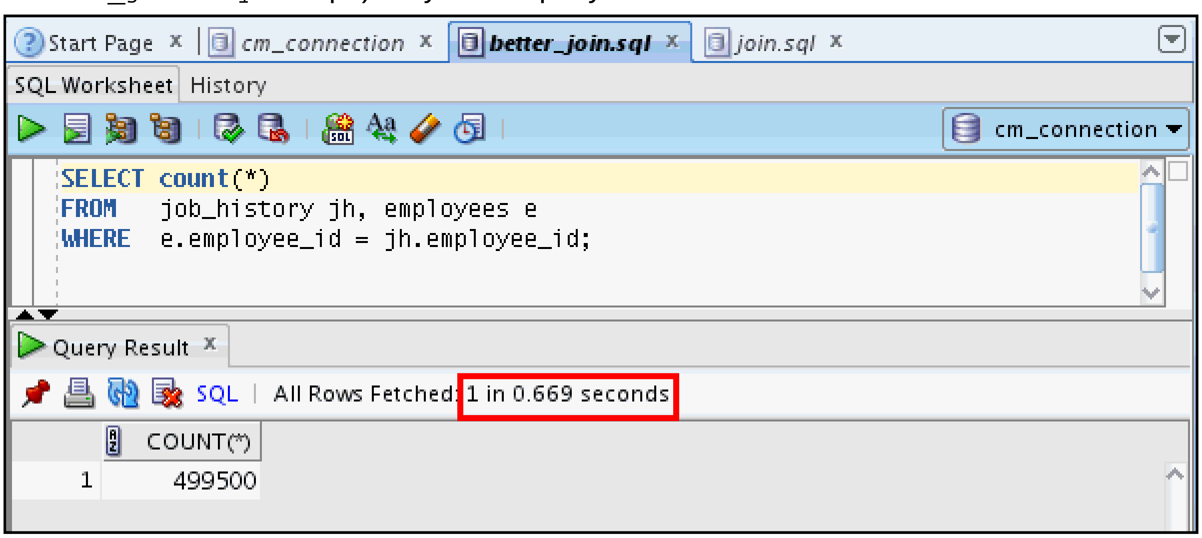
```
SELECT  count(*)
FROM    job_history jh, employees e
WHERE   substr(to_char(e.employee_id),1) =
substr(to_char(jh.employee_id),1);
```



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

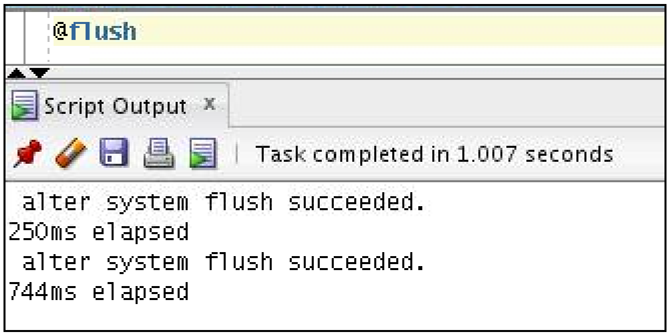


13. How would you rewrite the previous query for better performance? Test your solution. You should discuss this with your instructor. (Alternatively, open and execute the `better_join.sql` script.) Why is this query so much faster?



– The function on the join conditions in the first query prevents the use of a normal index. Removing the functions allows the indexes to be used.

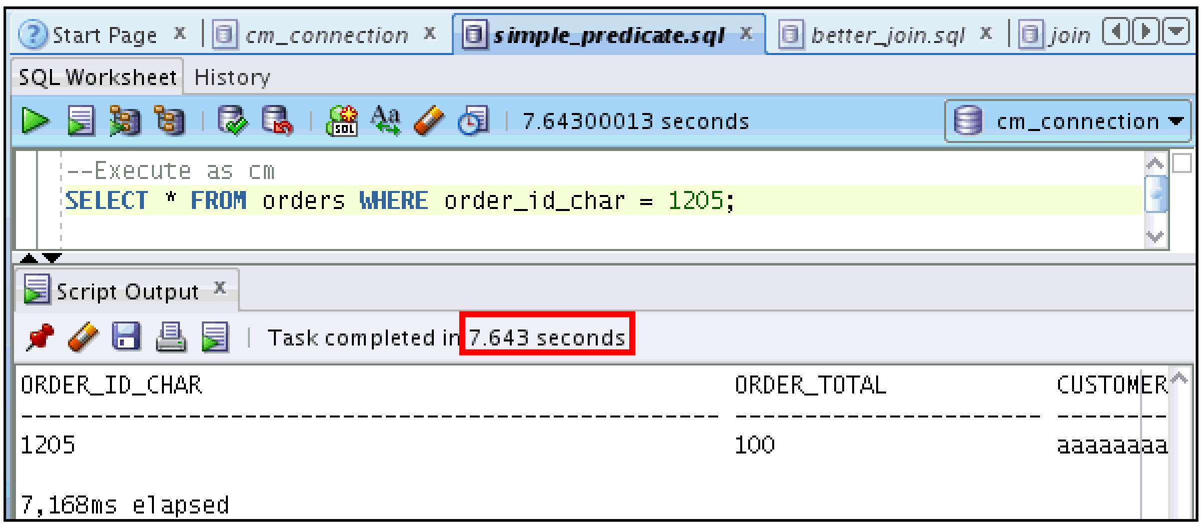
14. Before analyzing the third case, flush your environment again using the `flush.sql` script from your SQL*Plus session.



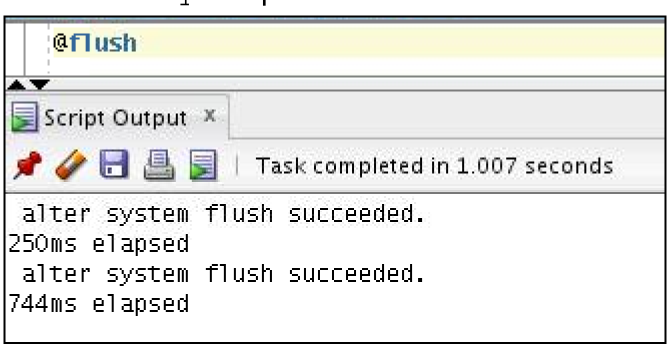
Case 3: Simple Predicate

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

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

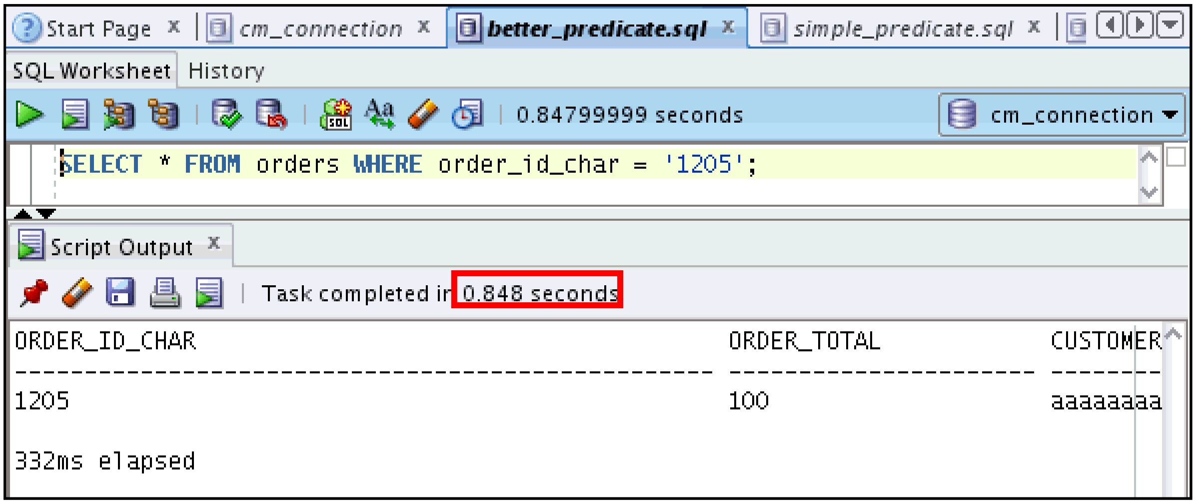


16. Before trying to fix the `SELECT` statement in step 20, flush your environment again using the `flush.sql` script.



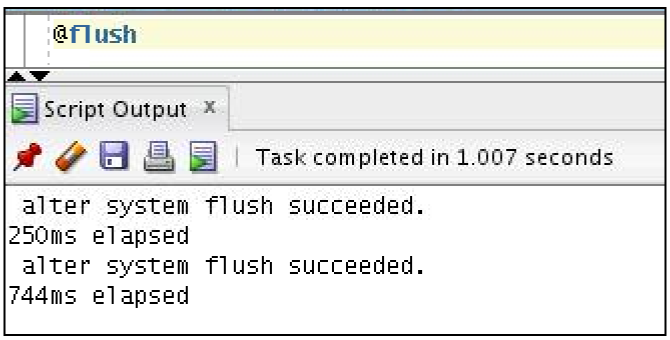
17. How would you rewrite the previous statement for better performance? Test your solution. You should discuss this with your instructor. (Alternatively, open and execute the `better_predicate.sql`.) Why does this query execute more quickly?

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



– This version runs faster because the data type `order_id_char` column matches the data type of the literal value.

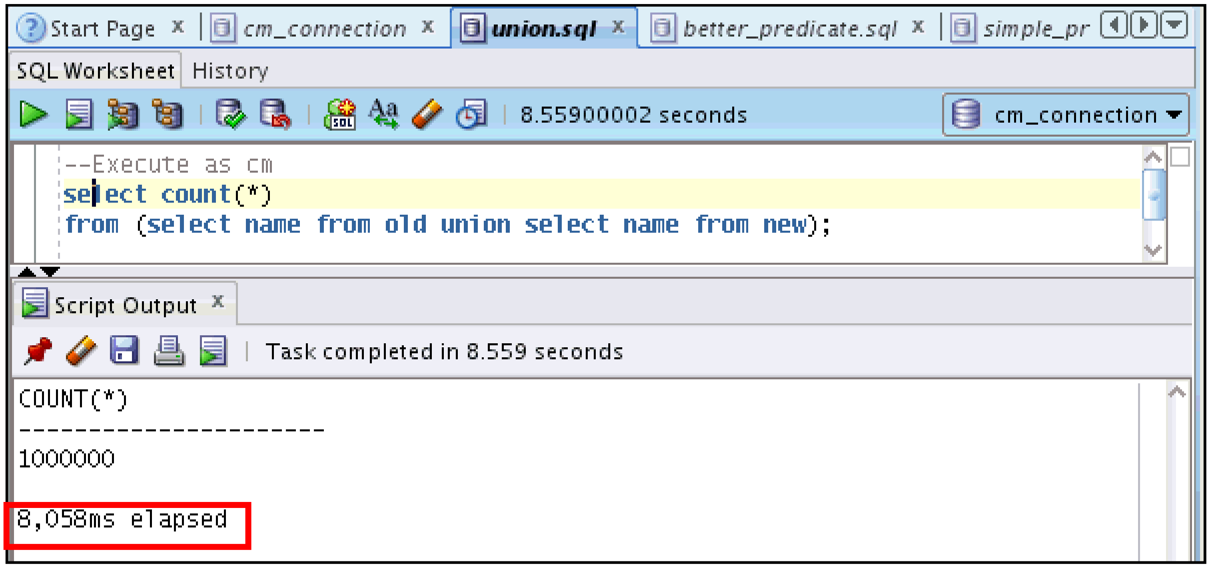
18. Before proceeding with the next analysis, flush your environment again using the `flush.sql` script.



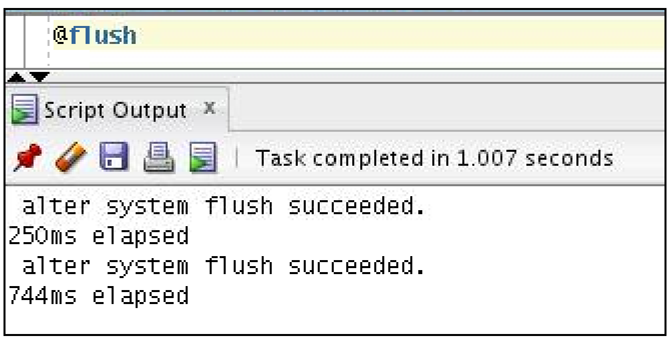
Case 4: Union

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

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



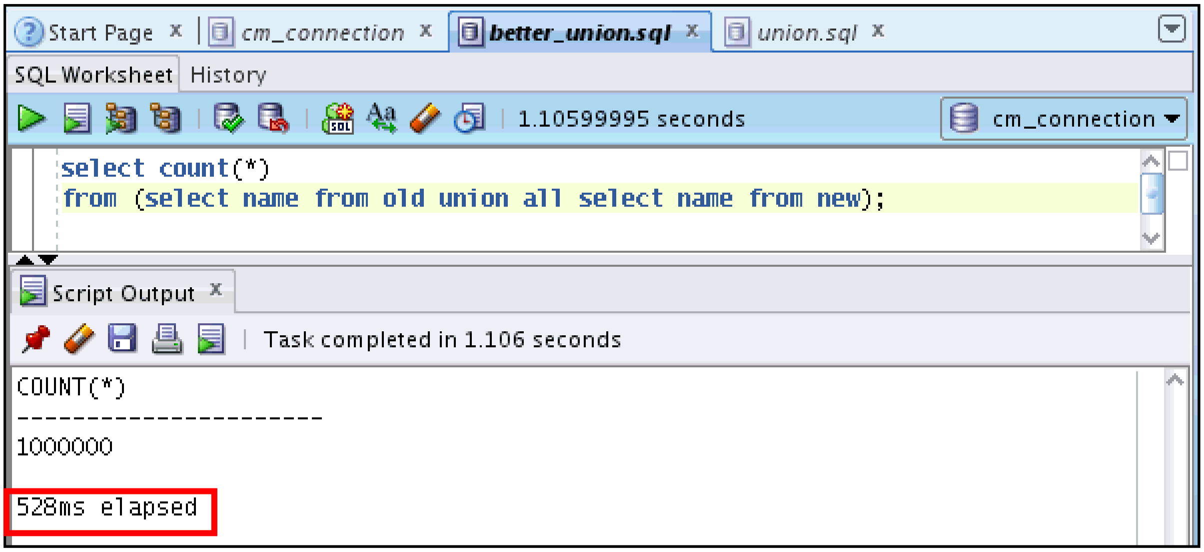
20. Before investigating a better solution, flush your environment again using the `flush.sql` script.



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

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

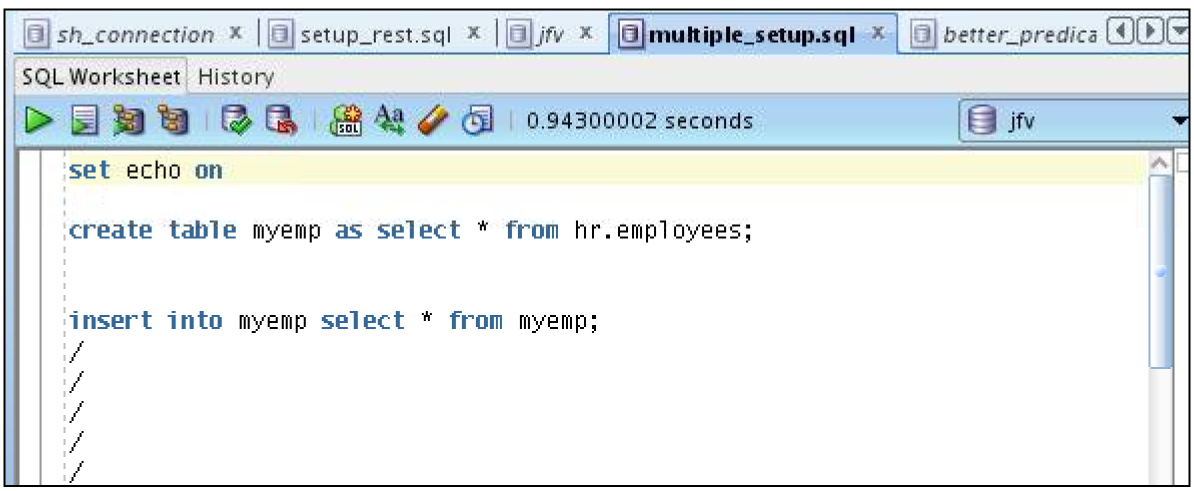
```
select count(*)
from (select name from old union all select name from new);
```



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

Case 5: Combining SQL Statements

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

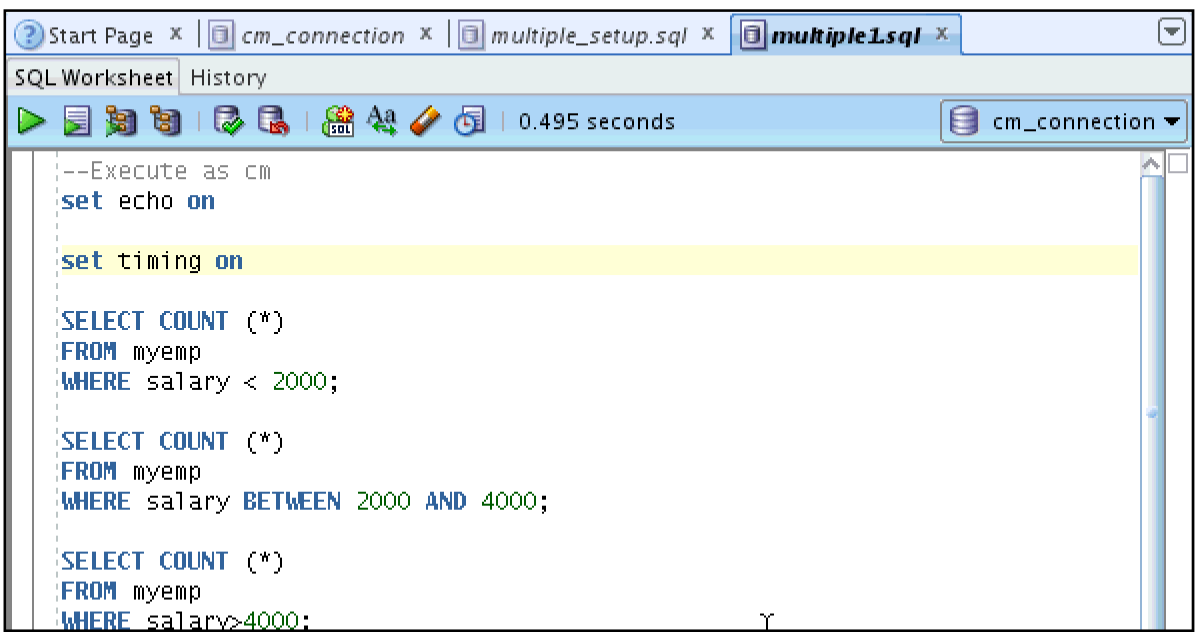


Output:

```
create table myemp as select * from hr.employees

create table succeeded.
397ms elapsed
insert into myemp select * from myemp

107 rows inserted
20ms elapsed
/
/
```

```
/
/
/
/
/
/
/
/
/
/
/
/
/
commit

commited
5ms elapsed
insert into myemp select * from myemp

214 rows inserted
3ms elapsed
commit

commited
2ms elapsed
```

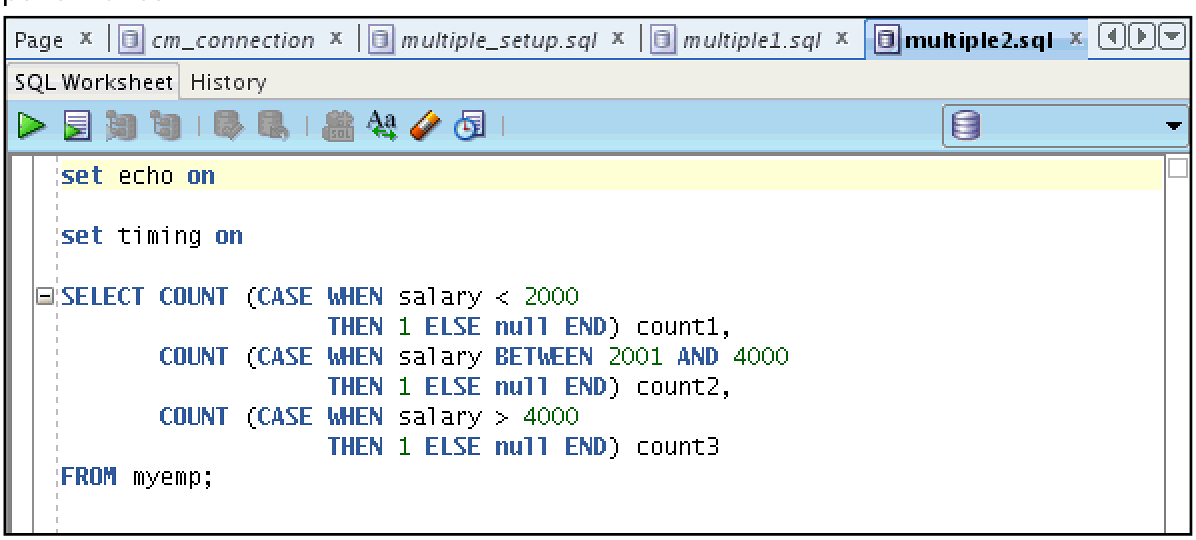
23. Open and execute the `multiple1.sql` script and note the total time it takes to execute.



Output:

```
set timing on

SELECT COUNT (*)
FROM myemp
WHERE salary < 2000

COUNT(*)
----------------------
0

7ms elapsed
SELECT COUNT (*)
FROM myemp
WHERE salary BETWEEN 2000 AND 4000

COUNT(*)
----------------------
172

4ms elapsed
SELECT COUNT (*)
FROM myemp
WHERE salary>4000

COUNT(*)
----------------------
256

8ms elapsed
set timing off
```

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



Output:

```
set timing on

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

COUNT1                 COUNT2                 COUNT3
---------------------- ---------------------- ----------------------
0                      172                    256

4ms elapsed
```

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

Case 6: Combining SQL Statements

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

```
$ cd $HOME/solutions/Common_Mistakes
$ ./bad_connect.sh
Wed Aug  4 06:54:28 GMT 2010
Wed Aug  4 06:55:07 GMT 2010
$

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

#!/bin/bash

cd /home/oracle/solutions/Common_Mistakes

STREAM_NUM=0
MAX_STREAM=500

date

while [ $STREAM_NUM -lt $MAX_STREAM ]; do

  # one more
  let STREAM_NUM="STREAM_NUM+1"
```

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

done

date

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


select count(*) from dba_users;
exit;
```

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

```
$ ./better_connect.sh
Wed Aug  4 06:57:12 GMT 2010
Wed Aug  4 06:57:14 GMT 2010
$

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

#!/bin/bash

cd /home/oracle/solutions/Common_Mistakes

date

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

date

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

declare
c number;
begin
for i in 1..500 loop
   select count(*) into c from dba_users;
end loop;
end;
/
exit;
```

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

Practices for Lesson 3

Chapter 3

Overview of Practices for Lesson 3

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.

Practice 3-1: Understanding Optimizer Decisions

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

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

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

```
$ cd $HOME/solutions/Trace_Event
$ ./te_setup.sh
…
SQL>
SQL> ALTER SESSION SET TRACEFILE_IDENTIFIER = 'MYOPTIMIZER';

Session altered.

SQL>
SQL> alter session set events '10053 trace name context forever, level
1';

Session altered.

SQL>
SQL> SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
  2  FROM   sh.sales s,sh.times t,sh.customers c,sh.channels ch
  3  WHERE  s.time_id = t.time_id AND
  4         s.cust_id = c.cust_id AND
  5         s.channel_id = ch.channel_id AND
  6         c.cust_state_province = 'CA' AND
  7         ch.channel_desc IN ('Internet','Catalog') AND
  8         t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
  9  GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc;

no rows selected

SQL> COMMIT;
```

```
Commit complete.

SQL>
SQL> exit;
Disconnected …
orcl_ora_28709_MYOPTIMIZER.trc
orcl_ora_28709_MYOPTIMIZER.trm
$
```

2. With the help of your instructor, see the generated trace file and interpret the important parts of the trace file. **Note:** This lab is only for demonstration purposes. Do *not* use it on your production system unless explicitly asked by Oracle Support Services.

   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.

```
$ cd $HOME/solutions/Trace_Event
$ cat myoptimizer.trc

Trace file
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_28709_MYOPTI
MIZER.trc
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 -
Production
With the Partitioning, OLAP, Data Mining and Real Application
Testing options
ORACLE_HOME = /u01/app/oracle/product/11.2.0/dbhome_1
System name:    Linux
Node name:      EDRSR10P1
Release:  2.6.18-164.el5
Version:  #1 SMP Thu Sep 3 02:16:47 EDT 2009
Machine:  i686
Instance name: orcl
Redo thread mounted by this instance: 1
Oracle process number: 39
Unix process pid: 28709, image: oracle@EDRSR10P1 (TNS V1-V3)



*** 2010-08-04 07:51:05.353
*** SESSION ID:(48.19088) 2010-08-04 07:51:05.353
*** CLIENT ID:() 2010-08-04 07:51:05.353
*** SERVICE NAME:(SYS$USERS) 2010-08-04 07:51:05.353
*** MODULE NAME:(sqlplus@EDRSR10P1 (TNS V1-V3)) 2010-08-04
07:51:05.353
```

```
*** ACTION NAME:() 2010-08-04 07:51:05.353

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


*** 2010-08-04 07:51:05.363
SPM: statement not found in SMB

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

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

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

```
CBQT - cost-based query transformation
JPPD - join predicate push-down
OJPPD - old-style (non-cost-based) JPPD
FPD - filter push-down
PM - predicate move-around
CVM - complex view merging
SPJ - select-project-join
SJC - set join conversion
SU - subquery unnesting
OBYE - order by elimination
OST - old style star transformation
ST - new (cbqt) star transformation
CNT - count(col) to count(*) transformation
JE - Join Elimination
JF - join factorization
SLP - select list pruning
DP - distinct placement
qb - query block
LB - leaf blocks
DK - distinct keys
LB/K - average number of leaf blocks per key
DB/K - average number of data blocks per key
CLUF - clustering factor
NDV - number of distinct values
Resp - response cost
Card - cardinality
Resc - resource cost
NL - nested loops (join)
SM - sort merge (join)
HA - hash (join)
CPUSPEED - CPU Speed
IOTFRSPEED - I/O transfer speed
IOSEEKTIM - I/O seek time
SREADTIM - average single block read time
MREADTIM - average multiblock read time
MBRC - average multiblock read count
MAXTHR - maximum I/O system throughput
SLAVETHR - average slave I/O throughput
dmeth - distribution method
  1: no partitioning required
  2: value partitioned
  4: right is random (round-robin)
```

```
  128: left is random (round-robin)
  8: broadcast right and partition left
  16: broadcast left and partition right
  32: partition left using partitioning of right
  64: partition right using partitioning of left
  256: run the join in serial
  0: invalid distribution method
sel - selectivity
ptn - partition
***************************************
PARAMETERS USED BY THE OPTIMIZER
********************************
  *************************************
  PARAMETERS WITH ALTERED VALUES
  ******************************
Compilation Environment Dump
_smm_min_size                       = 286 KB
_smm_max_size                       = 57344 KB
_smm_px_max_size                    = 143360 KB
Bug Fix Control Environment


  *************************************
  PARAMETERS WITH DEFAULT VALUES
  ******************************
Compilation Environment Dump
optimizer_mode_hinted               = false
optimizer_features_hinted           = 0.0.0
parallel_execution_enabled          = true
parallel_query_forced_dop           = 0
parallel_dml_forced_dop             = 0
parallel_ddl_forced_degree          = 0
parallel_ddl_forced_instances       = 0
_query_rewrite_fudge                = 90
optimizer_features_enable           = 11.2.0.1
_optimizer_search_limit             = 5
cpu_count                           = 1
…
parallel_hinted                     = none
_sql_compatibility                  = 0
_optimizer_use_feedback             = true
_optimizer_try_st_before_jppd       = true
```

```
Bug Fix Control Environment
    fix  3834770 = 1
    fix  3746511 = enabled
    fix  4519016 = enabled
    fix  3118776 = enabled
    fix  4488689 = enabled
    fix  2194204 = disabled
    …


  ***************************************
  PARAMETERS IN OPT_PARAM HINT
  ****************************
***************************************
Column Usage Monitoring is ON: tracking level = 1
***************************************

Considering Query Transformations on query block SEL$1 (#0)
**************************
Query transformations (QT)
**************************
CBQT: Validity checks passed for 70fqjd9u1zk7c.
CSE: Considering common sub-expression elimination in query
block SEL$1 (#0)
*************************
Common Subexpression elimination (CSE)
*************************
CSE:     CSE not performed on query block SEL$1 (#0).
OBYE:   Considering Order-by Elimination from view SEL$1 (#0)
***************************
Order-by elimination (OBYE)
***************************
OBYE:     OBYE bypassed: no order by to eliminate.
JE:   Considering Join Elimination on query block SEL$1 (#0)
*************************
Join Elimination (JE)
*************************
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
```

```
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
Query block SEL$1 (#0) unchanged
CNT:   Considering count(col) to count(*) on query block SEL$1
(#0)
*************************
Count(col) to Count(*) (CNT)
*************************
CNT:     COUNT() to COUNT(*) not done.
JE:   Considering Join Elimination on query block SEL$1 (#0)
*************************
Join Elimination (JE)
*************************
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
```

```
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
Query block SEL$1 (#0) unchanged
query block SEL$1 (#0) unchanged
Considering Query Transformations on query block SEL$1 (#0)
**************************
Query transformations (QT)
**************************
CSE: Considering common sub-expression elimination in query
block SEL$1 (#0)
*************************
Common Subexpression elimination (CSE)
*************************
CSE:     CSE not performed on query block SEL$1 (#0).
query block SEL$1 (#0) unchanged
apadrv-start sqlid=8087006336042125548
  :
    call(in-use=62772, alloc=81864), compile(in-use=73112,
alloc=77156), execution(in-use=3504, alloc=4060)

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

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

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

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

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

kkqctdrvTD-end:
    call(in-use=62772, alloc=81864), compile(in-use=115976,
alloc=117532), execution(in-use=3504, alloc=4060)

SJC: Considering set-join conversion in query block SEL$1 (#1)
*************************
Set-Join Conversion (SJC)
*************************
SJC: not performed
CNT:   Considering count(col) to count(*) on query block SEL$1
(#1)
*************************
Count(col) to Count(*) (CNT)
*************************
CNT:     COUNT() to COUNT(*) not done.
JE:   Considering Join Elimination on query block SEL$1 (#1)
*************************
```

```
Join Elimination (JE)
*************************
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
SQL:******* UNPARSED QUERY IS *******
SELECT "CH"."CHANNEL_CLASS" "CHANNEL_CLASS","C"."CUST_CITY"
"CUST_CITY","T"."CALENDAR_QUARTER_DESC"
"CALENDAR_QUARTER_DESC",SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S","SH"."TIMES" "T","SH"."CUSTOMERS"
"C","SH"."CHANNELS" "CH" WHERE "S"."TIME_ID"="T"."TIME_ID" AND
"S"."CUST_ID"="C"."CUST_ID" AND
"S"."CHANNEL_ID"="CH"."CHANNEL_ID" AND
"C"."CUST_STATE_PROVINCE"='CA' AND
("CH"."CHANNEL_DESC"='Internet' OR
"CH"."CHANNEL_DESC"='Catalog') AND
("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2') GROUP BY
"CH"."CHANNEL_CLASS","C"."CUST_CITY","T"."CALENDAR_QUARTER_DESC"
Query block SEL$1 (#1) unchanged
PM: Considering predicate move-around in query block SEL$1 (#1)
**************************
Predicate Move-Around (PM)
**************************
PM:     PM bypassed: Outer query contains no views.
PM:     PM bypassed: Outer query contains no views.
kkqctdrvTD-start on query block SEL$1 (#1)
kkqctdrvTD-start: :
    call(in-use=82748, alloc=98240), compile(in-use=118884,
alloc=121656), execution(in-use=3504, alloc=4060)

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

kkqctdrvTD-end:
```

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

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

Registered qb: SEL$1 0x610068 (COPY SEL$1)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature(): NULL
****************************************
 Cost-Based Group-By/Distinct Placement
****************************************
GBP/DP: Checking validity of GBP/DP for query block SEL$1 (#1)
GBP: Checking validity of group-by placement for query block
SEL$1 (#1)
GBP: Bypassed: QB has disjunction.
DP: Checking validity of distinct placement for query block
SEL$1 (#1)
DP: Bypassed: Query has invalid constructs.
kkqctdrvTD-cleanup: transform(in-use=9592, alloc=10596) :
    call(in-use=86820, alloc=98240), compile(in-use=139896,
alloc=146064), execution(in-use=3504, alloc=4060)

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

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

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

kkqctdrvTD-end:
```

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

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

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

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

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

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

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

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

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


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

```
kkoqbc-subheap (create addr=0x61b144)
****************
QUERY BLOCK TEXT
****************
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
SUM(s.amount_sold) sales_amount
FROM   sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE  s.time_id = t.time_id AND
       s.cust_id = c.cust_id AND
       s.channel_id = ch.channel_id
---------------------
QUERY BLOCK SIGNATURE
---------------------
signature (optimizer): qb_name=SEL$1 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-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2'
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-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc
sql_text_length=473
sql=SELECT ch.channel_class, c.cust_city,
t.calendar_quarter_desc, SUM(s.amount_sold) sales_amount
FROM   sh.sales s,sh.times t,sh.customers c,sh.channels ch
WHERE  s.time_id = t.time_id AND
       s.cust_id = c.cust_id AND
       s.channel_id = ch.channel_id
sql=AND
       c.cust_state_province = 'CA' AND
       ch.channel_desc IN ('Internet','Catalog') AND
       t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc
----- Explain Plan Dump -----
----- Plan Table -----

============
Plan Table
============
---------------------------------------------------+-----------
-----------------------+--------------+
| Id  | Operation                        | Name     | Rows  |
Bytes | Cost  | Time       | Pstart| Pstop |
```

```
-----------------------------------------------------+-----------
-----------------------+--------------+
| 0   | SELECT STATEMENT                  |          |       |
|   921 |           |       |       |
| 1   |  HASH GROUP BY                    |          |   572 |
47K |   921 |  00:00:12 |       |       |
| 2   |   HASH JOIN                       |          |  3116 |
256K |   920 |  00:00:12 |       |       |
| 3   |    TABLE ACCESS FULL              | CHANNELS |     2 |
42 |     3 |  00:00:01 |       |       |
| 4   |    HASH JOIN                      |          |  6231 |
383K |   916 |  00:00:11 |       |       |
| 5   |     PART JOIN FILTER CREATE       | :BF0000  |   183 |
2928 |    18 |  00:00:01 |       |       |
| 6   |      TABLE ACCESS FULL            | TIMES    |   183 |
2928 |    18 |  00:00:01 |       |       |
| 7   |     HASH JOIN                     |          |   49K |
2287K |   897 |  00:00:11 |       |       |
| 8   |      TABLE ACCESS FULL            | CUSTOMERS|   383 |
9958 |   405 |  00:00:05 |       |       |
| 9   |      PARTITION RANGE JOIN-FILTER |          |  897K |
18M |   489 |  00:00:06 | :BF0000| :BF0000|
| 10  |       TABLE ACCESS FULL           | SALES    |  897K |
18M |   489 |  00:00:06 | :BF0000| :BF0000|
-----------------------------------------------------+-----------
-----------------------+--------------+
Predicate Information:
----------------------
2 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
3 - filter(("CH"."CHANNEL_DESC"='Catalog' OR
"CH"."CHANNEL_DESC"='Internet'))
4 - access("S"."TIME_ID"="T"."TIME_ID")
6 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
"T"."CALENDAR_QUARTER_DESC"='1999-Q2'))
7 - access("S"."CUST_ID"="C"."CUST_ID")
8 - filter("C"."CUST_STATE_PROVINCE"='CA')


Content of other_xml column
===========================
nodeid/pflags: 10 513nodeid/pflags: 9 513  db_version     :
11.2.0.1
  parse_schema   : SYS
  plan_hash      : 593420798
  plan_hash_2    : 1128146253
  Outline Data:
```

```
  /*+
    BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('11.2.0.1')
      DB_VERSION('11.2.0.1')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      FULL(@"SEL$1" "C"@"SEL$1")
      FULL(@"SEL$1" "S"@"SEL$1")
      FULL(@"SEL$1" "T"@"SEL$1")
      FULL(@"SEL$1" "CH"@"SEL$1")
      LEADING(@"SEL$1" "C"@"SEL$1" "S"@"SEL$1" "T"@"SEL$1"
"CH"@"SEL$1")
      USE_HASH(@"SEL$1" "S"@"SEL$1")
      USE_HASH(@"SEL$1" "T"@"SEL$1")
      USE_HASH(@"SEL$1" "CH"@"SEL$1")
      PX_JOIN_FILTER(@"SEL$1" "T"@"SEL$1")
      SWAP_JOIN_INPUTS(@"SEL$1" "T"@"SEL$1")
      SWAP_JOIN_INPUTS(@"SEL$1" "CH"@"SEL$1")
      USE_HASH_AGGREGATION(@"SEL$1")
    END_OUTLINE_DATA
  */

Optimizer state dump:
Compilation Environment Dump
optimizer_mode_hinted              = false
optimizer_features_hinted          = 0.0.0
parallel_execution_enabled         = true
parallel_query_forced_dop          = 0
parallel_dml_forced_dop            = 0
parallel_ddl_forced_degree         = 0
parallel_ddl_forced_instances      = 0
_query_rewrite_fudge               = 90
optimizer_features_enable          = 11.2.0.1
…

Bug Fix Control Environment
    fix  3834770 = 1
    fix  3746511 = enabled
    fix  4519016 = enabled
    fix  3118776 = enabled
    fix  4488689 = enabled
    fix  2194204 = disabled
```

```
…

Query Block Registry:
SEL$1 0x5c3a04 (PARSER) [FINAL]

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

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

$
```

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

```
$ cd $HOME/solutions/Trace_Event
$ ./te_cleanup.sh
…
SQL>
SQL> exit;
…
$
```

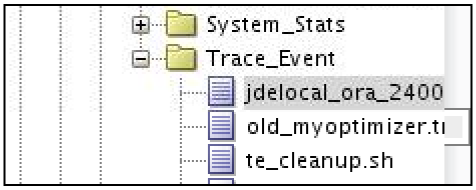
Practice 3-2: Working with Trace Files in SQL Developer

Overview

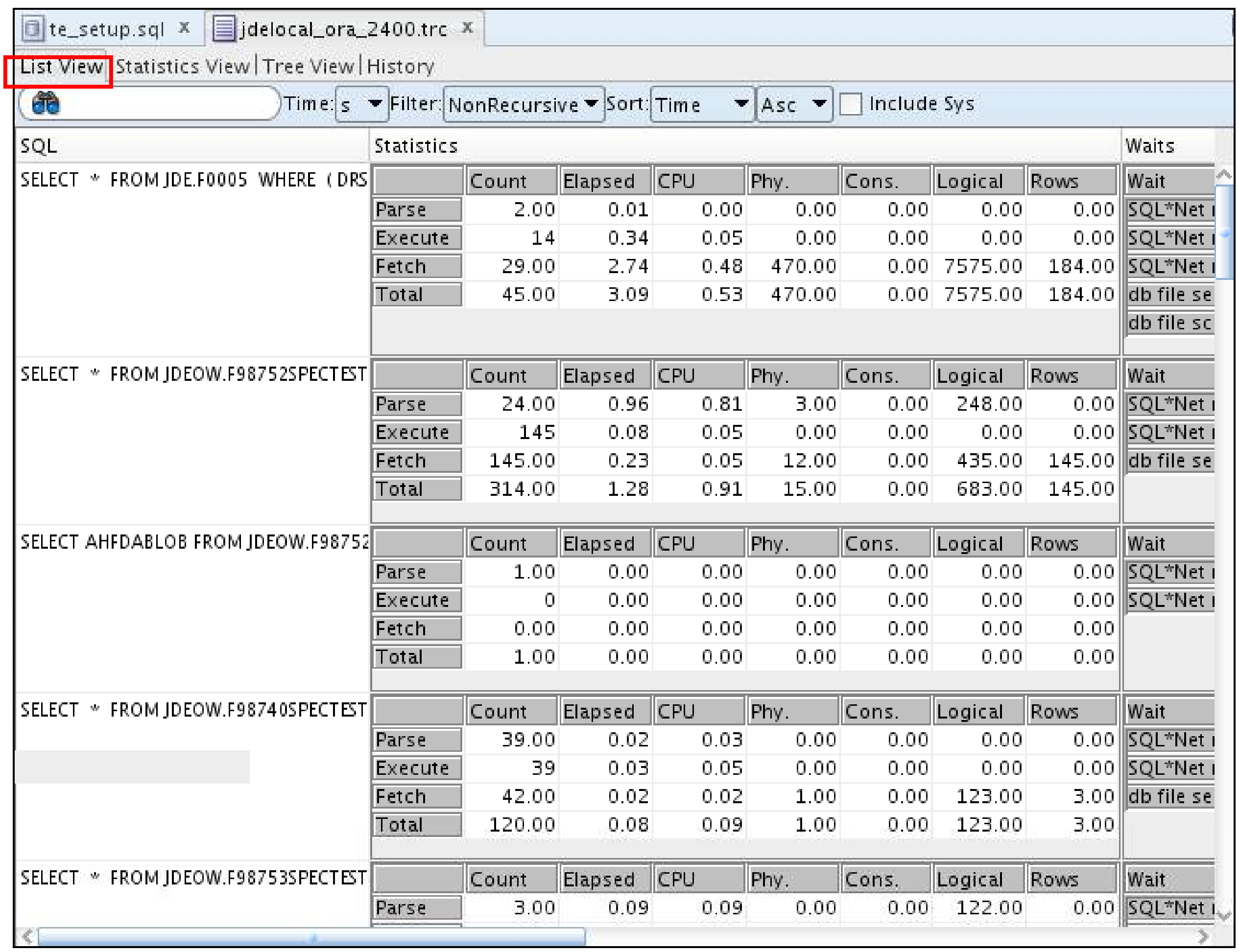
In this practice, you review a SQL trace file in SQL Developer

Tasks

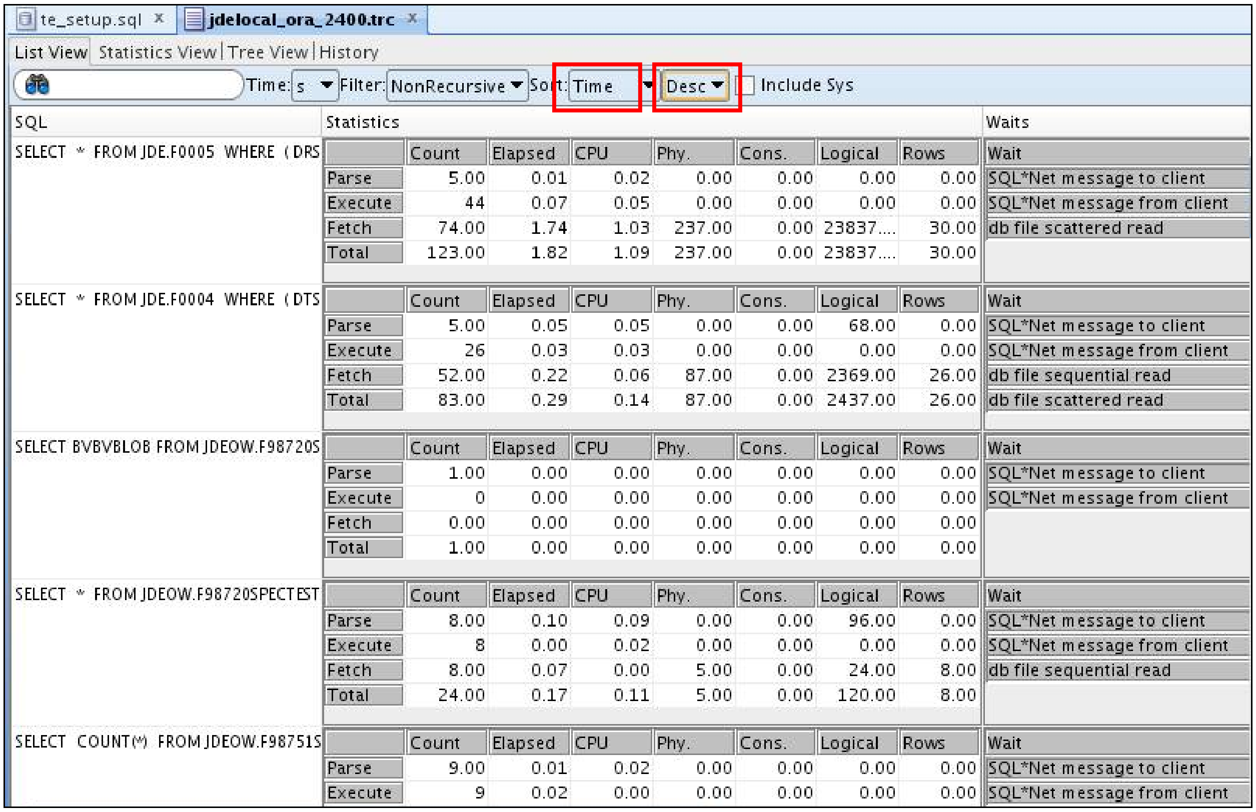
1. Open a large trace file, **`jdelocal_ora_2400.trc`**, in SQL Developer. View the trace file.
   a. Open the `$HOME/solutions/Trace_Event/jdelocal_ora_2400.trc` file.



   b. View its content in the List View. A report containing the trace data in a tabular form is displayed.

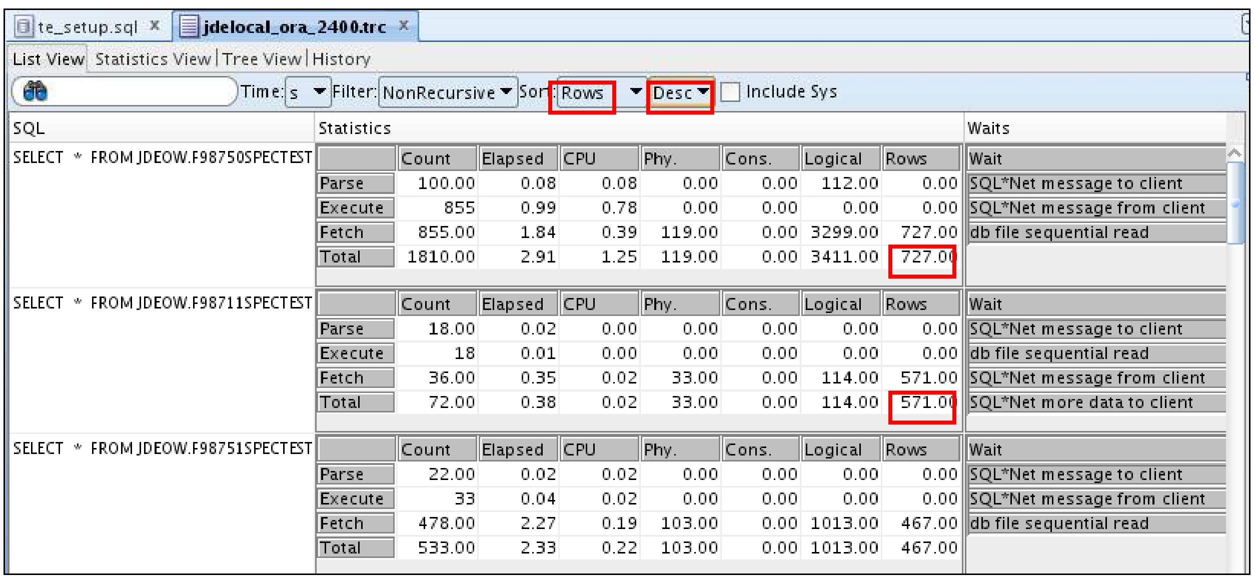


2. Display the slowest transaction by sorting the report by Time and in Desc order.



SQL		Count	Elapsed	CPU	Phy.	Cons.	Logical	Rows	Wait
SELECT * FROM JDE.F0005 WHERE (DRS	Parse	5.00	0.01	0.02	0.00	0.00	0.00	0.00	SQL*Net message to client
	Execute	44	0.07	0.05	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	74.00	1.74	1.03	237.00	0.00	23837....	30.00	db file scattered read
	Total	123.00	1.82	1.09	237.00	0.00	23837....	30.00	
SELECT * FROM JDE.F0004 WHERE (DTS	Parse	5.00	0.05	0.05	0.00	0.00	68.00	0.00	SQL*Net message to client
	Execute	26	0.03	0.03	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	52.00	0.22	0.06	87.00	0.00	2369.00	26.00	db file sequential read
	Total	83.00	0.29	0.14	87.00	0.00	2437.00	26.00	db file scattered read
SELECT BVBVBLOB FROM JDEOW.F98720S	Parse	1.00	0.00	0.00	0.00	0.00	0.00	0.00	SQL*Net message to client
	Execute	0	0.00	0.00	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Total	1.00	0.00	0.00	0.00	0.00	0.00	0.00	
SELECT * FROM JDEOW.F98720SPECTEST	Parse	8.00	0.10	0.09	0.00	0.00	96.00	0.00	SQL*Net message to client
	Execute	8	0.00	0.02	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	8.00	0.07	0.00	5.00	0.00	24.00	8.00	db file sequential read
	Total	24.00	0.17	0.11	5.00	0.00	120.00	8.00	
SELECT COUNT(*) FROM JDEOW.F98751S	Parse	9.00	0.01	0.02	0.00	0.00	0.00	0.00	SQL*Net message to client
	Execute	9	0.02	0.00	0.00	0.00	0.00	0.00	SQL*Net message from client

3. Display the transaction that fetched the highest number of rows by sorting the report by Rows and in Desc order.



SQL		Count	Elapsed	CPU	Phy.	Cons.	Logical	Rows	Wait
SELECT * FROM JDEOW.F98750SPECTEST	Parse	100.00	0.08	0.08	0.00	0.00	112.00	0.00	SQL*Net message to client
	Execute	855	0.99	0.78	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	855.00	1.84	0.39	119.00	0.00	3299.00	727.00	db file sequential read
	Total	1810.00	2.91	1.25	119.00	0.00	3411.00	727.00	
SELECT * FROM JDEOW.F98711SPECTEST	Parse	18.00	0.02	0.00	0.00	0.00	0.00	0.00	SQL*Net message to client
	Execute	18	0.01	0.00	0.00	0.00	0.00	0.00	db file sequential read
	Fetch	36.00	0.35	0.02	33.00	0.00	114.00	571.00	SQL*Net message from client
	Total	72.00	0.38	0.02	33.00	0.00	114.00	571.00	SQL*Net more data to client
SELECT * FROM JDEOW.F98751SPECTEST	Parse	22.00	0.02	0.02	0.00	0.00	0.00	0.00	SQL*Net message to client
	Execute	33	0.04	0.02	0.00	0.00	0.00	0.00	SQL*Net message from client
	Fetch	478.00	2.27	0.19	103.00	0.00	1013.00	467.00	db file sequential read
	Total	533.00	2.33	0.22	103.00	0.00	1013.00	467.00	

Practices for Lesson 4

Chapter 4

Overview of Practices for Lesson 4

Practices Overview

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

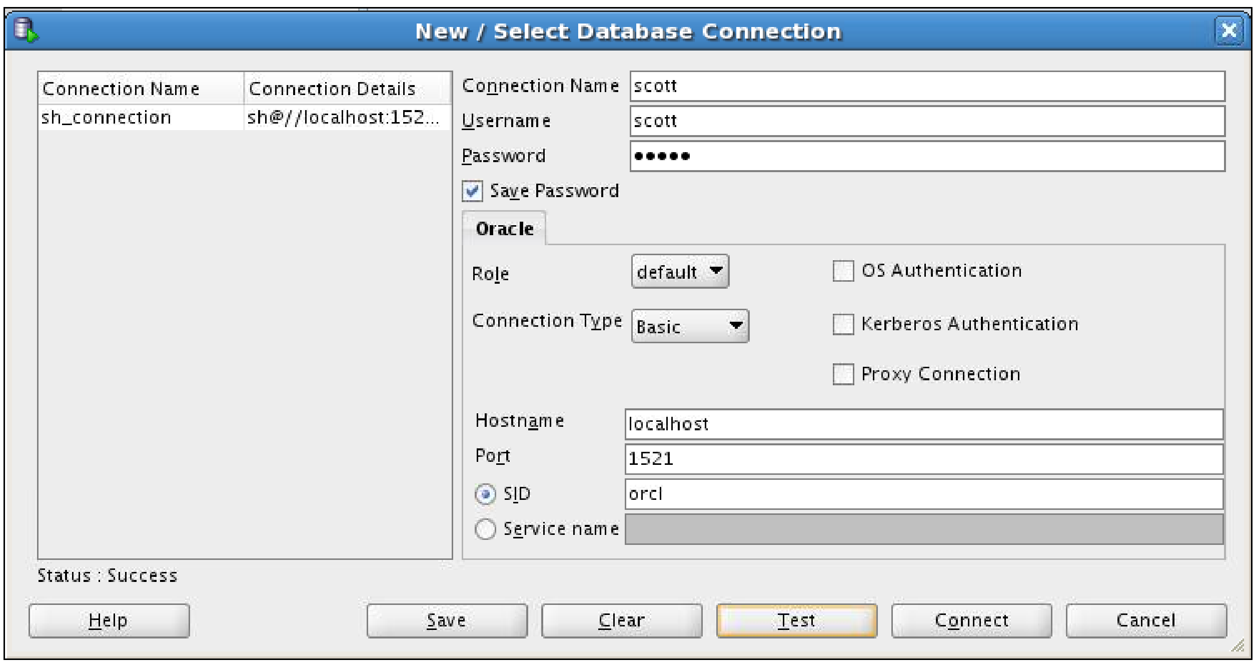
Practice 4-1: Extracting an Execution Plan Using SQL Developer

Overview

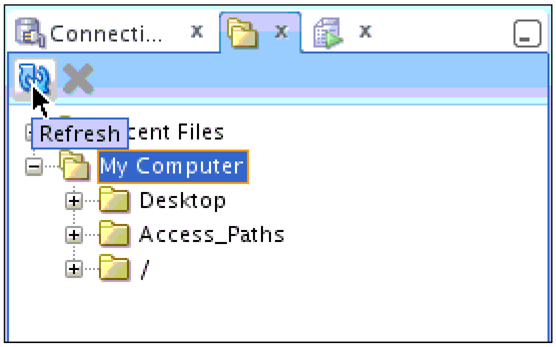
In this practice, you use SQL Developer to display the explain plan of a SQL query, and then use the Autotrace option.

Tasks

1. Create a connection for the `SCOTT` schema.
   a. Click the Add Connection button.
   b. Create a connection with the following specifications:
      Connection Name: `scott`
      Username: `scott`
      Password: `tiger`
      Select Save Password.
      `Sid: orcl`
      Click Test.
      Click Connect.

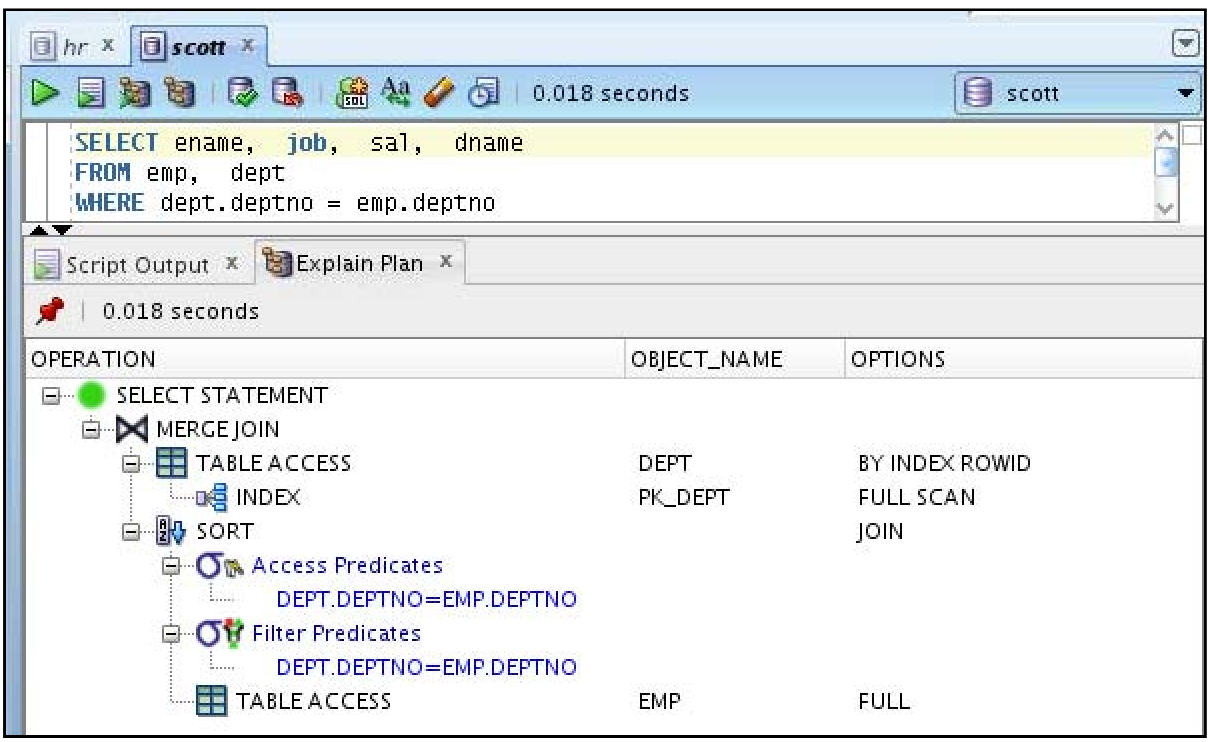


2. The scripts for this practice are in the `/home/oracle/solutions/Explain_Plan` directory. Set the SQL worksheet preferences to set the default directory.
   a. From the menu, select Tools > Preferences, expand the Database item, and select Worksheet. On the Worksheet, browse for the `/home/oracle/solutions/Explain_Plan` directory. Click Open, and then click OK.
   b. In the navigator pane, click the files tab. Select My Computer and click the Refresh button. The default Worksheet directory appears in the list.



3. Display the explain plan for the following query:
   a. On the SQL worksheet, enter the following command:
```
SELECT ename,  job,  sal,  dname
FROM emp,  dept
WHERE dept.deptno = emp.deptno
```
   You may instead open the `ep_join.sql` script in the `/home/oracle/solutions/Explain_Plan` directory.
   b. Click the **Explain Plan** button.

   c. Note the explain plan.

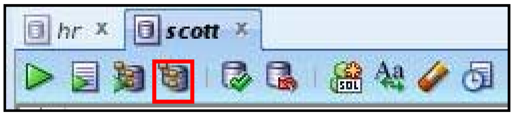


4. Change the query to the following and note the difference in the explain plan. You may instead open the `ep_not_exists.sql` script.
```
SELECT ename,  job,  sal,  dname
FROM emp,  dept
```

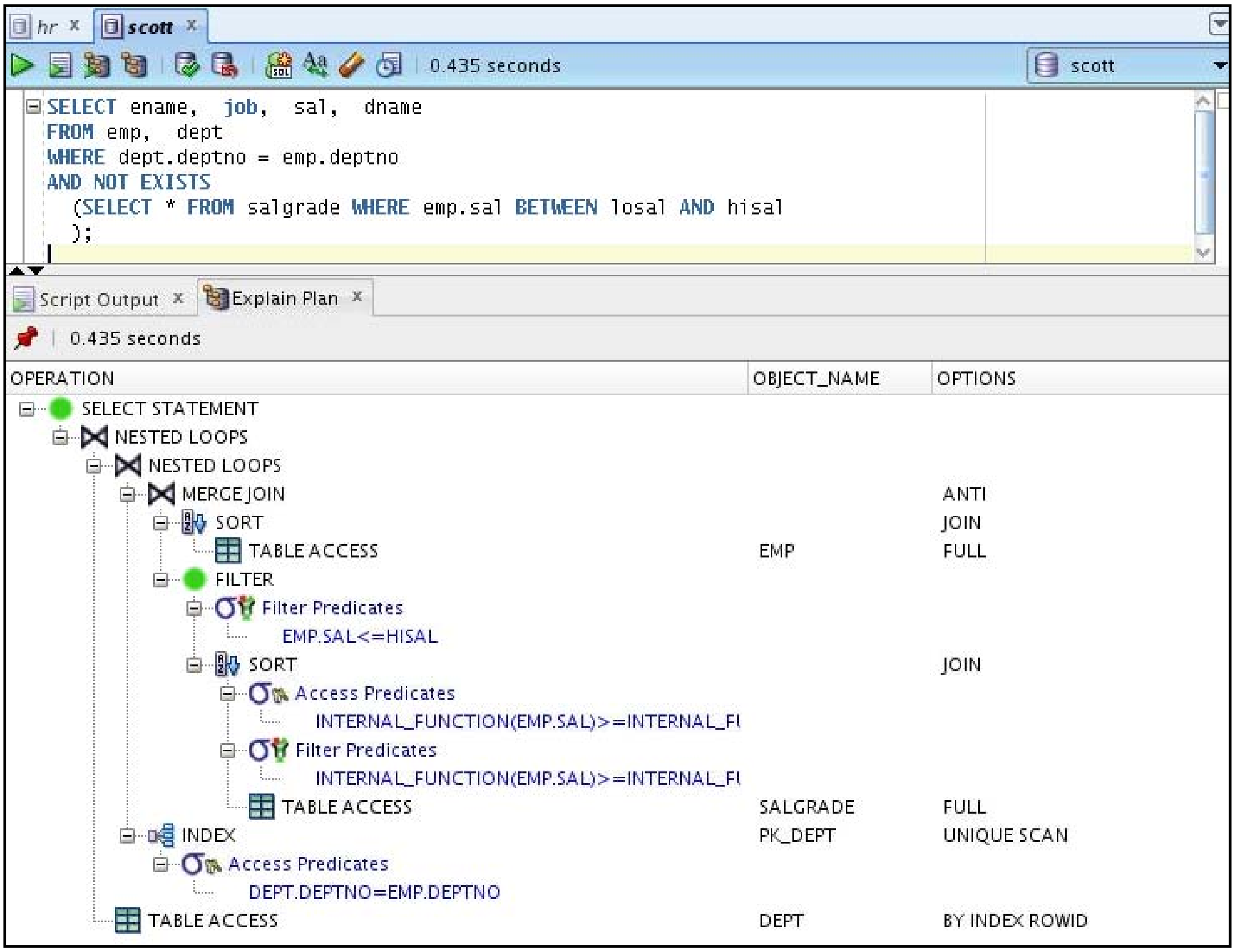
```
WHERE dept.deptno = emp.deptno
and not exists
(select * from salgrade where emp.sal between losal and hisal);
```

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

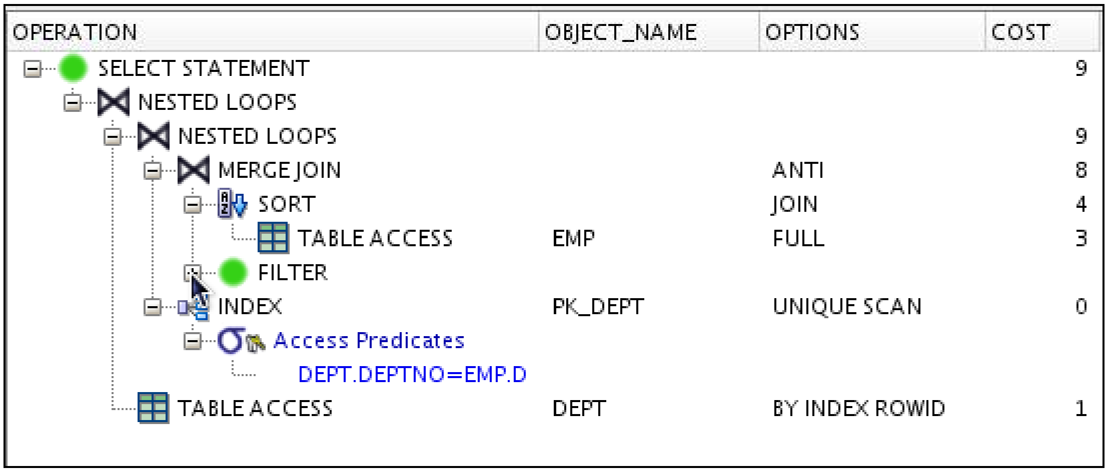
b. Click the **Explain Plan** button.



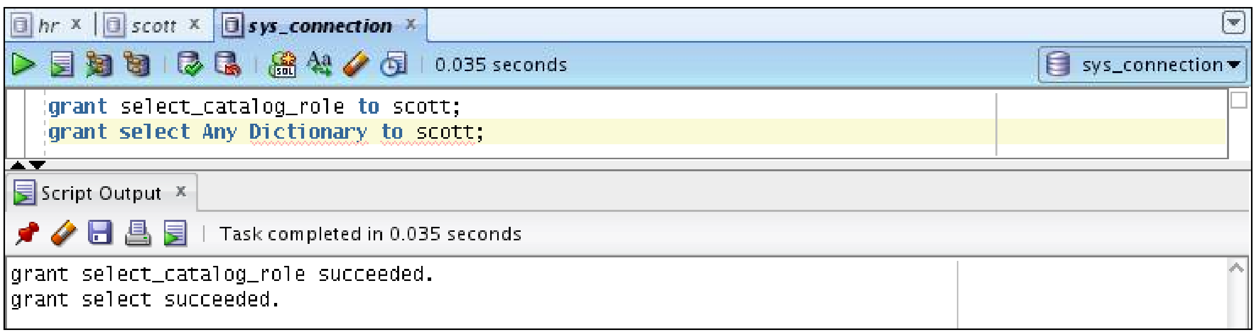
c. Note the difference in the explain plan.



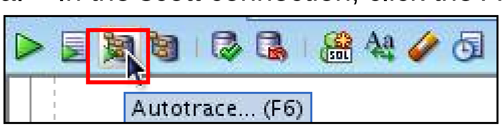
d. Collapse the information about the filters.



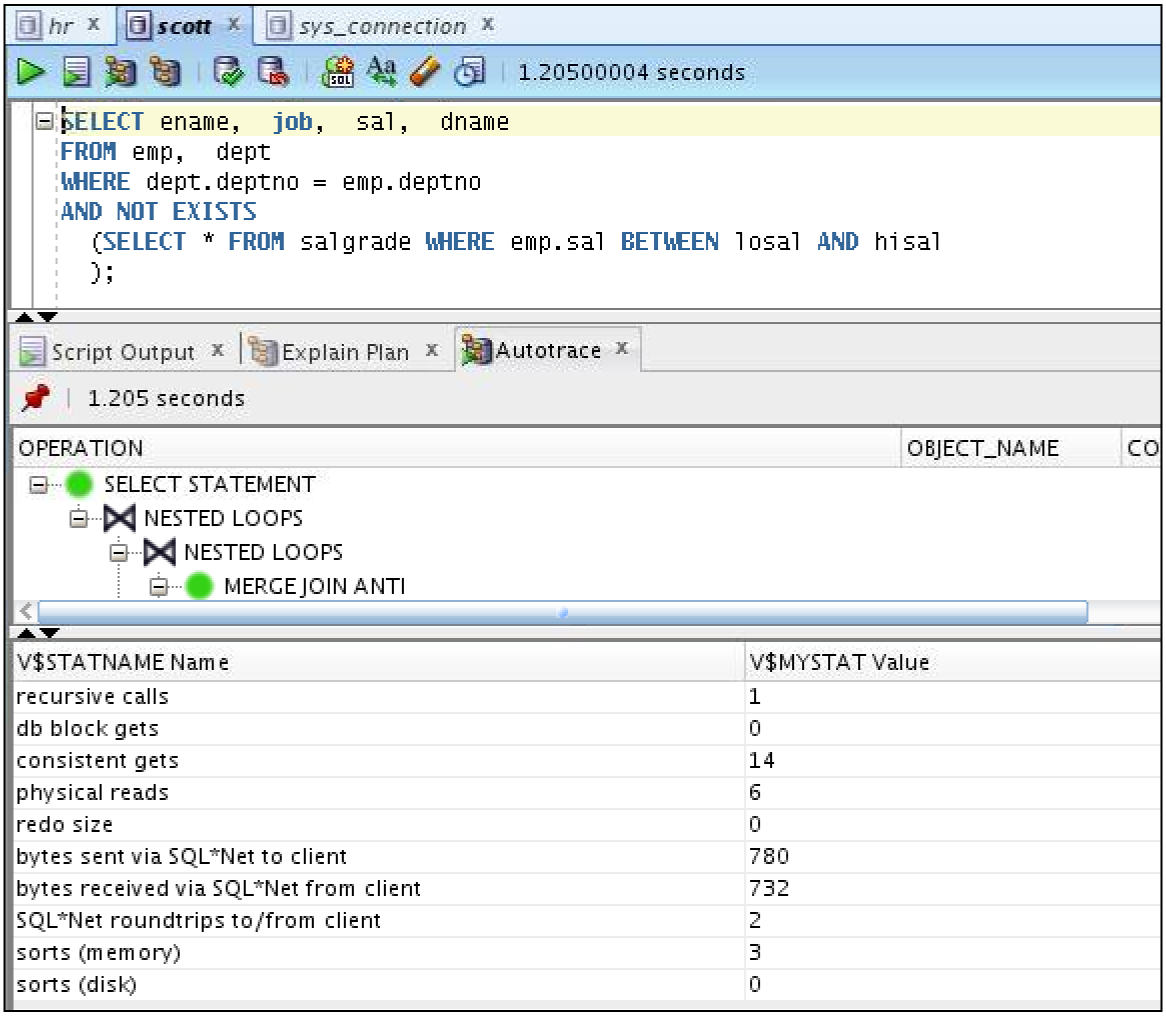
5. From `sys_connection`, grant `select_catalog_role` and `Select Any Dictionary` to `scott`.



6. Display the Autotrace for the query in the `ep_not_exists.sql` script.
   a. In the scott connection, click the Autotrace button.

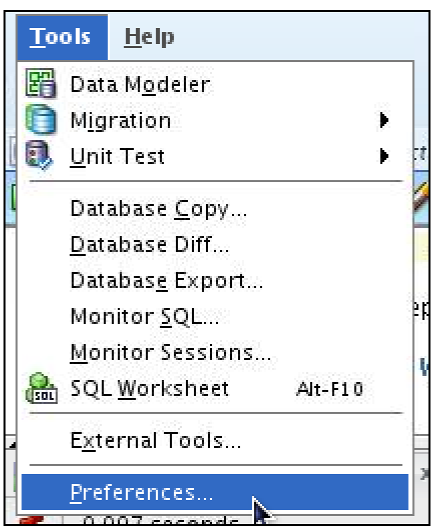


   b. Note the Autotrace output tab.

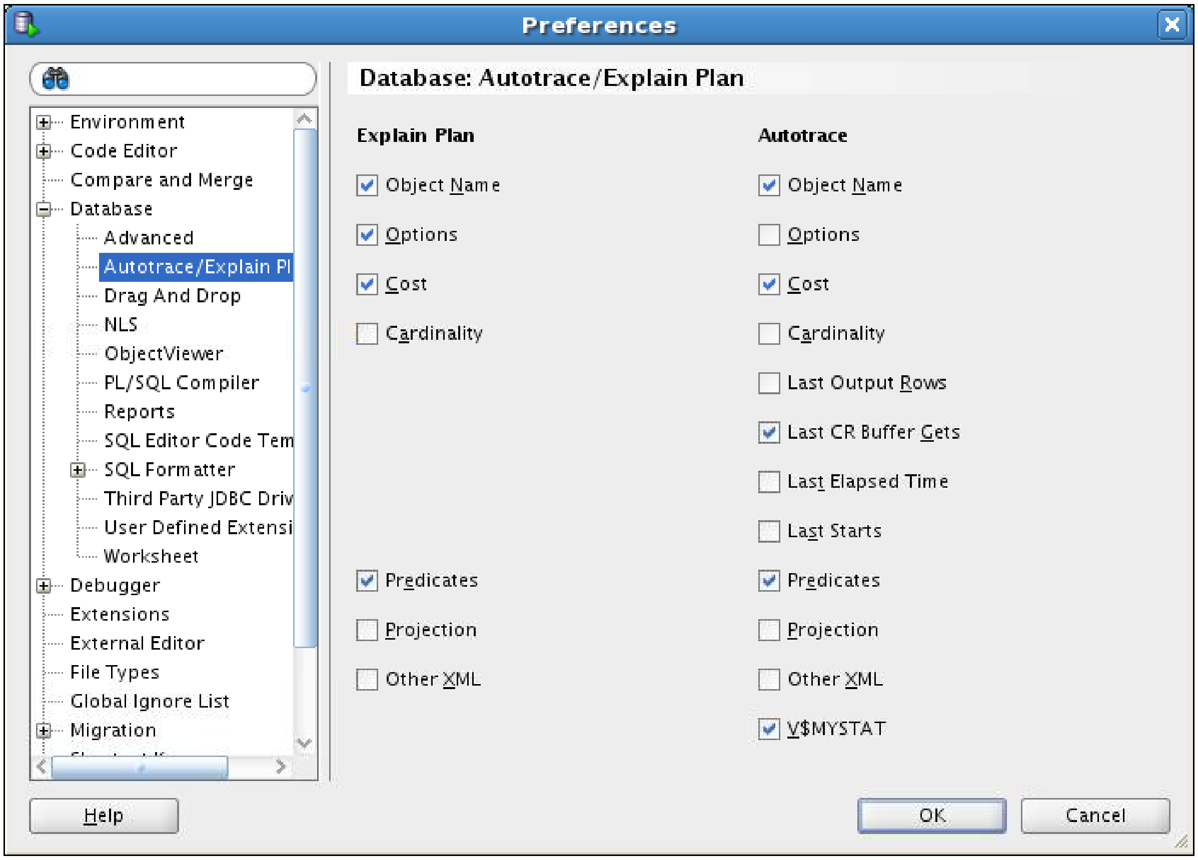


7. Customize the Explain Plan by adding Cardinality.

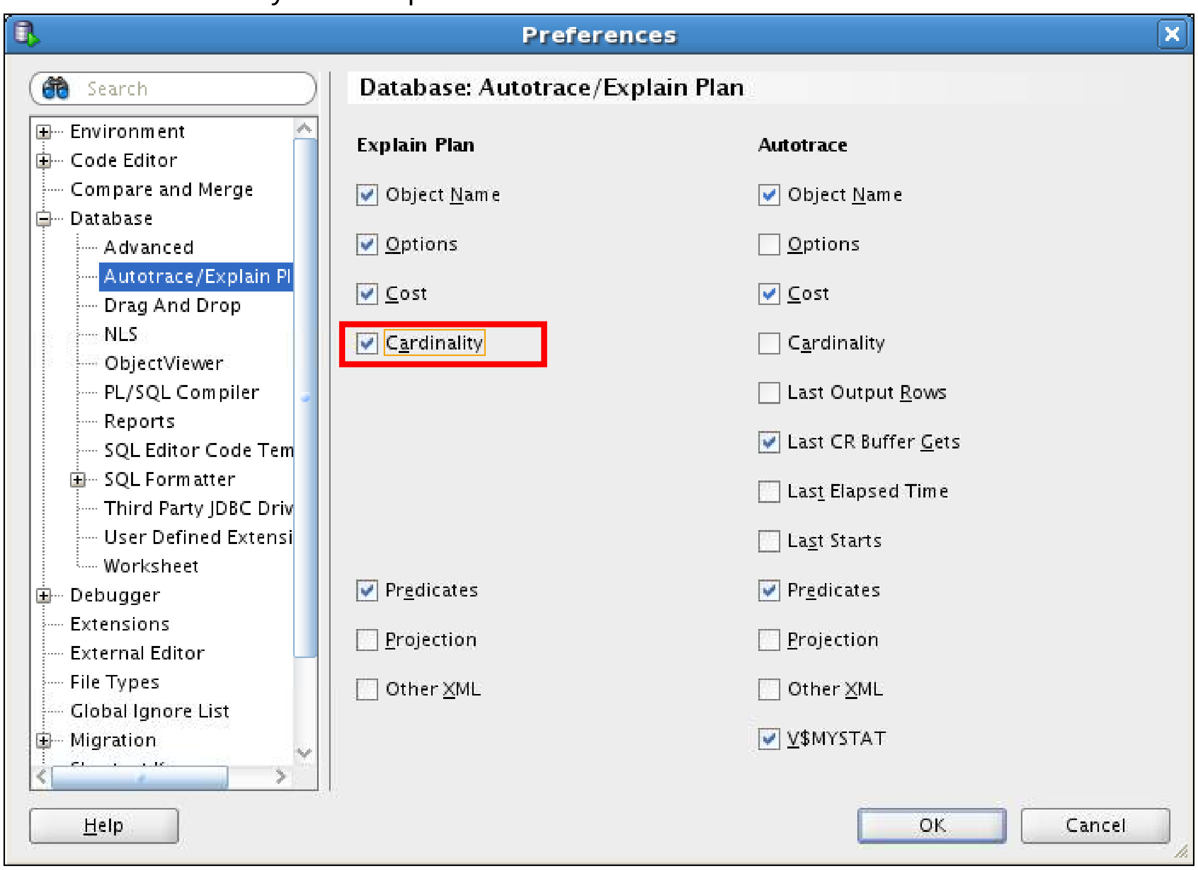
   a. Select Tools > Preferences.



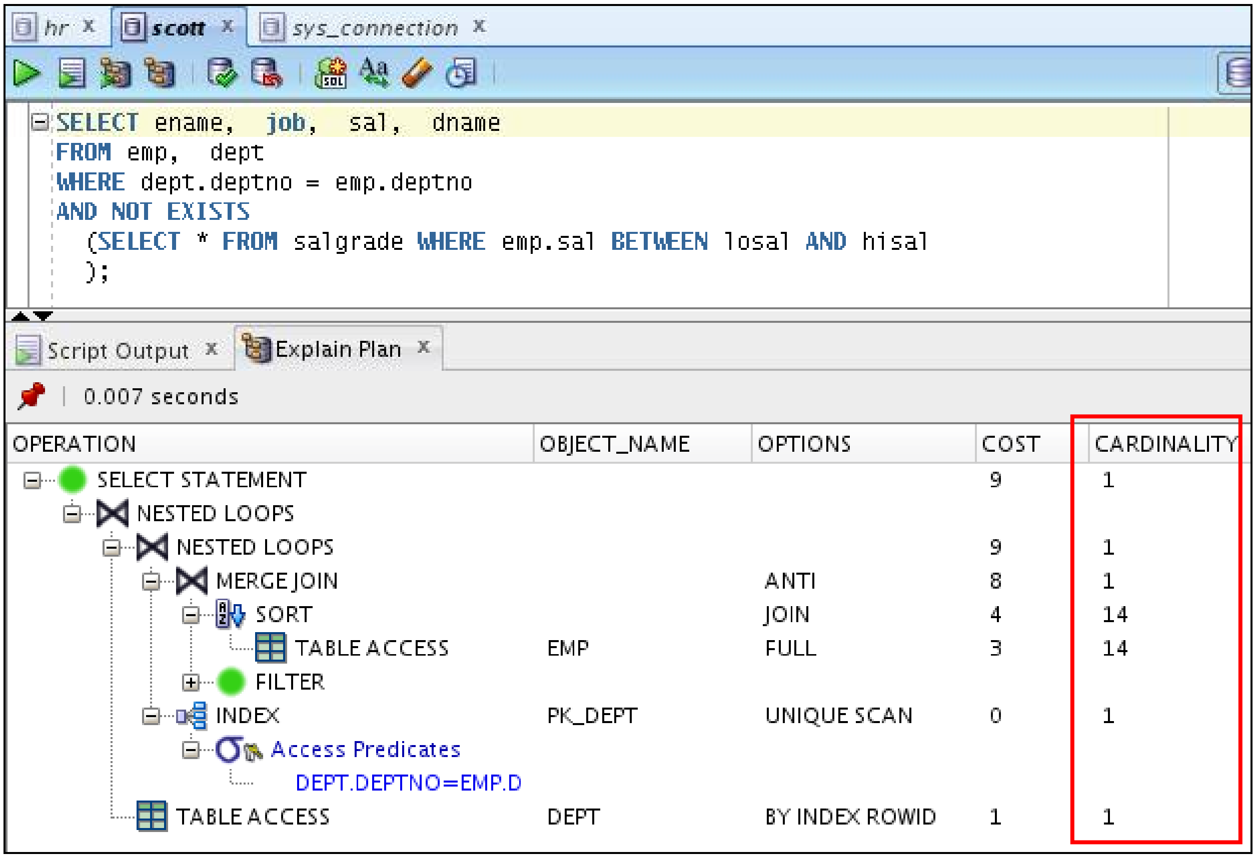
   b. Click Database > Autotrace/Explain Plan.



c. Select Cardinality in the Explain Plan column and click OK.



The Explain Plan now displays Cardinality.



Practice 4-2: 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 lab can be found in your `$HOME/solutions/Explain_Plan` directory.

1. Connected as the `oracle` user from a terminal session, change the working directory to `$HOME/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 lab.

```
$ cd $HOME/solutions/Explain_Plan
$ ./ep_startup.sh

…

SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> alter system flush buffer_cache;

System altered.

SQL>
SQL> set echo off
Disconnected …

$

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

#!/bin/bash

cd /home/oracle/solutions/Explain_Plan

sqlplus ep/ep @ep_startup.sql

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

set echo on

alter system flush shared_pool;

alter system flush buffer_cache;
```

```
set echo off
set term off

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

exit;
```

2. From the same terminal session (referred to as session 1 in the rest of this lab), 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

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

#!/bin/bash

cd /home/oracle/solutions/Explain_Plan

sqlplus ep/ep @ep_session_issue.sql

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

set echo off
set termout off

alter session set optimizer_mode=rule;

set termout on
set echo on

set timing on

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

exit;
```

3. From a second terminal session (referred to as session 2 in the rest of this lab), connect as the `oracle` user. After this, connect to a SQL*Plus session as the `SYS` user. From that

SQL*Plus session, be ready to use SQL Monitoring to monitor the execution plan used by session 1. You can execute the `ep_monitoring.sql` script for that purpose. Enter the command, but do not execute it yet. **Note:** Ensure that you understand the coordination between both sessions by prereading steps 4 and 5 before you continue.

```
Session 2:
----------

$ sqlplus /as sysdba
…
SQL>  @ep_monitor.sql

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

set echo on
set long 10000000
set longchunksize 10000000
set linesize 200
set pagesize 1000

exec dbms_lock.sleep(8);

select
dbms_sqltune.report_sql_monitor(sql_id=>'dkz7v96ym42c6',report_l
evel=>'ALL') from dual;
```

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

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.1.0.0 -
Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining
and Real Application Testing options

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

5. In session 2, press the return 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 done 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 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> 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_l
evel=>'ALL') from dual;

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',SESSION_
ID=>:SESSID,REPORT_LEVEL=>'ALL')
----------------------------------------------------------------
Global Information
------------------------------
 Status              :  DONE (ALL ROWS)
 Instance ID         :  1
 Session             :  EP (48:19978)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777217
 Execution Started   :  06/18/2010 12:24:27
 First Refresh Time  :  06/18/2010 12:24:36
 Last Refresh Time   :  06/18/2010 12:25:13
 Duration            :  46s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
```

```
 Program              :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)
 Fetch Calls          :  1

Global Stats
=================================================
| Elapsed |   Cpu   |  Other   | Fetch | Buffer |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  |
=================================================
|      47 |      32 |       14 |     1 |   780K |
=================================================


SQL Plan Monitoring Details (Plan Hash Value=1643938535)
================================================================
================================================================
=
| Id |      Operation       |    Name     |  Rows   | Cost |
Time    | Start  | Execs |   Rows    | Activity | Activity Detail
|
|    |                      |             | (Estim) |      |
Active(s) | Active | | (Actual) |   (%)    |   (# samples)    |
================================================================
================================================================
=
|  0 | SELECT STATEMENT     |             |         |      |
38 |     +9 |     1 |   1 |          |                  |
|  1 |   SORT AGGREGATE     |             |         |      |
40 |     +7 |     1 |   1 |     6.38 | Cpu (3)          |
|  2 |    NESTED LOOPS      |             |         |      |
38 |     +9 |     1 |     400M |              |                 |
|  3 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
38 |     +9 |     1 |    20000 |              |                 |
|  4 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
47 |     +1 | 20000 |     400M |        93.62 | Cpu (44)        |
================================================================
================================================================
=


SQL>
SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',SESSION_
ID=>:SESSID,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 (59:15805)
 SQL ID               :  dkz7v96ym42c6
 SQL Execution ID     :  16777218
 Execution Started    :  06/18/2010 12:29:51
 First Refresh Time   :  06/18/2010 12:29:59
 Last Refresh Time    :  06/18/2010 12:30:15
 Duration             :  24s
 Module/Action        :  SQL*Plus/-
 Service              :  SYS$USERS
 Program              :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      24 |      22 |     1.14 |   517K |
=========================================

SQL Plan Monitoring Details (Plan Hash Value=1643938535)
================================================================
================================================================
===
| Id   |      Operation       |    Name     |  Rows   | Cost |
Time     | Start  | Execs |   Rows    | Activity | Activity Detail
|
|      |                      |             | (Estim) |      |
Active(s) | Active |   | (Actual) |   (%)    |   (# samples)    |
================================================================
================================================================
===
|    0 | SELECT STATEMENT     |             |         |      |
|         |1 |           |          |                  |
```

```
| -> 1 |   SORT AGGREGATE     |              |         |      |
17 |     +8 |1 |        0 |     8.00 | Cpu (2)          |
| -> 2 |    NESTED LOOPS      |              |         |      |
17 |     +8 |1 |     265M |          |                  |
| -> 3 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
17 |     +8 |1 |    13248 |          |                  |
| -> 4 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
25 |     +0 | 13249 |        265M |    92.00 | Cpu (23)        |
=================================================================
=================================================================
===

SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',SESSION_
ID=>:SESSID,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 (59:15805)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777218
 Execution Started   :  06/18/2010 12:29:51
 First Refresh Time  :  06/18/2010 12:29:59
 Last Refresh Time   :  06/18/2010 12:30:21
 Duration            :  30s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
```

```
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      30 |      28 |     1.71 |   642K |
=========================================


SQL Plan Monitoring Details (Plan Hash Value=1643938535)
================================================================
================================================================
===
| Id   |       Operation        |    Name     |  Rows   | Cost |
Time    | Start  | Execs |   Rows   | Activity | Activity Detail
|
|      |                        |             | (Estim) |      |
Active(s) | Active |  | (Actual) |   (%)    |   (# samples)   |
================================================================
================================================================
===
|    0 | SELECT STATEMENT       |             |         |      |
|        |1 |          |          |                 |
| -> 1 |   SORT AGGREGATE       |             |         |      |
23 |     +8 |1 |        0 |    10.00 | Cpu (3)         |
| -> 2 |    NESTED LOOPS        |             |         |      |
23 |     +8 |1 |     329M |          |                 |
| -> 3 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
23 |     +8 |1 |    16465 |          |                 |
| -> 4 |     INDEX RANGE SCAN | TEST_C_INDX |         |      |
31 |     +0 | 16466 |        329M |    90.00 | Cpu (27)        |
================================================================
================================================================
===



SQL> /


DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',SESSION_
ID=>:SESSID,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 (59:15805)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777218
 Execution Started   :  06/18/2010 12:29:51
 First Refresh Time  :  06/18/2010 12:29:59
 Last Refresh Time   :  06/18/2010 12:30:27
 Duration            :  36s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)
 Fetch Calls         :  1

Global Stats
==================================================
| Elapsed |   Cpu   |  Other   | Fetch | Buffer |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  |
==================================================
|      36 |      34 |     2.12 |     1 |   780K |
==================================================


SQL Plan Monitoring Details (Plan Hash Value=1643938535)
==================================================================
==================================================================
=
| Id |      Operation       |    Name     |  Rows   | Cost |
Time     | Start   | Execs |    Rows    | Activity | Activity Detail
|
|    |                      |             | (Estim) |      |
Active(s) | Active  | | (Actual) |   (%)    |    (# samples)    |
==================================================================
==================================================================
=
|  0 | SELECT STATEMENT     |             |         |      |
29 |     +8 |     1 |    1 |          |                  |
```

```
|  1 |    SORT AGGREGATE      |               |          |      |
29 |      +8 |      1 |    1 |     11.11 | Cpu (4)          |
|  2 |     NESTED LOOPS       |               |          |      |
29 |      +8 |      1 |      400M |              |                 |
|  3 |      INDEX RANGE SCAN | TEST_C_INDX |          |      |
29 |      +8 |      1 |     20000 |              |                 |
|  4 |      INDEX RANGE SCAN | TEST_C_INDX |          |      |
37 |      +0 | 20000 |      400M |        88.89 | Cpu (32)        |
=================================================================
=================================================================
=


SQL>
```

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

```
Session 1:
----------

…

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

Elapsed: 00:00:35.95
SQL>
SQL> exit;
Disconnected …

$
```

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

```
Session 1:
----------

$ sqlplus ep/ep

…

SQL>
```

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

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

```
Session 1:
----------

SQL> @ep_explain
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 |
2042  (99)| 00:00:27 |
|   1 |  SORT AGGREGATE        |              |     1 |     6 |
|          |
|*  2 |   HASH JOIN            |              |  400M|  2288M|
2042  (99)| 00:00:27 |
|*  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.
SQL>

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

set echo on

set linesize 200 pagesize 1000

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

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

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:

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

Session 1:
----------

SQL> set autotrace on
SQL> @ep_execute
```

9. From your session 2, be ready to execute your 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> 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;

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

set echo on

set timing on

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

11. From your 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 done 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 can look at it. Then the probe is performed on the index again. This operation takes more time. Also, note that the cost information is provided in the execution plan.

```
Session 2:
----------

SQL> @ep_monitor
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_l
evel=>'ALL') from dual;

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',SESSION_
ID=>:SESSID,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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:08
 Duration            :  10s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|    8.15 |    7.48 |     0.67 |     61 |
=========================================

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         |             |         |
|           |        |     1 |     |     |         |
|
| -> 1 |   SORT AGGREGATE         |             |       1 |
|         3 |     +6 |     1 |       0 |     |     |
|
| -> 2 |    HASH JOIN             |             |    400M | 2042
|         9 |     +1 |     1 |      92M |  1M |   100.00 | Cpu
(9)           |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         1 |     +6 |     1 |    20000 |     |     |
|
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         3 |     +6 |     1 |     4608 |     |     |
|
=================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:10
 Duration            :  12s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
```

```
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      10 |    9.14 |     0.82 |     63 |
=========================================


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         |              |         |
|           |        |     1 |          |     |          |
|
| -> 1 |   SORT AGGREGATE         |              |       1 |
|         5 |     +6 |     1 |        0 |     |          |
|
| -> 2 |    HASH JOIN             |              |    400M | 2042
|        11 |     +1 |     1 |     113M |  1M |   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
|         5 |     +6 |     1 |     5632 |     |          |
|
================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:14
 Duration            :  15s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      14 |      12 |     1.59 |     67 |
=========================================

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         |             |         |
|           |        |     1 |     |     |         |
|
| -> 1 |   SORT AGGREGATE         |             |       1 |
|         9 |     +6 |     1 |       0 |     |     |
|
| -> 2 |    HASH JOIN             |             |    400M | 2042
|        14 |     +1 |     1 |     154M |  1M |   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 |     7680 |     |     |
|
================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:18
 Duration            :  20s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
```

```
 Program              :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)


Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      18 |      16 |     1.75 |     71 |
=========================================


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          |              |         |
|           |        |     1 |     |     |          |
|
| -> 1 |   SORT AGGREGATE          |              |       1 |
|        13 |     +6 |     1 |        0 |     |     |
|
| -> 2 |    HASH JOIN              |              |    400M | 2042
|        19 |     +1 |     1 |     195M |  1M |   100.00 | Cpu
(19)            |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         1 |     +6 |     1 |    20000 |     |     |
|
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|        13 |     +6 |     1 |     9728 |     |     |
|
================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:24
 Duration            :  26s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      24 |      22 |     1.98 |     78 |
=========================================

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          |               |         |
|          |         |     1 |      |      |          |
|
| -> 1 |   SORT AGGREGATE          |               |       1 |
|        19 |      +6 |     1 |        0 |      |      |
|
| -> 2 |    HASH JOIN              |               |    400M | 2042
|        25 |      +1 |     1 |     266M |  1M |   100.00 | Cpu
(25)          |
|     3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         1 |      +6 |     1 |    20000 |      |      |
|
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|        19 |      +6 |     1 |    13312 |      |      |
|
=================================================================
=================================================================
=============



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:30
 Duration            :  31s
 Module/Action       :  SQL*Plus/-
```

```
 Service              :  SYS$USERS
 Program              :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      30 |      27 |     2.32 |     85 |
=========================================

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         |              |         |
|           |        |     1 |          |     |          |
|
| -> 1 |   SORT AGGREGATE         |              |       1 |
|        25 |     +6 |     1 |        0 |     |     6.67 | Cpu
(2)             |
| -> 2 |    HASH JOIN             |              |    400M | 2042
|        30 |     +1 |     1 |     338M |  1M |    93.33 | Cpu
(28)            |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         1 |     +6 |     1 |    20000 |     |          |
|
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|        25 |     +6 |     1 |    16896 |     |          |
|
================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:32
 Duration            :  34s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
 Program             :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)

Global Stats
=========================================
| Elapsed |   Cpu   |  Other   | Buffer |
| Time(s) | Time(s) | Waits(s) |  Gets  |
=========================================
|      32 |      29 |     2.47 |     87 |
=========================================

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         |              |         |
|          |        |     1 |      |     |          |
|
| -> 1 |   SORT AGGREGATE         |              |       1 |
|        28 |     +6 |     1 |        0 |     |    12.12 | Cpu
(4)           |
|    2 |    HASH JOIN             |              |    400M | 2042
|        32 |     +1 |     1 |     358M |  1M |    87.88 | Cpu
(29)          |
|    3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|         1 |     +6 |     1 |    20000 |     |     |
|
| -> 4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12
|        27 |     +6 |     1 |    17920 |     |     |
|
=================================================================
======



SQL> /

DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID=>'DKZ7V96YM42C6',REPORT_L
EVEL=>'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 (24:32597)
 SQL ID              :  dkz7v96ym42c6
 SQL Execution ID    :  16777219
 Execution Started   :  06/18/2010 12:41:00
 First Refresh Time  :  06/18/2010 12:41:06
 Last Refresh Time   :  06/18/2010 12:41:35
 Duration            :  35s
 Module/Action       :  SQL*Plus/-
 Service             :  SYS$USERS
```

```
 Program              :  sqlplus@edrsr10p1.us.oracle.com (TNS V1-
V3)
 Fetch Calls          :  1

Global Stats
=================================================
| Elapsed |   Cpu   |  Other   | Fetch | Buffer |
| Time(s) | Time(s) | Waits(s) | Calls |  Gets  |
=================================================
|      35 |      32 |     2.61 |     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          |             |         |      |
30 |     +6 |        1 |        1 |       |          |
|
|  1 |   SORT AGGREGATE          |             |       1 |      |
30 |     +6 |        1 |        1 |       |    11.43 | Cpu (4)
|
|  2 |    HASH JOIN              |             |    400M | 2042 |
35 |     +1 |        1 |     400M |    1M |    88.57 | Cpu (31)
|
|  3 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12 |
1 |     +6 |        1 |    20000 |       |          |
|
|  4 |     INDEX FAST FULL SCAN | TEST_C_INDX |   20000 |   12 |
30 |     +6 |        1 |    20000 |       |          |
|
================================================================
======



SQL>
```

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.

```
Session 1:
----------

…

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

Elapsed: 00:00:35.04

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

----------------------------------------------------------------------
----------------
| Id  | Operation              | Name        | Rows  | Bytes | Cost
(%CPU)| Time     |
----------------------------------------------------------------------
----------------
|   0 | SELECT STATEMENT       |             |     1 |     6 |  2042
(99)| 00:00:27 |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |
|          |
|*  2 |   HASH JOIN            |             |   400M|  2288M|  2042
(99)| 00:00:27 |
|*  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
        422  bytes sent via SQL*Net to client
        419  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL>
```

13. In session 1, disable Autotrace.

```
Session 1:
----------

SQL> set autotrace off
SQL>
```

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

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

```
Session 1:
----------
SQL> @ep_execute_with_all
SQL> 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: 00:01:32.19
SQL>
```

15. From your session 1, retrieve all execution plans corresponding to all the queries you executed since the beginning of this lab. What is your conclusion?
    a. The easiest way to find out all the plans is to look at the content of the SGA 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 at step 4. There was no cost information due to the use of the rule-based optimizer instead of the cost-based one.

```
Session 1:
----------

SQL> @ep_retrieve_all_plans
SQL> set echo on
SQL>
SQL> set linesize 200 pagesize 1000
SQL>
SQL> col sql_text format a50
SQL>
SQL> select sql_id,plan_hash_value,sql_text from v$sql where sql_text
like '%from test t1, test t2%';

SQL_ID        PLAN_HASH_VALUE SQL_TEXT
------------- --------------- ----------------------------------------
----------
dkz7v96ym42c6      3253233075 select count(*) from test t1, test t2
where t1.c=t
                              2.c and t1.c=1

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

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

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

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

32fqwuk16uf23      3253233075 EXPLAIN PLAN SET
STATEMENT_ID='PLUS2140495' FOR se
                              lect count(*) from test t1, test t2
where t1.c=t2.
                              c and t1.c=1
```

```
6 rows selected.

Elapsed: 00:00:00.02
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       |             |       |       |  2042
(100)|          |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |
|        |
|*  2 |   HASH JOIN            |             |   400M|  2288M|  2042
(99)| 00:00:27 |
|*  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: 00:00:00.19
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 |        |
|  2042 (100)|          |      1 |00:01:32.15 |          92 |
|       |          |
```

```
|   1 |  SORT AGGREGATE        |              |      1 |      1 |     6
|           |          |      1 |00:01:32.15 |          92 |
|       |          |
|*  2 |   HASH JOIN            |              |      1 |    400M|
2288M|  2042  (99)| 00:00:27 |    400M|00:05:52.66 |          92 |
1155K|  1155K| 1142K (0)|
|*  3 |    INDEX FAST FULL SCAN| TEST_C_INDX |      1 |  20000 | 60000
|    12   (0)| 00:00:01 |  20000 |00:00:00.03 |          46 |
|       |          |
|*  4 |    INDEX FAST FULL SCAN| TEST_C_INDX |      1 |  20000 | 60000
|    12   (0)| 00:00:01 |  20000 |00:00:00.29 |          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('11.2.0.1')
      DB_VERSION('11.2.0.1')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      INDEX_FFS(@"SEL$1" "T1"@"SEL$1" ("TEST"."C"))
      INDEX_FFS(@"SEL$1" "T2"@"SEL$1" ("TEST"."C"))
      LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")
      USE_HASH(@"SEL$1" "T2"@"SEL$1")
      END_OUTLINE_DATA
  */

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


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

Column Projection Information (identified by operation id):
-----------------------------------------------------------
```

```
   1 - (#keys=0) COUNT(*)[22]
   2 - (#keys=1)
   3 - "T1"."C"[NUMBER,22]
   4 - "T2"."C"[NUMBER,22]


55 rows selected.

Elapsed: 00:00:00.12
SQL>

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


set echo on


set linesize 200 pagesize 1000


col sql_text format a50


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


select * from
table(dbms_xplan.display_cursor('dkz7v96ym42c6',null,'TYPICAL'));


select * from
table(dbms_xplan.display_cursor('8w580dd6ncgqw',null,'ADVANCED
ALLSTATS LAST'));
```

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 were stored in the AWR. **Note:** It is possible that a snapshot was taken during this practice. If so, some or all of your queries are stored in the AWR.

```
Session 1:
----------


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');

no rows selected

SQL>

Elapsed: 00:00:00.01
SQL>

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

set echo on

set linesize 200

SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext
WHERE SQL_ID in ('dkz7v96ym42c6','8w580dd6ncgqw');
```

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 the AWR. You can use `DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT` for this purpose.

```
Session 1:
----------

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

PL/SQL procedure successfully completed.

Elapsed: 00:00:05.68
SQL>

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

set echo on

EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT('ALL');
```

18. Verify that your solution works.

```
Session 1:
----------

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       |             |       |       |  2042
(100)|          |
|   1 |  SORT AGGREGATE        |             |     1 |     6 |
|          |
|   2 |   HASH JOIN            |             |   400M|  2288M|  2042
(99)| 00:00:27 |
|   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.

Elapsed: 00:00:00.20
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       |             |        |       |  2042
(100)|          |
|   1 |  SORT AGGREGATE        |             |      1 |     6 |
|          |
|   2 |   HASH JOIN            |             |    400M|  2288M|  2042
(99)| 00:00:27 |
|   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.

Elapsed: 00:00:00.04 SQL>

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

set echo on

set linesize 200 pagesize 1000

SELECT PLAN_TABLE_OUTPUT
FROM
TABLE (DBMS_XPLAN.DISPLAY_AWR('dkz7v96ym42c6'));

SELECT PLAN_TABLE_OUTPUT
FROM
TABLE (DBMS_XPLAN.DISPLAY_AWR('8w580dd6ncgqw',null,null,'TYPICAL
ALLSTATS LAST'));
```

19. Exit from both SQL*Plus sessions.

```
Session 1:
----------

SQL> exit
Disconnected …
$
```

Do not forget to exit from your session 2:

```
Session 2:
----------

SQL> exit
Disconnected …
$
```

Practices for Lesson 5

Chapter 5

Practices Overview

In these practices, you will explore the use of services to trace session activity.

Practice 5-1: Tracing Applications

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

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

```
set echo on

drop user trace cascade;

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

grant connect, resource, dba to trace;

drop tablespace tracetbs including contents and datafiles;

drop tablespace tracetbs3 including contents and datafiles;

create tablespace tracetbs
datafile '/u01/app/oracle/oradata/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

drop table sales purge;

create table sales as select * from sh.sales;


drop table sales2 purge;

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

drop table sales3 purge;

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

exit;
```

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

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

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

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

# tnsnames.ora Network Configuration File:
/u01/app/oracle/product/11.2.0/dbhome_1/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = node)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl.example.com)
    )
  )


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

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

```
$ sqlplus / as sysdba

…
SQL>
```

4. In your SQL*Plus session, create a new service called TRACESERV with no domain name.

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

NAME
----------------------------------------------------------------
SYS$BACKGROUND
SYS$USERS
orcl.example.com
orclXDB
orcl.us.oracle.com

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

PL/SQL procedure successfully completed.

SQL>
SQL> select name from dba_services;

NAME
----------------------------------------------------------------
SYS$BACKGROUND
SYS$USERS
orcl.example.com
TRACESERV
orclXDB
orcl.us.oracle.com

6 rows selected.

SQL>
```

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

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

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

PL/SQL procedure successfully completed.

SQL>
SQL> show parameter service_names

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

6. Exit from your SQL*Plus session.

```
SQL> exit;
Disconnected …

$
```

7. Open a browser window and connect to Enterprise Manager Database Control as the `SYS` user. Navigate to the Top Services page.
   a. Log in to Enterprise Manager as the `SYS` user.
      The URL is `https://localhost:1158/em`.
      Username: `SYS`
      Password: `oracle_4U`
      Connect as: `SYSDBA`
   b. On the Database Home page, click the Performance tab.
   c. On the Performance page, click the Top Consumers link in the Additional Monitoring links section of the page.
   d. On the Top Consumers page, click the Top Services tab. This takes you to the Top Services page.
8. You now execute seven workload scripts that are traced. All workload scripts run under the `TRACESERV` service. Your goal is to analyze the generated trace files to interpret what happens in the seven cases. From your terminal session, execute the `run_tracep0.sh` script. This script is used to trigger the generation of statistics for your `TRACESERV` service so it can be viewed from within Enterprise Manager. As soon as you start the execution of the `run_tracep0.sh` script, move to the next step of this lab.

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

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

Session altered.

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

...

$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

sqlplus / as sysdba @run_tracep0.sql

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

set echo on

connect trace/trace@TRACESERV

alter session set tracefile_identifier='mytraceP0';

set termout off

select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
```

```
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;
select count(*) from dba_objects;

exec dbms_lock.sleep(60);

set termout on

exit;
```

9. Go back to the Top Consumers page in your Enterprise Manager session. Wait until you see the `TRACESERV` service in the Active Services table, and enable tracing for that service.
   a. When you see `TRACESERV` in the Active Services table on the Top Services page, select it, and click the Enable SQL Trace button.
   b. On the Enable SQL Trace page, make sure that Waits is set to `TRUE`, and Binds is set to `FALSE`. Click OK.
   c. Back on the Top Services page, you should see a confirmation message near the top of the Top Consumers page.
10. When tracing for `TRACESERV` is enabled, and `run_tracep0.sh` has finished, execute the `run_tracep1.sh` script from your terminal session. Observe the Enterprise Manager Top Consumers/Top Services page.

```
$ ./run_tracep1.sh
…

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

Session altered.

SQL>
SQL> alter session set sort_area_size=50000;

Session altered.

SQL>
SQL> alter session set hash_area_size=5000;
```

```
Session altered.

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

Session altered.

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

  COUNT(*)
----------
172878975

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

Session altered.

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

  COUNT(*)
----------
172878975

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

…
```

```
$

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

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

```
$ ./run_tracep2.sh
…
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceP2';

Session altered.

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

PL/SQL procedure successfully completed.

Elapsed: 00:00:49.36
SQL>
SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
```

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

Session altered.

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

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.86
SQL>
SQL> exit;
Disconnected …

$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

sqlplus / as sysdba @run_tracep2.sql

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

set echo on
```

```
connect trace/trace@TRACESERV

alter session set tracefile_identifier='mytraceP2';

set timing on

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;
/



connect trace/trace@TRACESERV

alter session set tracefile_identifier='mytraceS2';

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;
/

exit;
```

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

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

Session altered.

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

2 rows updated.

SQL>
SQL> commit;

Commit complete.

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

Index created.

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

Session altered.

SQL>
```

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

2 rows updated.

SQL>
SQL> commit;

Commit complete.

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

Index dropped.

SQL>
SQL>
SQL> exit;
Disconnected …
$
```

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

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

Session altered.

SQL>
SQL> set timing on
SQL>
SQL> DECLARE
  2    TYPE SalesCurTyp  IS REF CURSOR;
  3    v_sales_cursor    SalesCurTyp;
  4    sales_record      sh.sales%ROWTYPE;
  5    v_stmt_str         VARCHAR2(200);
  6  BEGIN
  7    -- Dynamic SQL statement with placeholder:
  8    v_stmt_str := 'select * from sh.sales where amount_sold>0';
```

```
  9
10    -- Open cursor and specify bind argument in USING clause:
11    OPEN v_sales_cursor FOR v_stmt_str;
12
13    -- Fetch rows from result set one at a time:
14    LOOP
15      FETCH v_sales_cursor INTO sales_record;
16      EXIT WHEN v_sales_cursor%NOTFOUND;
17    END LOOP;
18
19    -- Close cursor:
20    CLOSE v_sales_cursor;
21  END;
22  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:26.84
SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
SQL>
SQL> alter session set tracefile_identifier='mytraceS4';

Session altered.

Elapsed: 00:00:00.00
SQL>
SQL> set timing on
SQL>
SQL> DECLARE
  2    TYPE SalesCurTyp  IS REF CURSOR;
  3    TYPE SalesList IS TABLE OF sh.sales%ROWTYPE;
  4    v_sales_cursor    SalesCurTyp;
  5    sales_List        SalesList;
  6    v_stmt_str         VARCHAR2(200);
  7  BEGIN
  8    -- Dynamic SQL statement with placeholder:
  9    v_stmt_str := 'select /* S4 */ * from sh.sales where
amount_sold>0';
10
11    -- Open cursor:
12    OPEN v_sales_cursor FOR v_stmt_str;
13
14    -- Fetch rows from result set one at a time:
```

```
15    LOOP
16      FETCH v_sales_cursor BULK COLLECT INTO Sales_List LIMIT 10000;
17      EXIT WHEN v_sales_cursor%NOTFOUND;
18    END LOOP;
19
20    -- Close cursor:
21    CLOSE v_sales_cursor;
22  END;
23  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:02.09
SQL>
SQL>
SQL> exit;
Disconnected …
$
```

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

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

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

Session altered.

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

1837686 rows created.

SQL> commit;

Commit complete.

SQL>
SQL>
SQL> connect trace/trace@TRACESERV
Connected.
```

```
SQL>
SQL> alter session set tracefile_identifier='mytraceS5';

Session altered.

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

1837686 rows created.

SQL> commit;

Commit complete.

SQL>
SQL> exit;
Disconnected …
$
```

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

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

Session altered.

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

918843 rows updated.

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

PL/SQL procedure successfully completed.
```

```
SQL>
SQL> set termout off
Disconnected …
$
PL/SQL procedure successfully completed.

SQL>
SQL> rollback;

Rollback complete.

SQL>
SQL> exit;
Disconnected …

$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

sqlplus / as sysdba @run_tracep6a.sql &

sqlplus / as sysdba @run_tracep6b.sql

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


set echo on

connect trace/trace

update sales set amount_sold=amount_sold+1;

exec dbms_lock.sleep(60);

rollback;

exit;

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

set echo on
```

```
connect trace/trace@TRACESERV

alter session set tracefile_identifier='mytraceP6';

exec dbms_lock.sleep(30);

set termout off

select cust_id, sum(amount_sold) from sales group by cust_id order by
cust_id;

set tournout on

exit;
```

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

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

Session altered.

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

```
PL/SQL procedure successfully completed.

SQL>
SQL> connect trace/trace
Connected.
SQL>
SQL> create index sales_cust_indx on sales(cust_id);

Index created.

SQL>
SQL> exit;
Disconnected …
$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

export ORACLE_SID=orcl

export ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1

export
PATH=/u01/app/oracle/product/11.1.0/db_1/bin:/bin:/usr/bin:/usr/local/
bin:/usr/X11R6/bin:/usr/java/jdk1.5.0_11/bin:/bin

sqlplus / as sysdba @run_tracep7.sql

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

set echo on

connect trace/trace@TRACESERV

alter session set tracefile_identifier='mytraceP7';

declare
   c number := dbms_sql.open_cursor;
   custid  number;
   amount  number;
   ignore integer;
begin
```

```
   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
   dbms_sql.define_column(c, 1, custid);
   dbms_sql.define_column(c, 2, amount);
   ignore := dbms_sql.execute(c);
   if dbms_sql.fetch_rows(c)>0 then
     dbms_sql.column_value(c, 1, custid);
     dbms_sql.column_value(c, 2, amount);
   end if;
end;
/

connect trace/trace

create index sales_cust_indx on sales(cust_id);

exit;
```

17. Disable tracing for your database.
    a. Go back to the Top Services page in your Enterprise Manager session.
    b. Ensure that `TRACESERV` is selected, and click the Disable SQL trace button.
    c. Back on the Top Consumers page, you should see a confirmation message near the top of the page.
18. Try to find out all the trace files that were generated to handle the previous seven cases.

```
$ ./show_mytraces.sh
orcl_ora_19237_mytraceP0.trc
orcl_ora_19237_mytraceP0.trm
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.trc
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
$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

ls /u01/app/oracle/diag/rdbms/orcl/orcl/trace | grep mytrace
```

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

```
$ ./merge_traces.sh
$

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

#!/bin/bash

trcsess output=mytrace.trc service=TRACESERV.example.com
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/*.trc
```

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

```
$ ./tkprof_traces.sh

TKPROF: Release 11.2.0.1.0 - Development on Mon Jun 14 04:27:51 2010

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



$

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

#!/bin/bash
```

```
cd /home/oracle/solutions/Application_Tracing

tkprof mytrace.trc myreport.txt
```

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

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

TKPROF: Release 11.2.0.1.0 - Development on Mon Jun 14 04:29:59 2010

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

$
```

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

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

    Hint: Search for the first occurrence of the string 'ORDERED USE_HASH(s2) ' and examine the trace. Then find the second 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 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 using the default SQL area parameters, which were sized automatically by the system to handle the needs. You can see a high disk value as well as a high number of waits for temporary segments for the first execution, compared to the second one. Compare the statistics direct path read temp, and direct path write temp in the two statement traces. These indicate activity from SQL work areas to temporary segments.

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

```
SQL ID: 5h4bydz30hwf7
Plan Hash: 2354142265
select /*+ ORDERED USE_HASH(s2) */ count(*)
from
 sales s1, sales s2 where s1.cust_id=s2.cust_id



call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2    105.91     118.33     806501       8874          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
```

```
total       4   105.92    118.34    806501     8874        0         1
Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 95

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

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

23. Interpret the trace generated for case 2 (step 11). In this trace, the first statement of interest has the string 'use literal'. **Note:** This trace is very long. What do you observe, and what are your conclusions?

    a. For this case, the almost same 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 includes the time 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)

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

24. Interpret the trace generated for case 3 (step 12). This trace starts at the string ‘update sales set amount_sold=20000’. What do you observe, and what are your conclusions?
    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 and the alternate case is much better as 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
```

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

25. Interpret the trace generated for case 4 (step 13). This trace can be found below the string `'mytracep4'`. What do you observe, and what are your conclusions?
    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 alternate 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
********************************************************************************
```

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

   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.

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

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


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

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

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

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

28. Interpret the trace generated for case 7 (step 16). What do you observe, and what are your conclusions?
    a. For case 7, you should compare the content in `myreport.txt` with the content in `myreport2.txt`. You should see that an index was added after the first trace was generated. This situation can cause confusion especially if the trace does not contain an execution plan to begin with. This is what you can see from within `myreport.txt`:

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


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

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

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


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

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

```
$ ./at_cleanup.sh
…
SQL> Drop index trace.sales_cust_indx;

Index dropped.

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

PL/SQL procedure successfully completed.

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

PL/SQL procedure successfully completed.

SQL>
SQL> exit;
…
$

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

#!/bin/bash

cd /home/oracle/solutions/Application_Tracing

rm /u01/app/oracle/diag/rdbms/orcl/orcl/trace/*mytrace*.*

rm mytrace.trc
```

```
rm myreport.txt

rm myreport2.txt

rm $ORACLE_HOME/network/admin/tnsnames.ora

mv $ORACLE_HOME/network/admin/tnsnames.ora.bak1
$ORACLE_HOME/network/admin/tnsnames.ora

sqlplus / as sysdba @at_cleanup.sql

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

set echo on

Drop index trace.sales_cust_indx;

exec DBMS_SERVICE.STOP_SERVICE('TRACESERV');

exec DBMS_SERVICE.DELETE_SERVICE('TRACESERV');

exit;
```

Practices for Lesson 6

Chapter 6

Overview of Practices for Lesson 6

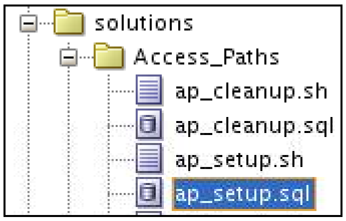
Practices Overview

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

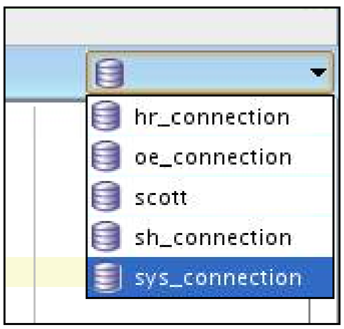
Practice 6-1: Using Different Access Paths

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

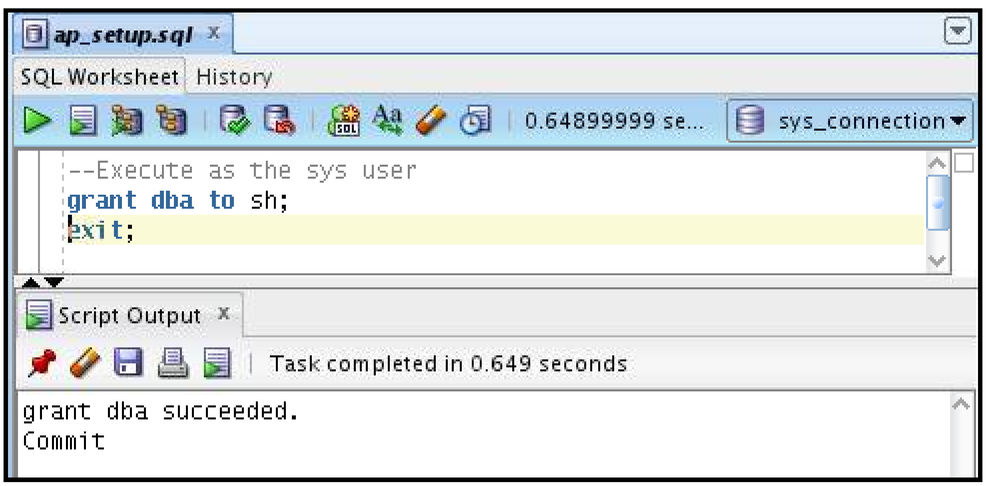
1. **Case 1: With and Without Index**
   a. Use the SQL Developer files tab and open `$HOME/solutions/Access_Paths/ap_setup.sql` script.



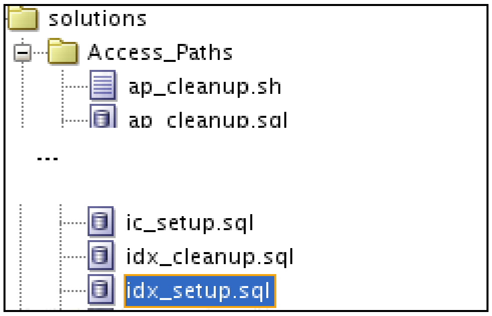
   b. Select `sys_connection`.



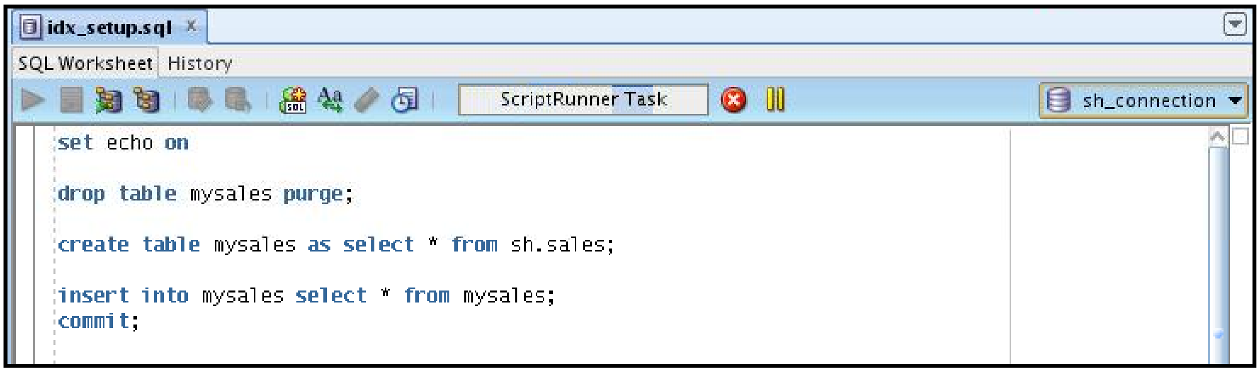
   c. Execute the script (F5).



2. Open `idx_setup.sql`.



3. Execute it with `sh_connection`.



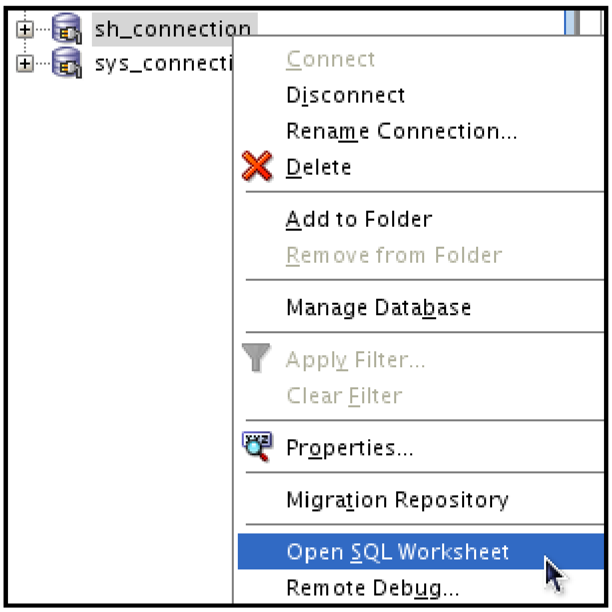
– Output

```
drop table mysales purge

Error starting at line 3 in command:
drop table mysales purge
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
create table mysales as select * from sh.sales

create table succeeded.
insert into mysales select * from mysales
918843 rows inserted
commit
commited
insert into mysales select * from mysales
1837686 rows inserted
commit
commited
insert into mysales select * from mysales
3675372 rows inserted
commit
```

```
commited
insert into mysales select * from mysales
7350744 rows inserted
commit
commited
insert into mysales select * from mysales
14701488 rows inserted
commit
commited
insert into mysales values (0,0,sysdate,0,0,0,0)
1 rows inserted
commit
commited
exec dbms_stats.gather_schema_stats('SH')
anonymous block completed
```

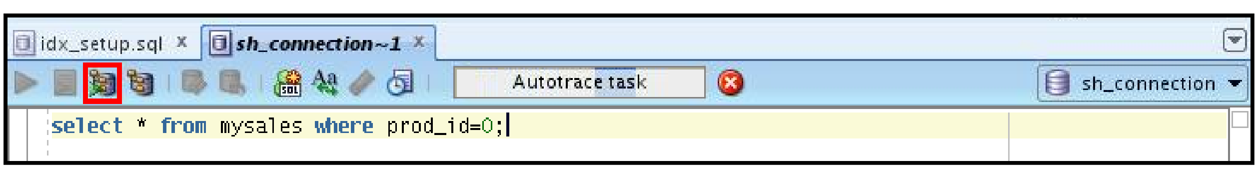
4. What do you observe when you execute the following query in the `sh_connection` worksheet.

   a. Right-click `sh_connection` and select Open SQL Worksheet.

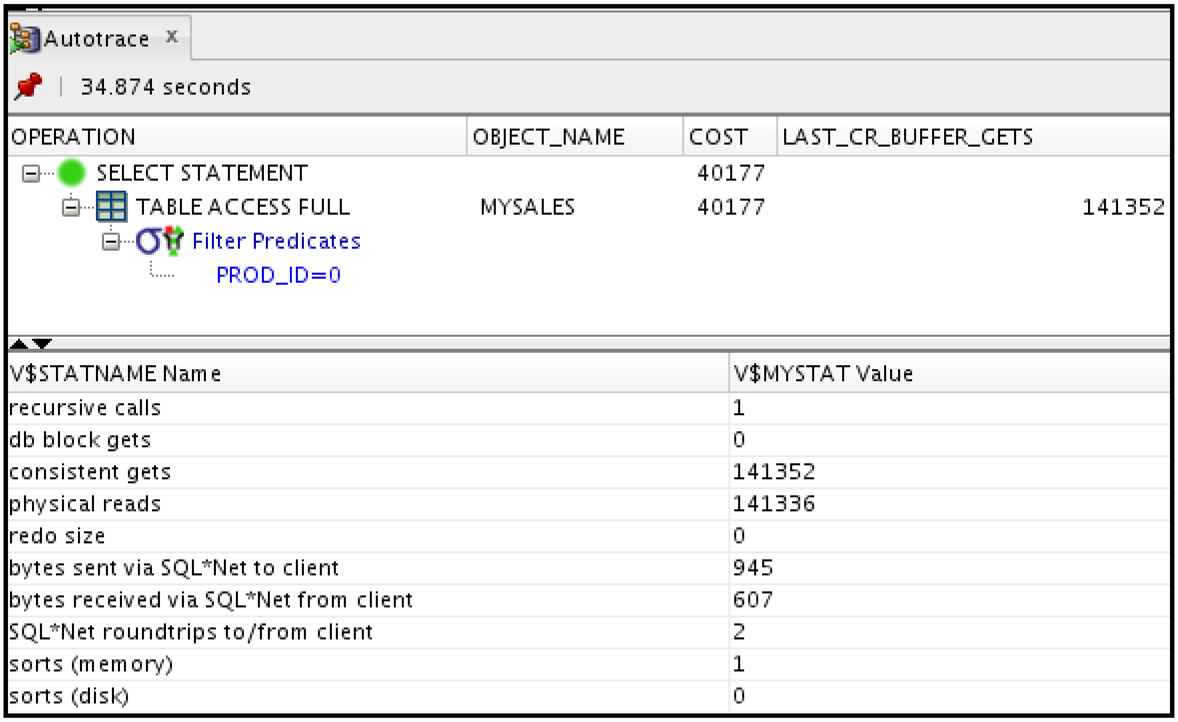


   b. Autotrace the query.

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



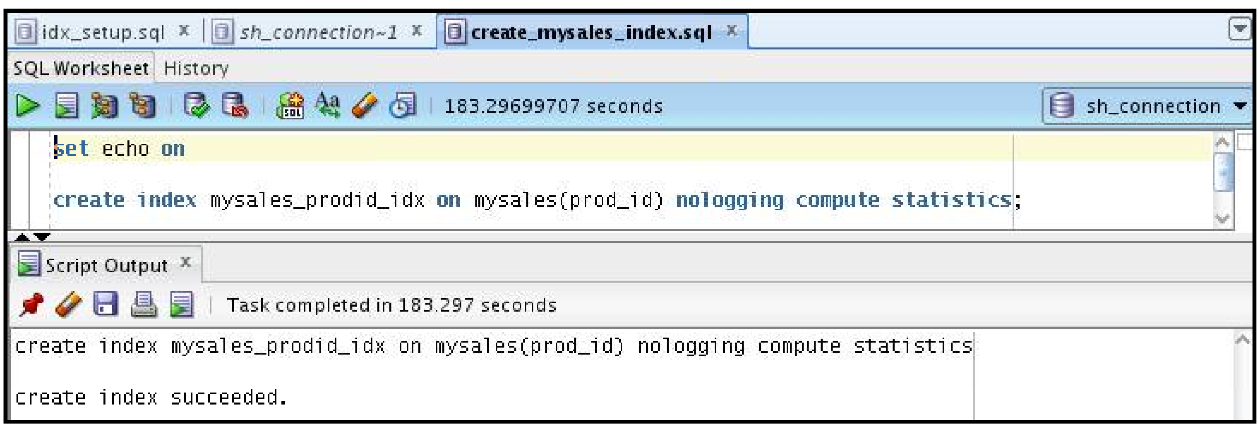
   c. Observe the output.



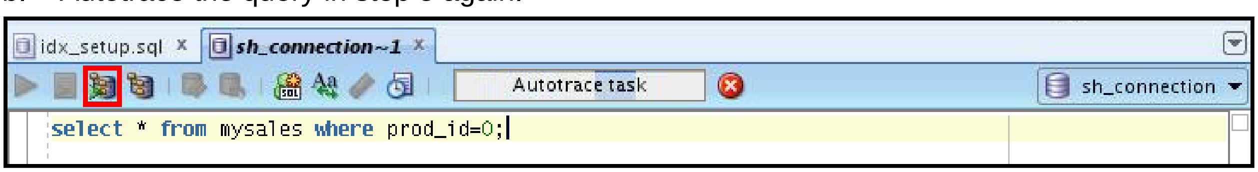
- Basically, there are no indexes on the `MYSALES` table.
- The only possibility for the optimizer is to use the full table scan to retrieve only one row. You can see that the full table scan takes a long time.

5. To enhance the performance of the query in step 4, reexecute the query in step 4 after executing `create_mysales_index.sql`.

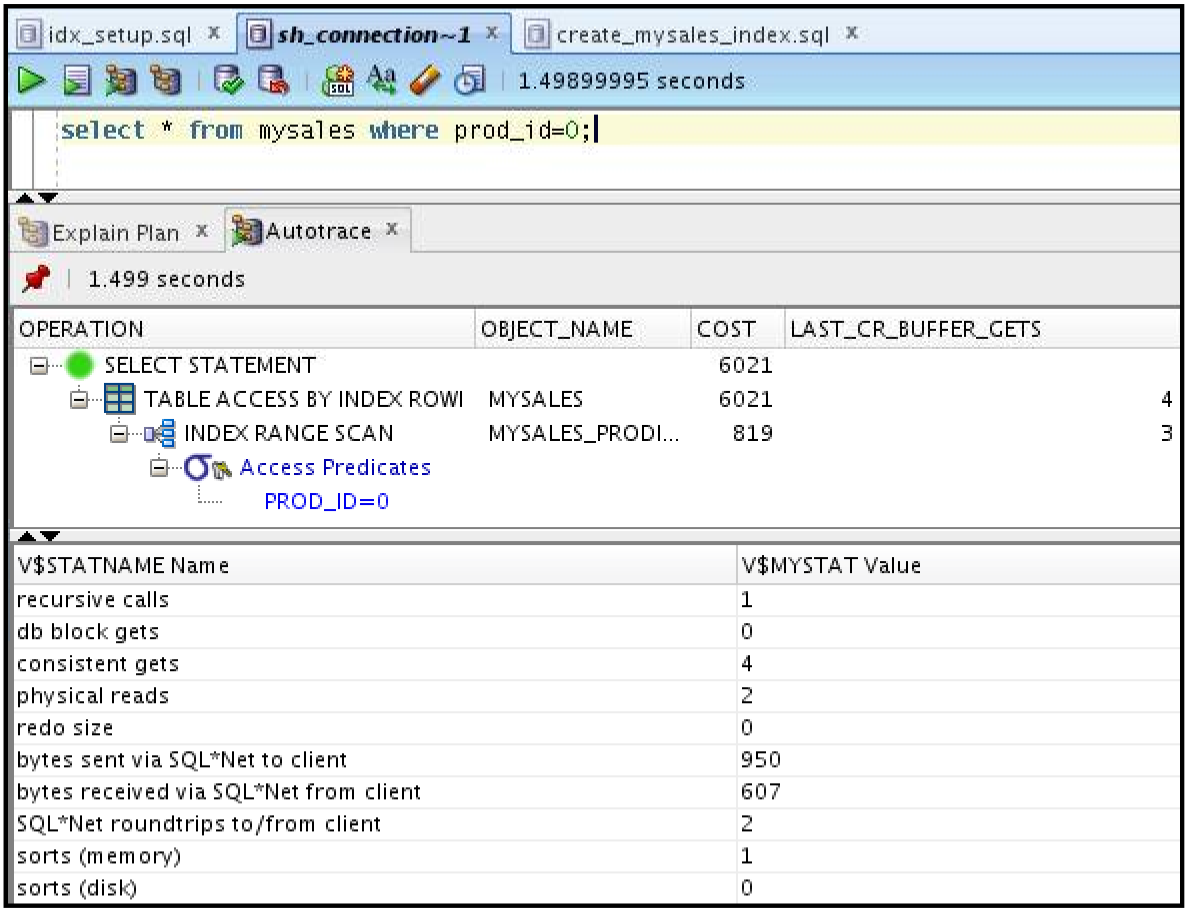
   a. Open the `create_mysales_index.sql` file and execute it with `sh_connection`.



   b. Autotrace the query in step 5 again.

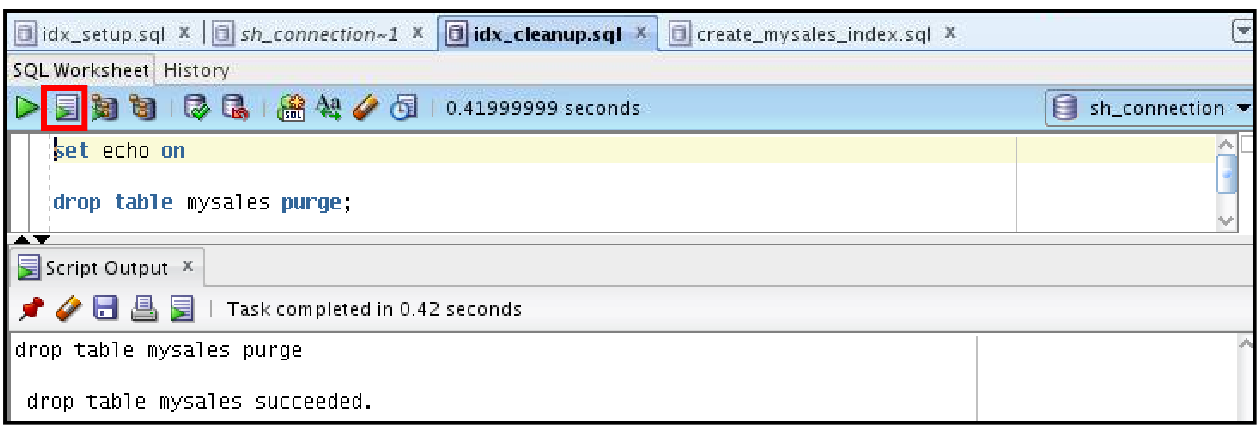


   c. Observe the output.



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

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

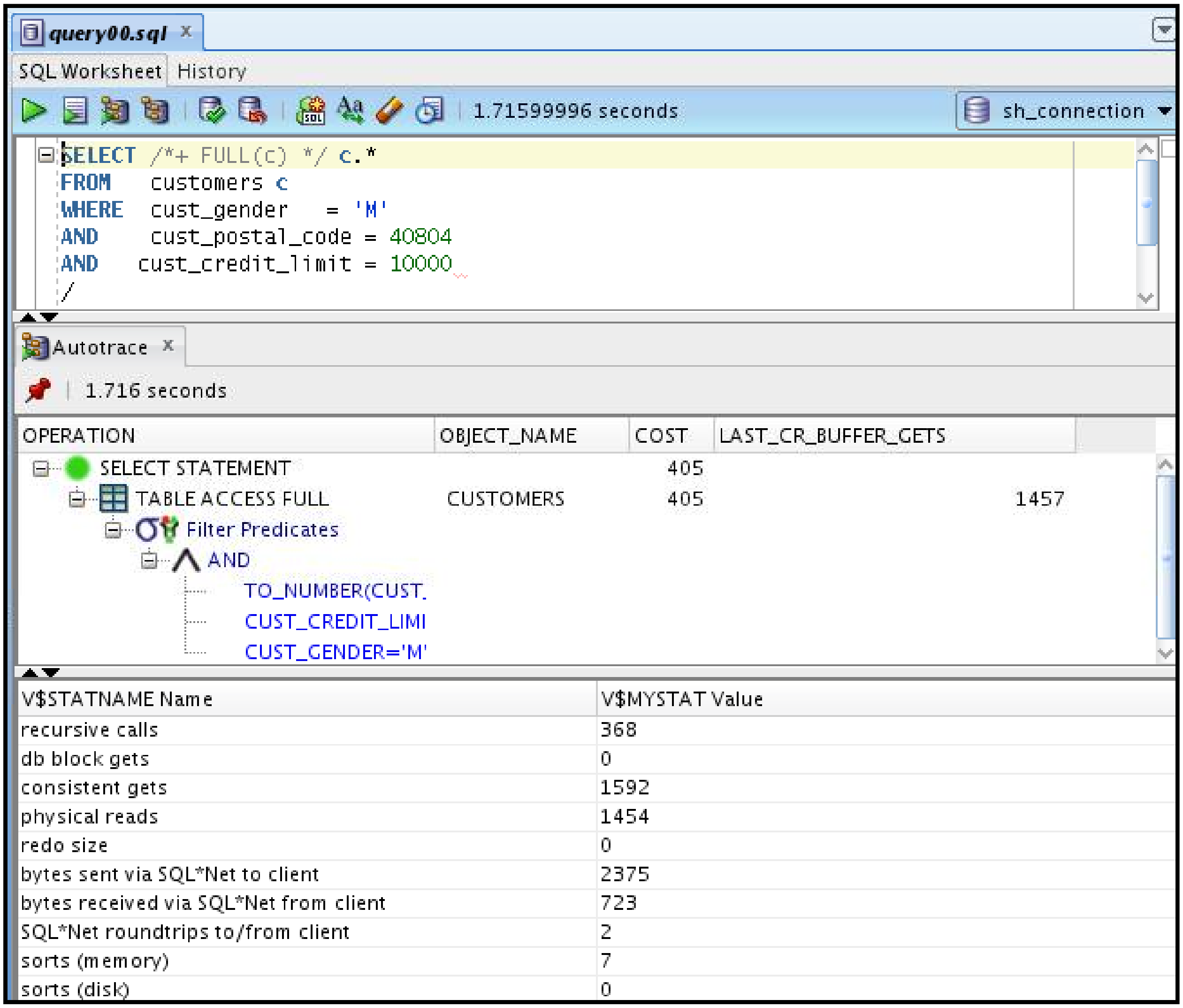


7. **Case 2: Compare Single Column Index Access:** Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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

…
```

```
Connected to:…

SQL> @drop_customers_indexes.sql
```

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



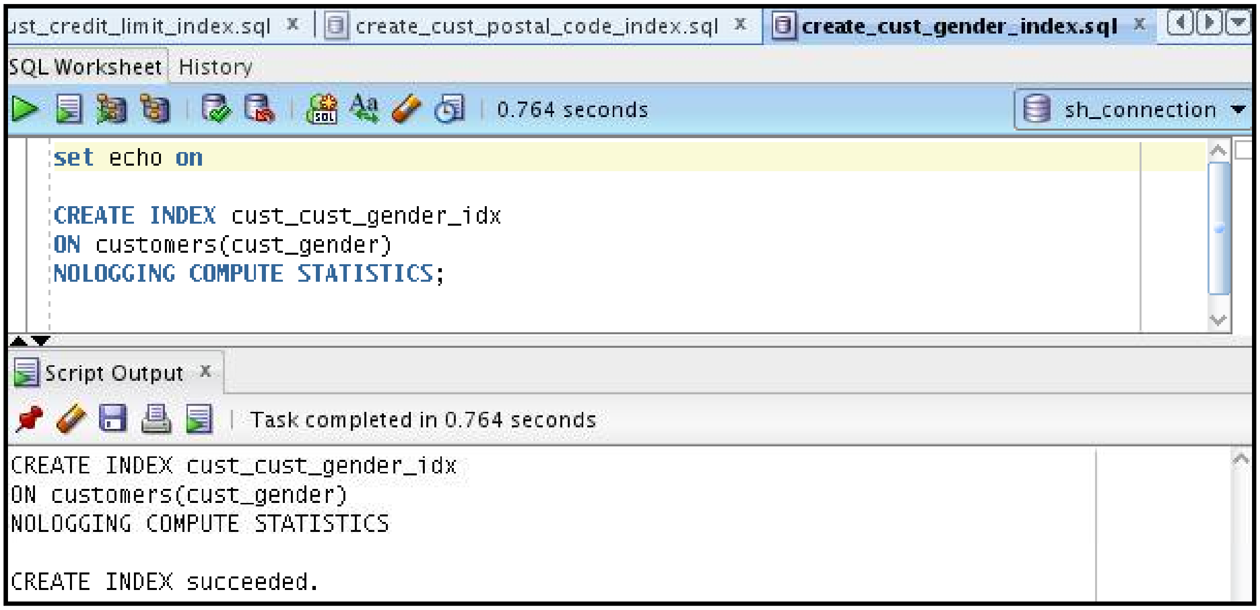
9. Create three B*-tree indexes on the following `CUSTOMERS` table columns by using `sh_connection`:

   `cust_gender`

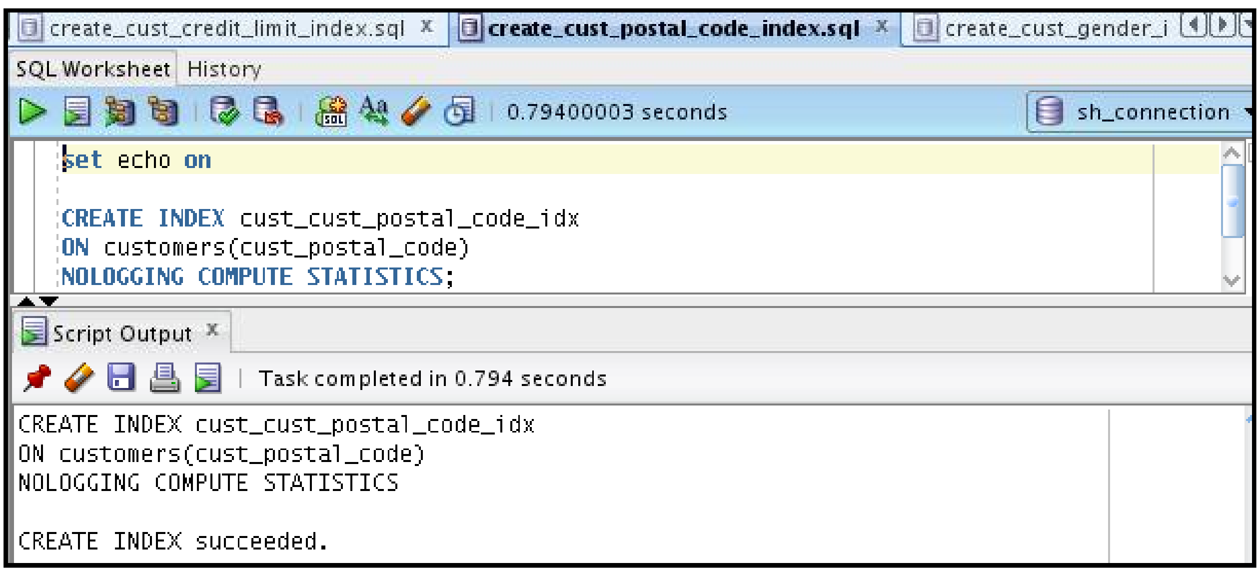
   `cust_postal_code`

   `cust_credit_limit`

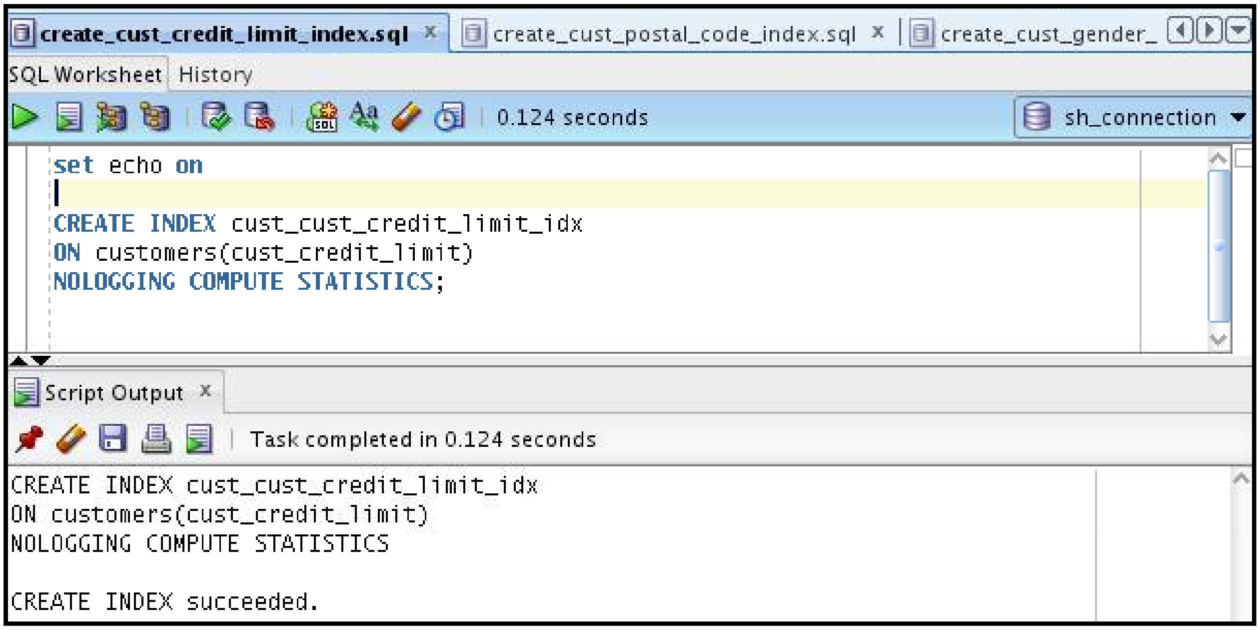
   a. Open and execute the `create_cust_gender_index.sql` script.



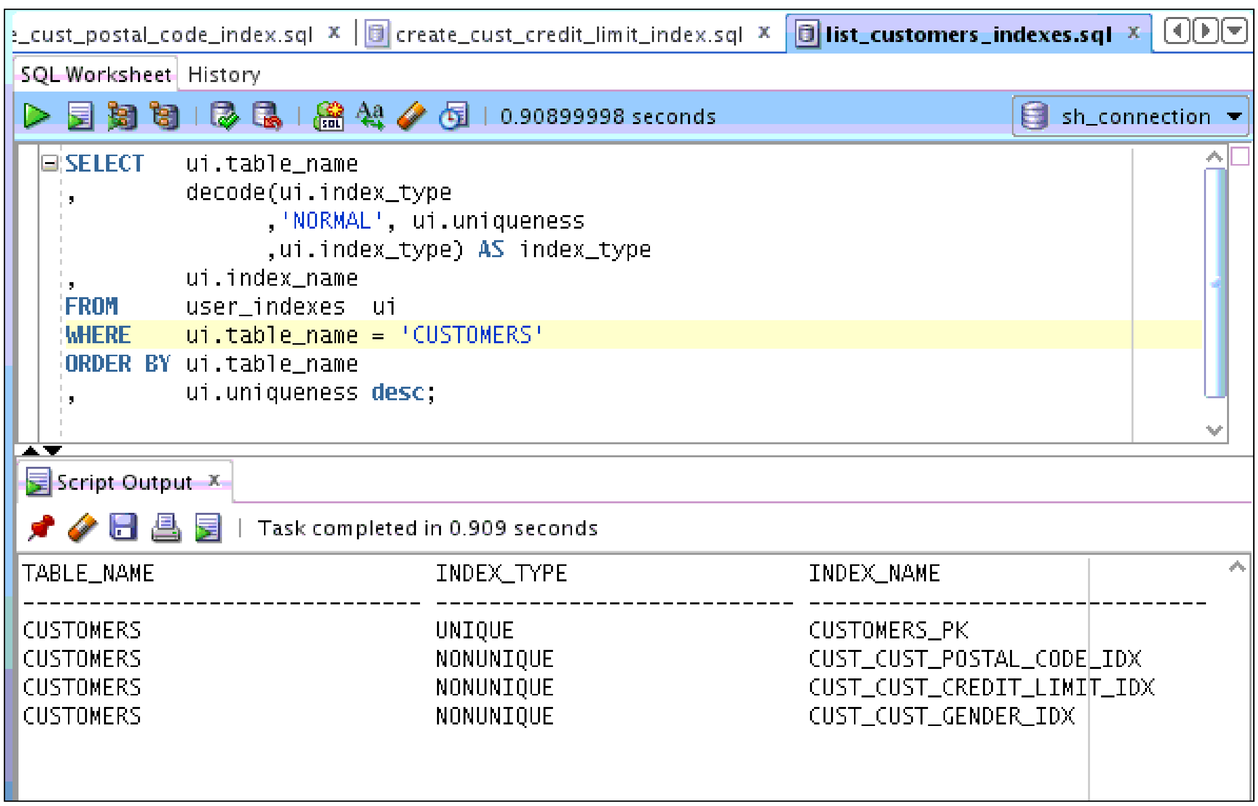
b. Open and execute the `create_cust_postal_code_index.sql` script.



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

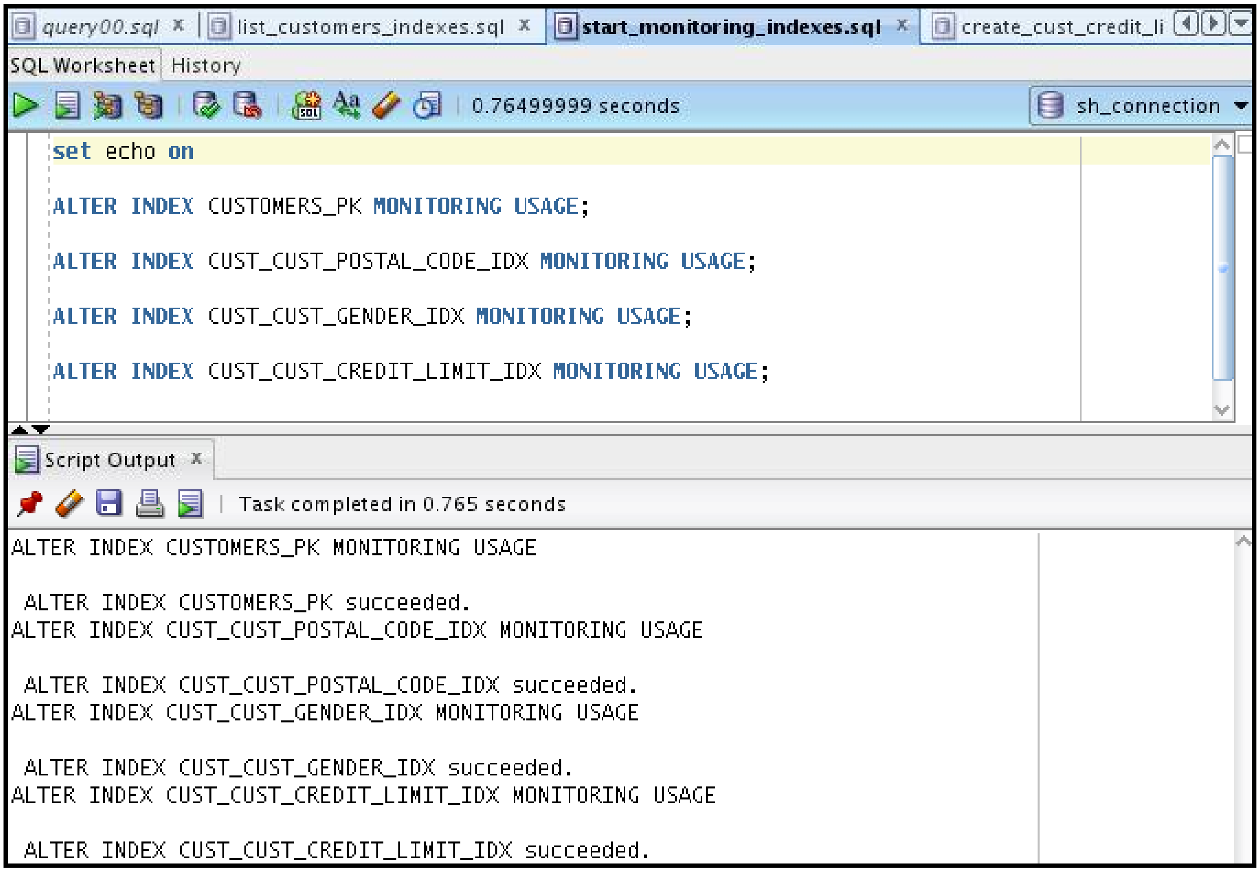


d. To verify that the indexes exist, execute `list_customers_indexes.sql`.

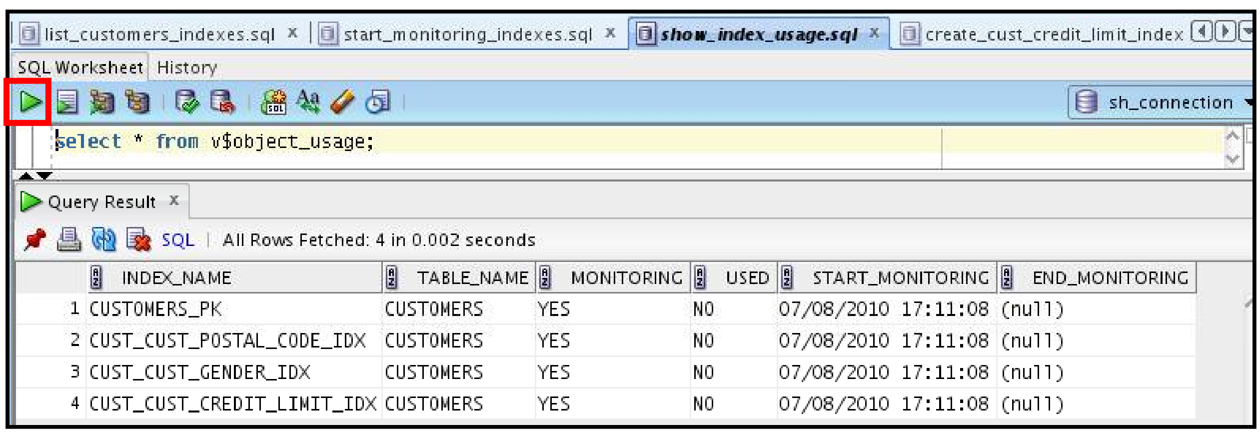


10. Start monitoring all the `CUSTOMERS` indexes. Notice that the value in the `USED` column is `NO`.

a. Open and execute the `start_monitoring_indexes.sql` script using `sh_connection`.

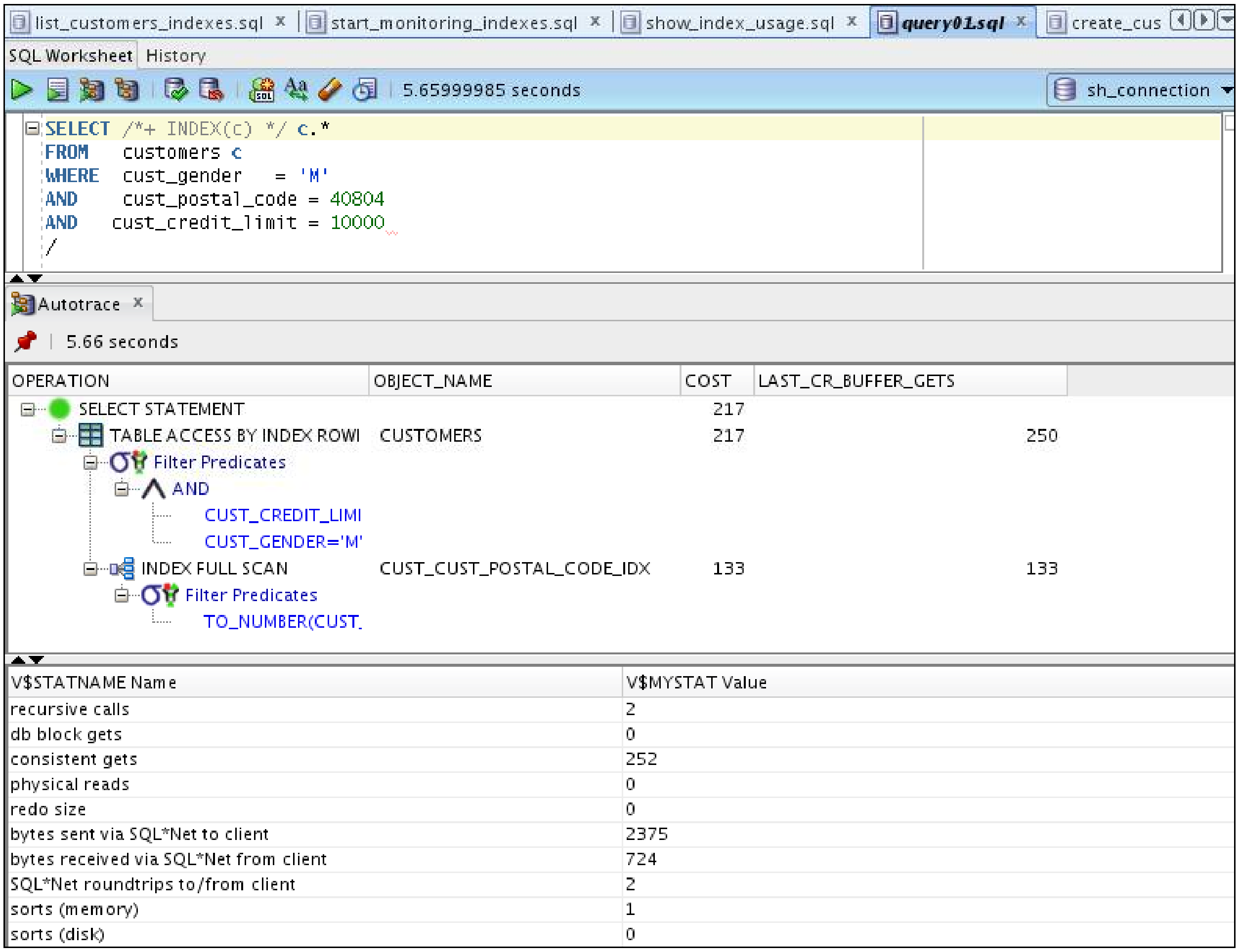


b. Open and execute the statement (Ctrl + Enter) in the `show_index_usage.sql` script.



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

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.

12. Autotrace the query in `query02.sql` 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;
```

OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		465	
TABLE ACCESS BY INDEX ROWID	CUSTOMERS	465	887
Filter Predicates			
TO_NUMBER(CUST_POSTAL_CODE)=40			
BITMAP CONVERSION TO ROWIDS			83
BITMAP AND			83
BITMAP CONVERSION FROM ROWIDS			14
INDEX RANGE SCAN	CUST_CUST_CREDIT_LIMIT_IDX	14	14
Access Predicates			
CUST_CREDIT_LIMIT=10			
BITMAP CONVERSION FROM ROWIDS			69
INDEX RANGE SCAN	CUST_CUST_GENDER_IDX	51	69
Access Predicates			
CUST_GENDER='M'			

Autotrace: 2.684 seconds

V$STATNAME Name	V$MYSTAT Value
recursive calls	3
db block gets	6
consistent gets	891
physical reads	81
redo size	972
bytes sent via SQL*Net to client	2375
bytes received via SQL*Net from client	723
SQL*Net roundtrips to/from client	2
sorts (memory)	1
sorts (disk)	0

– 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.

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

```
select * from v$object_usage;
```

Query Result: All Rows Fetched: 4 in 0.652 seconds

	INDEX_NAME	TABLE_NAME	MONITORING	USED	START_MONITORING	END_MONITORING
1	CUSTOMERS_PK	CUSTOMERS	YES	NO	07/08/2010 17:11:08	(null)
2	CUST_CUST_POSTAL_CODE_IDX	CUSTOMERS	YES	YES	07/08/2010 17:11:08	(null)
3	CUST_CUST_GENDER_IDX	CUSTOMERS	YES	YES	07/08/2010 17:11:08	(null)
4	CUST_CUST_CREDIT_LIMIT_IDX	CUSTOMERS	YES	YES	07/08/2010 17:11:08	(null)

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

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


…
Connected to:…
```

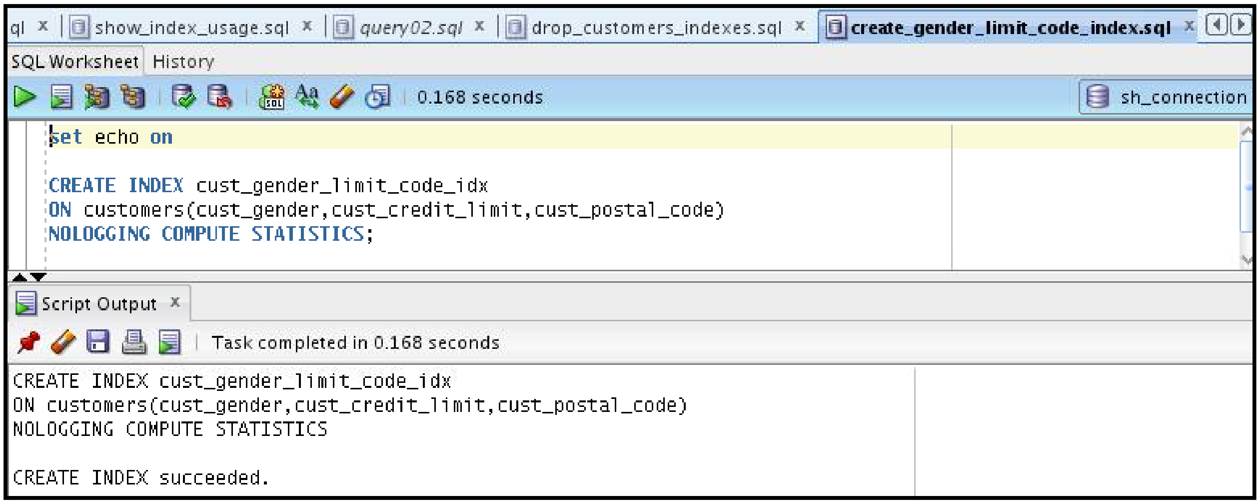
```
SQL> @drop_customers_indexes.sql
```

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

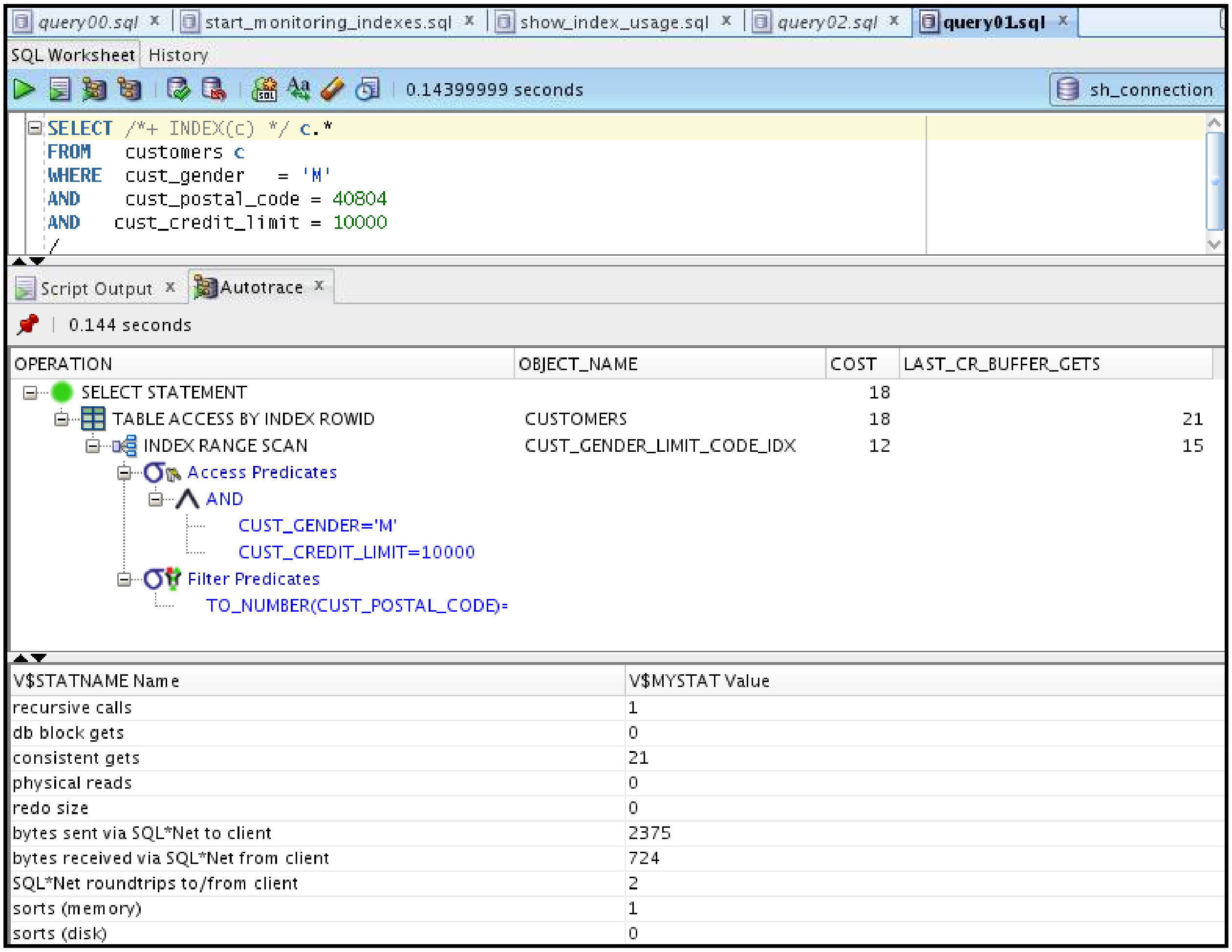
    `cust_gender`

    `cust_credit_limit`

    `cust_postal_code`

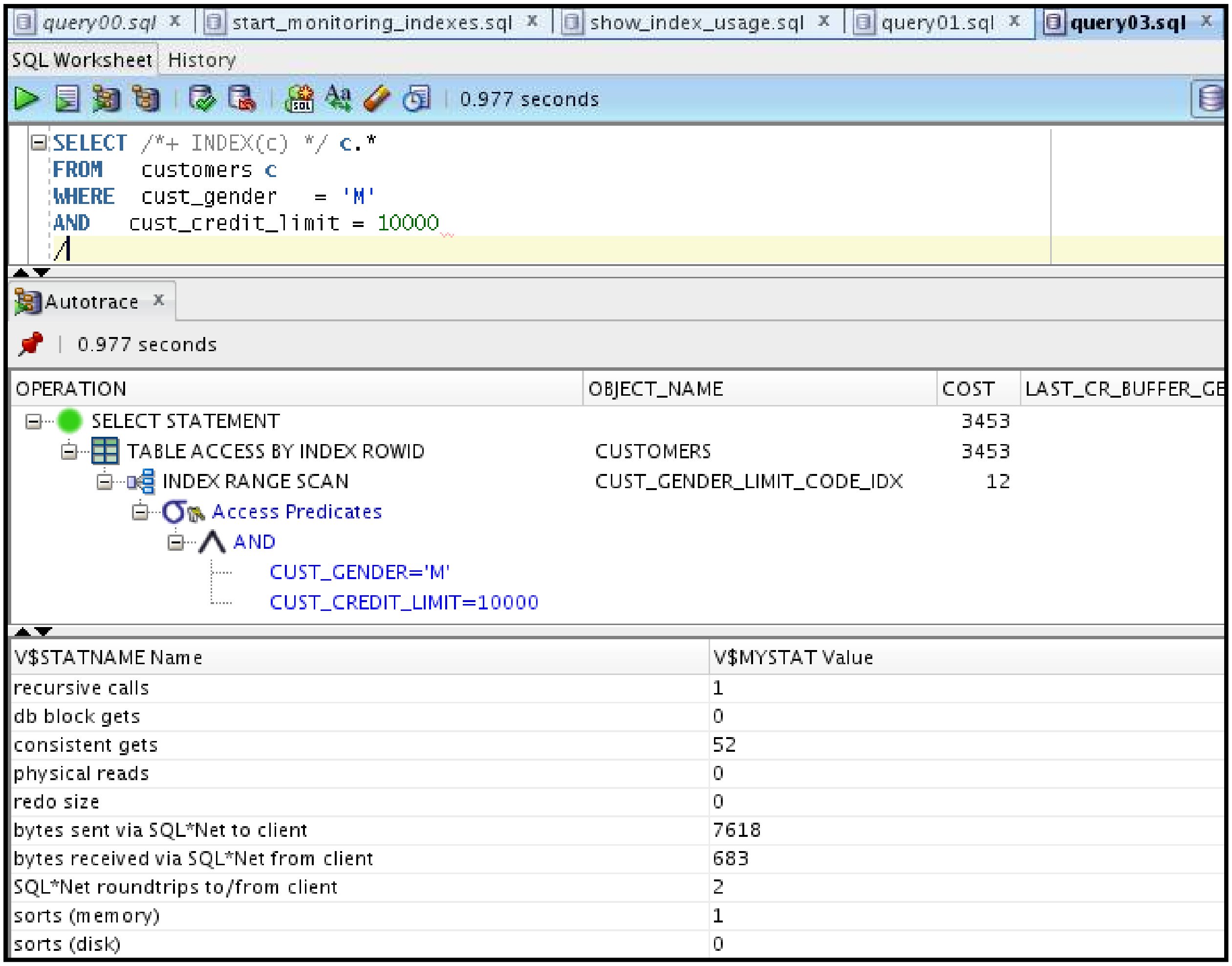


16. Autotrace the query in `query01.sql`. What do you observe?



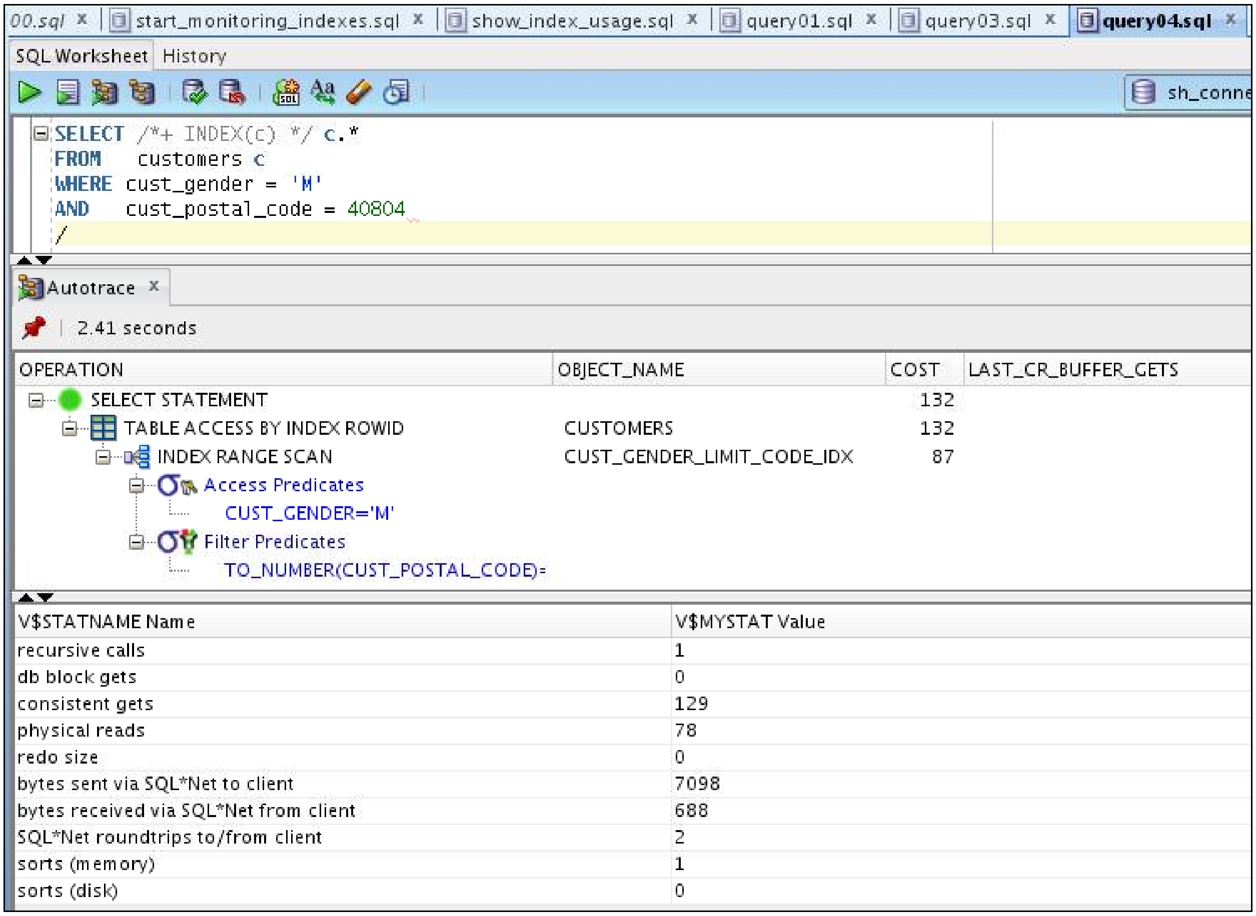
– The optimizer uses your concatenated index, and the resulting cost is by far the best compared to the previous steps.

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



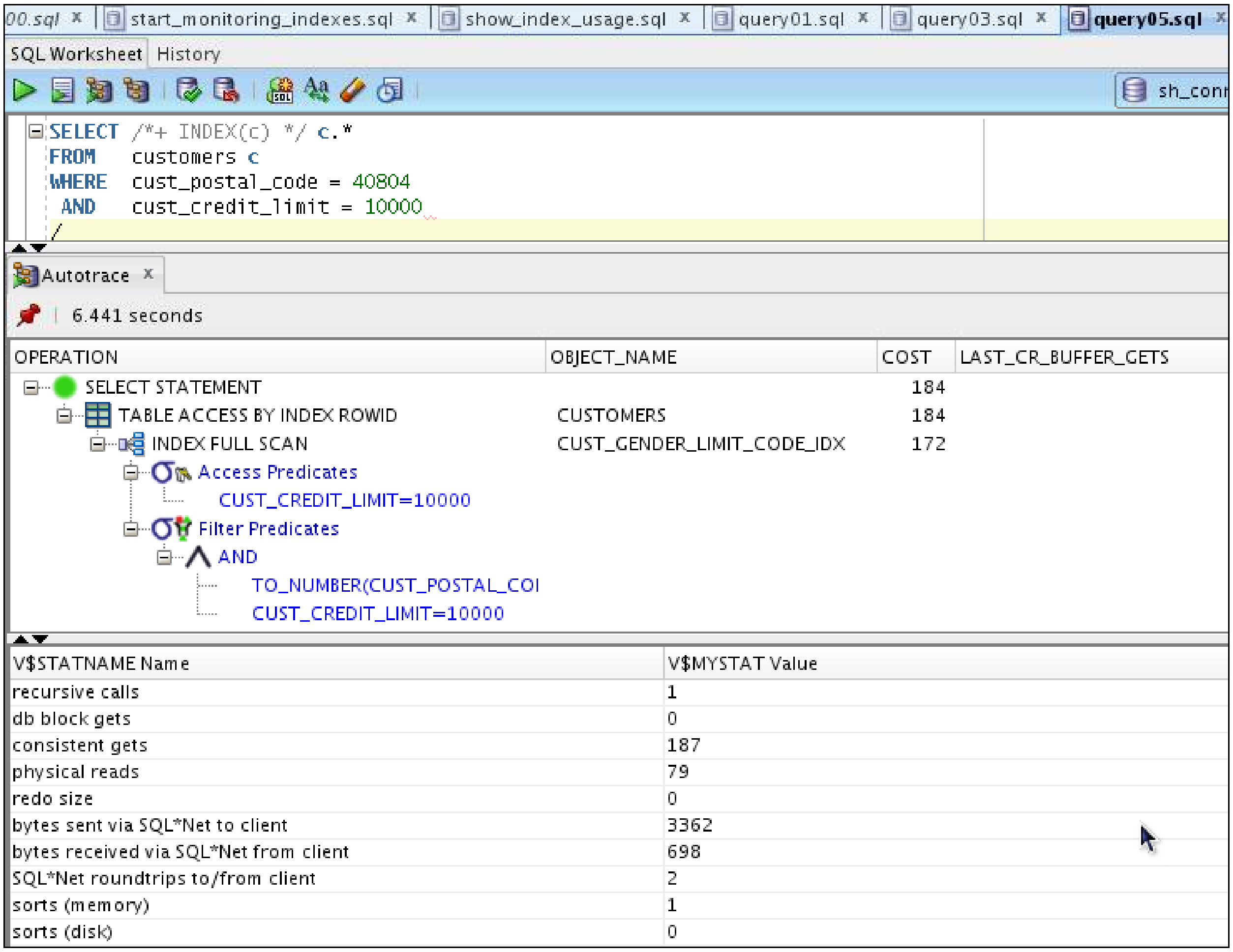
- The query is almost the same as in the previous step, but the predicate on `cust_postal_code` is removed. The optimizer can still use the concatenated index, but the resulting cost is much higher because neither `cust_credit_limit` nor `cust_gender` are very selective.

18. 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.

19. 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 a full index scan.

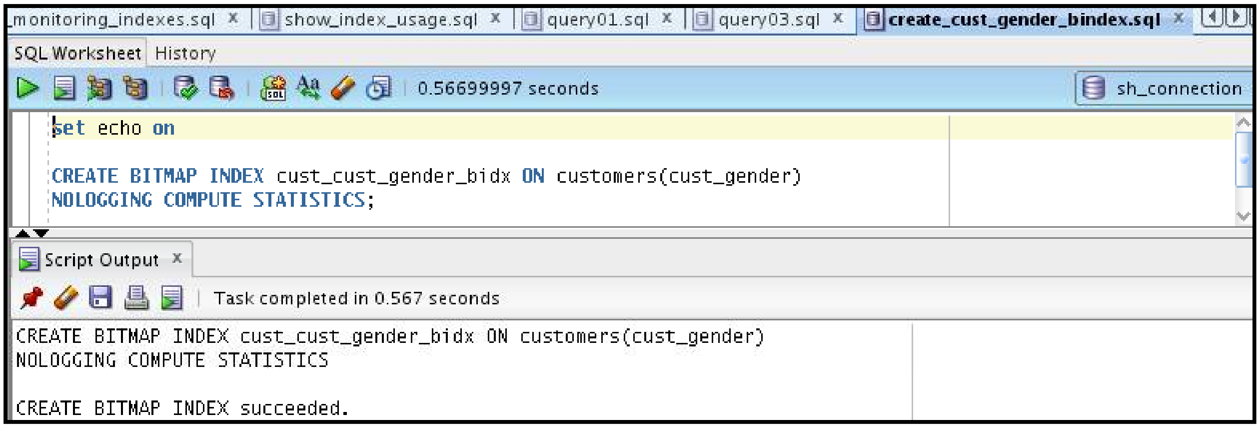
20. **Case 4: Bitmap Index Access:** Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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

…
Connected to:…

SQL> @drop_customers_indexes.sql
```

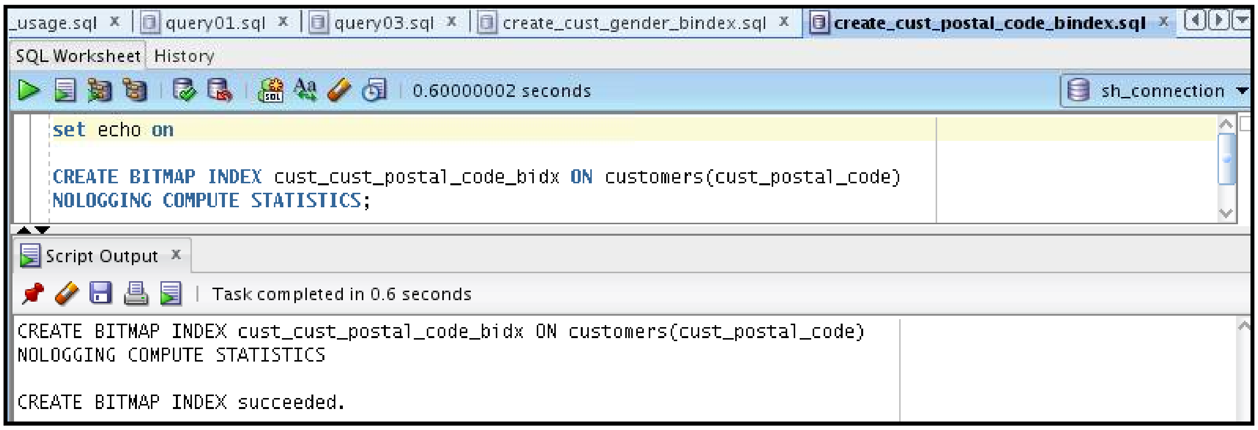
21. Open and execute three scripts using `sh_connection` to create three different bitmap indexes on the following columns of the `CUSTOMERS` table:

```
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)
```

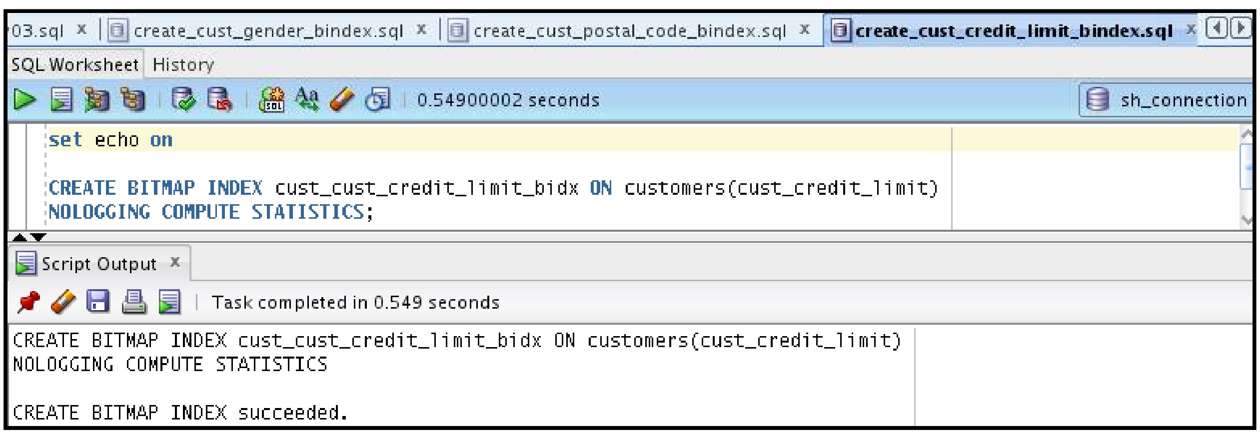
a. Open and execute the `create_cust_gender_bindex.sql` script.



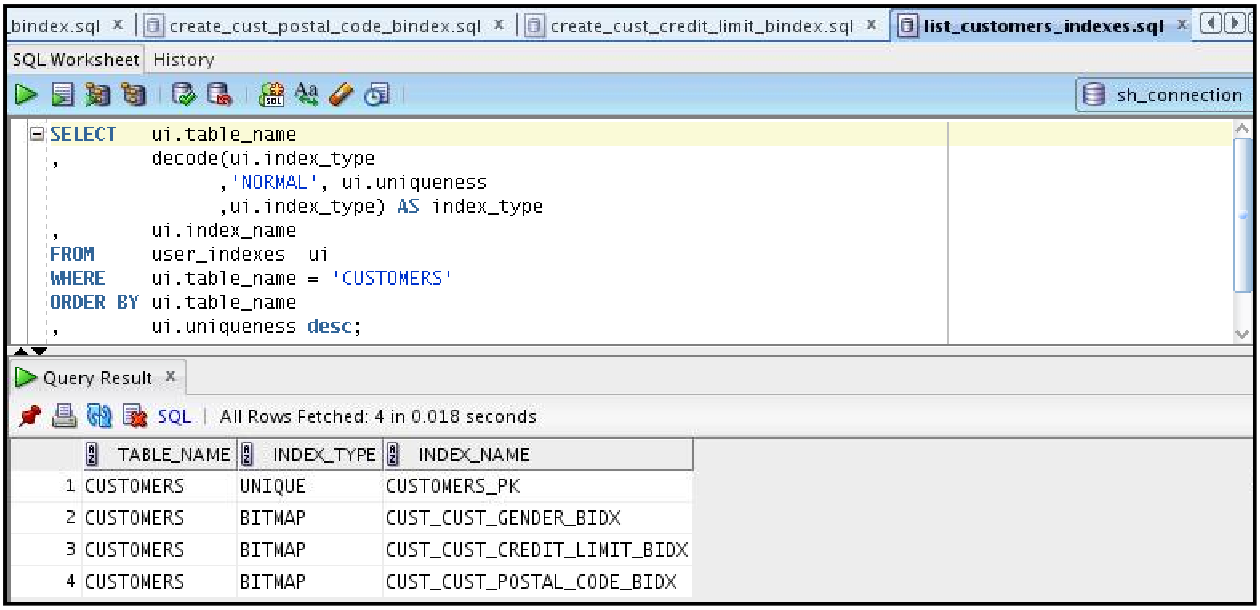
b. Open and execute the `create_cust_postal_code_bindex.sql` script.



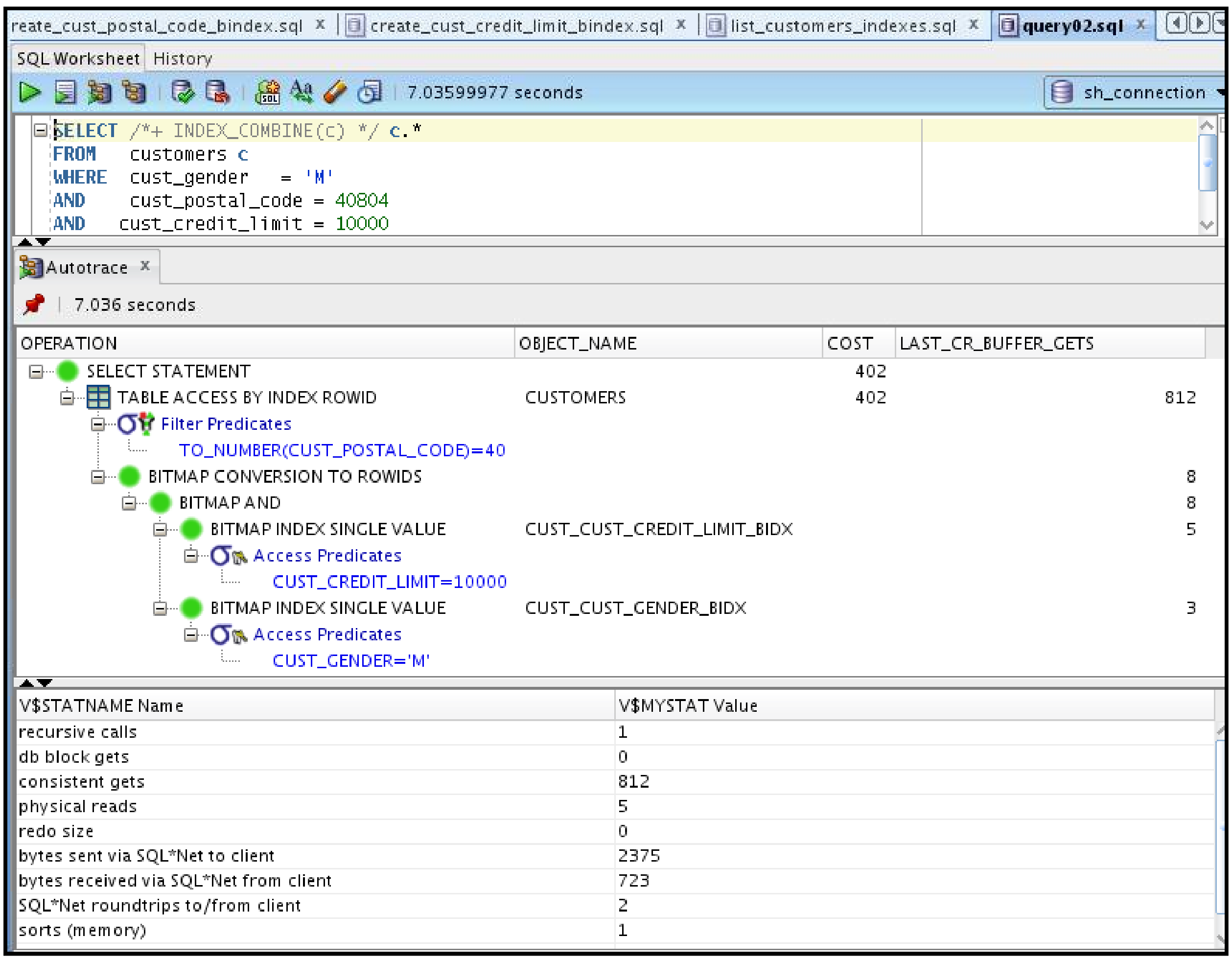
c. Open and execute the `create_cust_credit_limit_bindex.sql` script.



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



22. Autotrace the query in `query02.sql`. What do you observe?

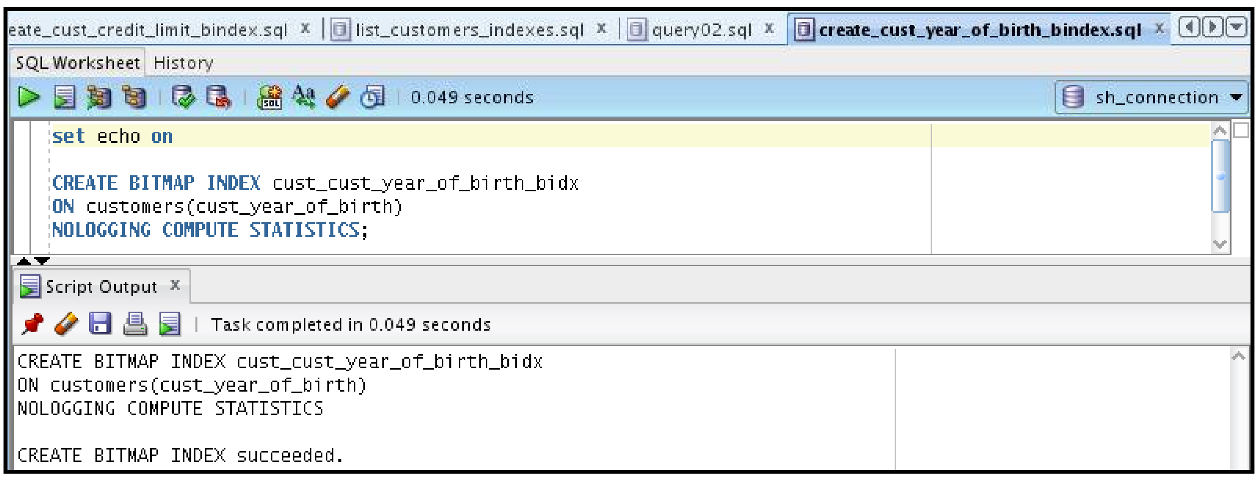


– The optimizer uses only two bitmap indexes to solve this query. However, the cost is not good. The cost is a little lower than the cost of the full table scan.

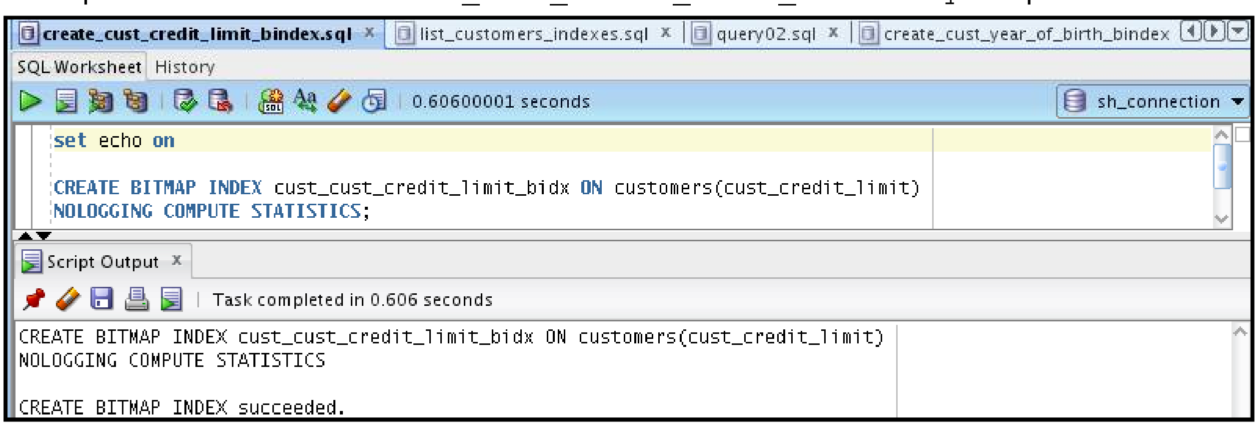
23. **Case 5: Complex Predicate with Bitmap Indexes:** Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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


…
Connected to:…

SQL> @drop_customers_indexes.sql
```

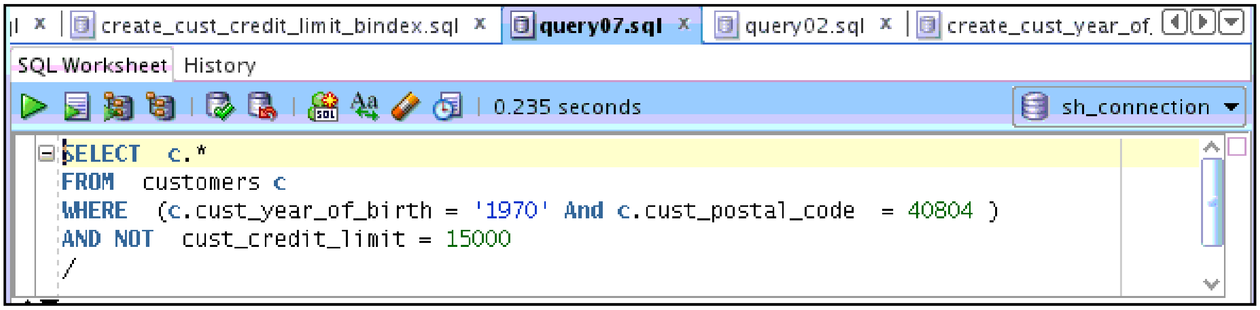
24. After this, create two bitmap indexes on the following columns of the `CUSTOMERS` table:
`cust_year_of_birth (create_cust_year_of_birth_bindex.sql)`
`cust_credit_limit (create_cust_credit_limit_bindex.sql)`
    a. Open and execute the `create_cust_year_of_birth_bindex.sql` script.

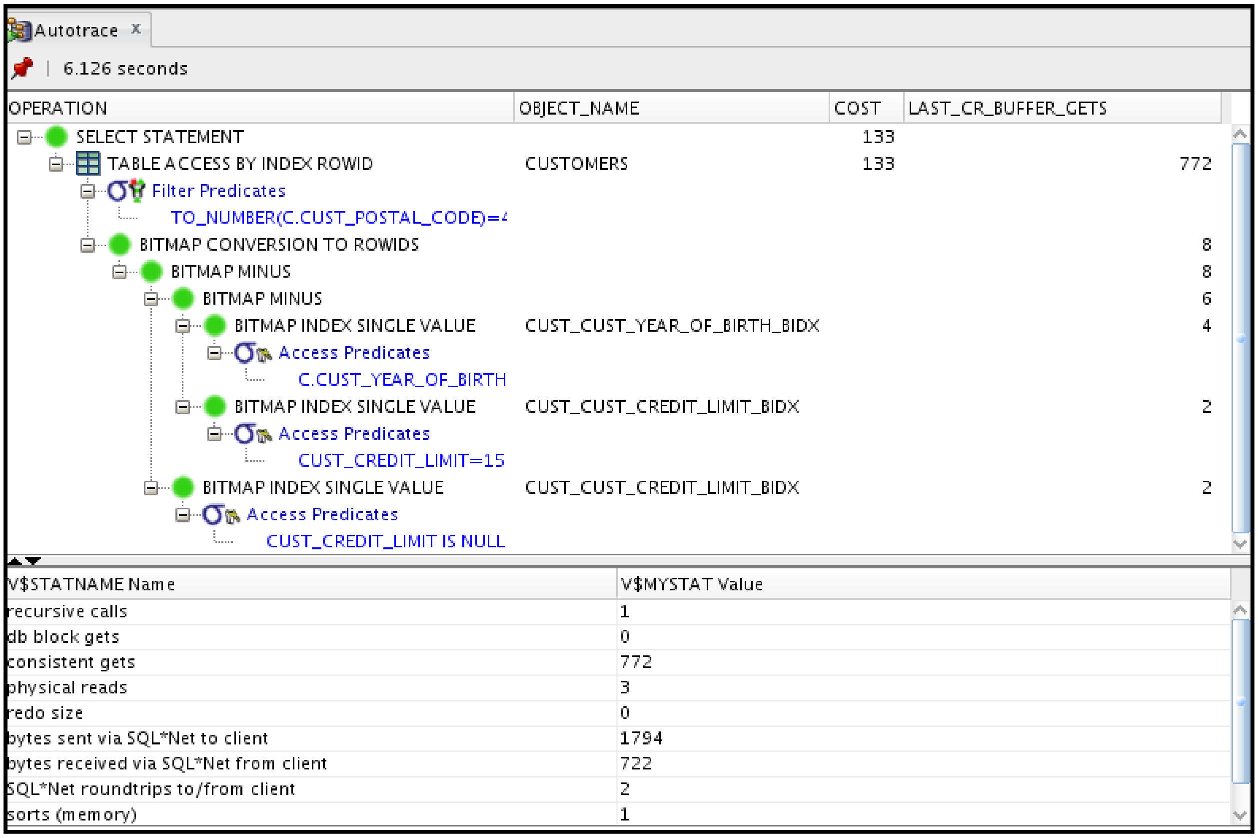


    b. Open and execute the `create_cust_credit_limit_bindex.sql` script.



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



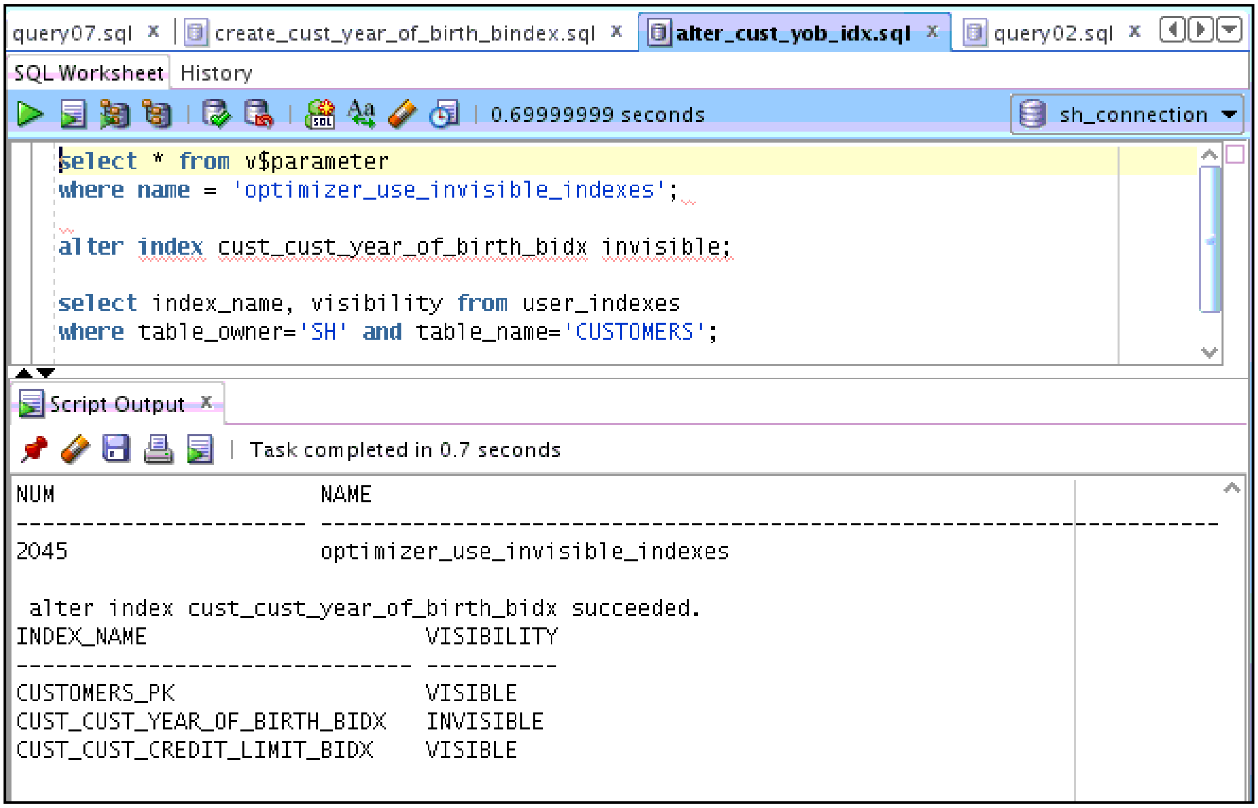


– The query has a complex `WHERE` clause that is well suited for using bitmap indexes. The optimizer uses two bitmap indexes and the resulting cost is better than the full table scan cost.

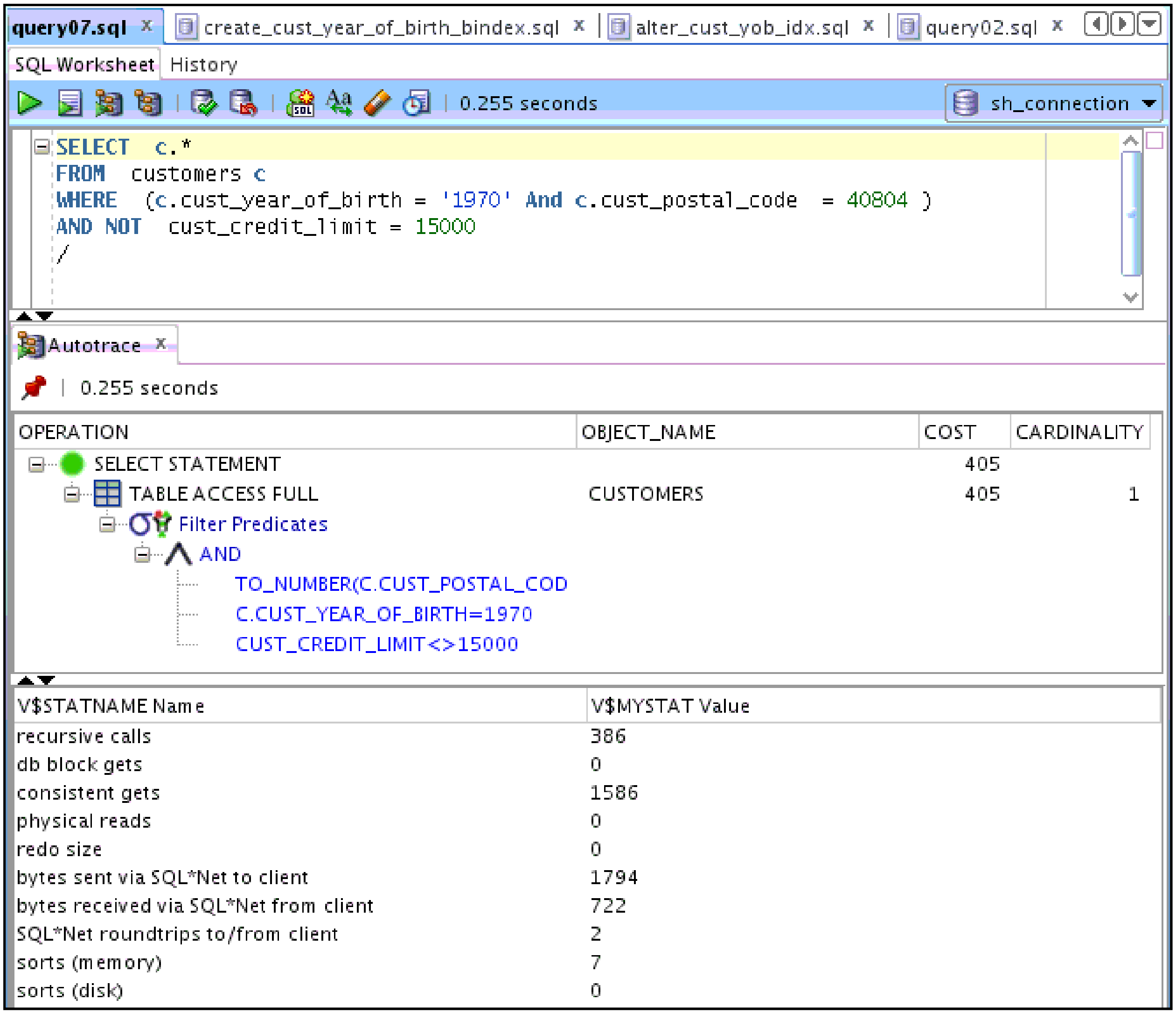
26. Make sure that the optimizer can no longer use the bitmap index you created on the `cust_year_of_birth` column.

The best solution is to render it invisible. Open and execute the `alter_cust_yob_idx.sql` script. Verify that the `optimizer_use_invisible_indexes` parameter is set to `FALSE`.

```
select * from v$parameter
where name = 'optimizer_use_invisible_indexes';

alter index cust_cust_year_of_birth_bidx invisible;

select index_name, visibility from user_indexes
where table_owner='SH' and table_name='CUSTOMERS';
```



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



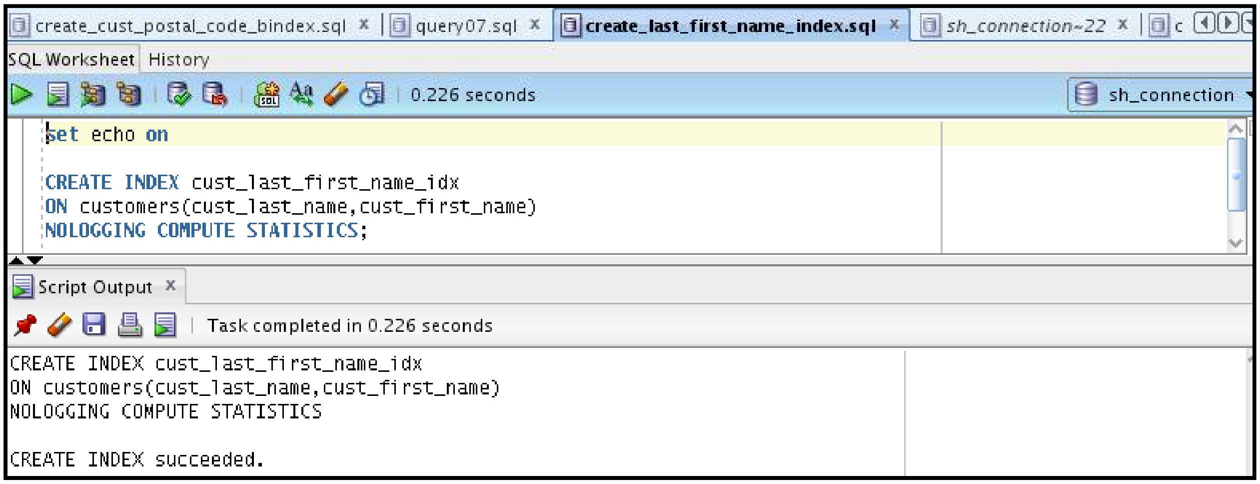
– This is the same query as in the previous step. However, the optimizer can no longer find a good plan that uses bitmap indexes.

28. **Case 6: Index Only Access**: Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

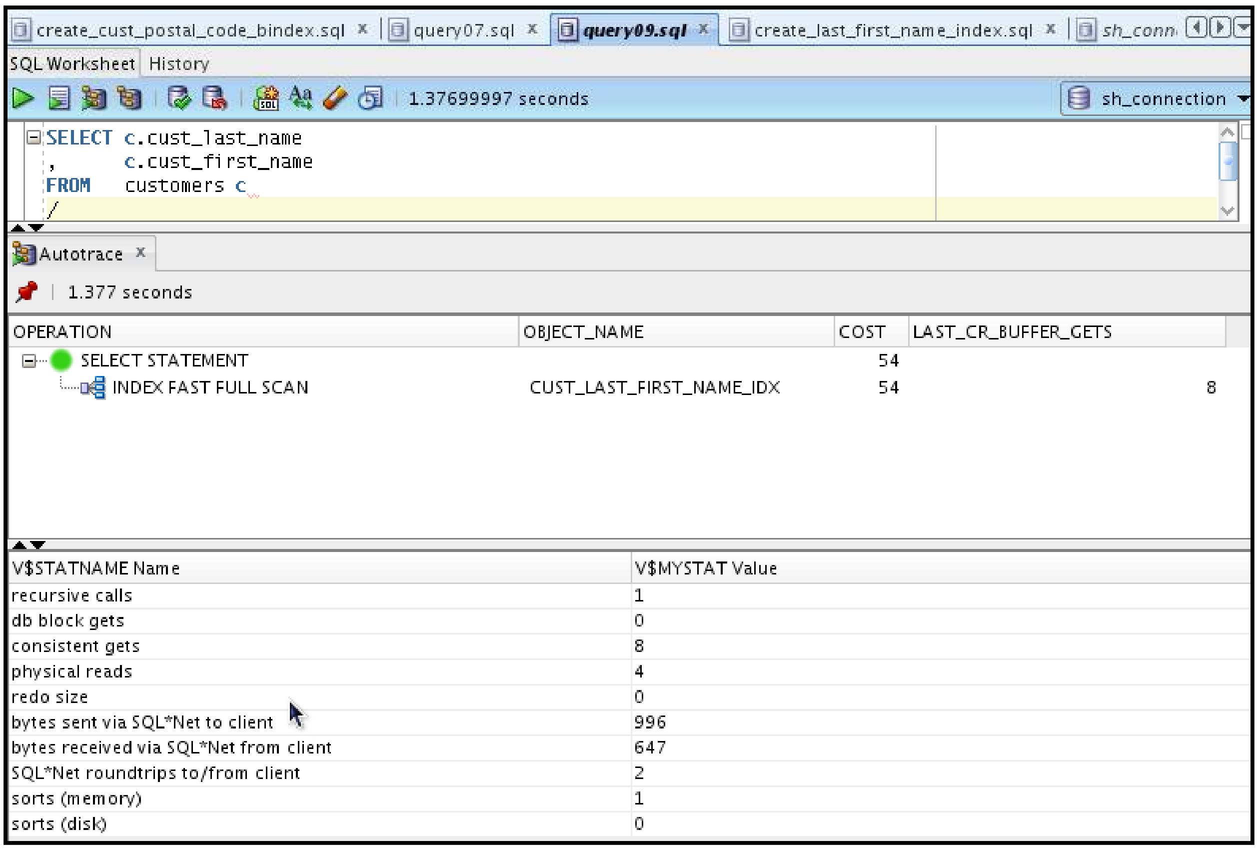
```
$ cd /home/oracle/solutions/Access_Paths/
$ sqlplus sh/sh

…
Connected to:…

SQL> @drop_customers_indexes.sql
```

29. 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 here:
`cust_last_name`

`cust_first_name`



30. 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.

31. **Case 7: Index Join:** Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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

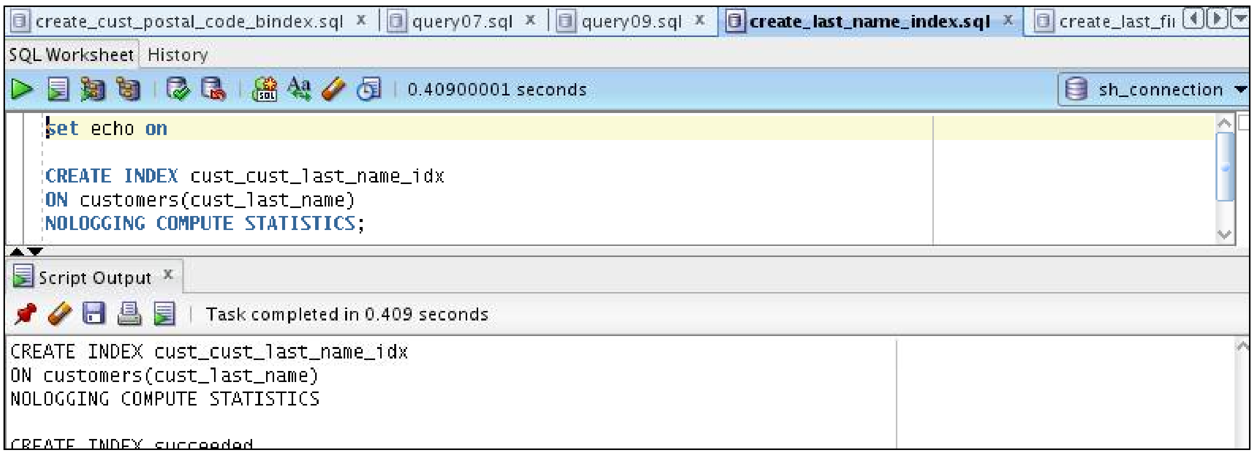
```
…
Connected to:…

SQL> @drop_customers_indexes.sql
```

32. After this, create two B*-tree indexes on the following columns of the `CUSTOMERS` table:

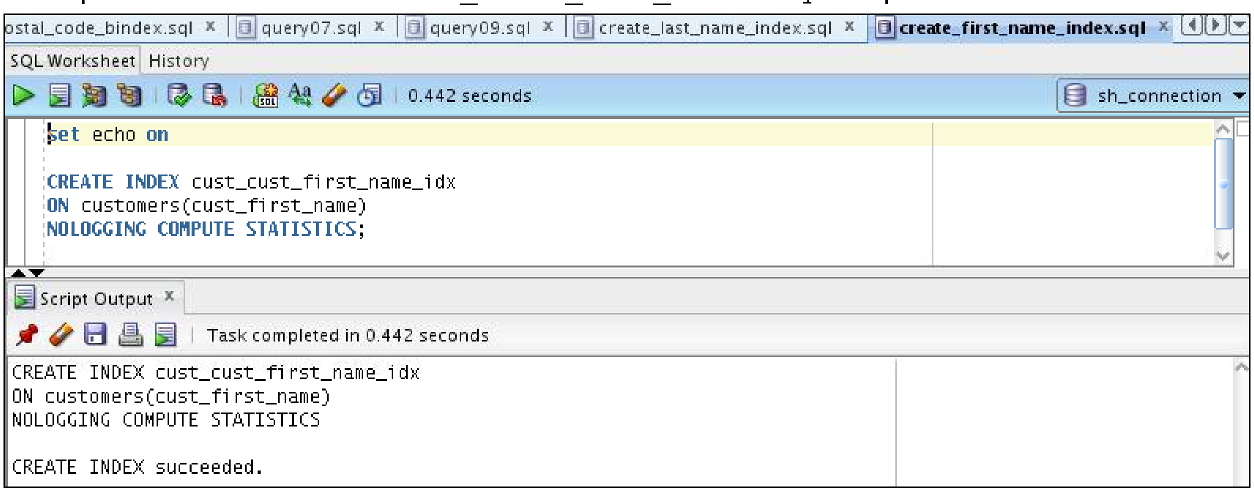
`cust_last_name (create_last_name.index.sql)`

`cust_first_name (create_first_name.index.sql)`

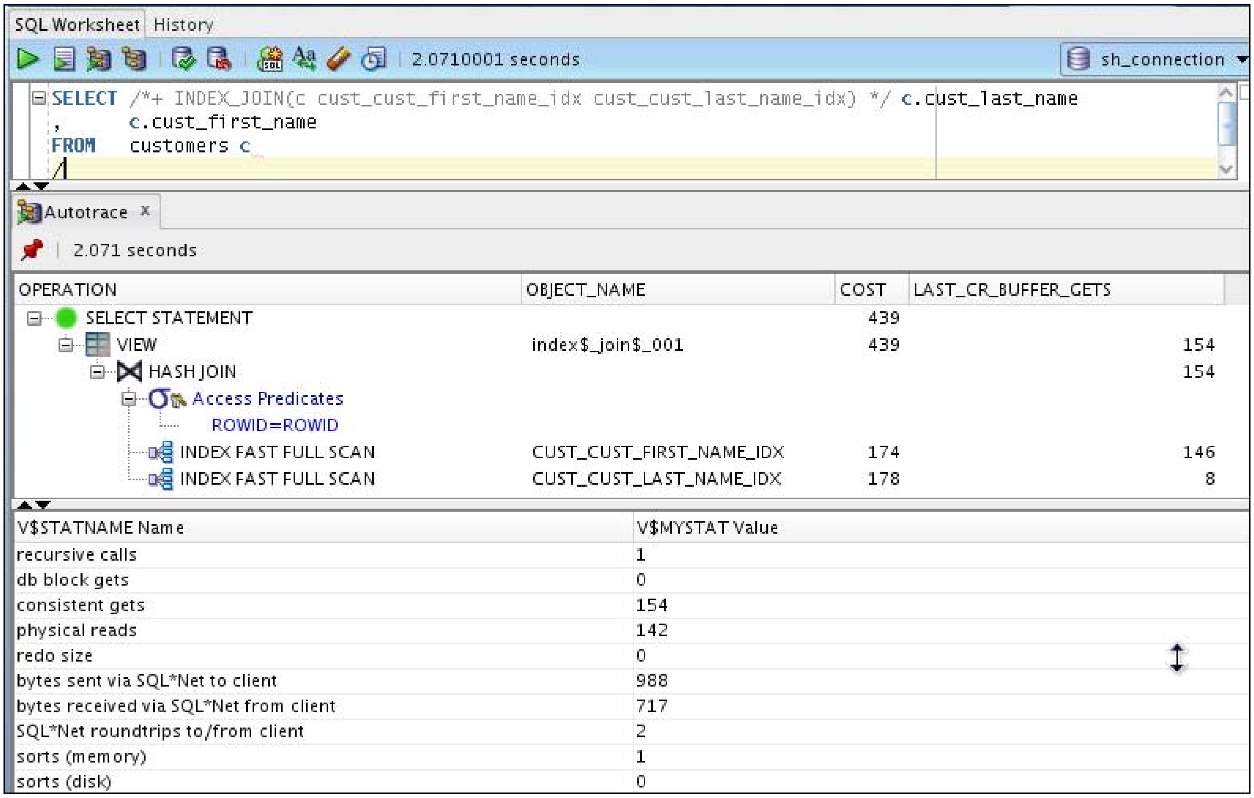
a. Open and execute the `create_last_name_index.sql` script.



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



33. Autotrace the query in the `query10.sql` script. What do you observe?

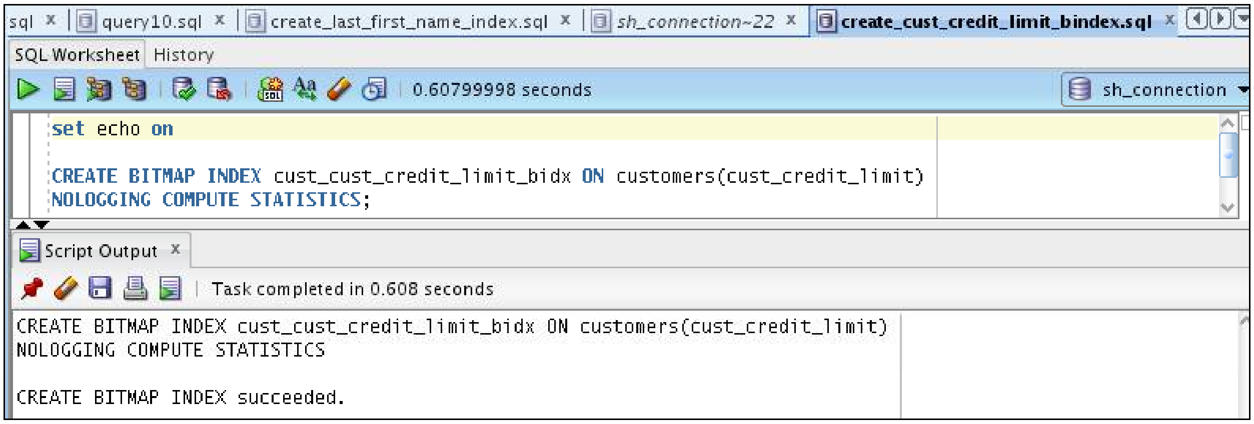


– Although the optimizer can use both the indexes, the resulting cost is not better than the concatenated index.

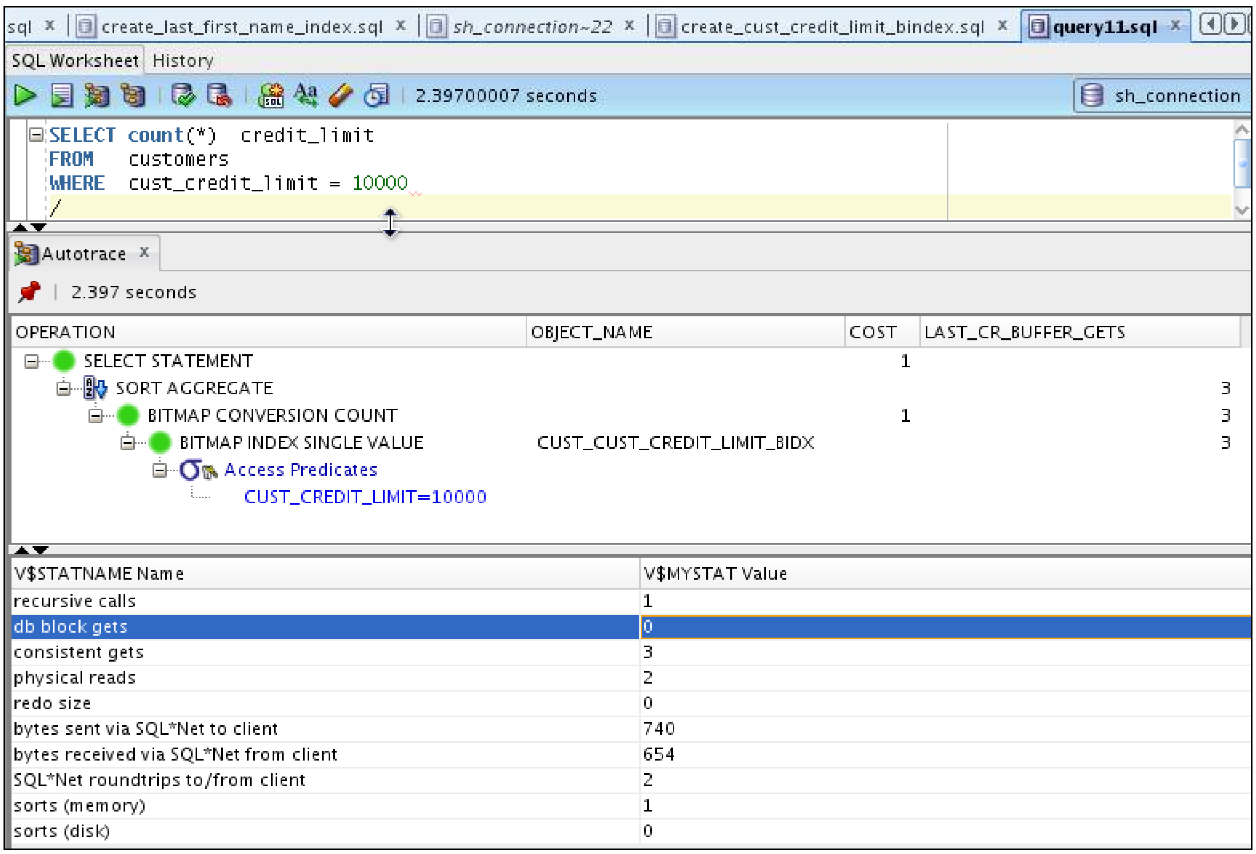
34. **Case 8: Bitmap Index Only Access:** Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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


…
Connected to:…

SQL> @drop_customers_indexes.sql
```

35. 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.



36. 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.

37. **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 indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

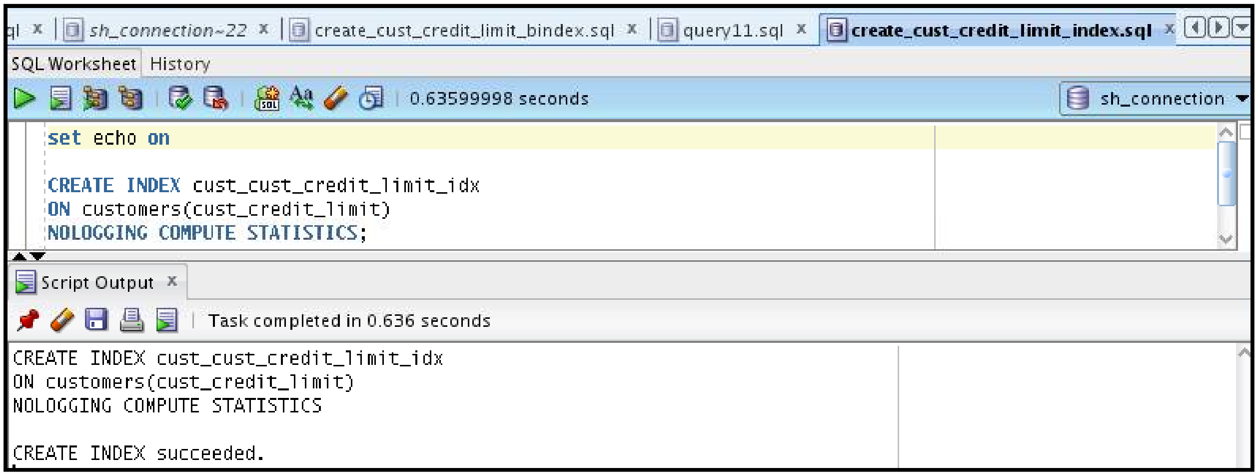
```
$ cd /home/oracle/solutions/Access_Paths/
$ sqlplus sh/sh

…
Connected to:…
```

```
SQL> @drop_customers_indexes.sql
```

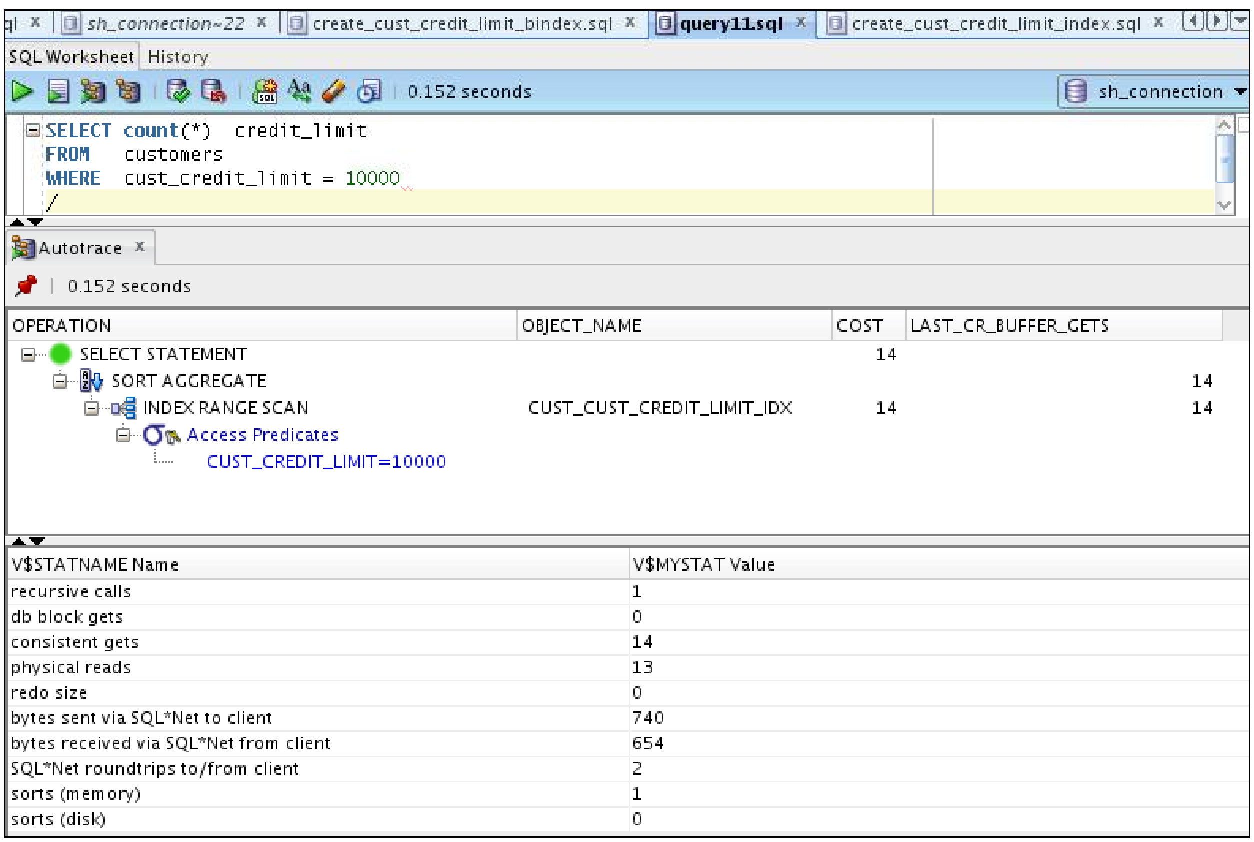
38. After this, create one B*-tree index on the following column of the `CUSTOMERS` table:

    `cust_credit_limit (create_cust_credit_limit_index.sql)`

    Open and execute the `create_cust_credit_limit_index.sql` script.



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



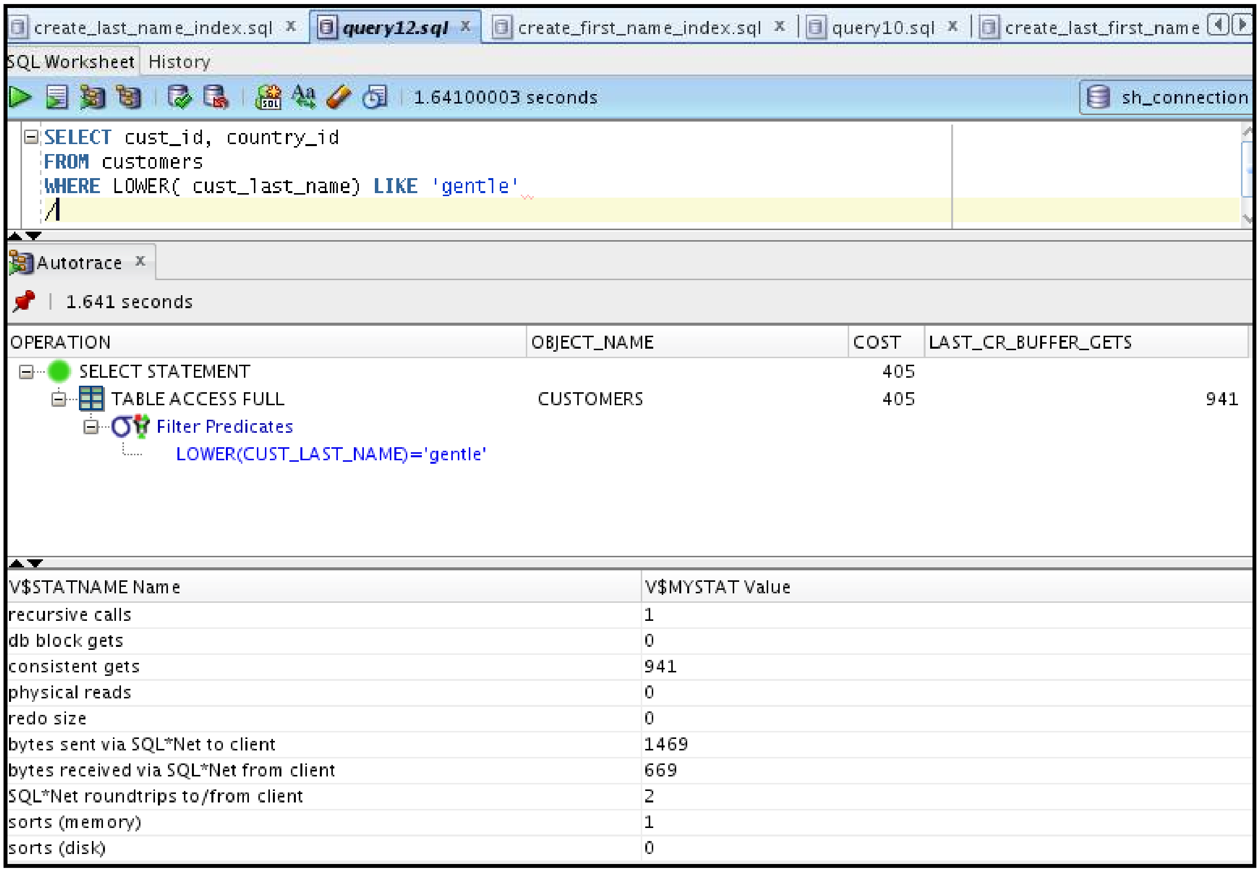
    – The optimizer uses the B*Tree index; however, this is less efficient compared to the corresponding bitmap index from the previous case.

40. **Case 10: Function Based Index:** Drop all the `CUSTOMERS` indexes except its primary key index. Open a terminal window. Connect to `sh` and drop all indexes currently created on the `CUSTOMERS` table except its primary key index by executing `drop_customers_indexes.sql`.

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


…
Connected to:…

SQL> @drop_customers_indexes.sql
```

41. After this, create one B*-tree index on the following column of the `CUSTOMERS` table: `cust_last_name (create_last_name_index.sql)`

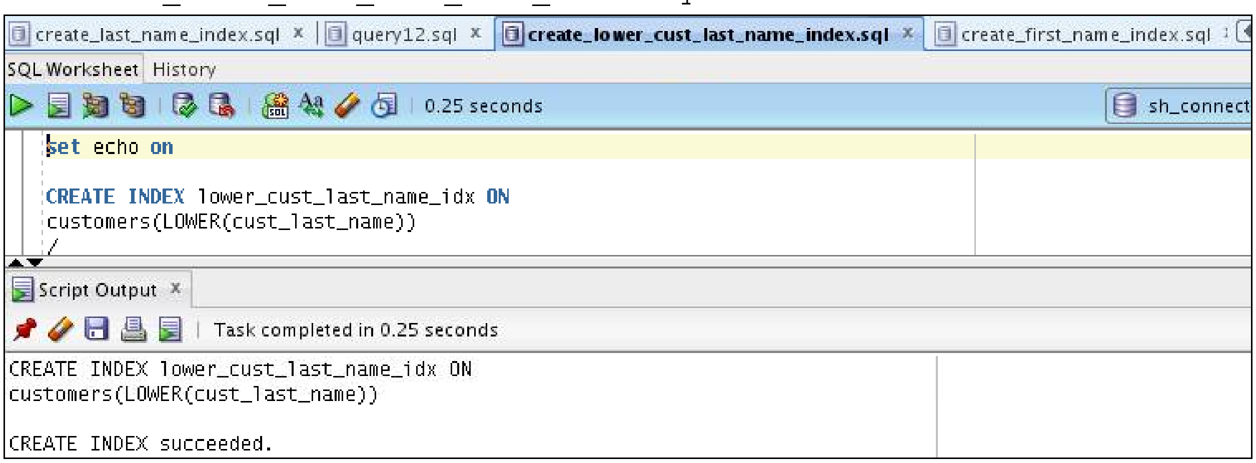
42. Autotrace the query in `query12.sql`. What do you observe?



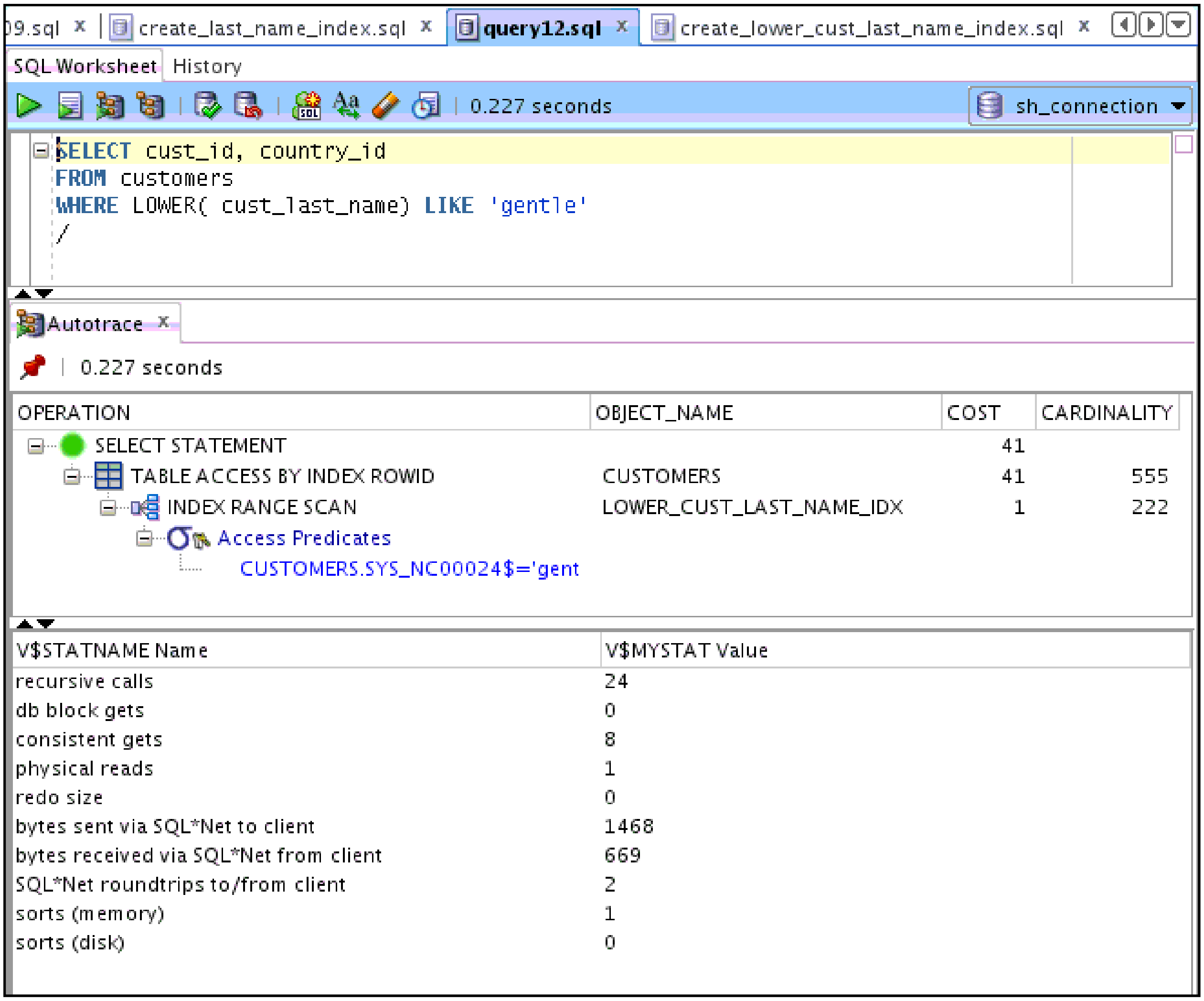
– Although there is an index, it cannot be used because its column is modified by a function.

43. How can you enhance the performance of the previous query without modifying the statement itself? Implement your solution.

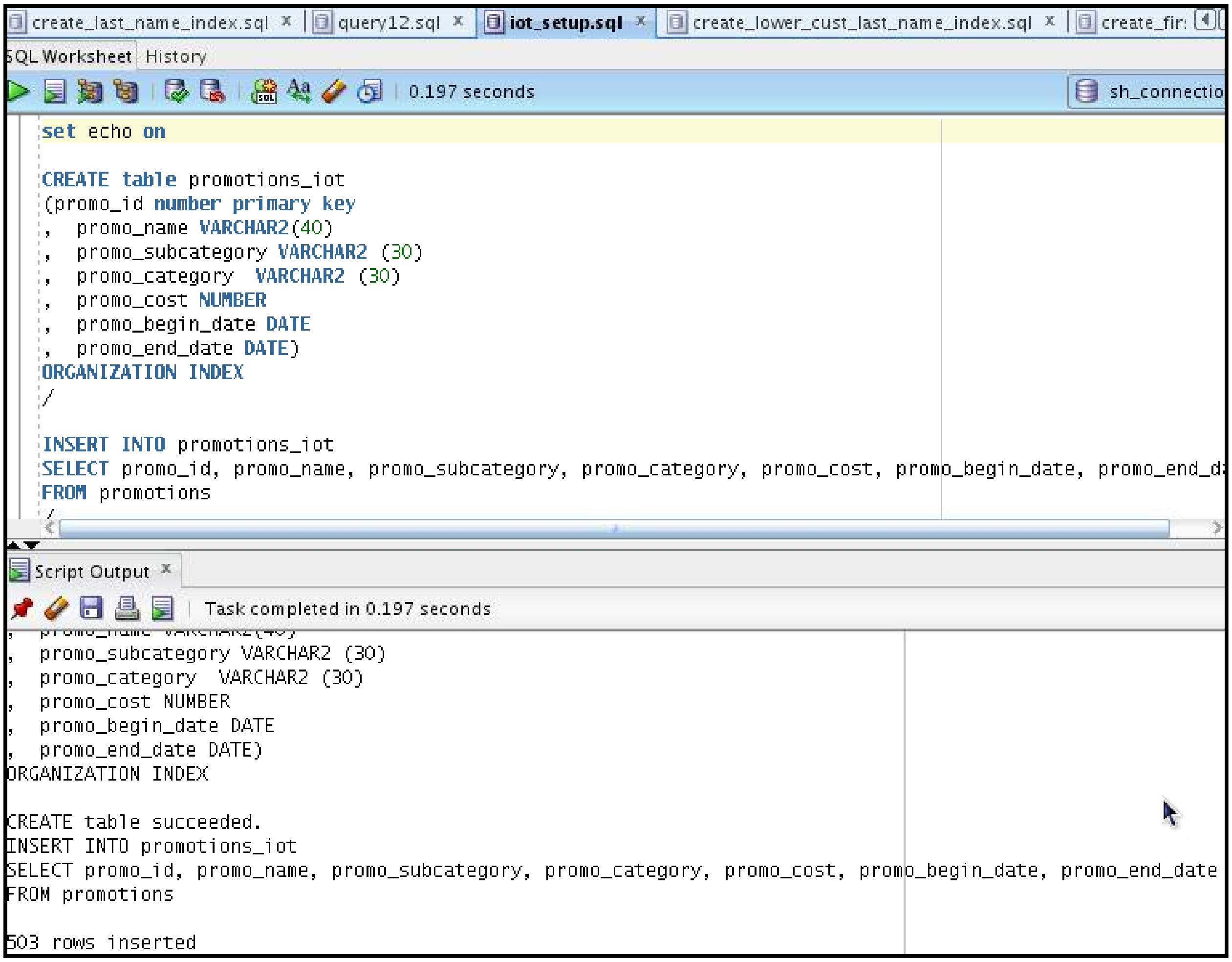
a. You can create a function-based index using `create_lower_cust_last_name_index.sql`.



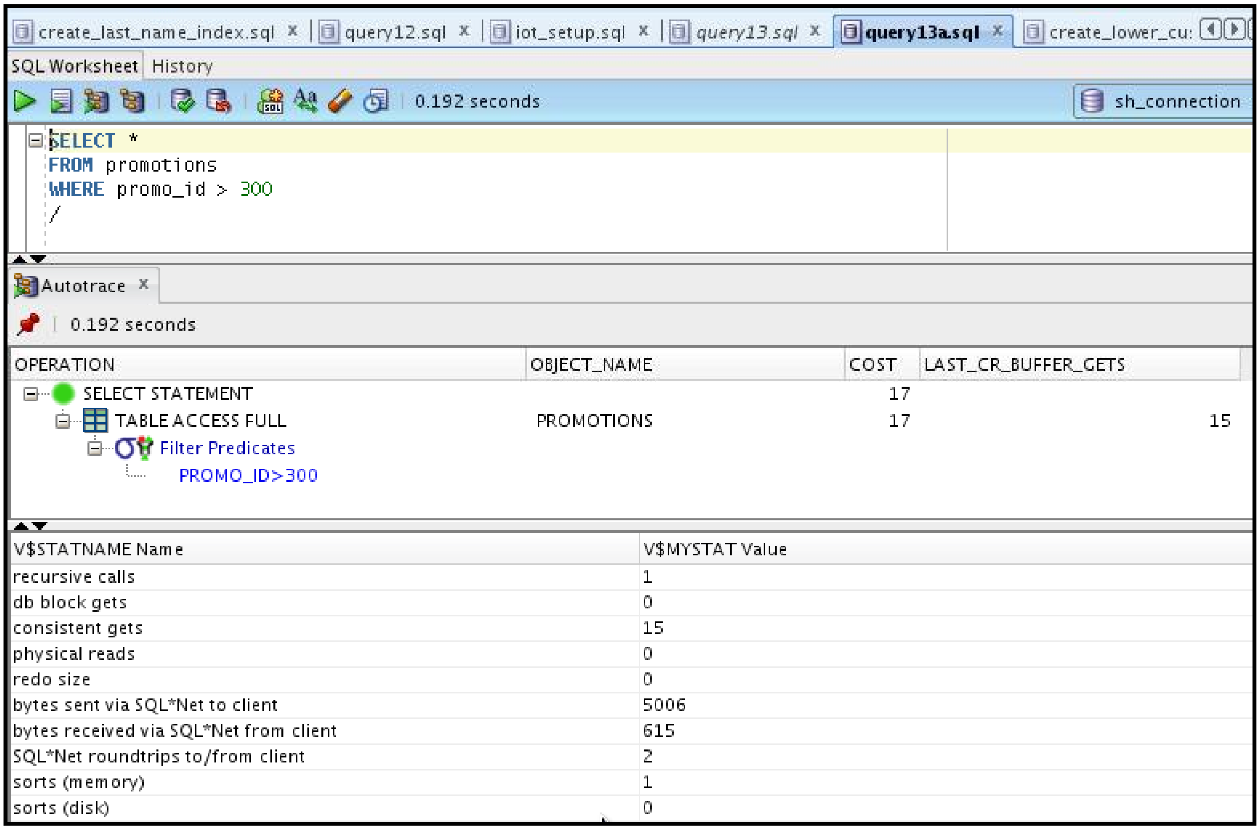
44. Check if your solution executes faster than in the case of the query in step 42.

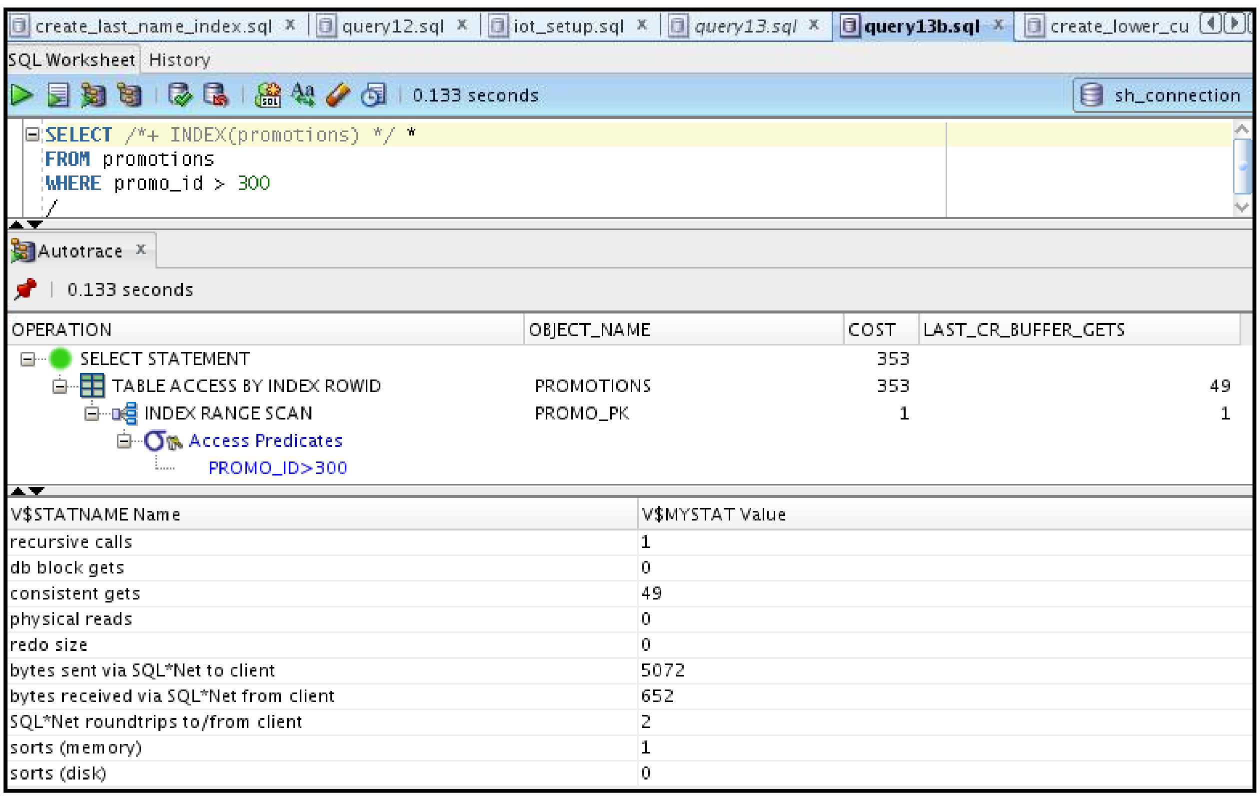


45. **Case 11: Index Organized Table:** Execute the `iot_setup.sql` script to set up the environment for this case.



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



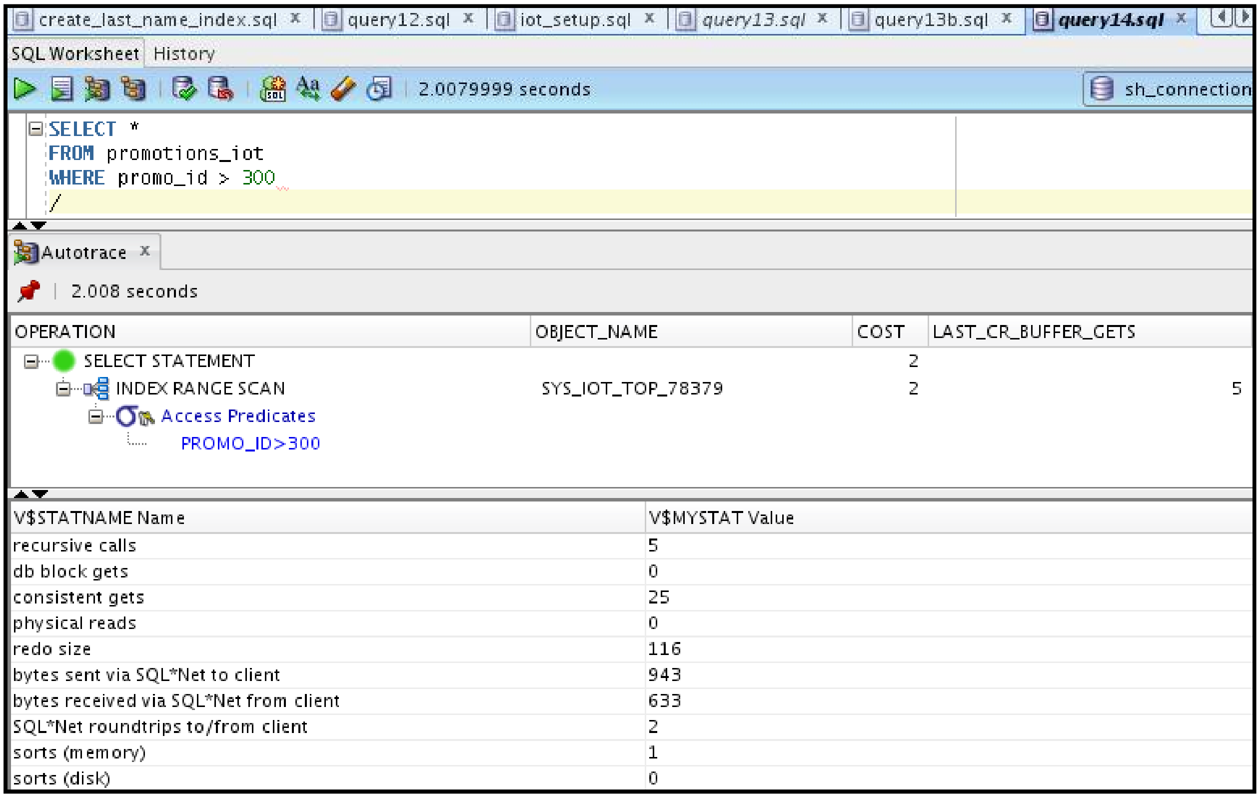


- The first lets the optimizer decide the plan, and the best it can find is to do a full table scan. Forcing the use of the index is not a good idea because it takes longer to execute.

47. Autotrace the query in `query14.sql`.

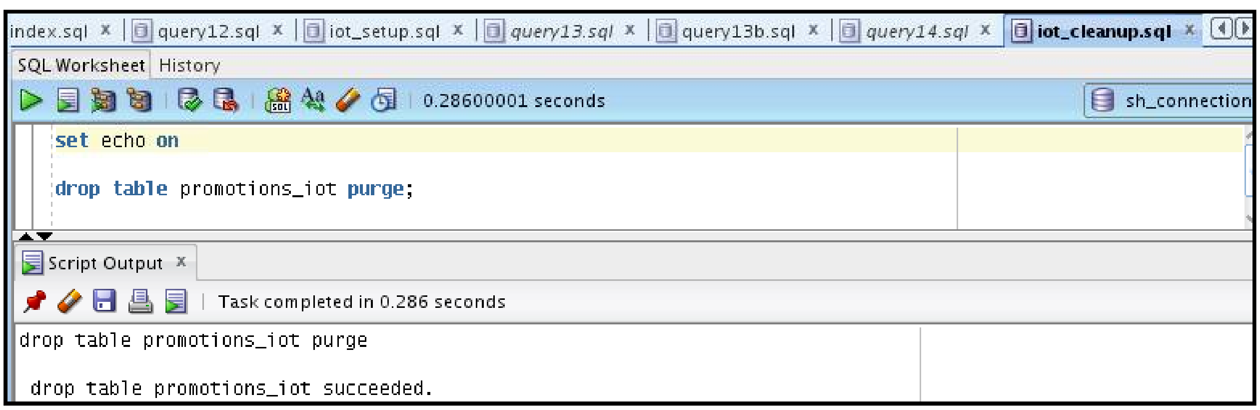
```
SELECT *
FROM promotions_iot
WHERE promo_id > 300;
```

What do you observe?

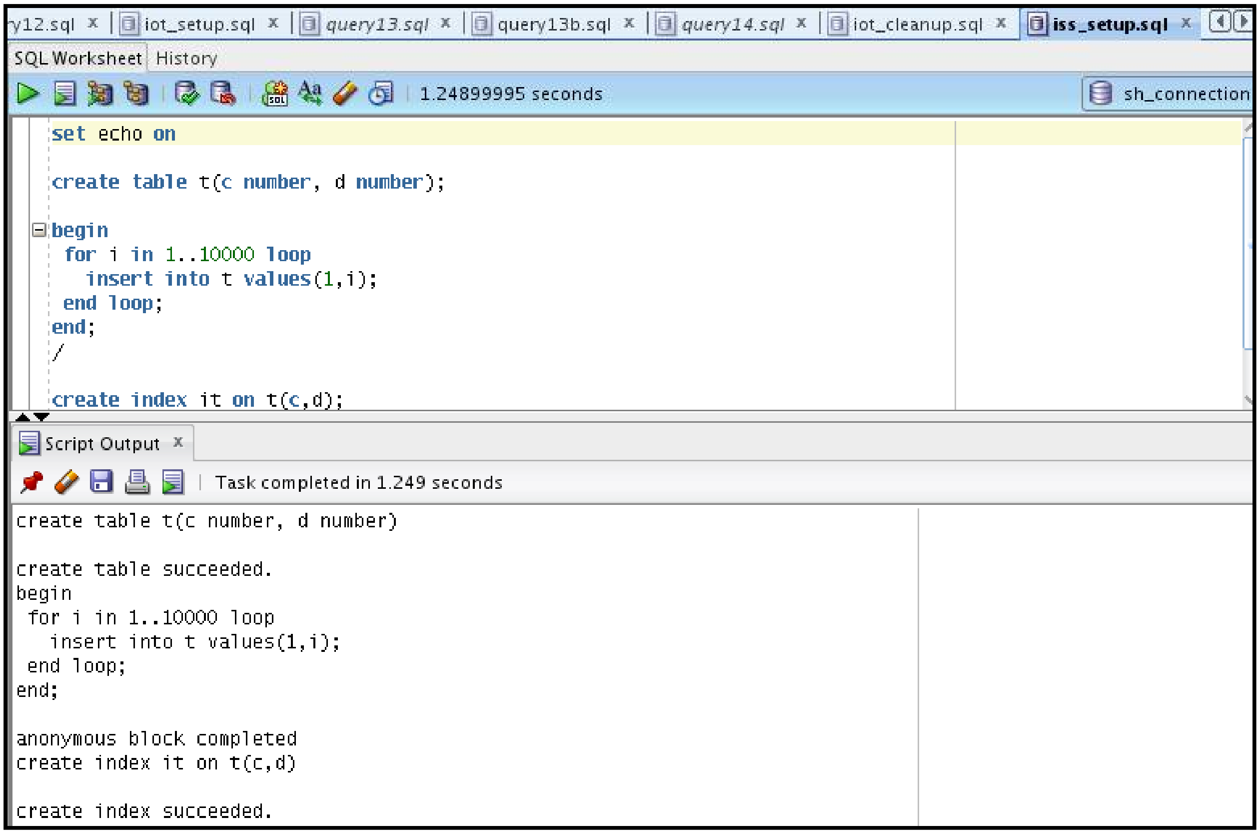


– The optimizer directly uses the index-organized structure, which is extremely efficient in this case compared to the previous step.

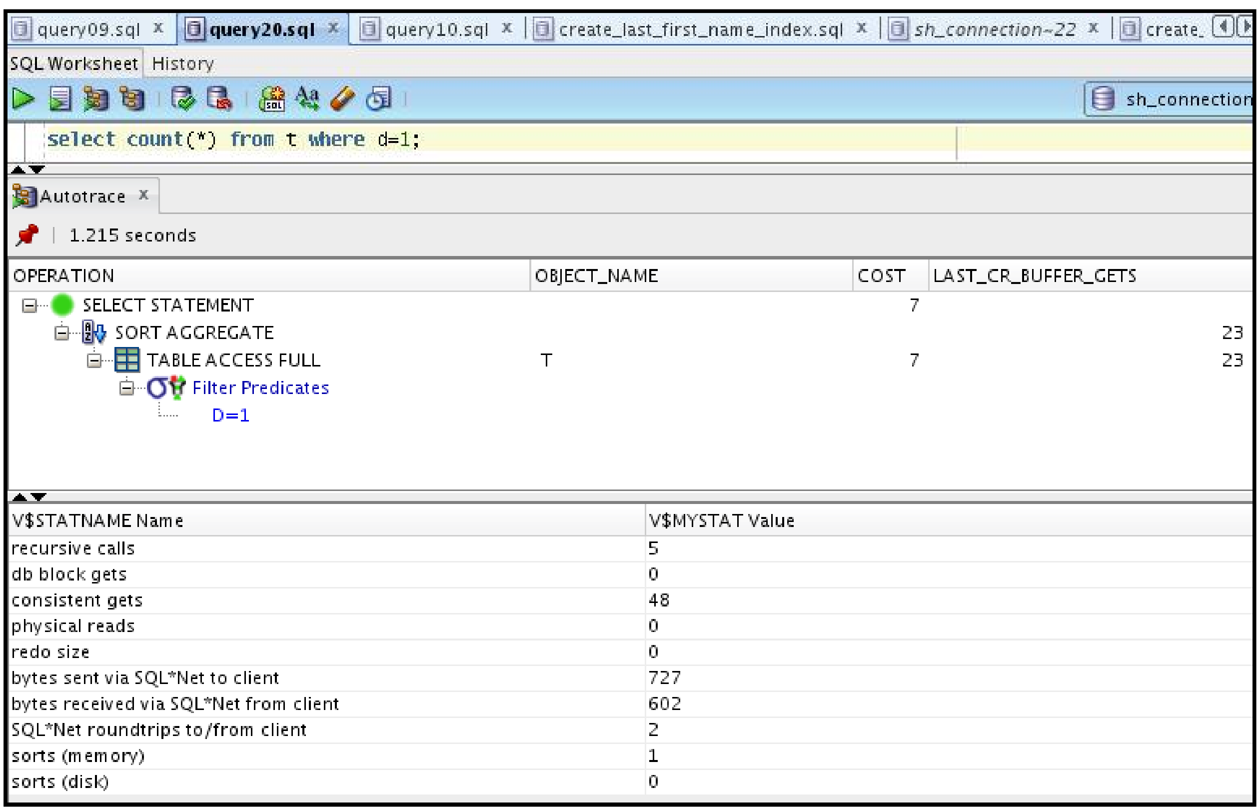
48. Open and execute the `iot_cleanup.sql` script to clean up your environment.



49. **Case 12: Index Skip Scan:** Execute the `iss_setup.sql` script to set up your environment for this lab.



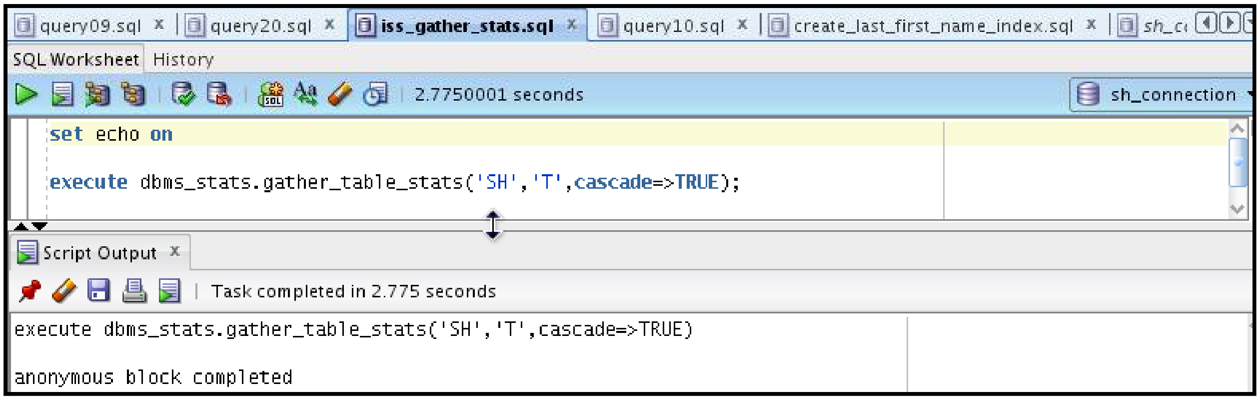
50. Autotrace the query in `query20.sql`. What do you observe?



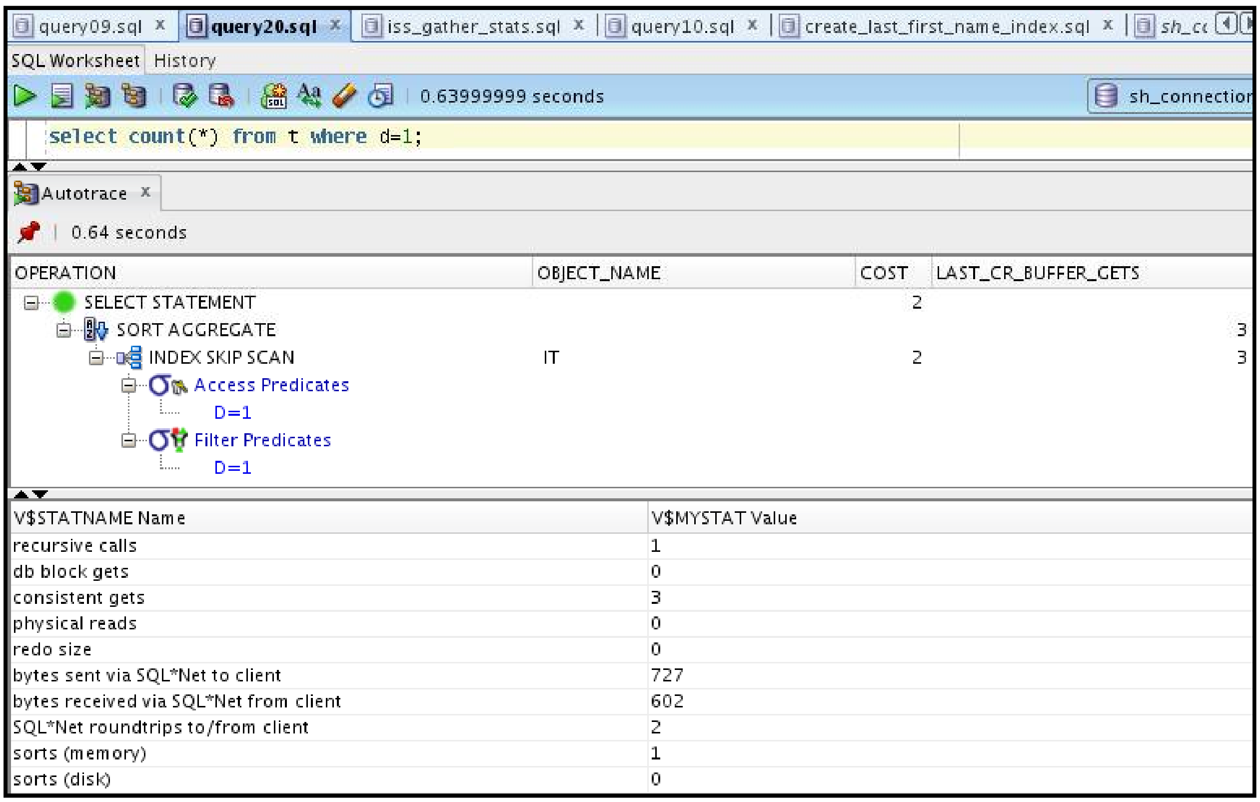
– The optimizer is not using the index and does a full table scan.

51. 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.



52. Autotrace the query in `query20.sql`. What do you observe?

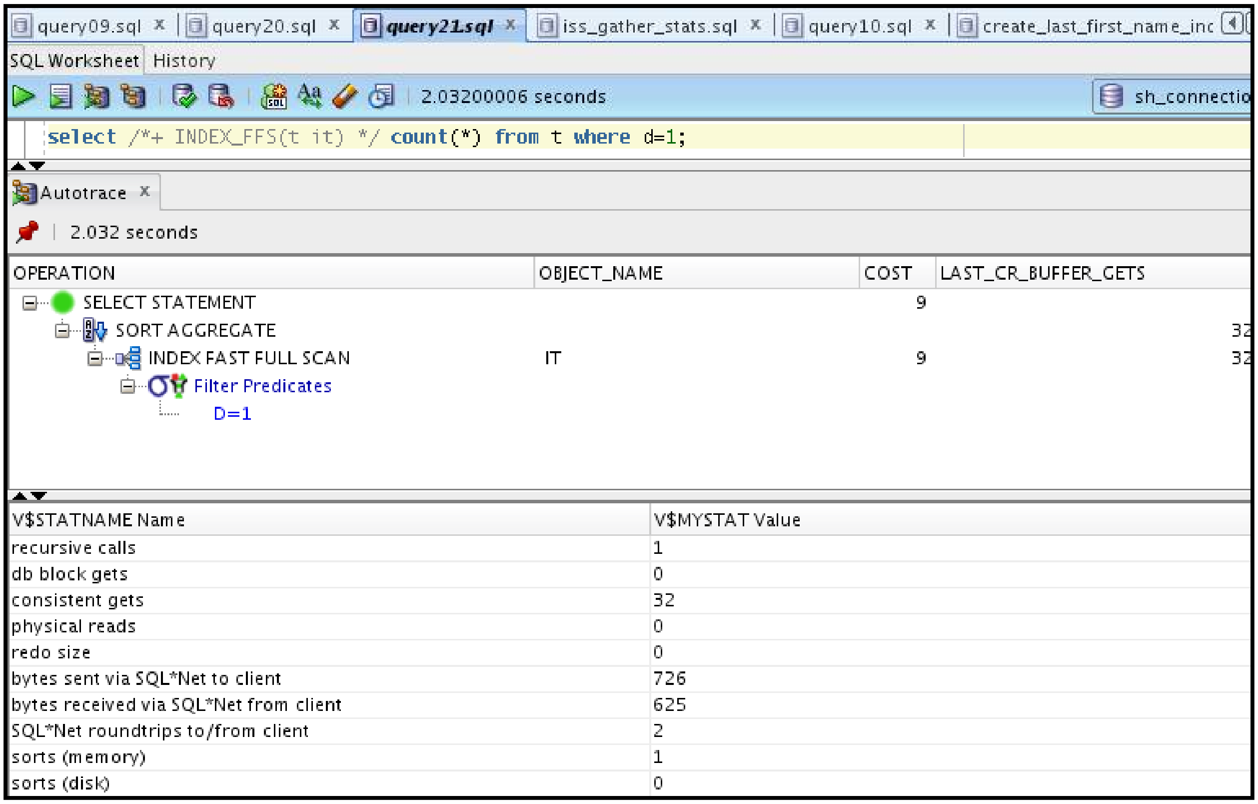


   – The optimizer now uses the index to perform an index skip scan.

53. Compare the result of executing `query20.sql` with the result you obtain when you execute 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.

54. Execute the `iss_cleanup.sql` script to clean up your environment for this case.

Practices for Lesson 7

Chapter 7

Overview of Practices for Lesson 7

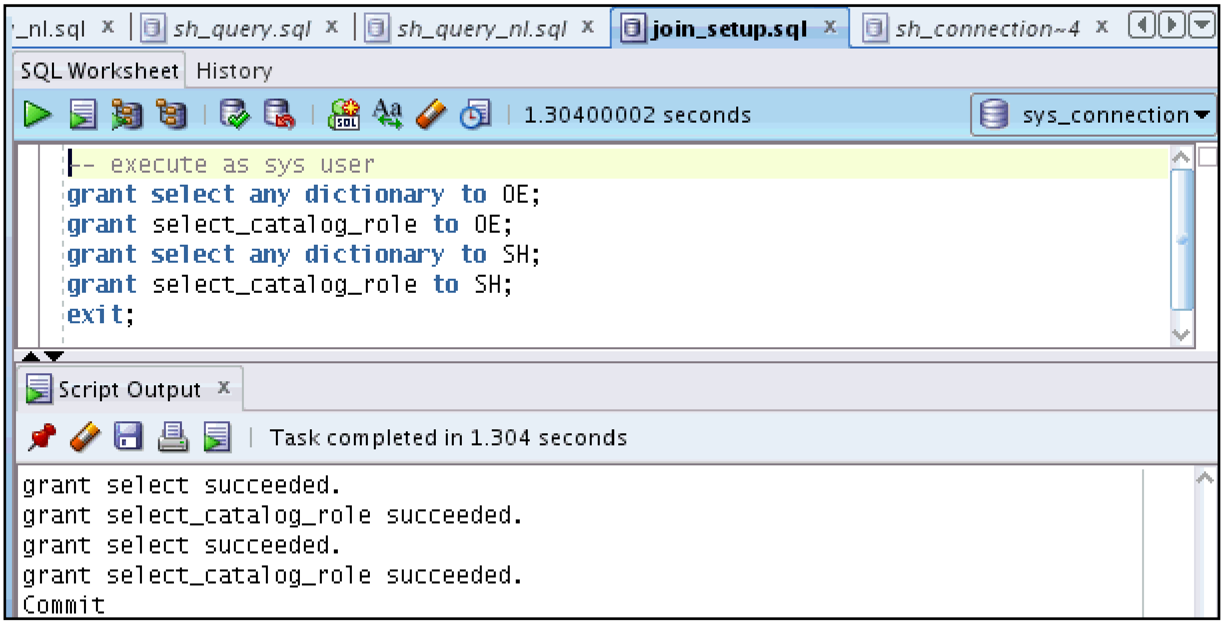
Practices Overview

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

Practice 7-1: Using Join Paths

In this practice, you explore various access paths 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 lab can be found in your `$HOME/solutions/Access_Paths` directory.

1. Open and execute the `join_setup.sql` script using `sys_connection`.



2. Open the connection named `scott` that you created in Practice 4. If you have not created this connection, create it now with the following attributes.

   Create a connection with the following specifications:

   `Name: scott`

   `Username: scott`

   `Password: tiger`

   Select Save Password.

   `Sid: orcl`

   Click Test.

   Click Connect.

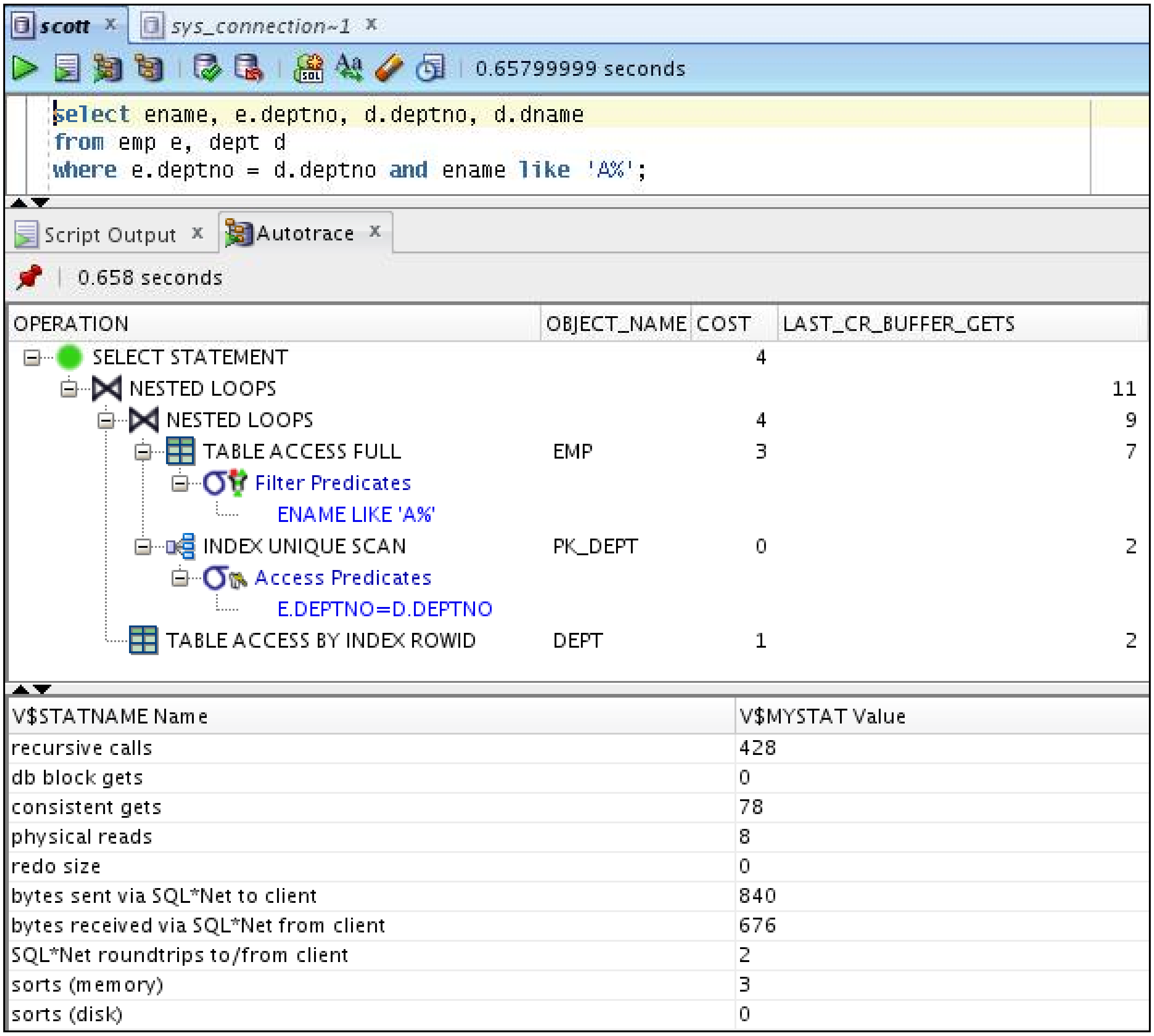
3. **Case 1: Nested Loops Join Path:** The nested loop join method works well with very small tables, and larger tables with indexes on the join column and at least one table that produces a small row source. The nested loop method will produce joined rows as soon as a match is found, so is sometimes the preferred method for the FIRST_ROWS access goal.

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

   ```
   select ename, e.deptno, d.deptno, d.dname
   from emp e, dept d
   where e.deptno = d.deptno and ename like 'A%';
   ```

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

   b. The optimizer chooses the nested loop join method for this query by default. The cost is low. This is expected because both tables in the query are very small.

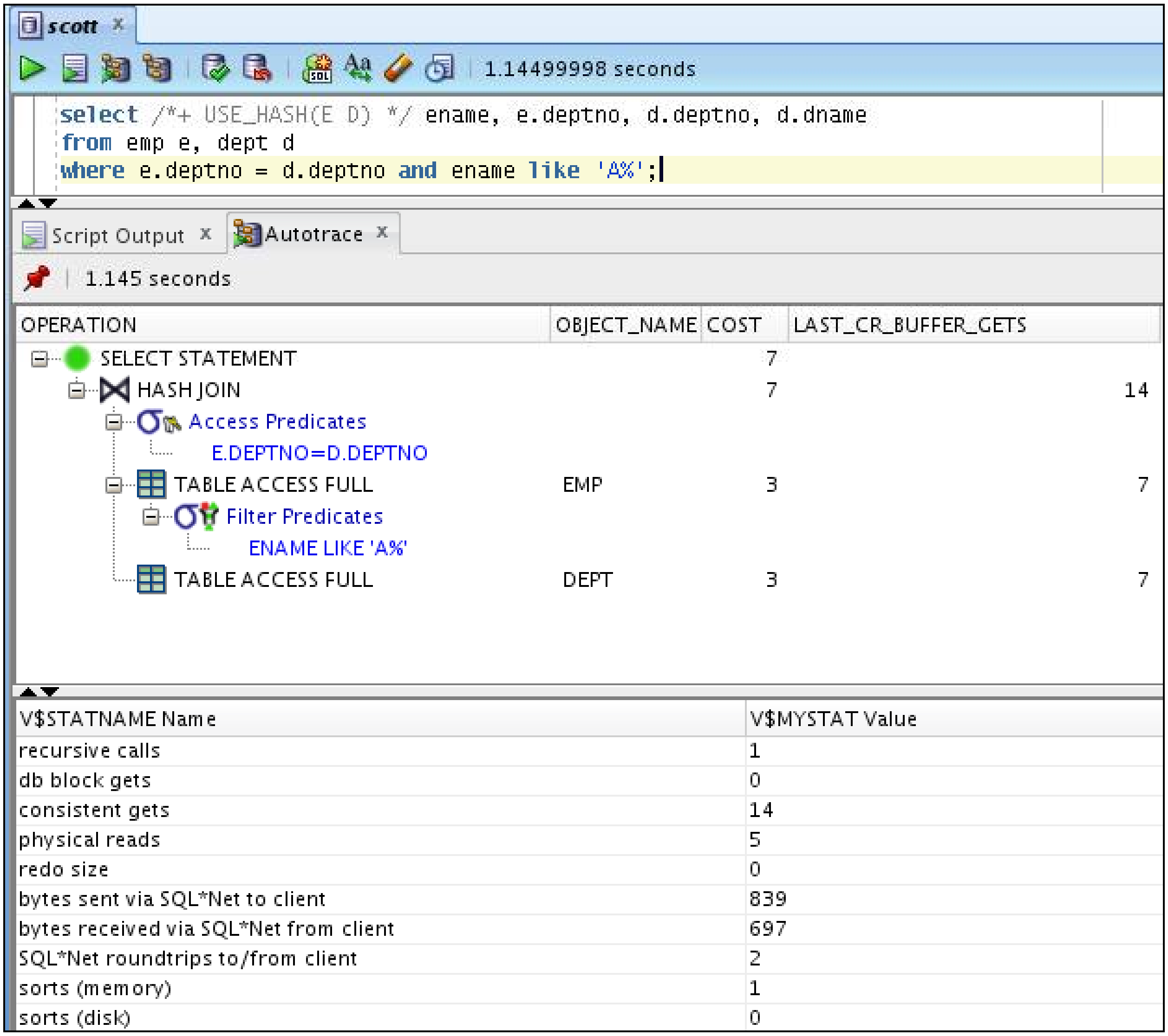


c. Enter the same query into SQL*Developer or open the `small_tables_hash_join.sql` script, but force the use of a hash join by using the following query with hints. Then use Autotrace to observe the differences in execution plan and statistics.

```
select /*+ USE_HASH(E D) */ ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
```

What do you observe?

The optimizer is forced to use the hash join method. The number of consistent gets is higher, and the estimated cost is higher. Note that the elapsed time may vary depending on any other activity on the machine.

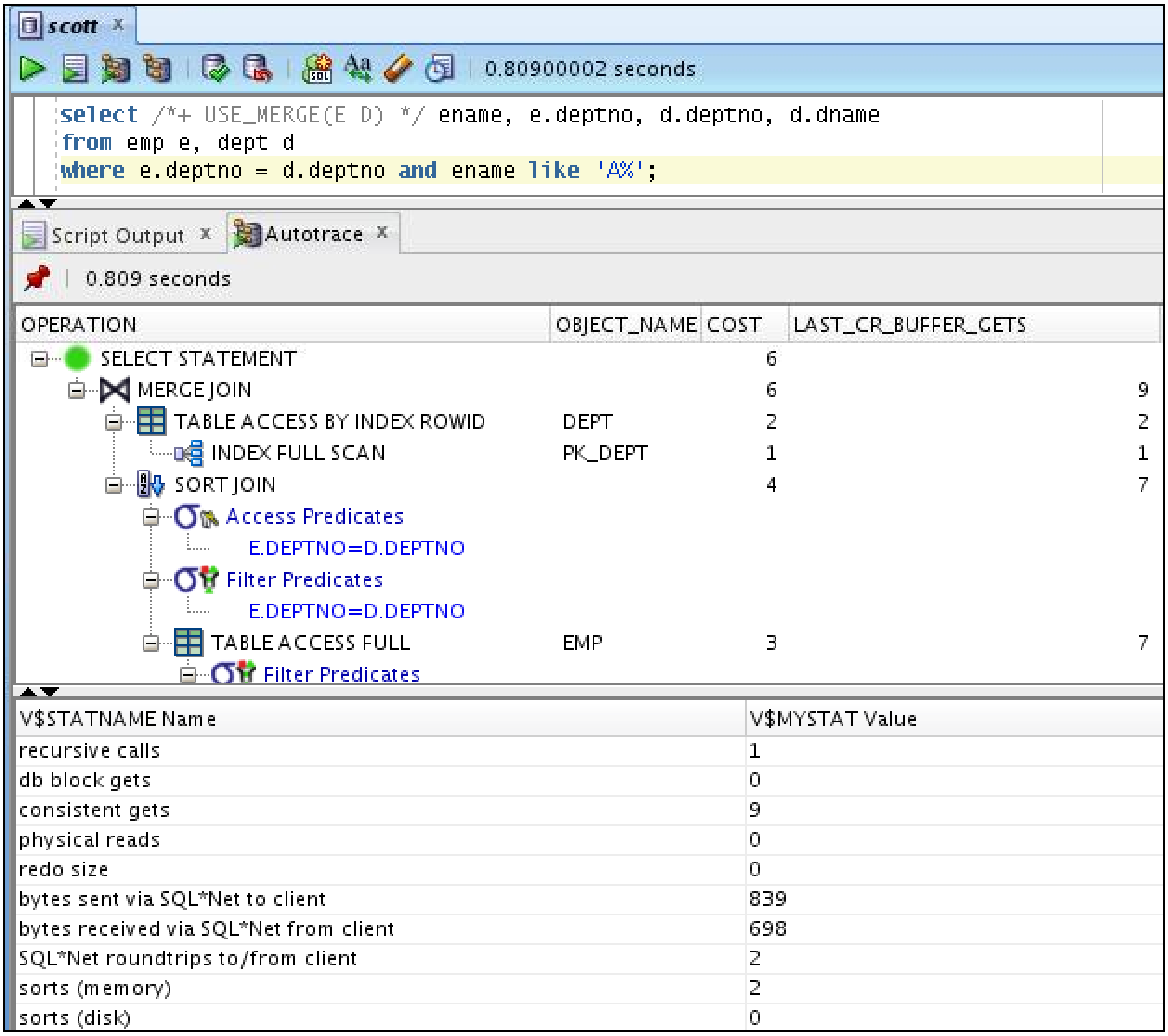


d. Enter the same query into SQL*Developer or open `small_tables_merge_join.sql`, but force the use of a merge-sort join by using following query with hints. Then use Autotrace to observe the differences in execution plan and statistics.

```
select /*+ USE_MERGE(E D) */ ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
```

What do you observe?

The optimizer is forced to use the merge join method. The number of consistent gets is lower than nested loops, but there is the additional cost of the sorts that is not included in the consistent gets. The estimated cost is higher. Note that the elapsed time may vary depending on any other activity on the machine.



4. Open the connection named `oe_connection` that you created in Practice 2. If you have not created this connection, create it now with the following attributes.
   a. Click the Add Connection button.
   b. Create a connection with following specifications:

      `Name: oe_connection`

      `Username: oe`

      `Password: oe`

      Select Save Password.

      `Sid: orcl`

      Click Test.

      Click Connect.

5. **Case 2: Merge Join:** The merge join is a general purpose join method that can work when other methods cannot. The merge join must sort each of the two row sources before performing the join. Because both row sources must be sorted before the first joined row is returned, the merge join method is not well suited for use with the FIRST_ROWS goal.

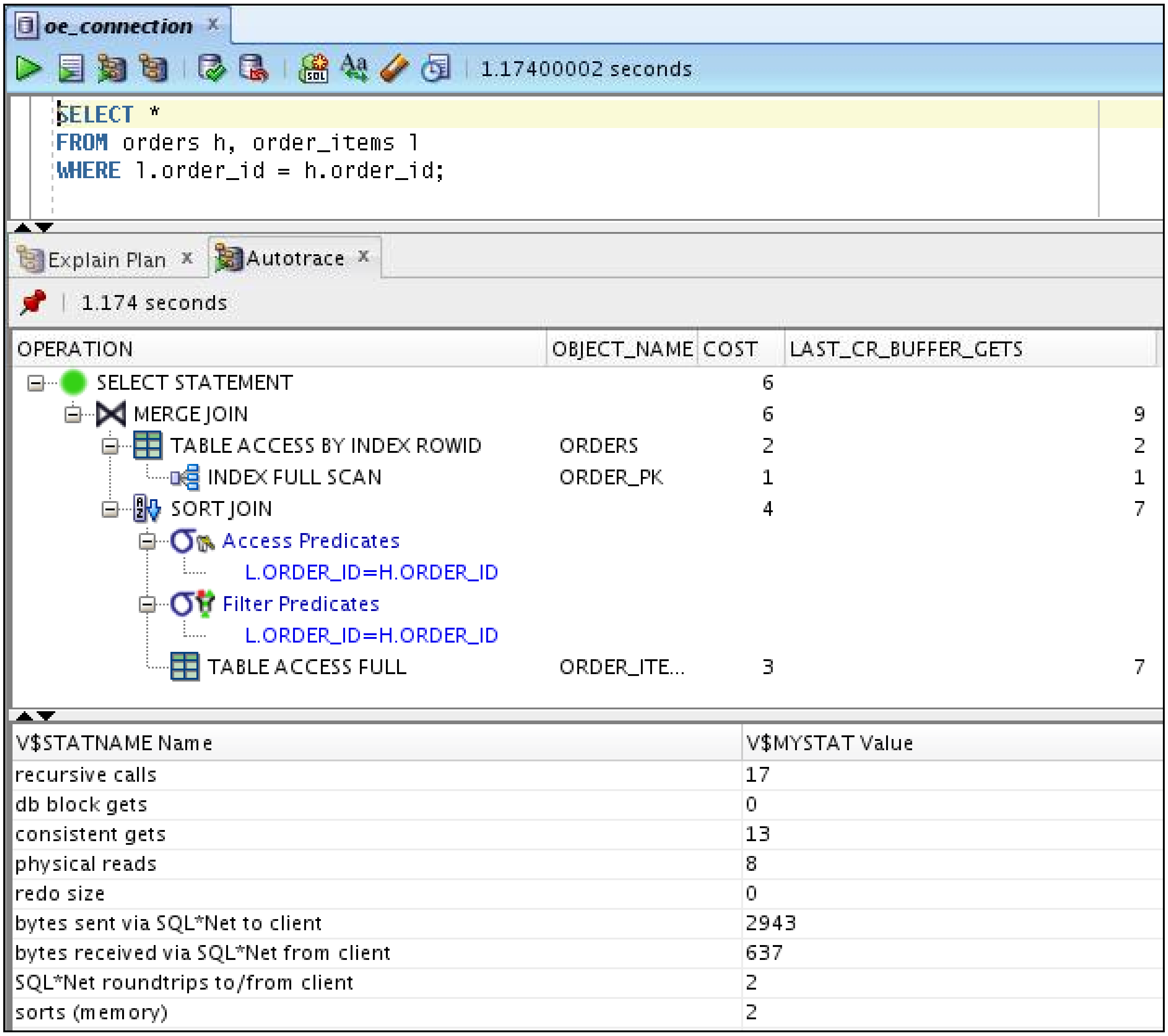
a. Enter the following query in a SQL Developer worksheet connected as the `OE` user or open the `oe_query.sql` script.

```
SELECT *
FROM orders h, order_items l
WHERE l.order_id = h.order_id;
```

b. Autotrace this query twice to warm the buffer cache.

What do you observe?

Notice that the merge join is chosen by the optimizer.

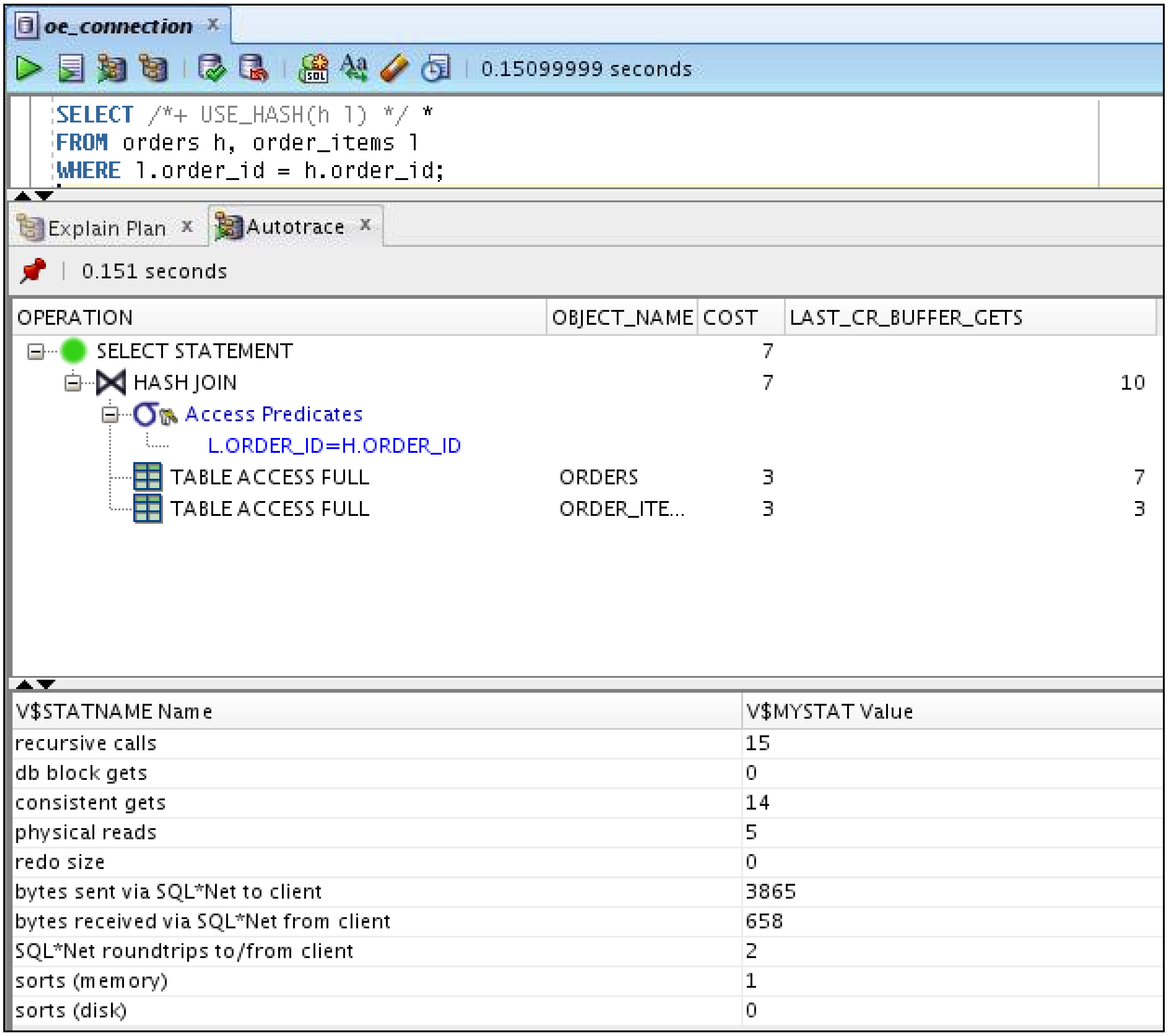


c. Enter the same query with a hint to force the use of a hash join or open the `oe_query_hash.sql` script.

```
SELECT /*+ USE_HASH(h l) */ *
FROM orders h, order_items l
WHERE l.order_id = h.order_id;
```

What do you observe?

Compare the estimated cost to the merge join above, 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.



OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		7	
HASH JOIN		7	10
Access Predicates			
L.ORDER_ID=H.ORDER_ID			
TABLE ACCESS FULL	ORDERS	3	7
TABLE ACCESS FULL	ORDER_ITE...	3	3

V$STATNAME Name	V$MYSTAT Value
recursive calls	15
db block gets	0
consistent gets	14
physical reads	5
redo size	0
bytes sent via SQL*Net to client	3865
bytes received via SQL*Net from client	658
SQL*Net roundtrips to/from client	2
sorts (memory)	1
sorts (disk)	0

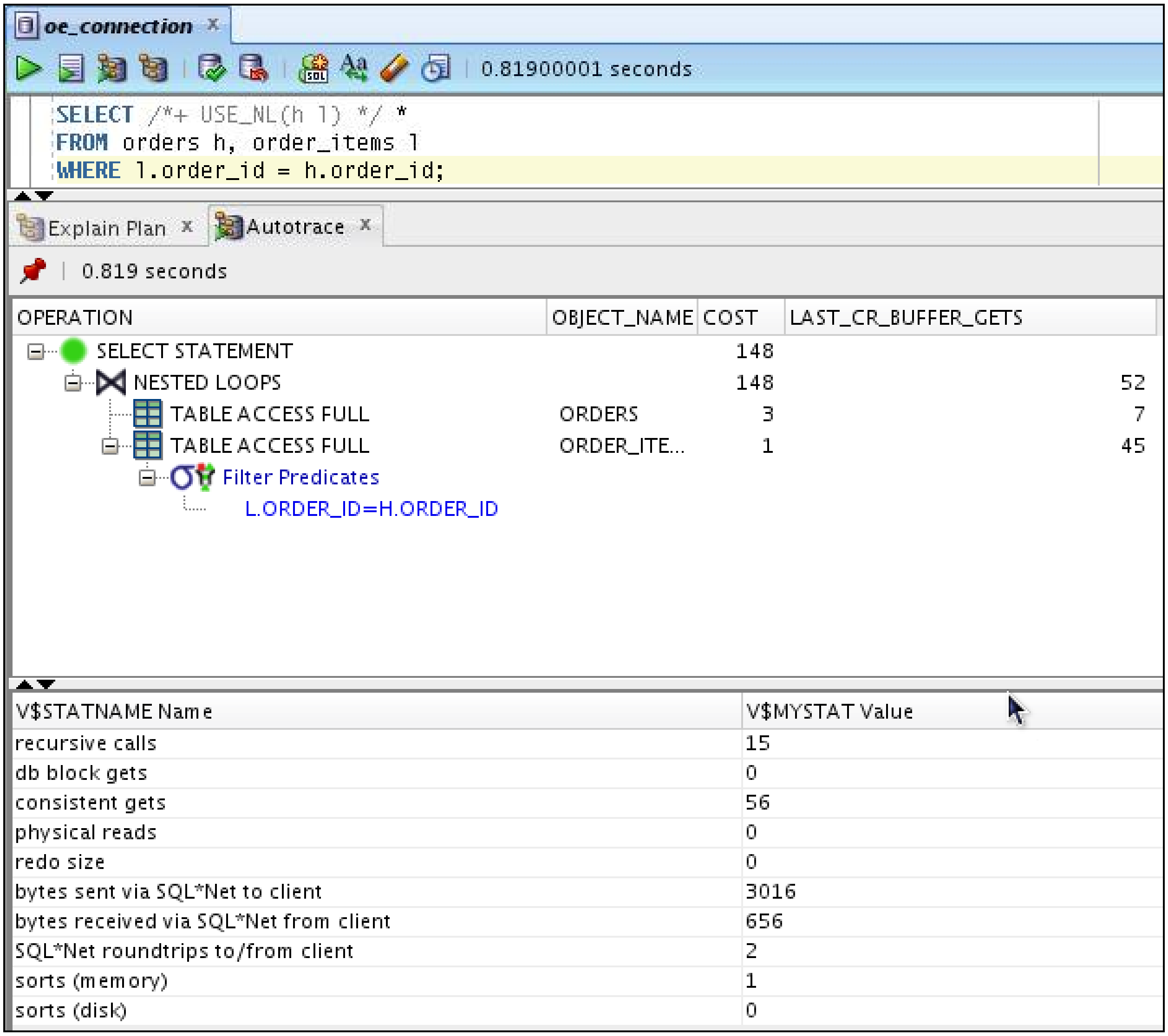
d. Enter the same query again changing the hint to force the use of nested loops, or open the `oe_query_nl.sql` script.

```
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 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.



6. Open the connection named `sh_connection` that you created in Practice 2. If you have not created this connection, create it now with the following attributes.
   a. Click the Add Connection button.
   b. Create a connection with the following specifications:
      `Name: sh_connection`
      `Username: sh`
      `Password: sh`
      Select Save Password.
      `Sid: orcl`
      Click Test.
      Click Connect.
7. **Case 3: Hash Join:** The hash join method performs very well on large row sources and on row sources where one row source is smaller. The hash join builds a hash table in memory (and overflows to temp tablespace if the row source is too large) on the smaller of the row sources. The procedure then reads the second row source probing for the hash value in the

first. Because the rows are joined as soon as a match is found, the hash join method is also preferred for the FIRST_ROWS goal.

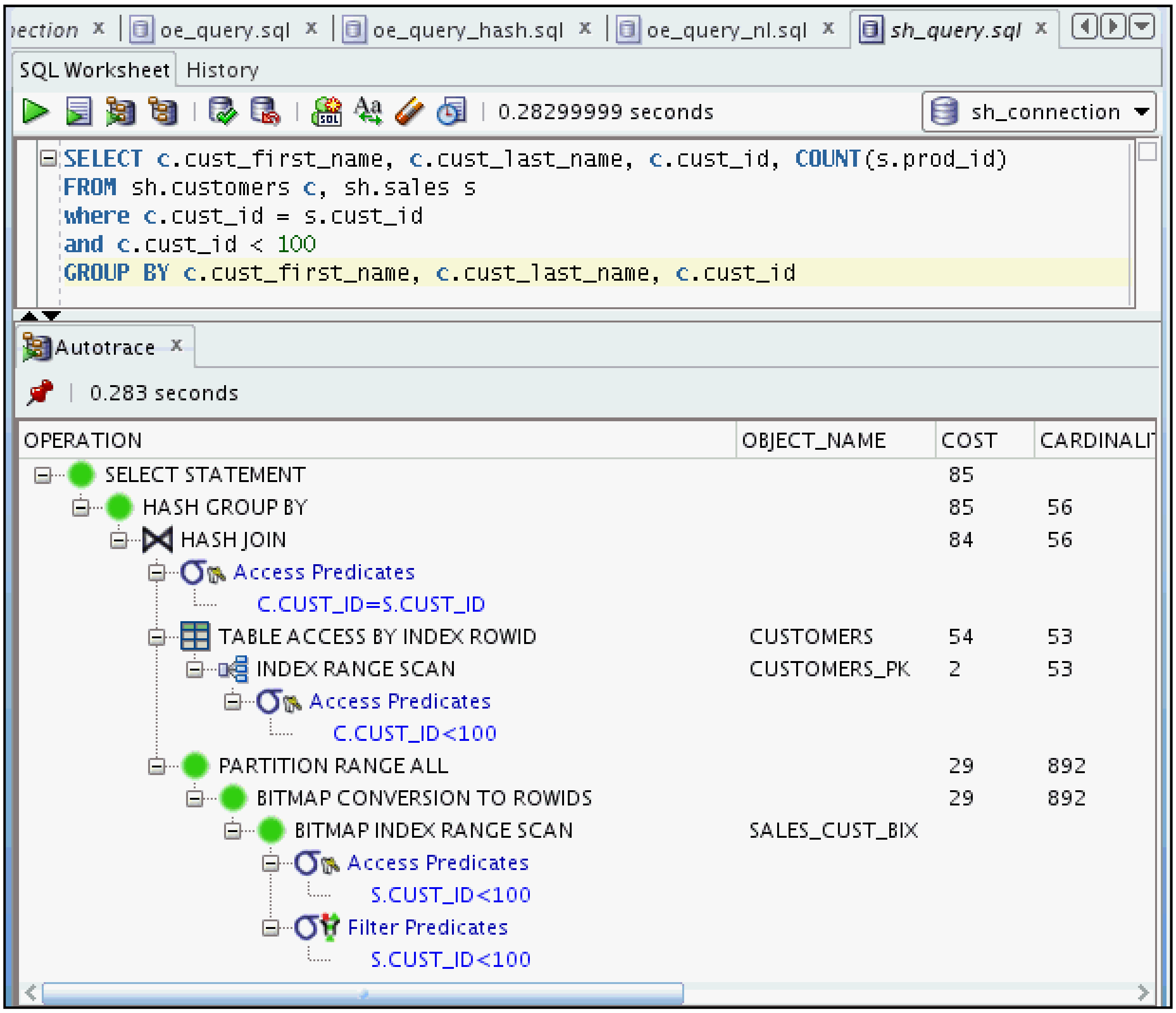
a. Enter the following query in SQL Developer or open `sh_query.sql`:

```
SELECT c.cust_first_name, c.cust_last_name, c.cust_id,
COUNT(s.prod_id)
FROM sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_id < 100
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id
```

Run Autotrace twice to warm the buffer cache.

What do you observe?

Notice the consistent gets and the estimated cost.



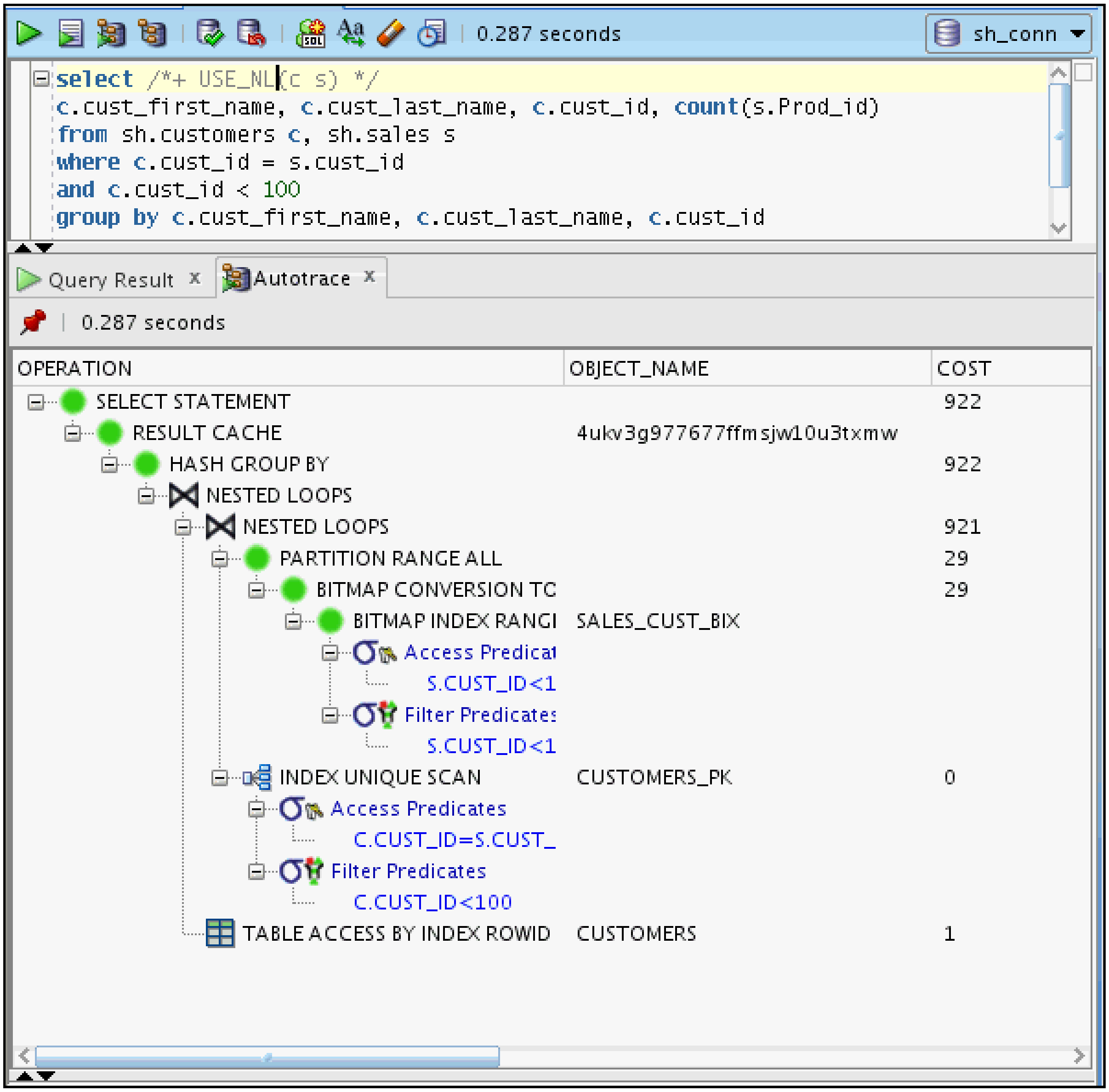
V$STATNAME Name	V$MYSTAT Value
recursive calls	2
db block gets	0
consistent gets	144
physical reads	0
redo size	0
bytes sent via SQL*Net to client	2100
bytes received via SQL*Net from client	778
SQL*Net roundtrips to/from client	2
sorts (memory)	1
sorts (disk)	0

b. Enter the same query again, but use a hint to force the nested loops method;

```
SELECT /*+ USE_NL(c s) */
c.cust_first_name, c.cust_last_name,
c.cust_id, COUNT(s.prod_id)
FROM sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_id < 100
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id
```

Run Autotrace. What do you observe?

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



V$STATNAME Name	V$MYSTAT Value
recursive calls	1
db block gets	0
consistent gets	5516
physical reads	0
redo size	0
bytes sent via SQL*Net to client	2083
bytes received via SQL*Net from client	798
SQL*Net roundtrips to/from client	2
sorts (memory)	1
sorts (disk)	0

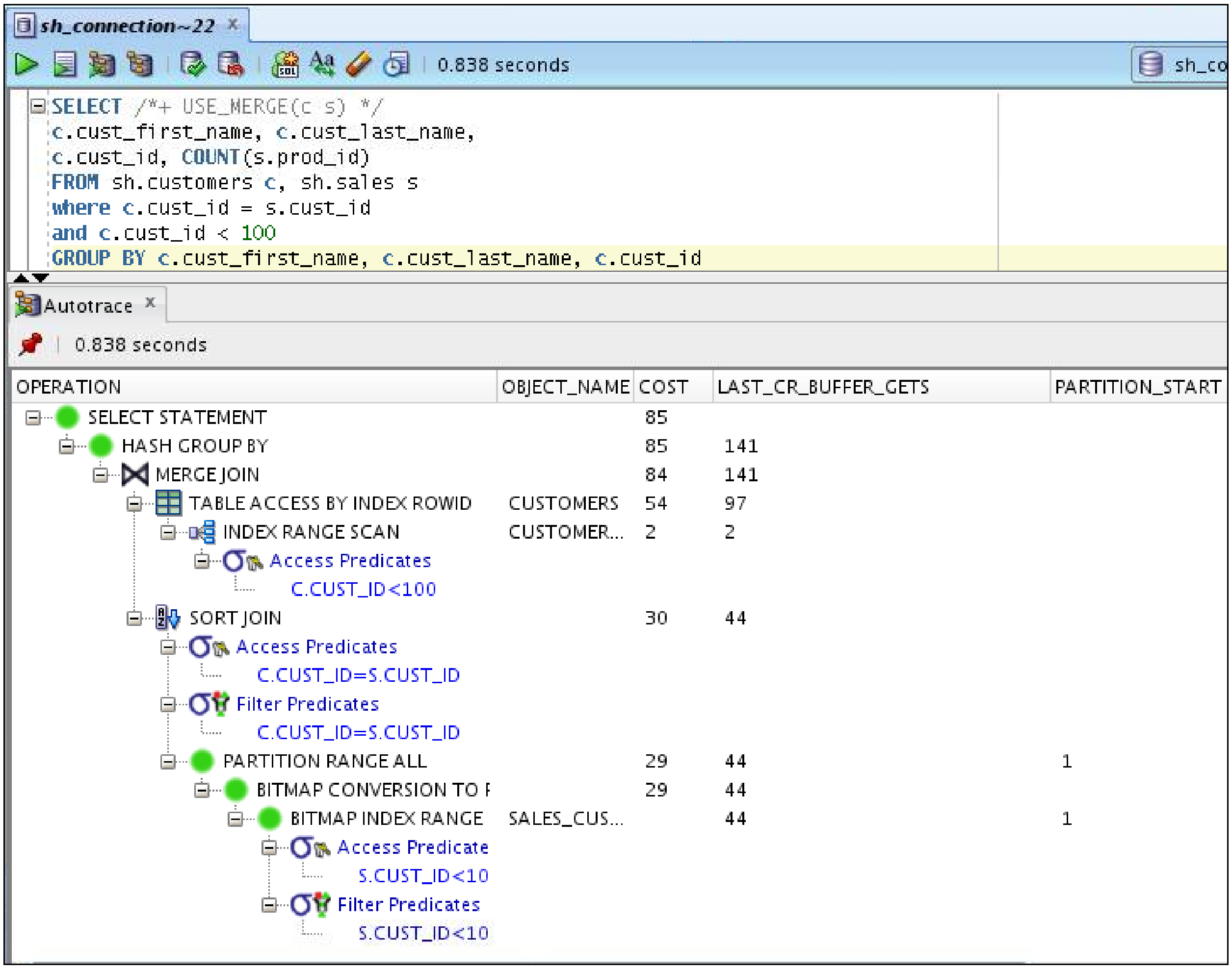
c. Enter the same query again. This time use a hint to force the merge join method or open the `sh_query_merge.sql` script.

```
SELECT /*+ USE_MERGE(c s) */
c.cust_first_name, c.cust_last_name,
```

```
c.cust_id, COUNT(s.prod_id)
FROM sh.customers c, sh.sales s
where c.cust_id = s.cust_id
and c.cust_id < 100
GROUP BY c.cust_first_name, c.cust_last_name, c.cust_id
```

Run Autotrace. What do you observe?

The consistent gets and the estimated cost values are higher than the hash join method, but still much lower than the values for the nested loops method.



V$STATNAME Name	V$MYSTAT Value
recursive calls	1
db block gets	0
consistent gets	141
physical reads	0
redo size	0
bytes sent via SQL*Net to client	2073
bytes received via SQL*Net from client	801
SQL*Net roundtrips to/from client	2
sorts (memory)	2
sorts (disk)	0

Practices for Lesson 8

Chapter 8

Overview of Practices for Lesson 8

Practices Overview

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

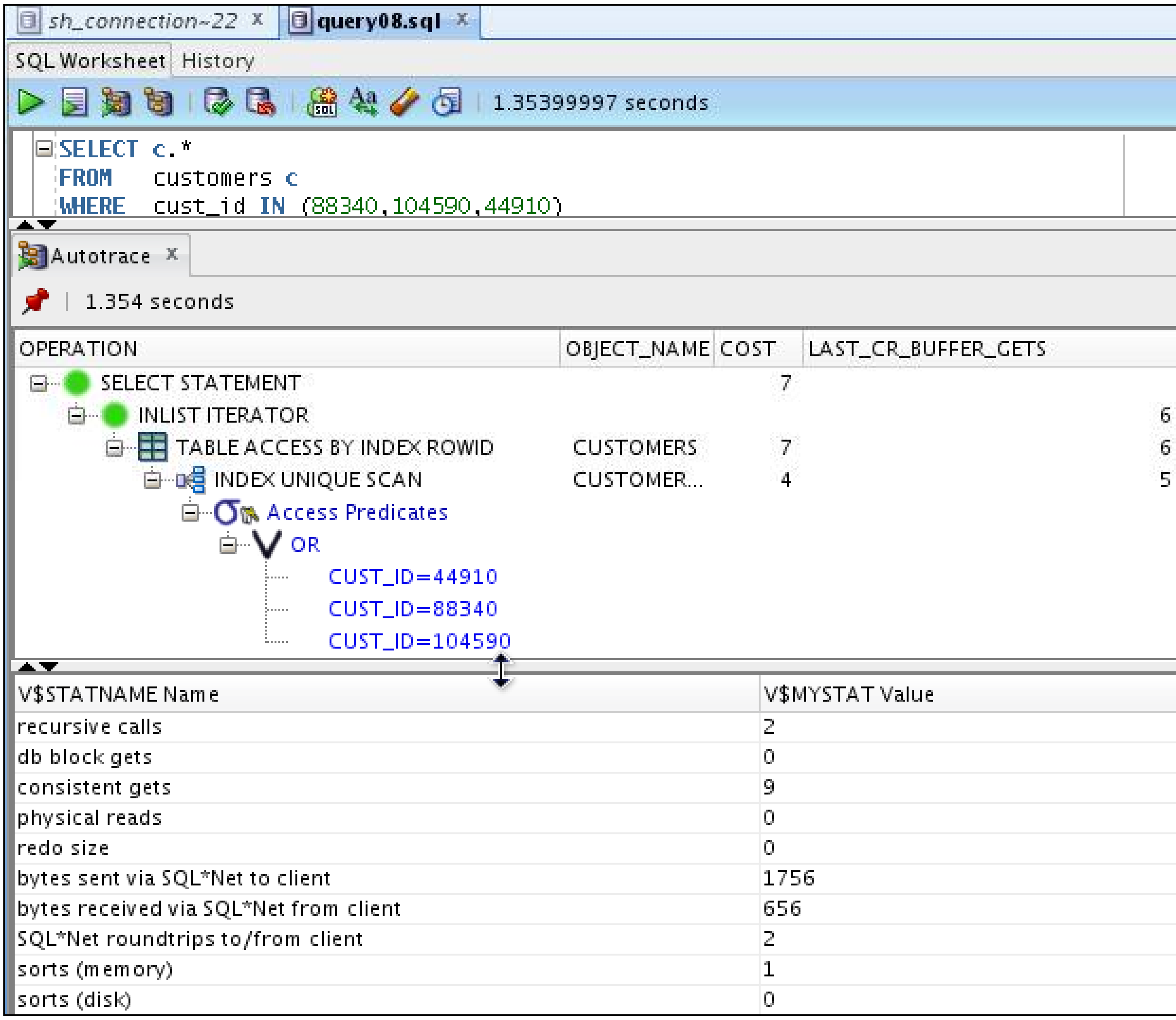
Practice 8-1: Using Other Access Paths

In this practice, you explore various access paths 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 lab can be found in your `$HOME/solutions/Access_Paths` directory.

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 Add Connection button.
   b. Create a connection with the following specifications:
      `Name: sh_connection`
      `Username: sh`
      `Password: sh`
      Select Save Password.
      `Sid: orcl`
      Click Test.
      Click Connect.
2. **Case 1: Inlist Iterator:** Open the `$HOME/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?



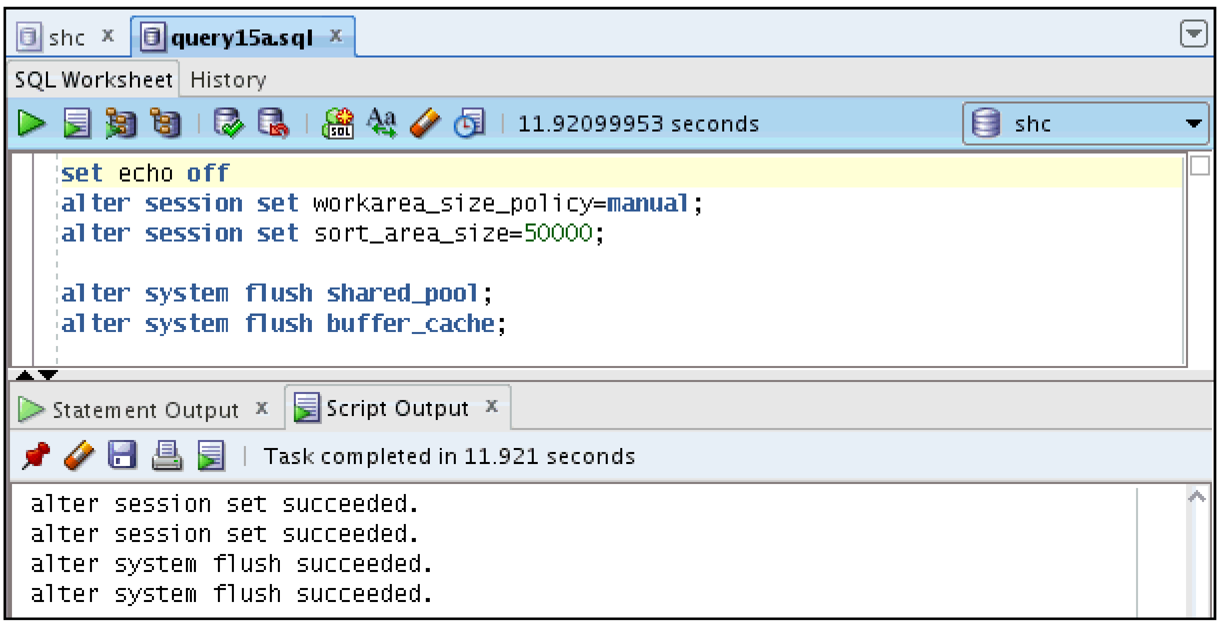
– The optimizer can use the `CUSTOMERS` primary key index to resolve this query. The cost is very low for the resulting plan.

3. **Case 2: Using Hash Clusters:** The `SHC` schema was created and populated in the practice setup using `$HOME/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 here:

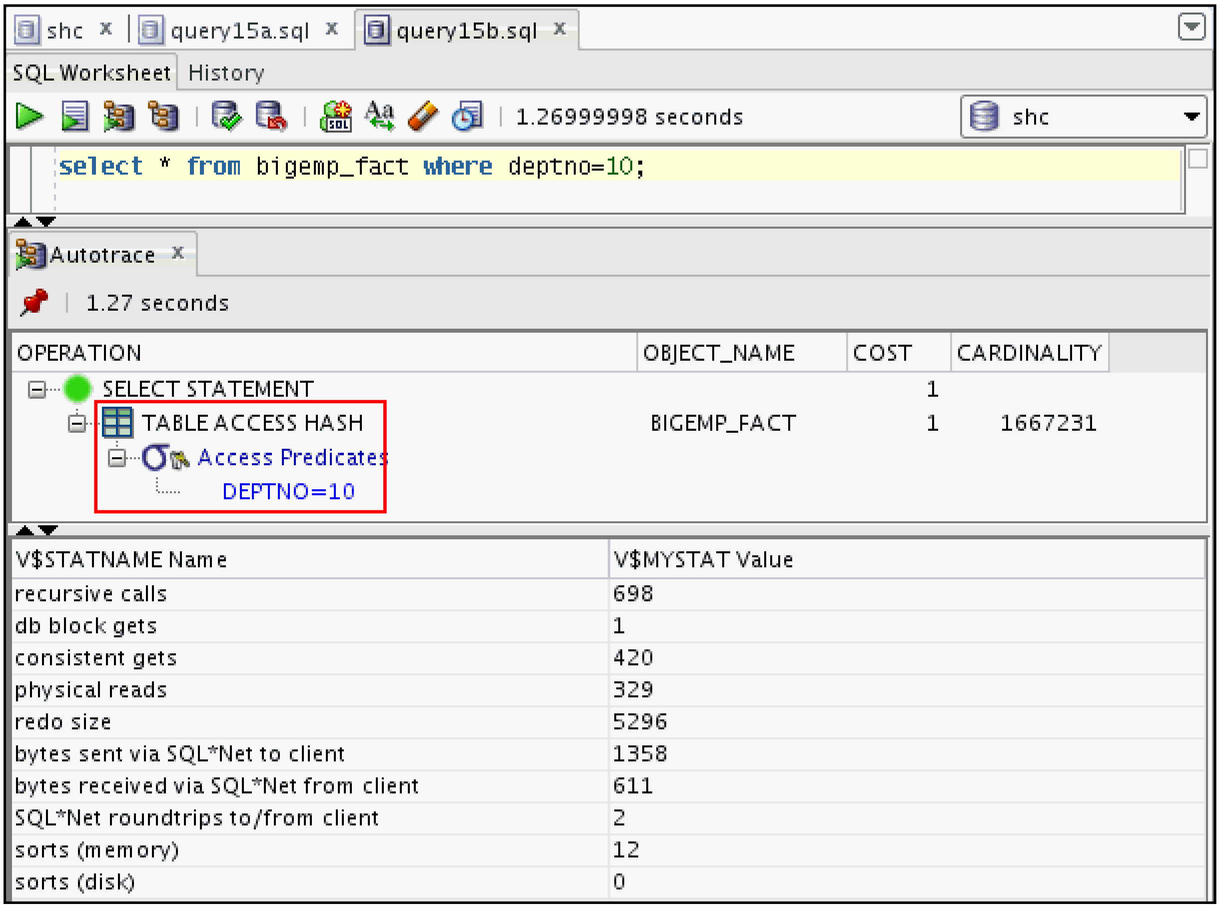
```
-- run with sqlplus /nolog @shc_setup.sql
connect / as sysdba

drop user shc cascade;

create user shc identified by shc;
Grant DBA to SHC;
GRANT select_catalog_role to SHC;
GRANT select any dictionary to SHC;
```

```
connect shc/shc

set echo on

set linesize 200

drop cluster bigemp_cluster including tables;

CREATE CLUSTER bigemp_cluster
(deptno number, sal number sort)
HASHKEYS 10000
single table HASH IS deptno SIZE 50
tablespace users;

create table bigemp_fact (
empno number primary key, sal number sort, job varchar2(12) not
null,
deptno number not null, hiredate date not null)
CLUSTER bigemp_cluster (deptno, sal);


begin
for i in 1..1400000 loop
 insert into bigemp_fact values(i,i,'J1',10,sysdate);
end loop;
commit;
end;
/

begin
for i in 1..1400000 loop
 insert into bigemp_fact values(1400000+i,i,'J1',20,sysdate);
end loop;
commit;
end;
/


exec dbms_stats.gather_schema_stats('SH');

exit
```

4. Create a connection for the `shc` user. Use this connection for all the scripts in the Hash Cluster case.
   a. Click the Add Connection button.
   b. Create a connection with the following specifications:
      `Name: shc`
      `Username: shc`
      `Password: shc`
      Check the `Remember Password`
      `Sid: orcl`
      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.

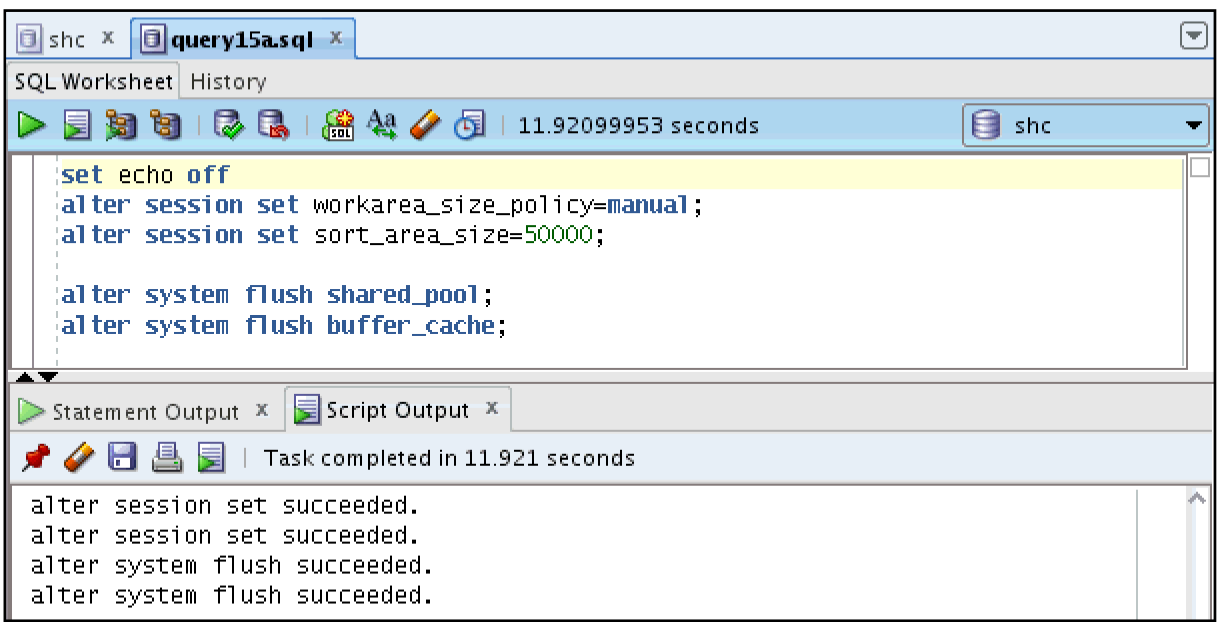


6. Use the `shc` connection to Autotrace the query in the `query15b.sql` script. What do you observe?

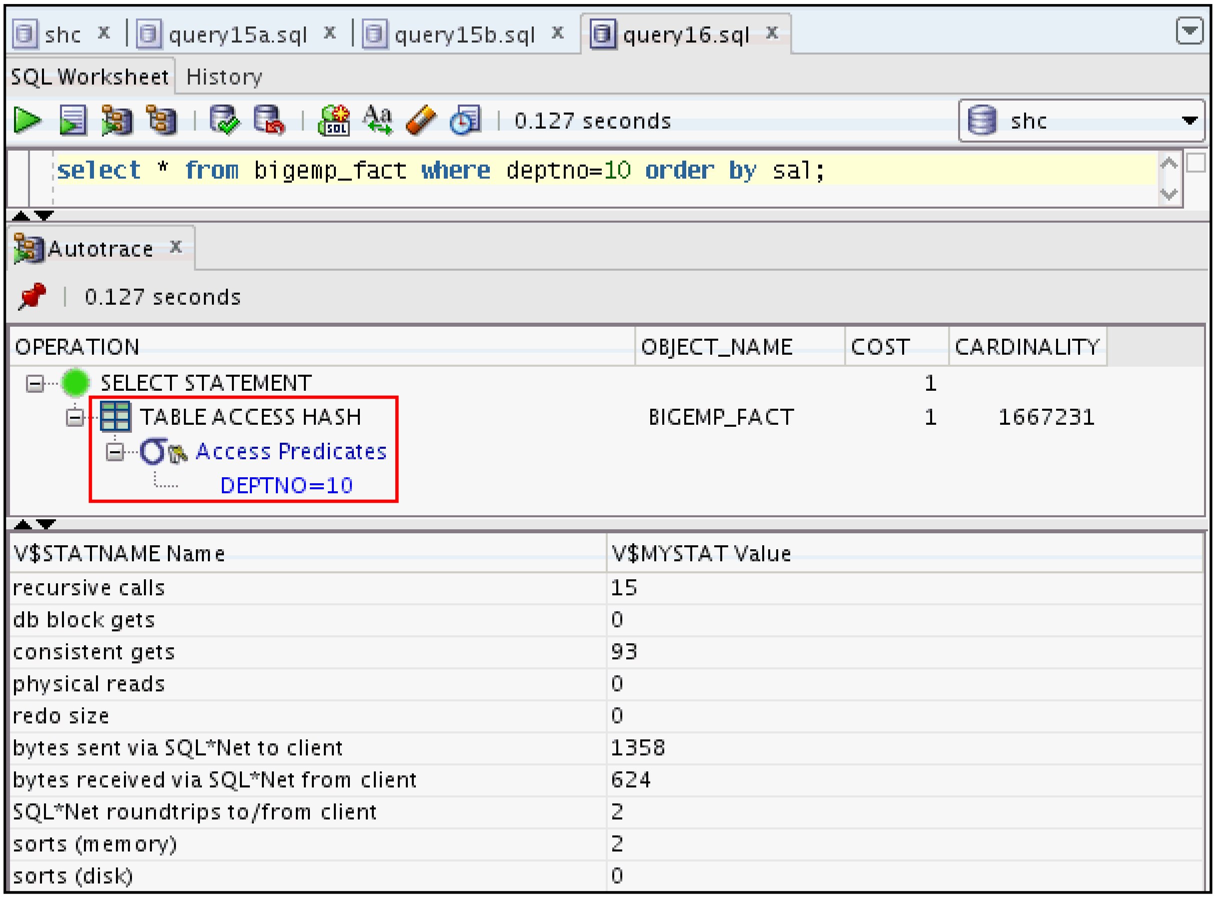


– The optimizer decides to use the cluster access path to retrieve the data. The cost is minimal.

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

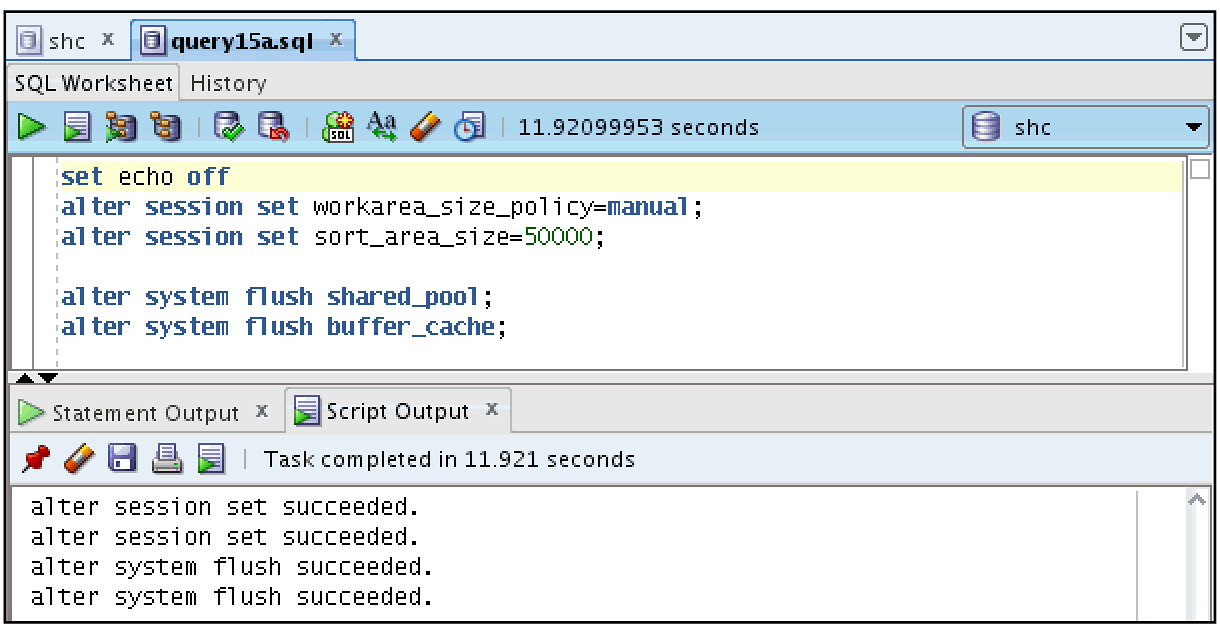


8. Open and Autotrace the query in the `query16.sql` script as the `SHC` user. What do you observe?

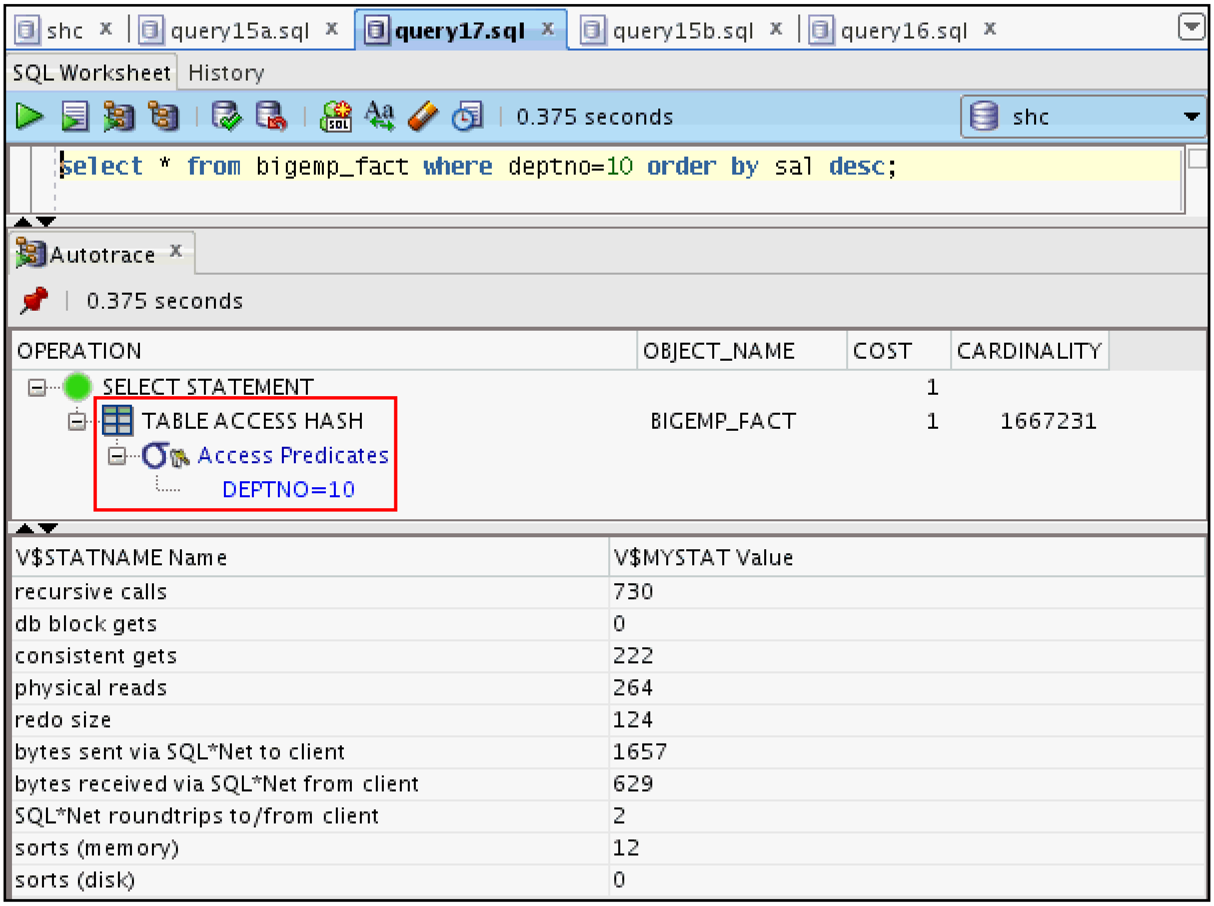


– The script executes a slightly different query, one that requires ordering the result based on the sorted `sal` column. The optimizer can still use the cluster access path without sorting the data. The cost is still minimal.

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

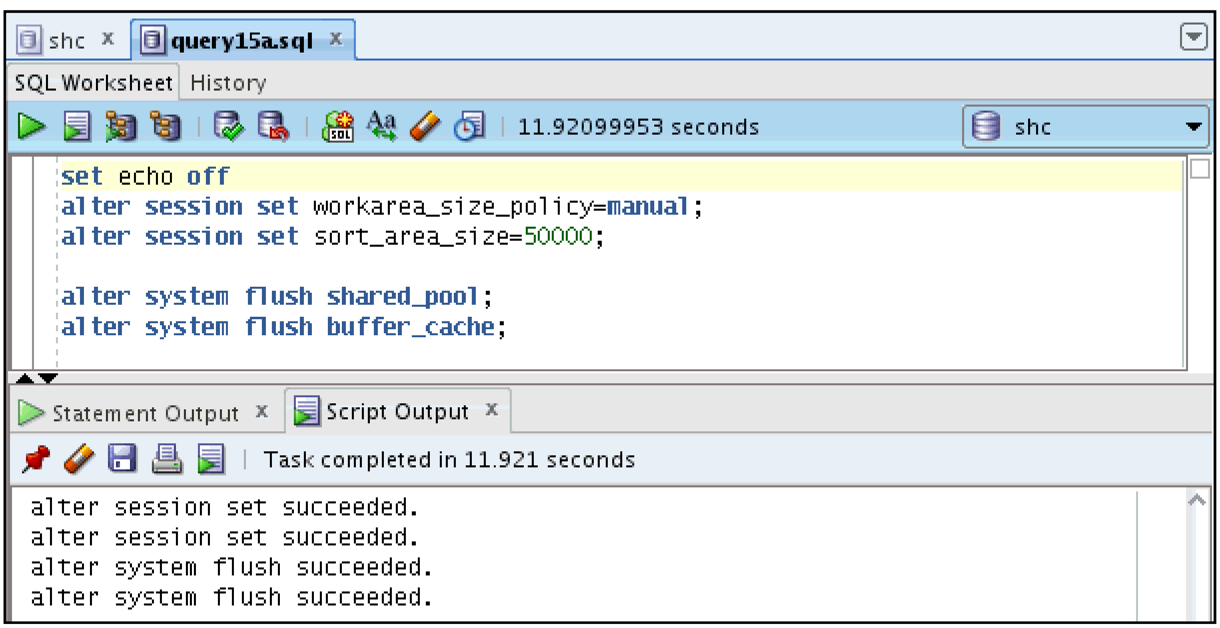


10. Autotrace the query in the `query17.sql` script. What do you observe?

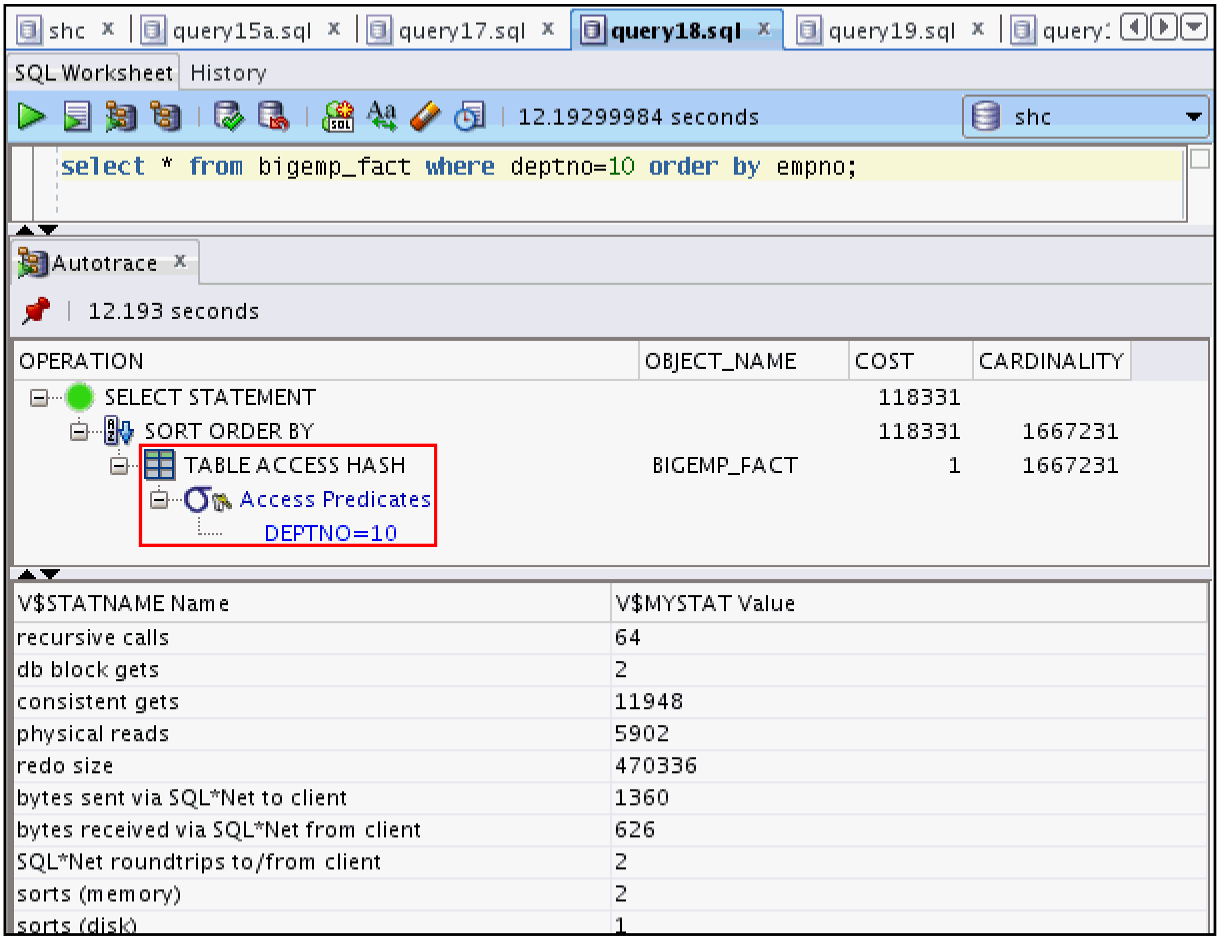


– The `query17.sql` script executes the query, the difference is the result is ordered based on the sorted `sal` column in the descending order. The optimizer can still use the cluster access path without sorting the data. The cost is still minimal.

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

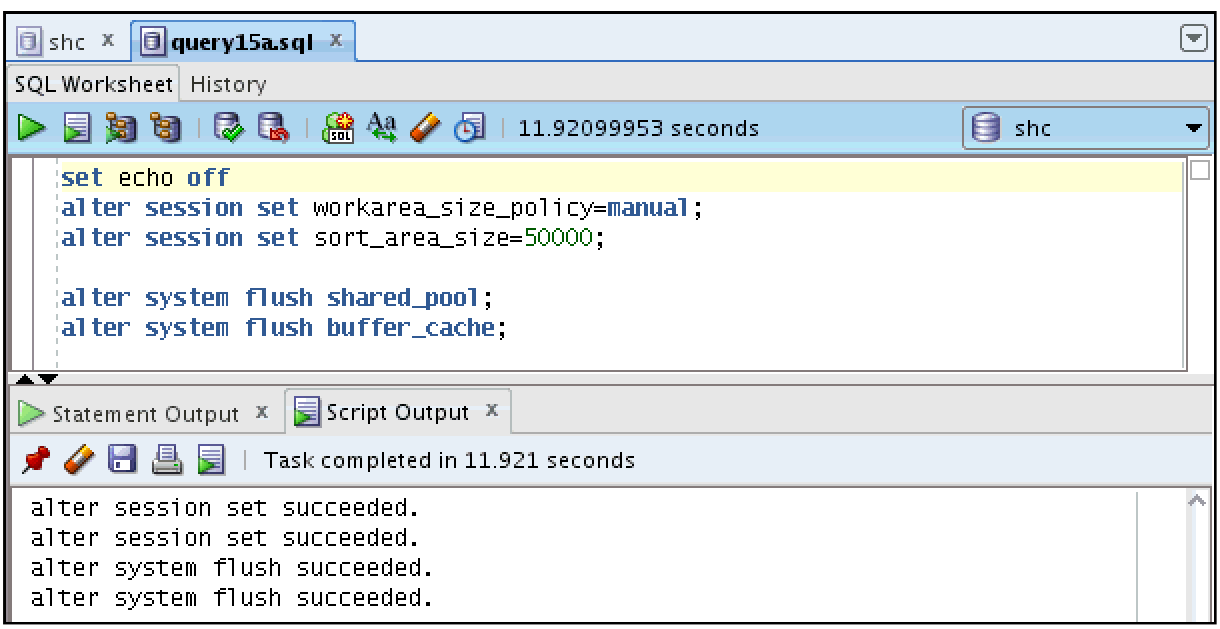


12. Open and Autotrace the query in the `query18.sql` script as the `SHC` user. What do you observe?

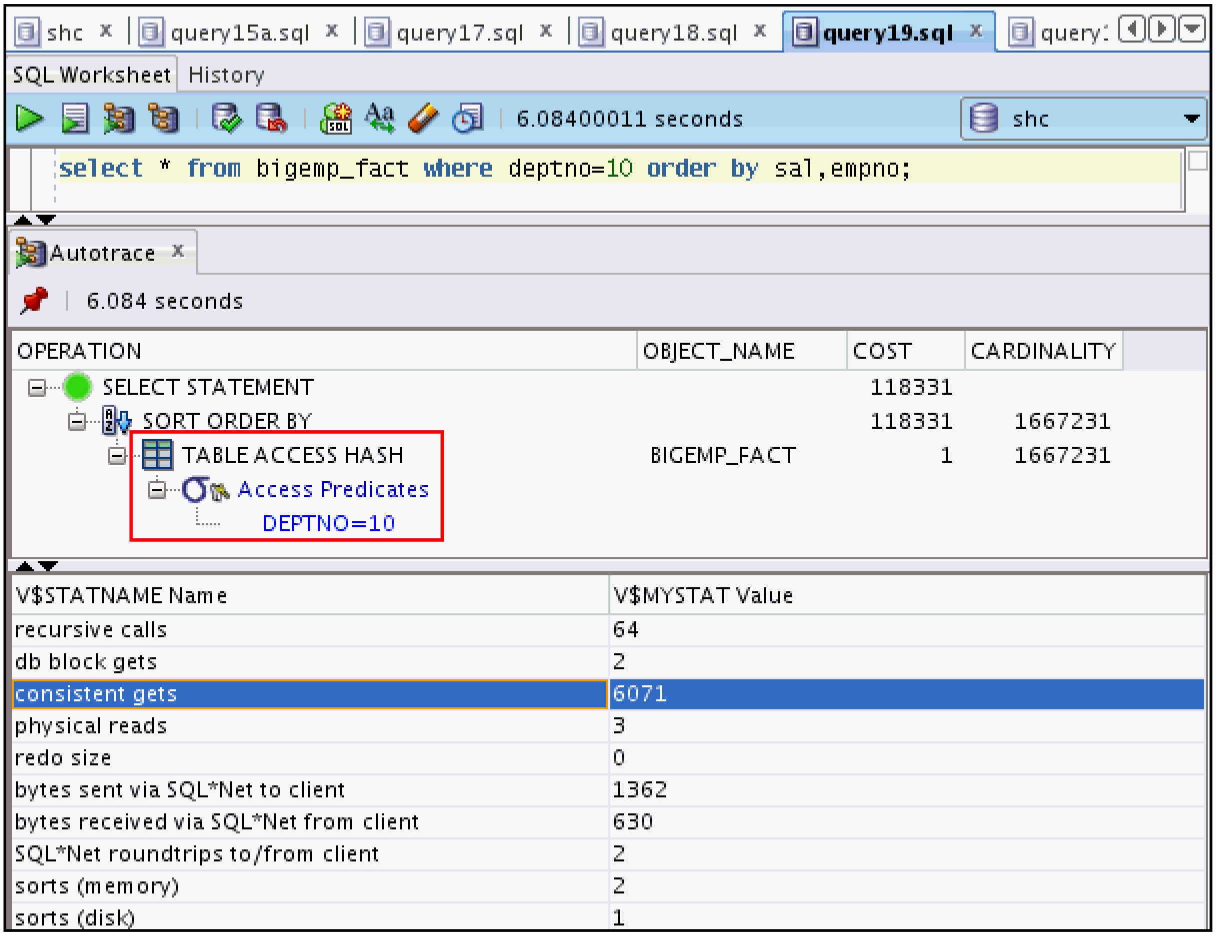


– The script executes the same query, but this time asks to order the result based on the nonsorted `empno` column. The optimizer can still make use of the cluster access path, but must sort the data making the cost of the query higher.

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.

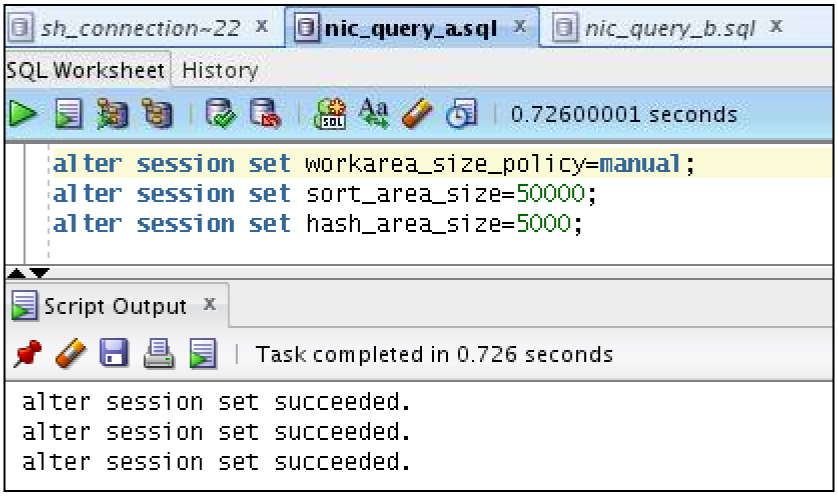
15. **Case 3 Using Index Cluster:** The `nic_setup.sql` script was executed as part of the setup for this practice. This script creates and populates the `nic` schema to set up your environment for this case. This script creates large `emp` and `dept` tables. Each of these tables has an index on the `deptno` value. The `nic_setup.sql` script is listed here:

```
-- run with sqlplus /nolog
connect / as sysdba

DROP user NIC cascade;

create user nic identified by nic;
GRANT DBA to NIC;
GRANT SELECT_CATALOG_ROLE to NIC;
GRANT select any dictionary to NIC;

connect nic/nic

set echo on
```

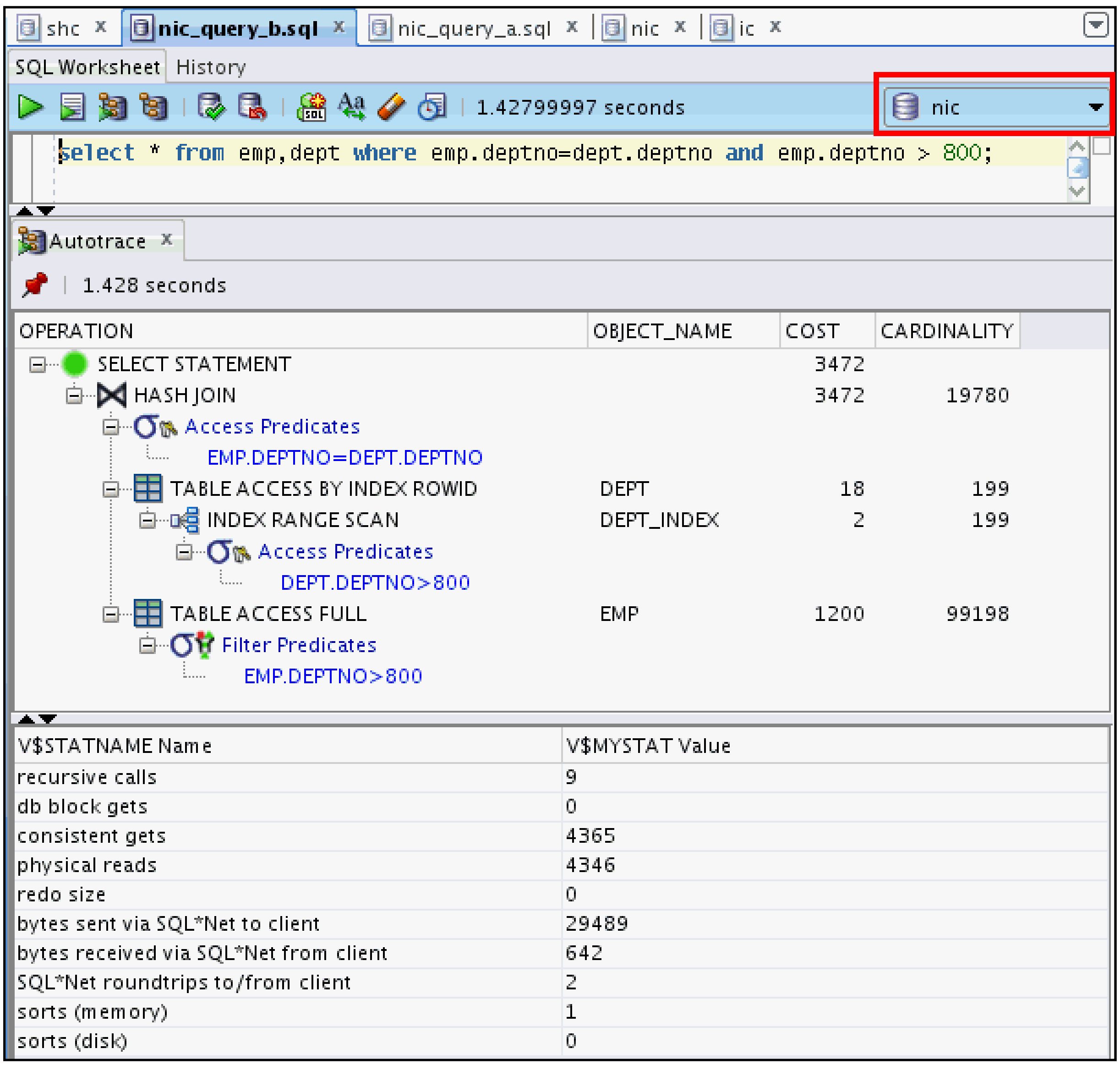
```
drop cluster emp_dept including tables;

drop table emp purge;
drop table dept purge;


CREATE TABLE emp (
   empno     NUMBER(7)            ,
   ename     VARCHAR2(15) NOT NULL,
   job       VARCHAR2(9)          ,
   mgr       NUMBER(7)            ,
   hiredate  DATE                 ,
   sal       NUMBER(7)          ,
   comm      NUMBER(7)          ,
   deptno    NUMBER(3)
);

CREATE TABLE dept (
   deptno NUMBER(3) ,
   dname  VARCHAR2(14),
   loc    VARCHAR2(14),
   c      VARCHAR2(500)
);

CREATE INDEX emp_index
   ON emp(deptno)
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);

CREATE INDEX dept_index
   ON dept(deptno)
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);


begin
 for i in 1..999 loop
```

```
  insert into dept values
(i,'D'||i,'L'||i,dbms_random.string('u',500));
 end loop;
 commit;
end;
/



begin
 for i in 1..500000 loop
  insert into emp values
(i,dbms_random.string('u',15),dbms_random.string('u',9),i,sysdate,i,i,
mod(i,999));
 end loop;
 commit;
end;
/

exec dbms_stats.gather_schema_stats('SH');

exit;
```

16. Create a connection for the `nic` user.
    a. Click the Add Connection button.
    b. Create a connection with the following specifications:
       `Name: nic`
       `Username: nic`
       `Password: nic`
       Select Save Password.
       `Sid: orcl`
       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.

a. The ic user was created in the setup of the practices with the ic_setup.sql script to create an index cluster to store the two tables. These tables have exactly the same rows in both the nic and ic schemas. This script creates an index cluster containing the emp and dept tables. The rows of these tables are stored together clustered by the deptno value. This script is listed here:

```
-- run with sqlplus /nolog

connect / as sysdba

DROP user IC cascade;

CREATE USER ic IDENTIFIED BY ic;

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

connect ic/ic

set echo on

drop table emp purge;
drop table dept purge;

drop cluster emp_dept including tables;

CREATE CLUSTER emp_dept (deptno NUMBER(3))
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      PCTINCREASE 33);

CREATE TABLE emp (
   empno     NUMBER(7)            ,
   ename     VARCHAR2(15) NOT NULL,
   job       VARCHAR2(9)          ,
   mgr       NUMBER(7)            ,
   hiredate  DATE                 ,
   sal       NUMBER(7)          ,
   comm      NUMBER(7)          ,
   deptno    NUMBER(3))
```

```
   CLUSTER emp_dept (deptno);

CREATE TABLE dept (
   deptno NUMBER(3) ,
   dname  VARCHAR2(14),
   loc    VARCHAR2(14),
   c      VARCHAR2(500))
   CLUSTER emp_dept (deptno);

CREATE INDEX emp_dept_index
   ON CLUSTER emp_dept
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);

begin
 for i in 1..999 loop
  insert into dept values
(i,'D'||i,'L'||i,dbms_random.string('u',500));
 end loop;
 commit;
end;
/



begin
 for i in 1..500000 loop
  insert into emp values
(i,dbms_random.string('u',15),dbms_random.string('u',9),i,sysdate,i,i,
mod(i,999));
 end loop;
 commit;
end;
/

exec dbms_stats.gather_schema_stats('SH');

exit;
```

20. Create a connection for the `ic` user.
    a. Click the Add Connection button.
    b. Create a connection with following specifications:
       `Name: ic`
       `Username: ic`

`Password: ic`

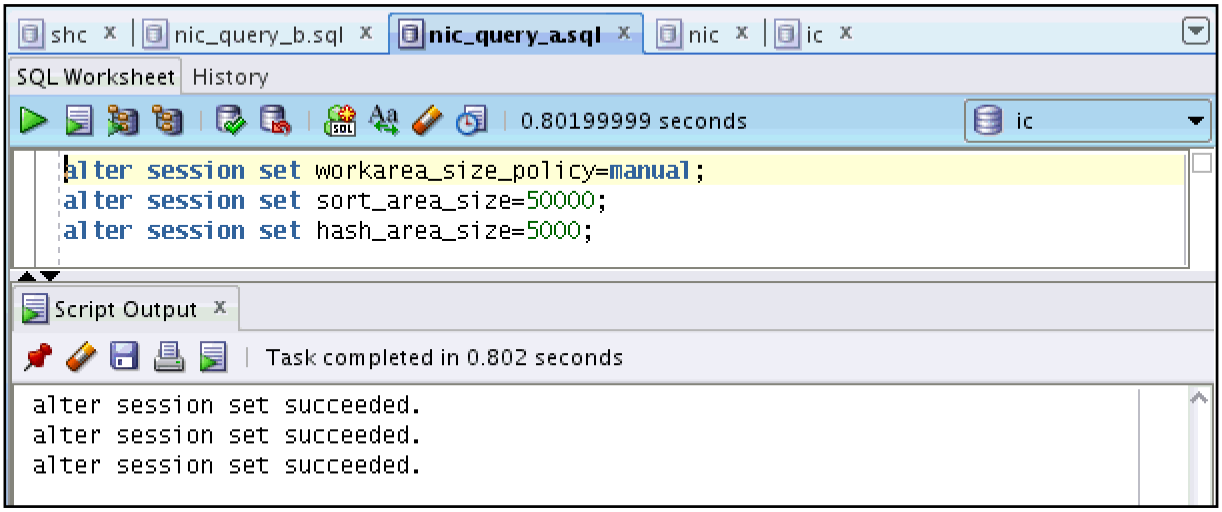
Check the `Remember Password`

`Sid: orcl`

Click Test.

Click Connect.

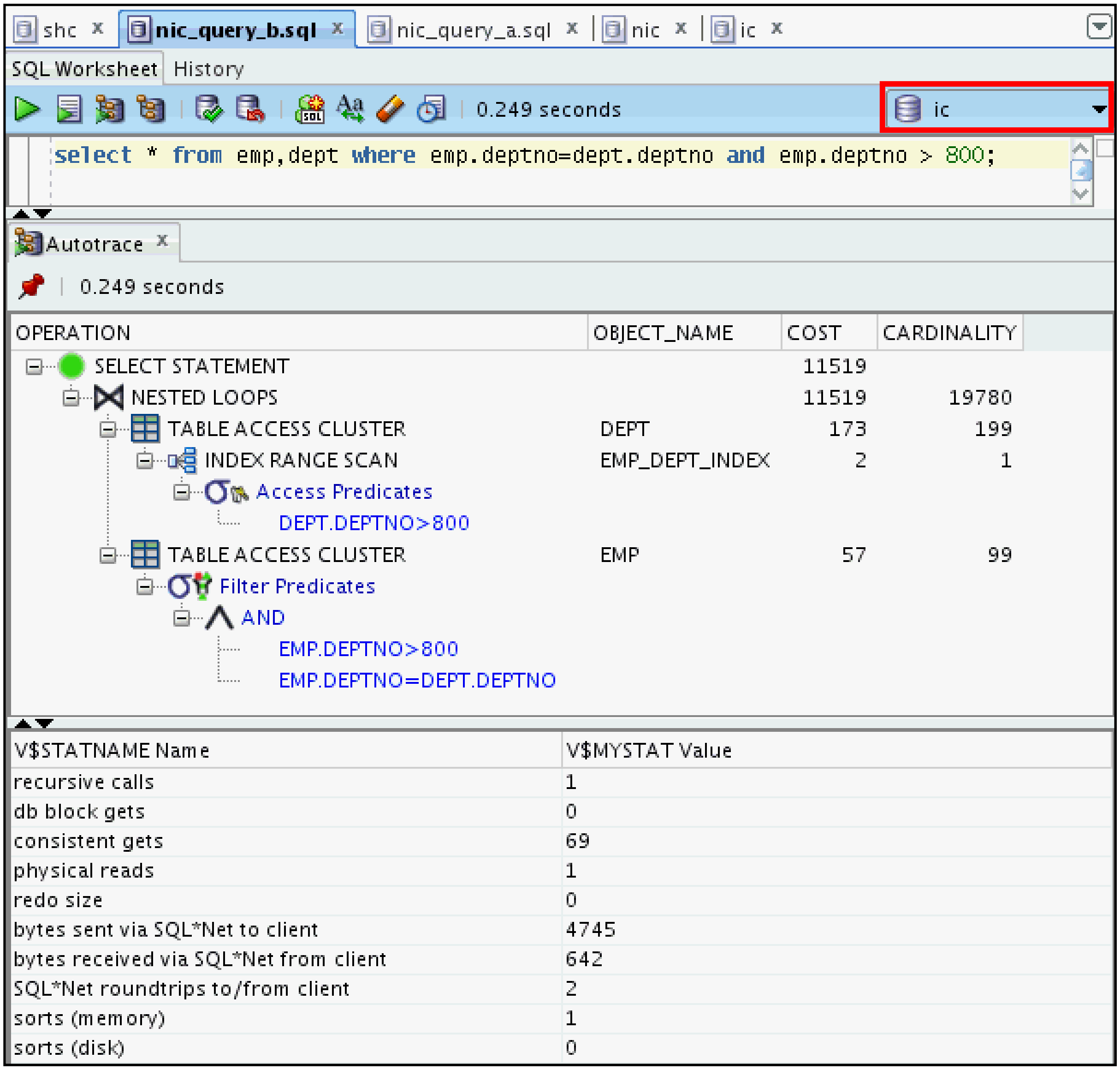
21. Open and execute the `nic_query_a.sql` script as the `ic` user.



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 connected as the `SH` user, and then execute this script again.

```
$ ./ap_cleanup.sh
…
Connected to:…


SQL>
```

```
SQL> @sh_main sh example temp oracle_4U
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_histor
y/ /home/oracle/ v3
SQL> Rem
SQL> Rem $Header: sh_main.sql 06-mar-2008.15:00:45 cbauwens Exp
$
SQL> Rem
SQL> Rem sh_main.sql
SQL> Rem
SQL> Rem Copyright (c) 2001, 2008, Oracle. All rights reserved.
SQL> Rem
SQL> Rem    NAME
SQL> Rem         sh_main.sql - Main schema creation and load
script
SQL> Rem
SQL> Rem    DESCRIPTION
SQL> Rem         SH is the Sales History schema of the Oracle
Sample
SQL> Rem        Schemas
SQL> Rem
SQL> Rem    NOTES
SQL> Rem        CAUTION: use absolute pathnames as parameters 5
and 6.
SQL> Rem        Example (UNIX) echo
$ORACLE_HOME/demo/schema/sales_history
SQL> Rem        Please make sure that parameters 5 and 6 are
specified
SQL> Rem        INCLUDING the trailing directory delimiter,
since the
SQL> Rem        directory parameters and the filenames are
concatenated
SQL> Rem        without adding any delimiters.
SQL> Rem        Run this as SYS or SYSTEM
SQL> Rem
SQL> Rem    MODIFIED   (MM/DD/YY)
SQL> Rem        cbauwens        03/06/08 - NLS settings for load
SQL> Rem        cbauwens        07/10/07 - NLS fix bug 5684394
SQL> Rem        glyon   06/28/07 - grant CWM_USER role, if it
exists
SQL> Rem        cbauwens        02/23/05 - 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:


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 …
$
```

Practice 8-2: Using the Result Cache

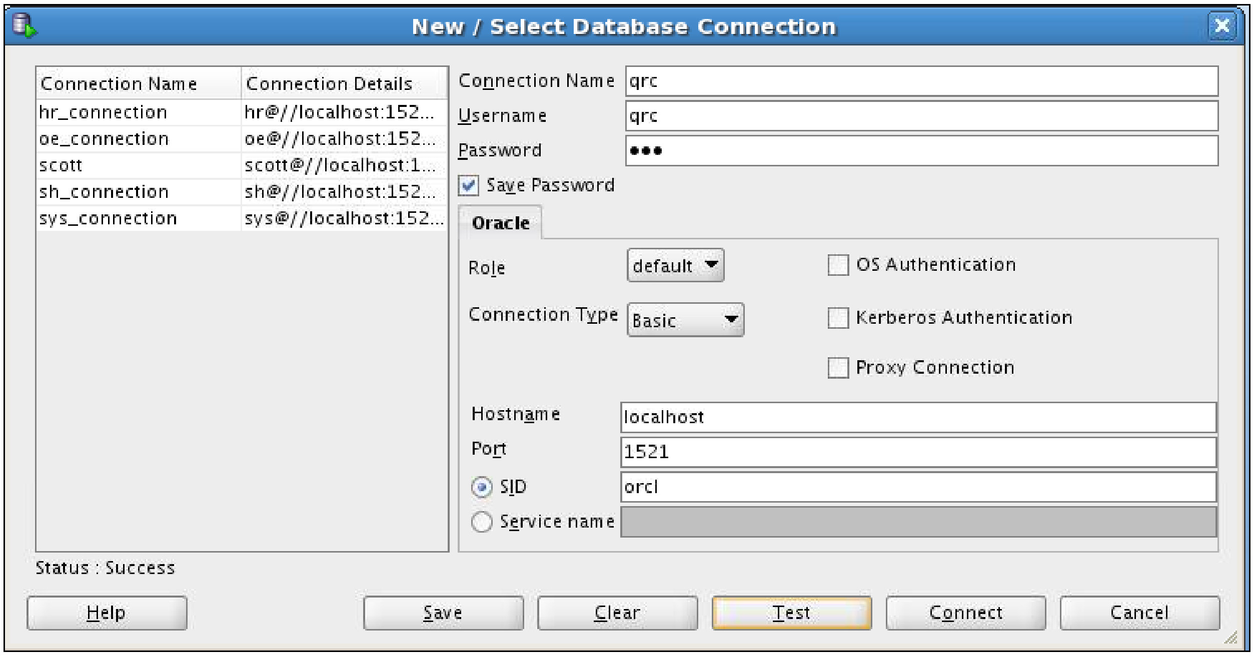
In this practice, you explore the various possibilities of caching query results in the System Global Area (SGA). Perform the following steps to understand the use of Query Result Cache. All the scripts for this practice can be found in the `$HOME/solutions/Query_Result_Cache/` directory.

1. The `result_cache_setup.sh` script was executed in the setup of this class as the `SYS` user to create and populate the QRC schema. This script is listed here:

```
#!/bin/bash

cd /home/oracle/solutions/Query_Result_Cache

sqlplus / as sysdba <<FIN!

set echo on

drop user qrc cascade;

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

grant connect, resource, dba to qrc;

connect qrc/qrc

exec dbms_result_cache.flush;

drop table cachejfv purge;

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

insert into cachejfv
values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
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 values('b');

commit;

alter system flush buffer_cache;

FIN!

$
```

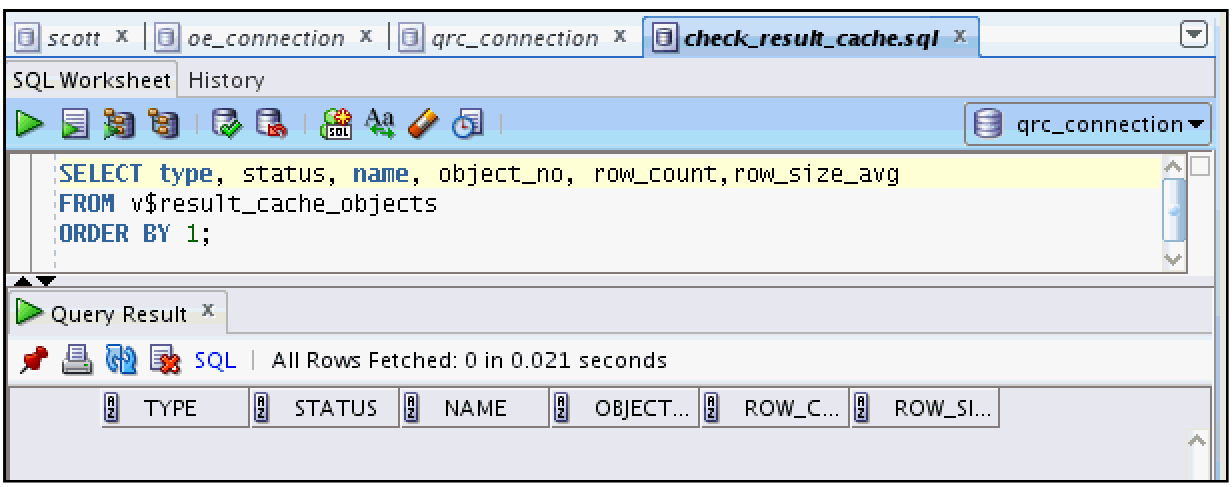
2. Create a connection for the `qrc` user.
   a. Click the Add Connection button.
   b. Create a connection with the following specifications:
      `Name: qrc`
      `Username: qrc`
      `Password: qrc`
      Select Save Password.
      `Sid: orcl`
      Click Test.
      Click Connect.



3. As the qrc user, determine the current content of the query cache using the following statement or open and execute the statement in the `check_result_cache.sql` file in the `$HOME/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?

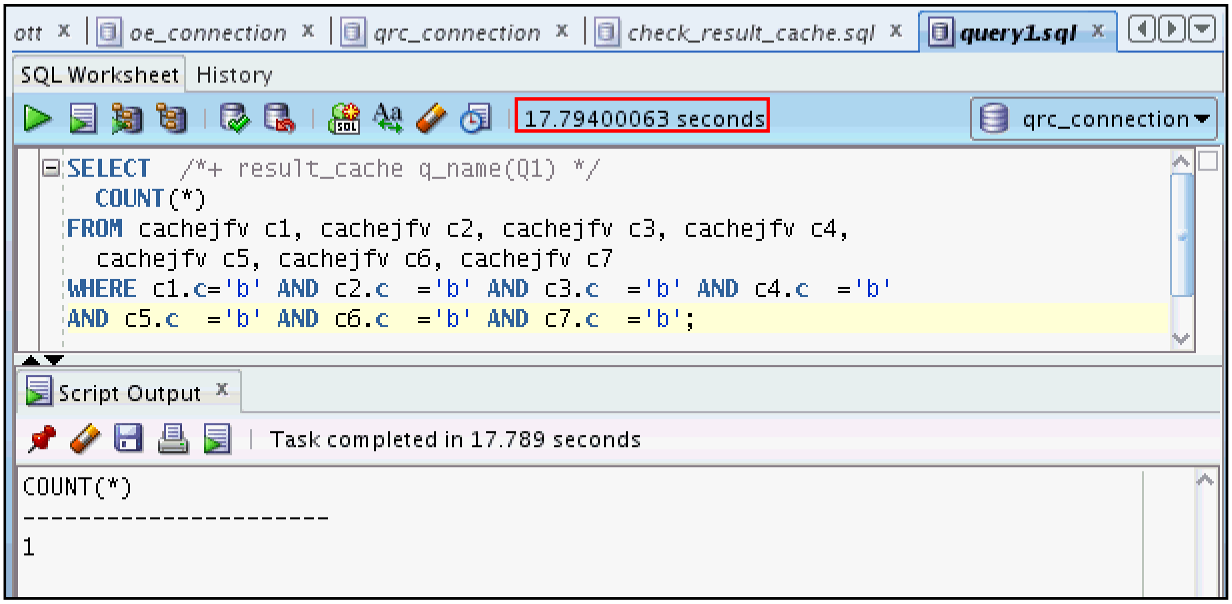


– No row is selected because the query cache is empty.

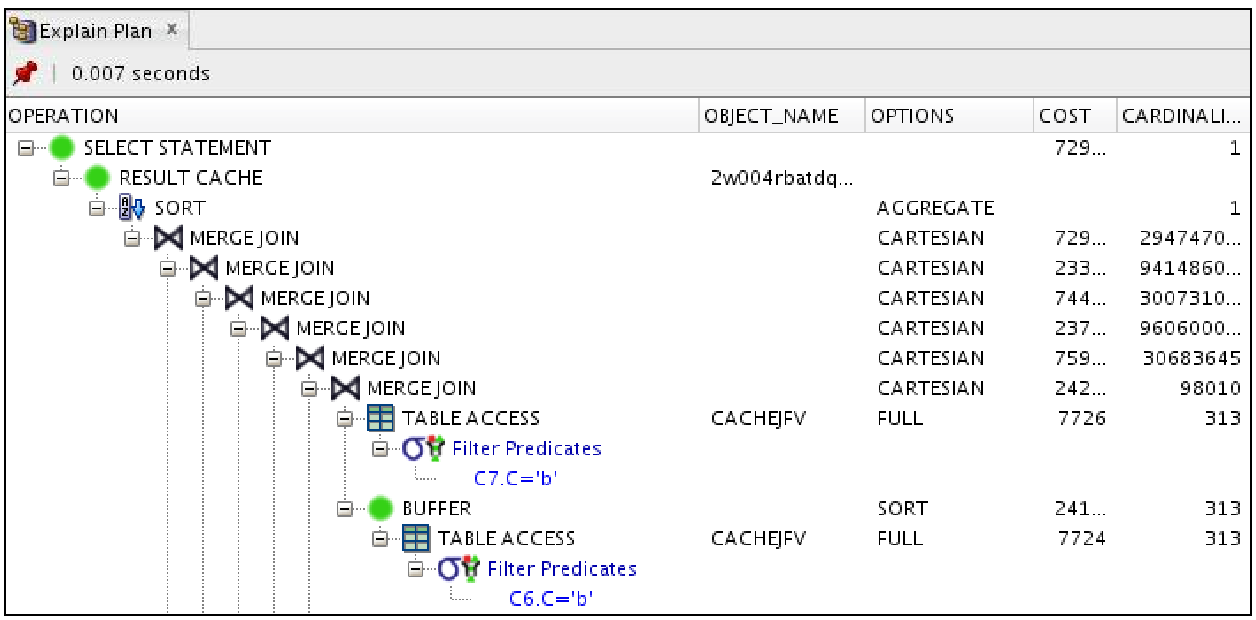
4. Open and execute the `query1.sql` script. Note the time that it takes for this statement to execute.

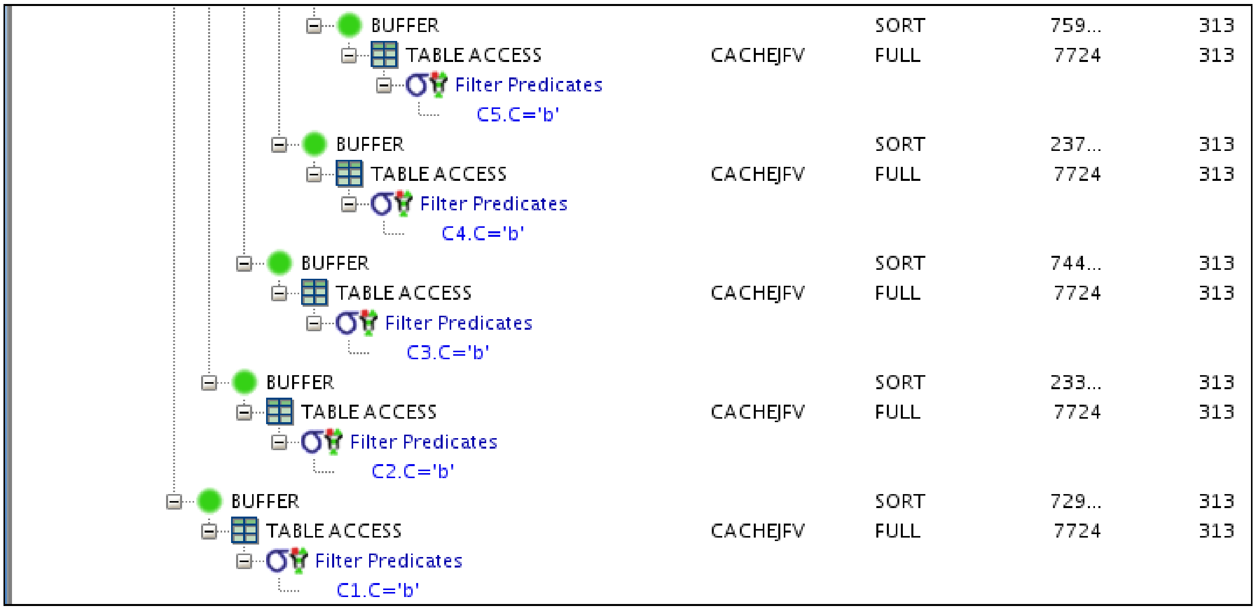
```
SELECT  /*+ result_cache q_name(Q1) */
  COUNT(*)
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,
  cachejfv c5, cachejfv c6, cachejfv c7
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'
```

```
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```



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

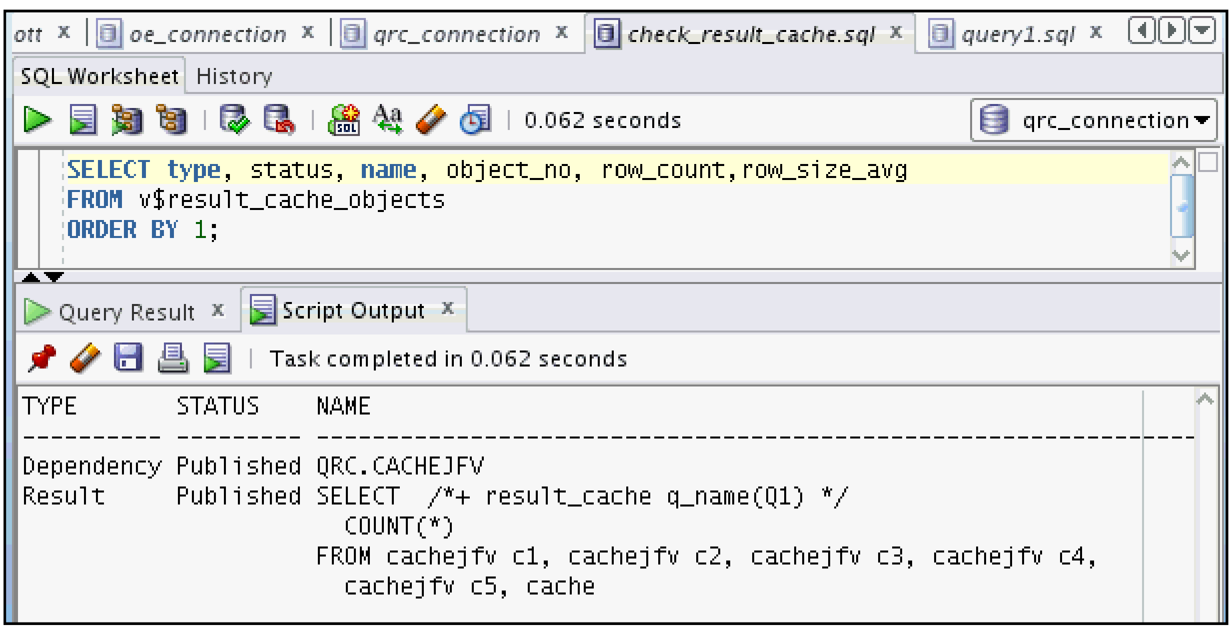




– Because of the `result_cache` hint, the result of the query is computed using the result cache.

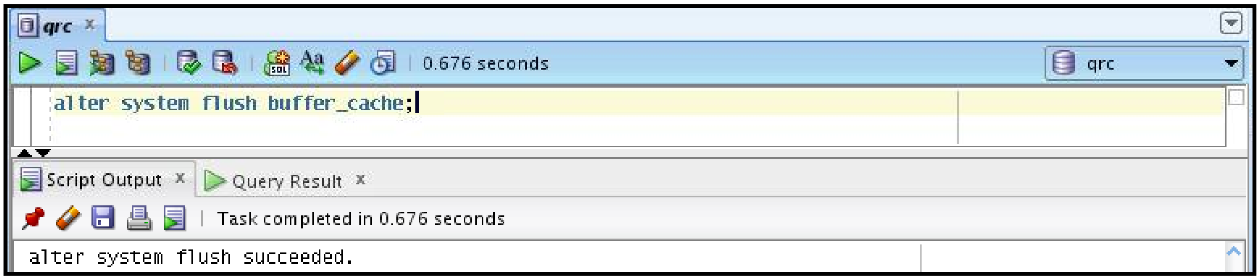
6. As the `qrc` user, determine the current content of the query cache using the following statement or the `check_result_cache.sql` script. What do you observe?

```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects order by 1;
```

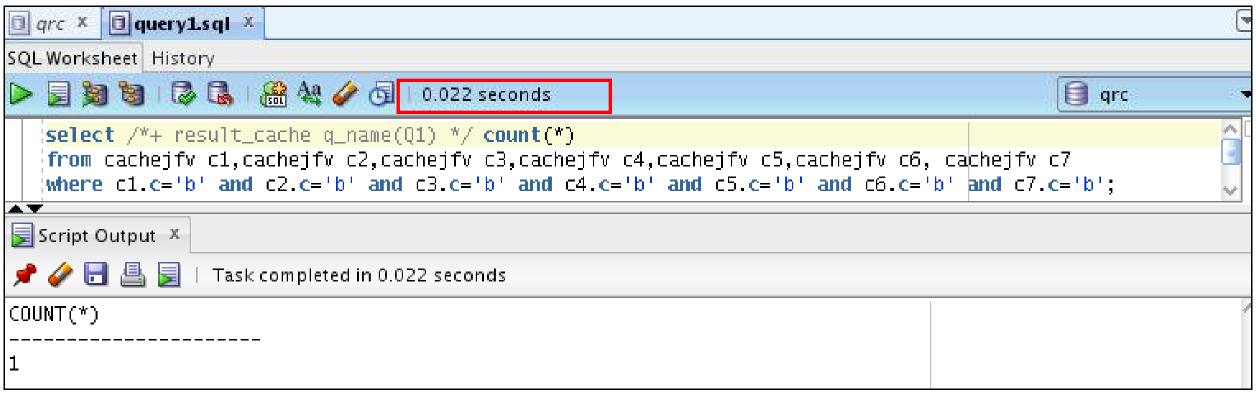


– You can now see that the result of your query is cached.

7. Flush the buffer cache of your instance.



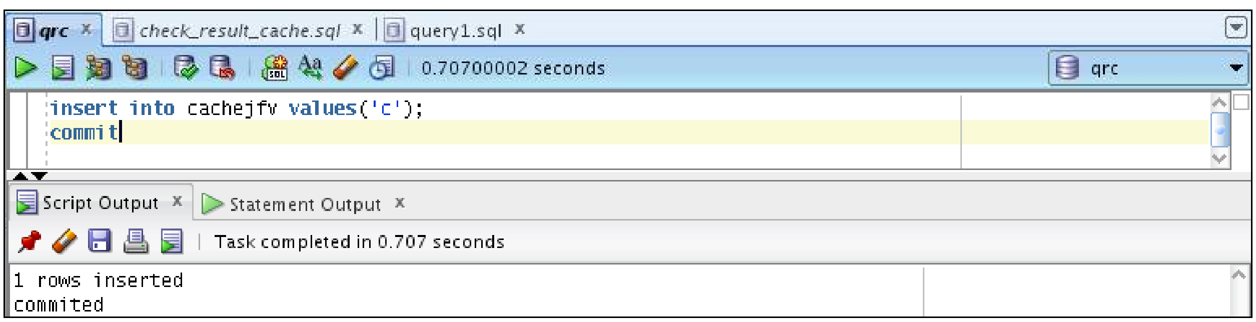
8. Rerun `query1.sql`. What do you observe?



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

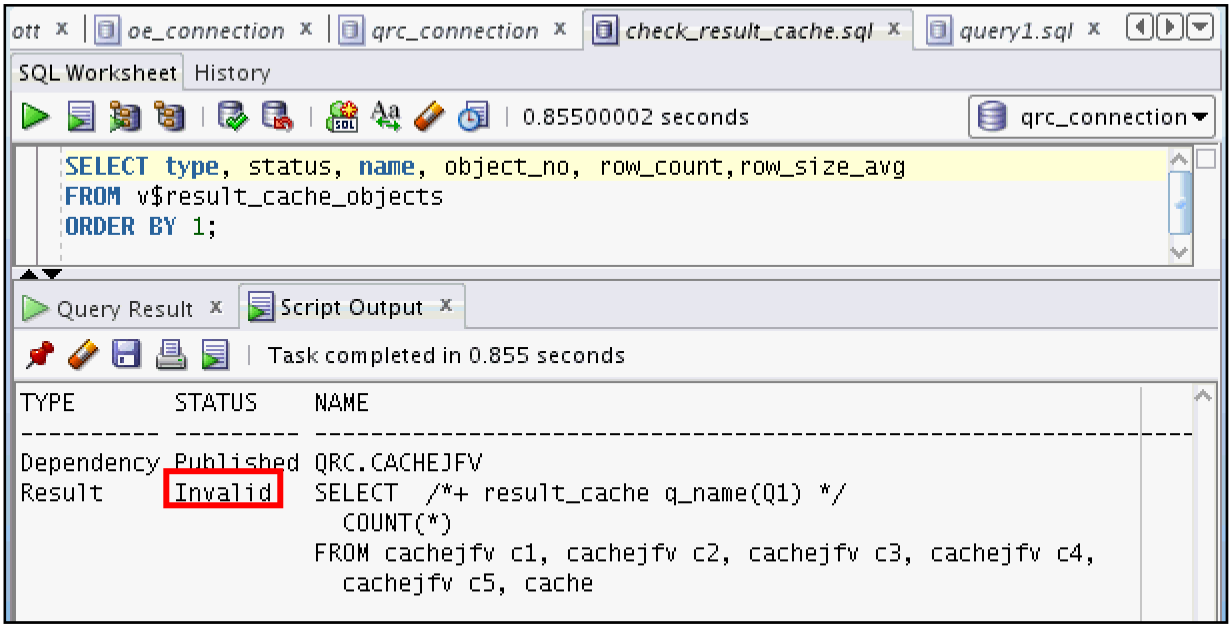
9. Insert a new row into the `CACHEJFV` table using the following statement:

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



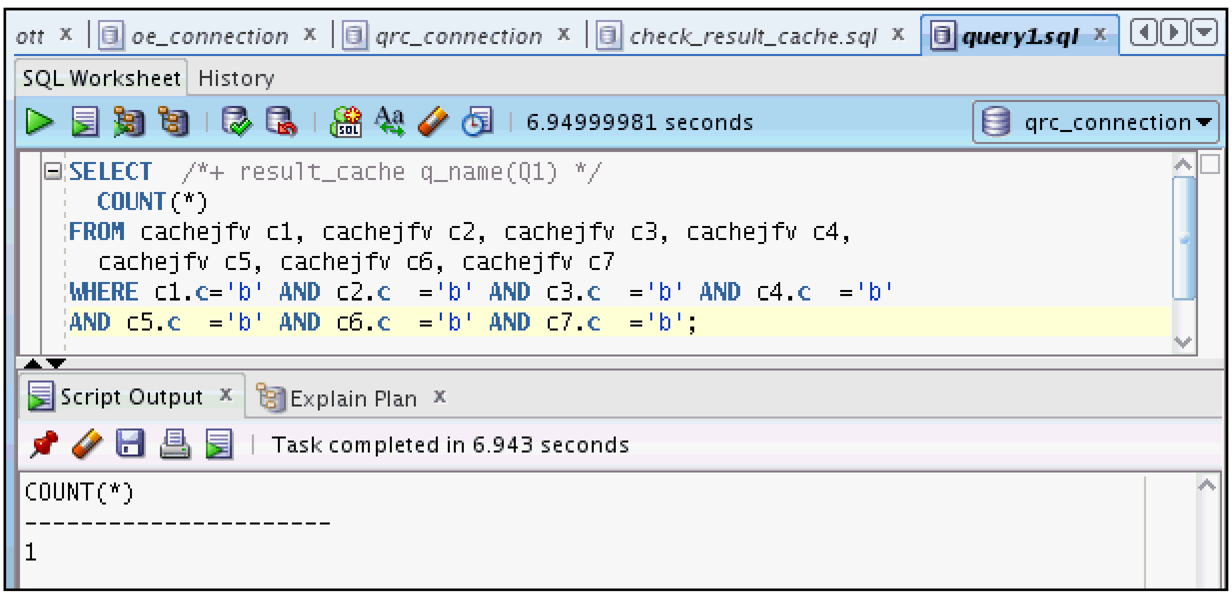
10. As the `qrc` user, determine the current content of the query cache using the following statement or the `check_result_cache.sql` script. What do you observe?

```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects order by 1;
```



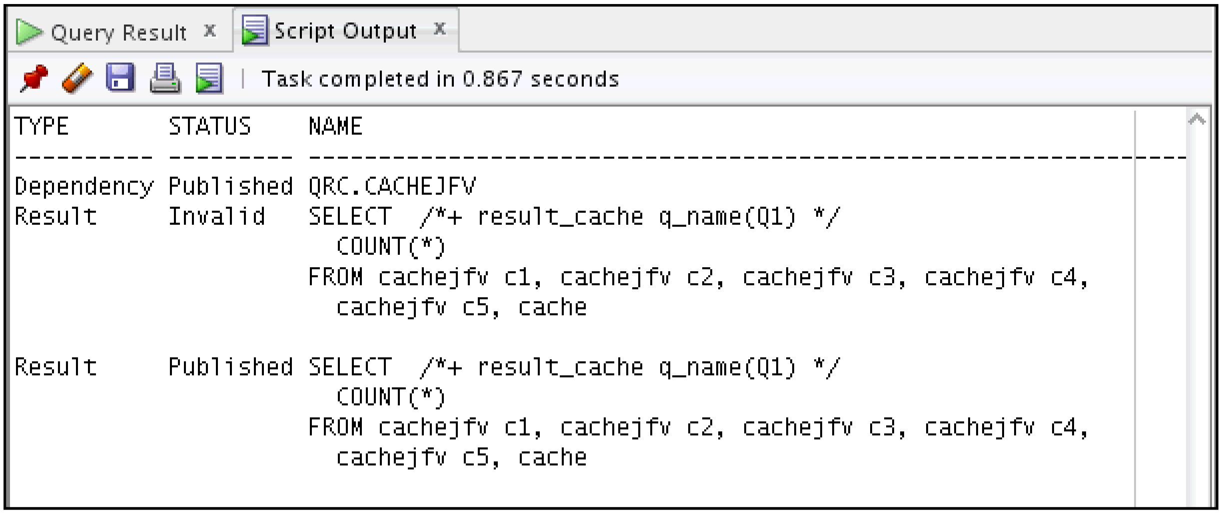
– The corresponding result cache entry is automatically invalidated.

11. Rerun `query1.sql`.



12. As the `qrc` user, determine the current content of the query cache using the following statement or execute the `check_result_cache.sql` script. What do you observe?

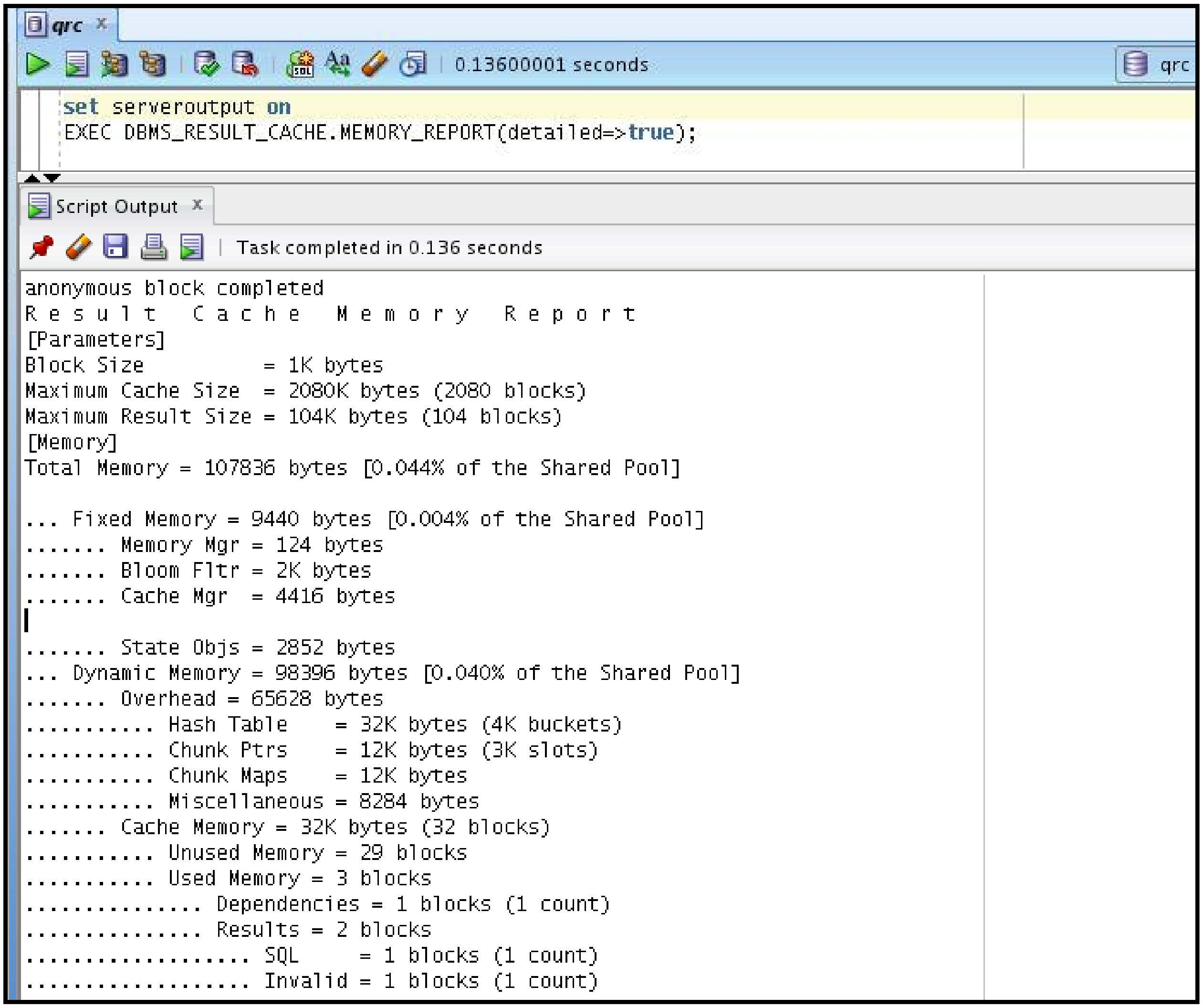
```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects order by 1;
```



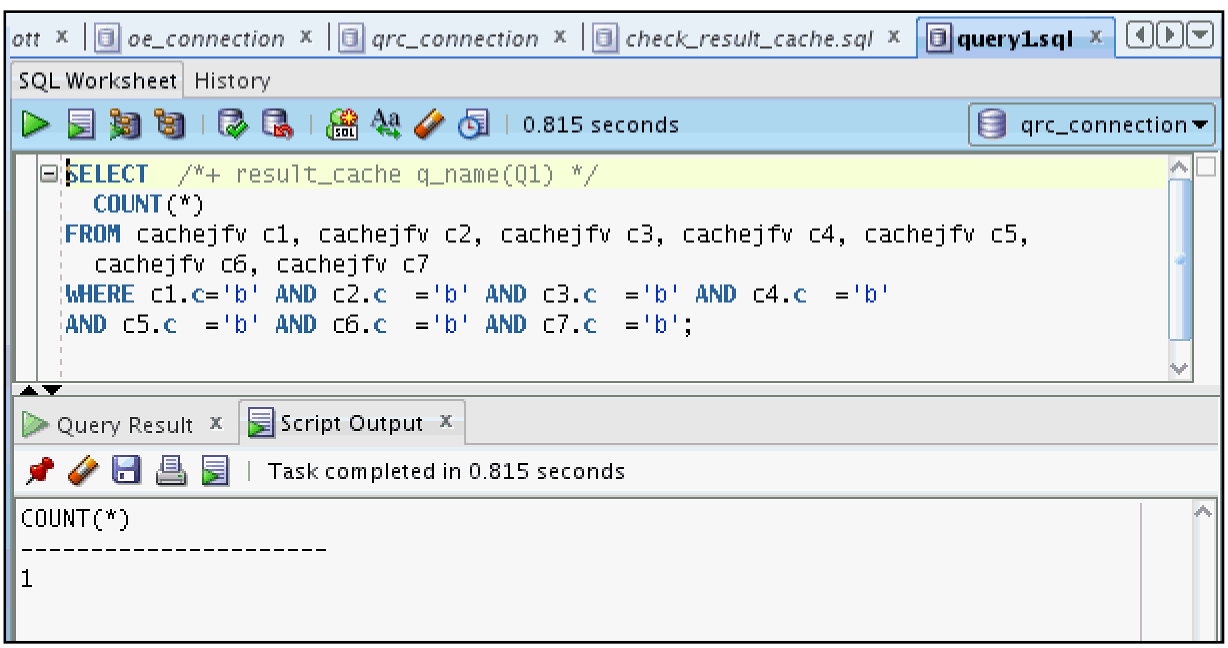
– Again, it takes some time to execute the query. The result cache shows that a new entry has been added for the new result.

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

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



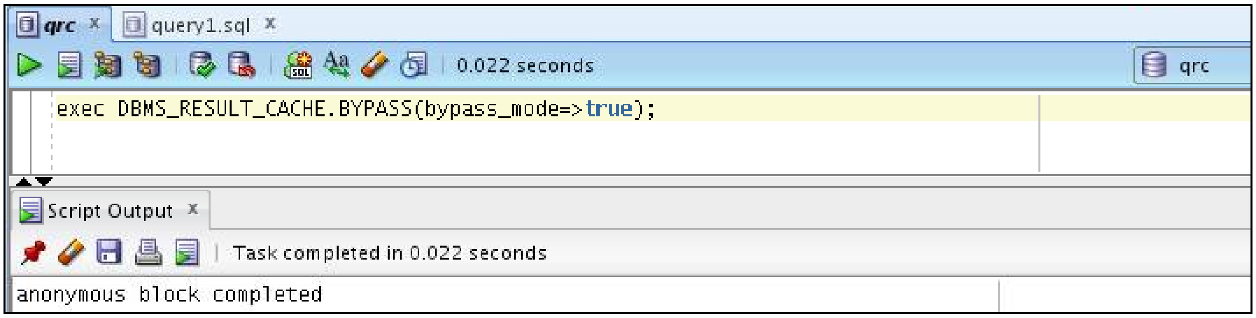
14. Rerun the `query1.sql` script. What do you observe?



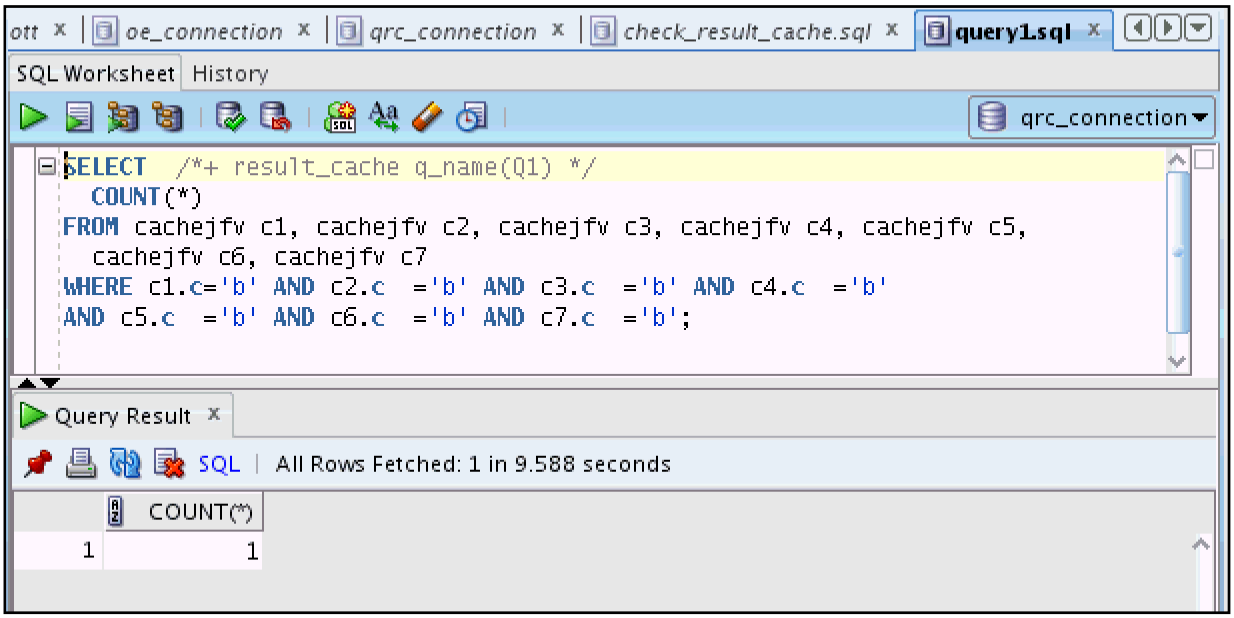
- The query again uses the result that was previously cached.

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

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



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

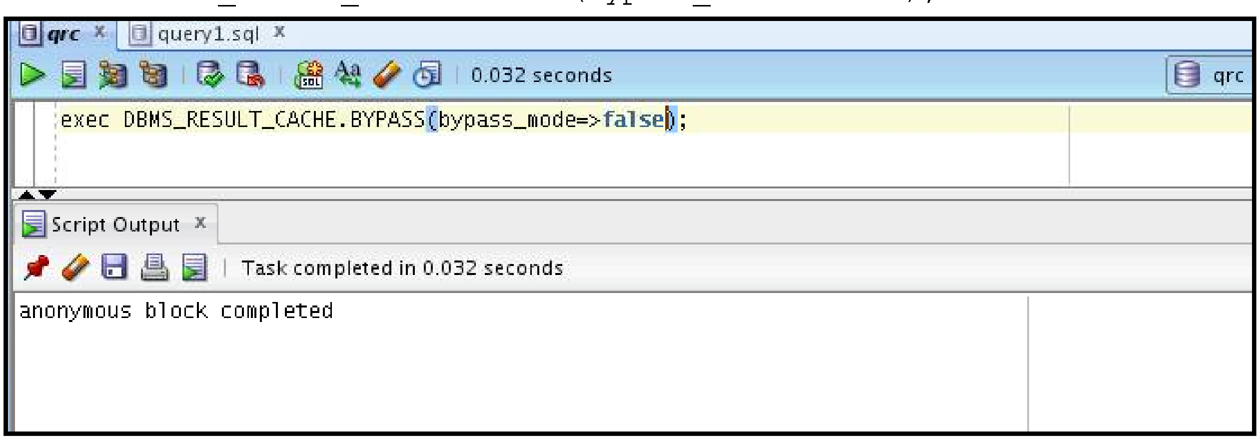


- The query again takes longer to execute because it no longer uses the result cache.

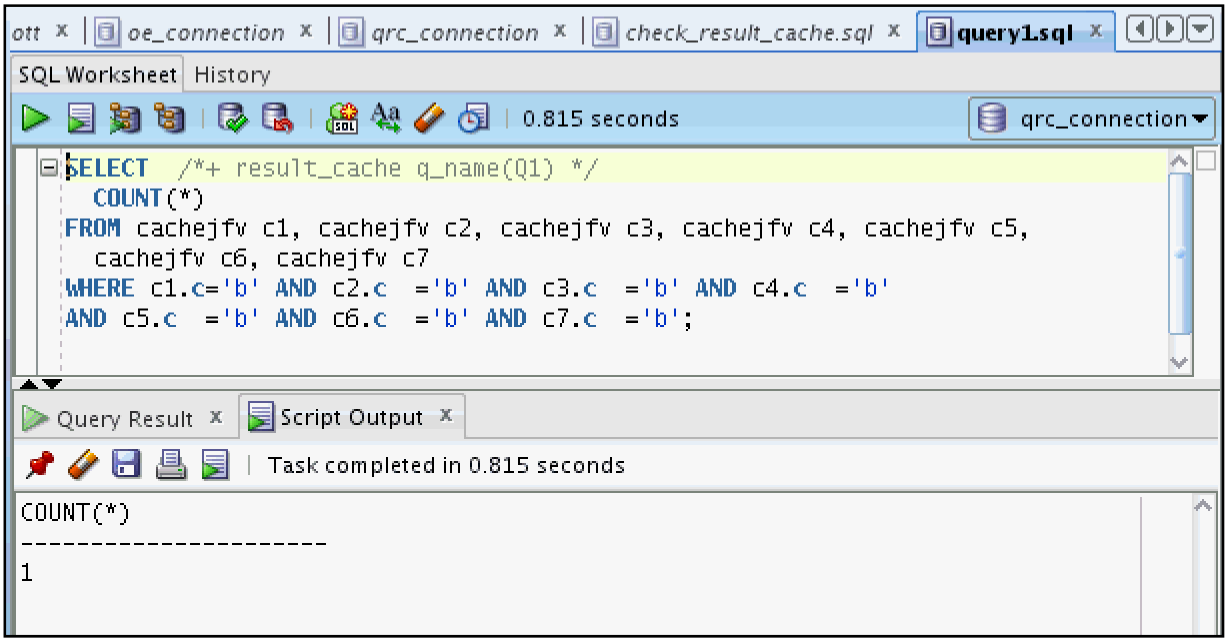
17. Ensure that you no longer bypass the result cache and check whether your query uses it again.

    a. Execute the following statement:

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



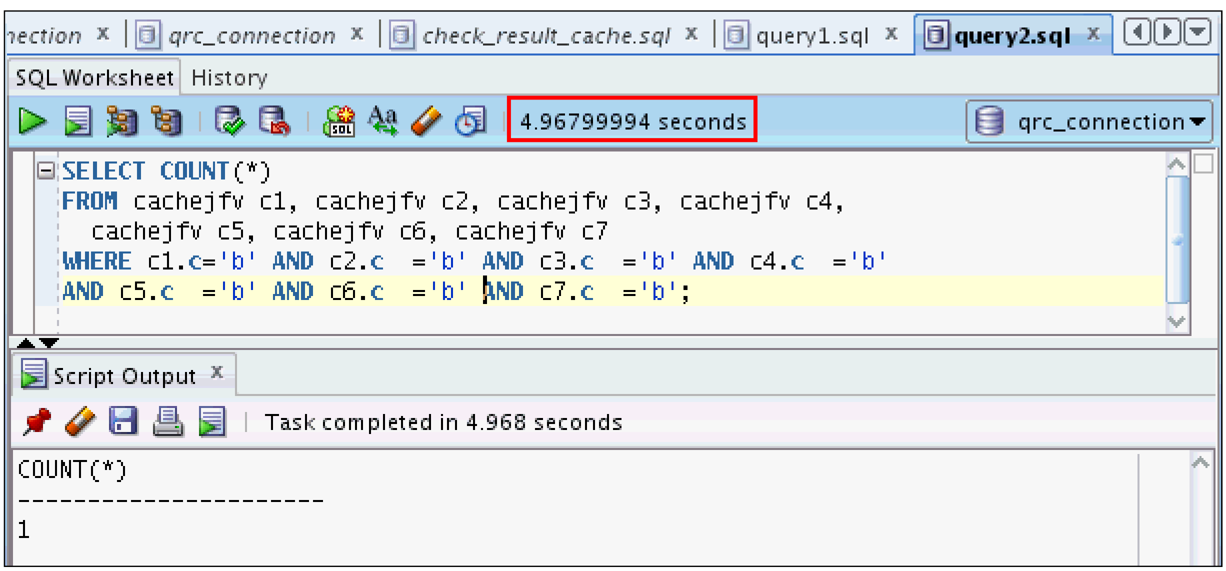
b. Rerun `query1.sql`.



18. Open and execute the `query2.sql` script or enter the following query:

```
SELECT COUNT(*)
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,
  cachejfv c5, cachejfv c6, cachejfv c7
WHERE c1.c='b' AND c2.c  ='b' AND c3.c  ='b' AND c4.c  ='b'
AND c5.c  ='b' AND c6.c  ='b' AND c7.c  ='b';
```

What do you observe?

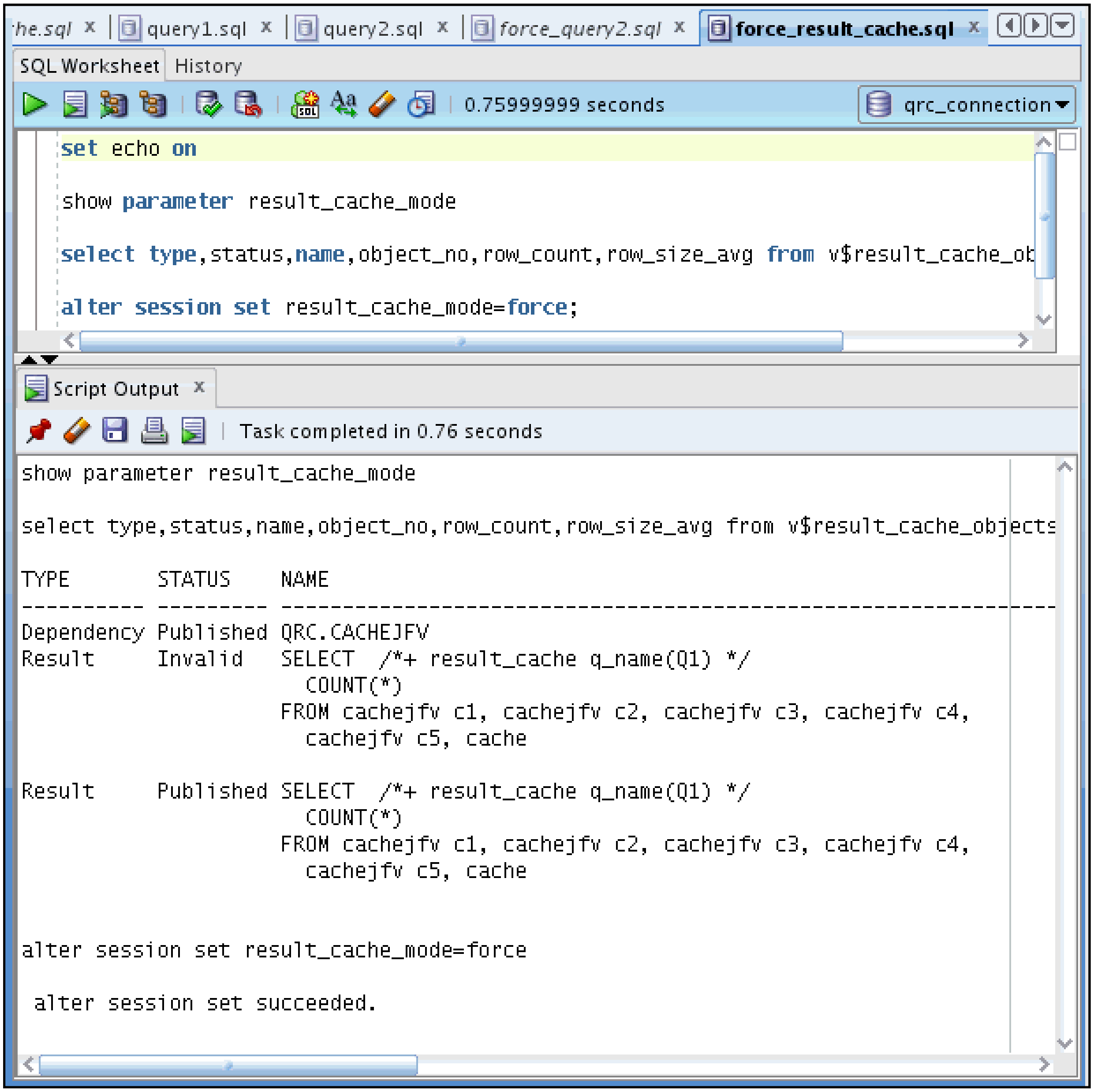


– Although the query is the same as the one in `query1.sql`, it is not recognized as cached because it does not contain the hint. So its execution time is long again.

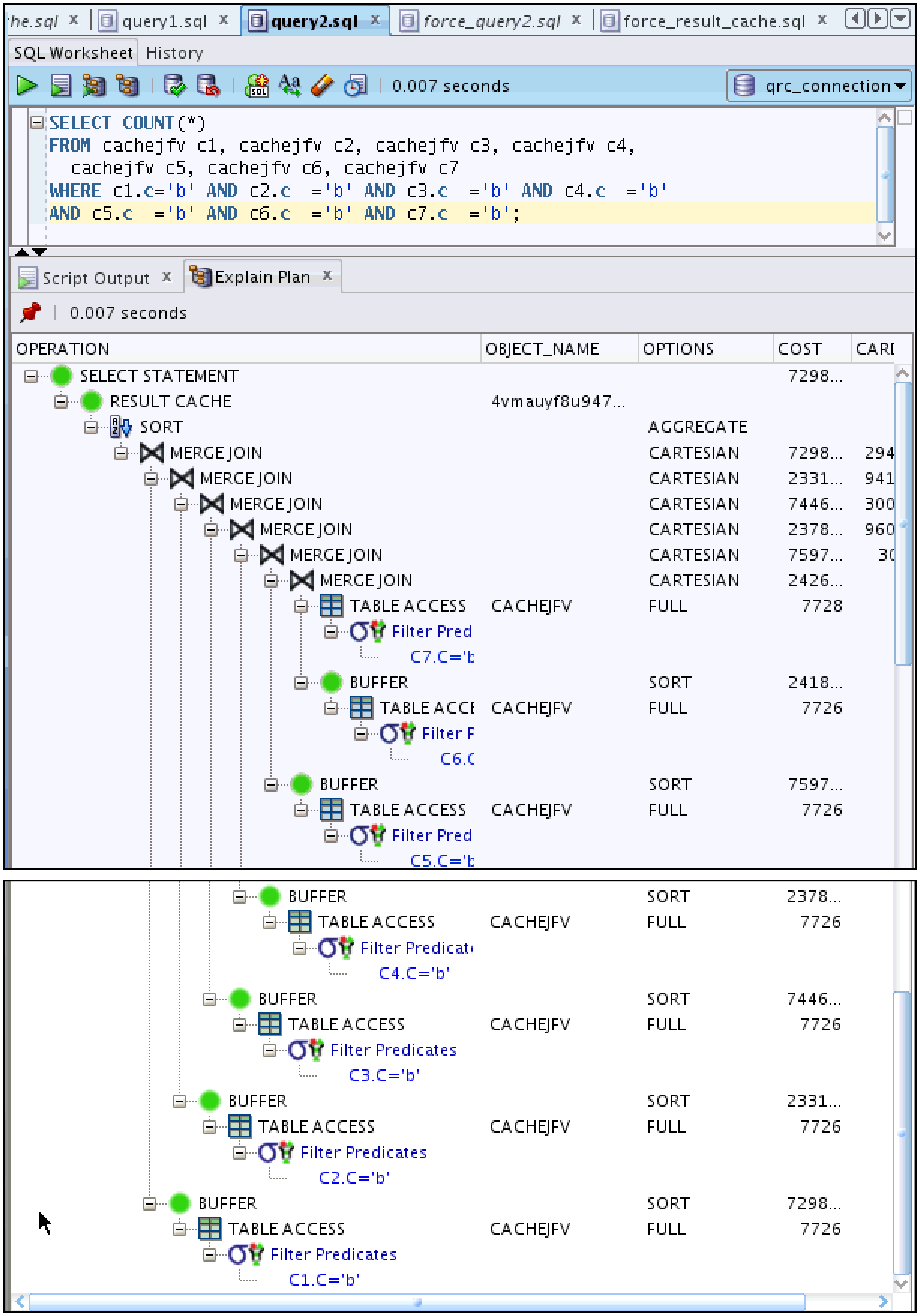
19. How would you force the previous query to use the cached result without using hints?

a. Use the following code to force the usage of 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;
```

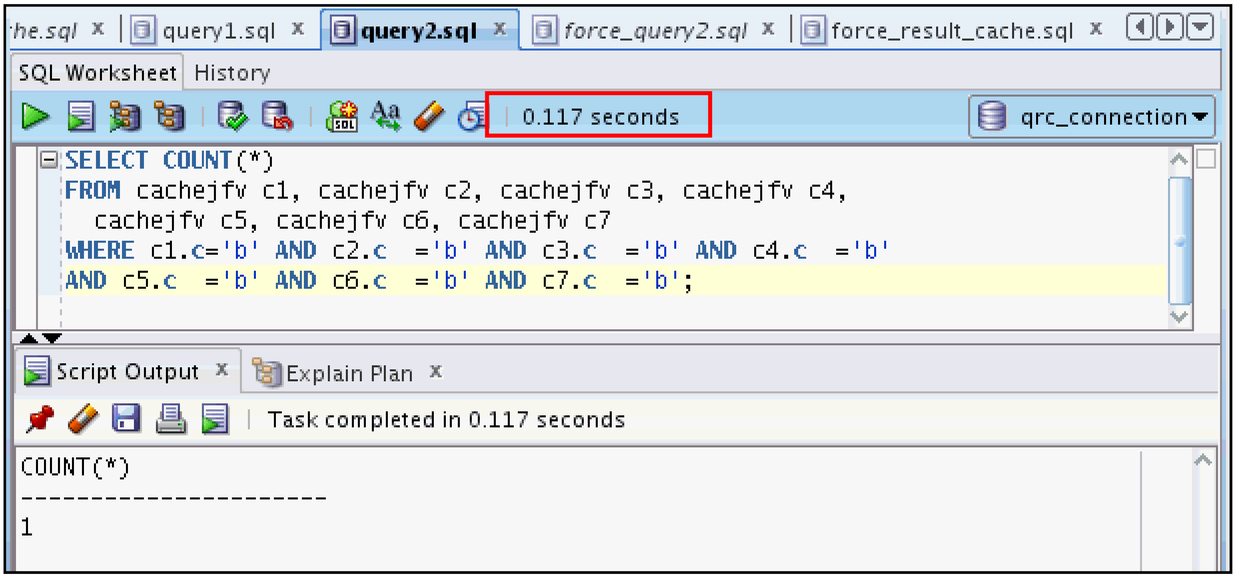


b. View the explain plan for the `query2.sql` script.



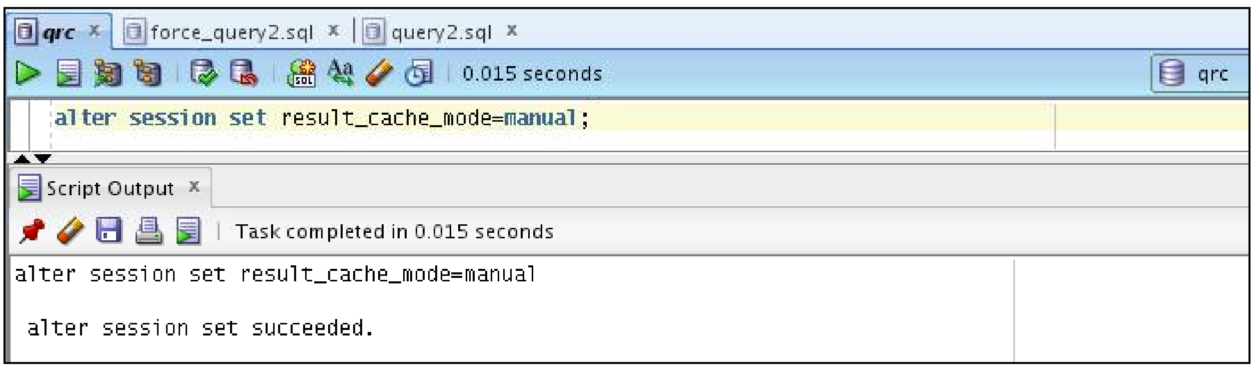
he.sql
query1.sql
query2.sql
force_query2.sql
force_result_cache.sql
SQL Worksheet
History
0.007 seconds
qrc_connection
SELECT COUNT(*)
FROM cachejfv c1, cachejfv c2, cachejfv c3, cachejfv c4,
cachejfv c5, cachejfv c6, cachejfv c7
WHERE c1.c='b' AND c2.c ='b' AND c3.c ='b' AND c4.c ='b'
AND c5.c ='b' AND c6.c ='b' AND c7.c ='b';
Script Output
Explain Plan
0.007 seconds
OPERATION
OBJECT_NAME
OPTIONS
COST
CARI
SELECT STATEMENT 7298...
RESULT CACHE 4vmauyf8u947...
SORT AGGREGATE
MERGE JOIN CARTESIAN 7298... 294
MERGE JOIN CARTESIAN 2331... 941
MERGE JOIN CARTESIAN 7446... 300
MERGE JOIN CARTESIAN 2378... 960
MERGE JOIN CARTESIAN 7597... 30
MERGE JOIN CARTESIAN 2426...
TABLE ACCESS CACHEJFV FULL 7728
Filter Pred
C7.C='b
BUFFER SORT 2418...
TABLE ACCE CACHEJFV FULL 7726
Filter F
C6.C
BUFFER SORT 7597...
TABLE ACCESS CACHEJFV FULL 7726
Filter Pred
C5.C='b
BUFFER SORT 2378...
TABLE ACCESS CACHEJFV FULL 7726
Filter Predicat
C4.C='b'
BUFFER SORT 7446...
TABLE ACCESS CACHEJFV FULL 7726
Filter Predicates
C3.C='b'
BUFFER SORT 2331...
TABLE ACCESS CACHEJFV FULL 7726
Filter Predicates
C2.C='b'
BUFFER SORT 7298...
TABLE ACCESS CACHEJFV FULL 7726
Filter Predicates
C1.C='b'

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



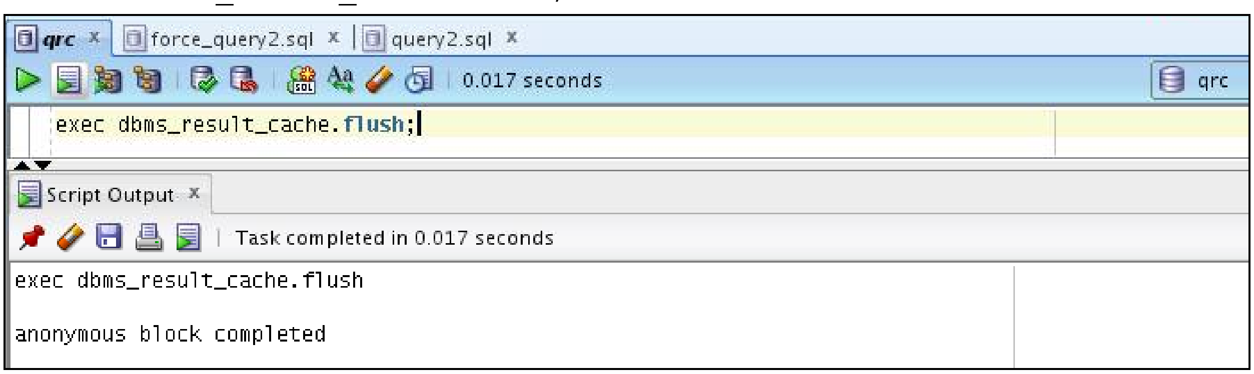
d. Finally, undo your change.

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

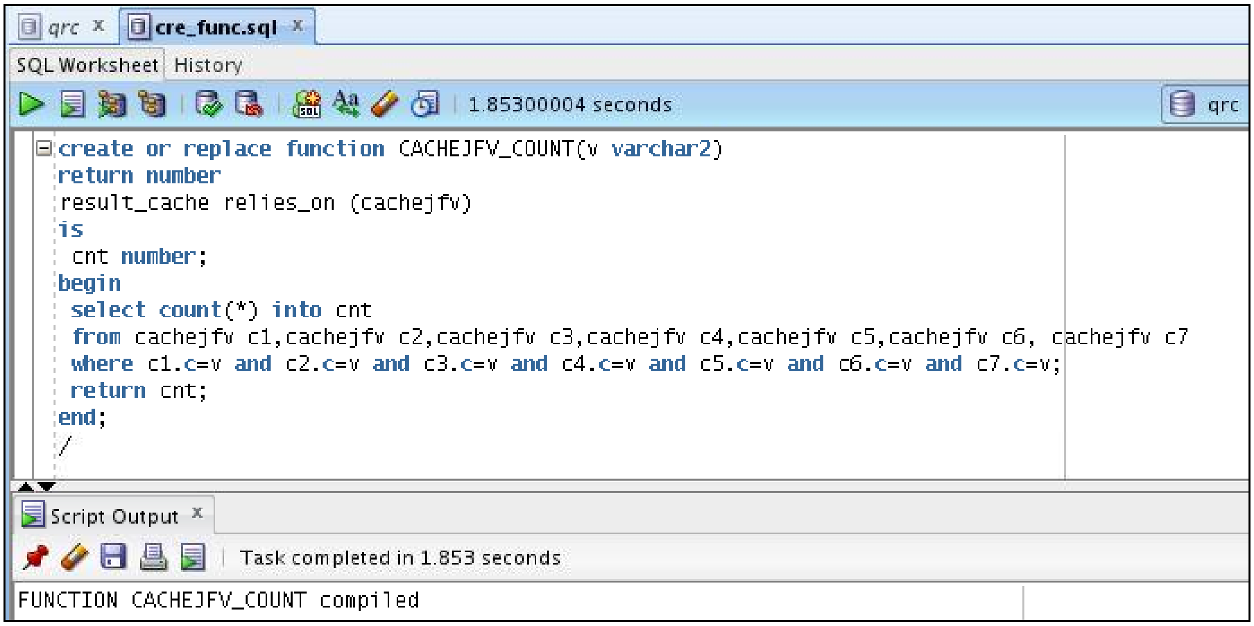


20. Clear the result cache. Query `V$RESULT_CACHE_OBJECTS` to verify the clear operation.

```
exec dbms_result_cache.flush;
```

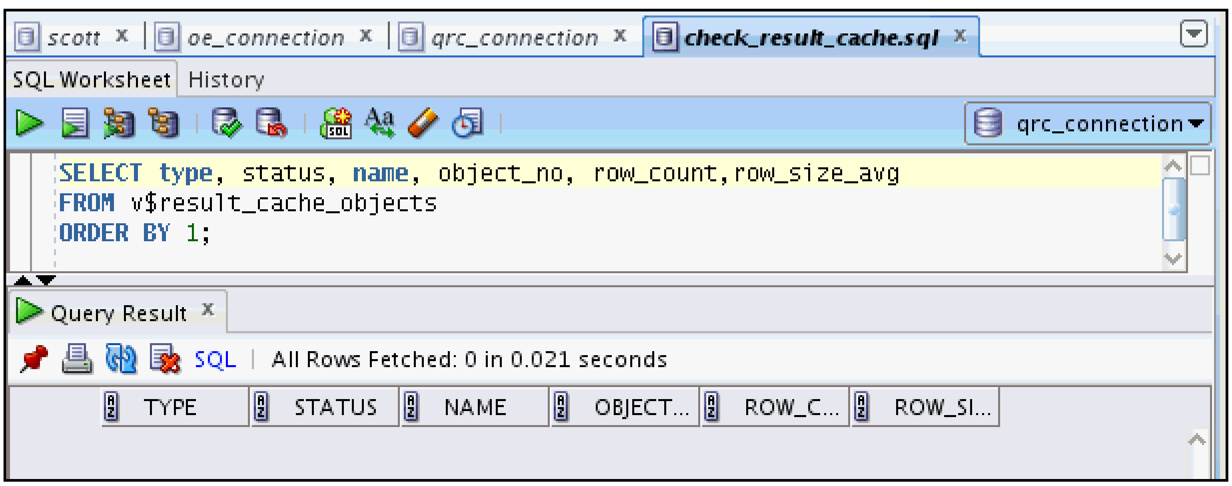


21. Open and execute the `cre_func.sql` script. This script creates a PL/SQL function that uses the result cache.



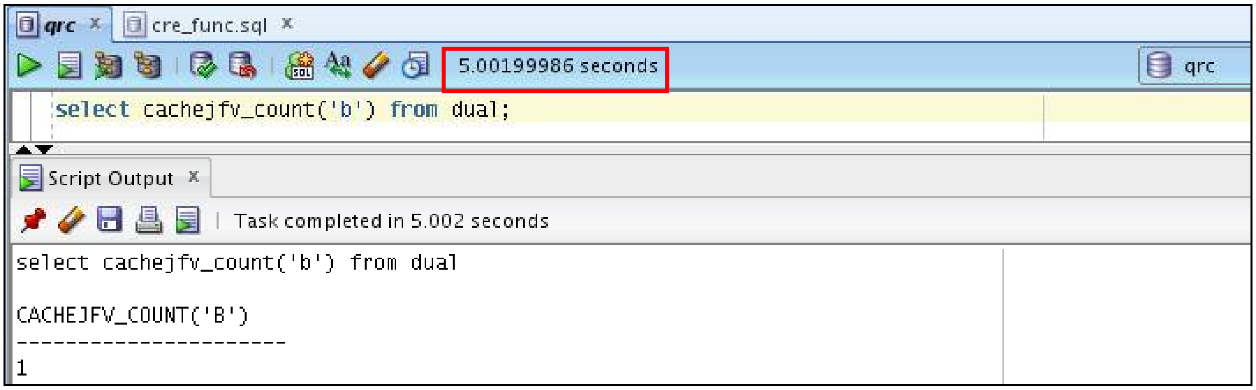
22. Determine what is in the result cache by querying `V$RESULT_CACHE_OBJECTS`. (This is the `check_result_cache.sql` script.)

```
select type,status,name,object_no,row_count,row_size_avg
from v$result_cache_objects
order by 1;
```



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

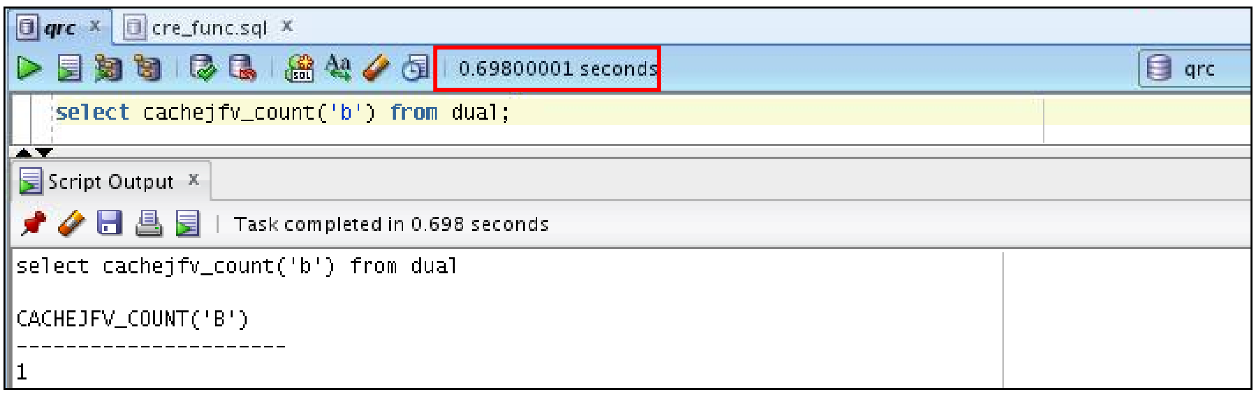
```
select cachejfv_count('b') from dual;
```



– 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.

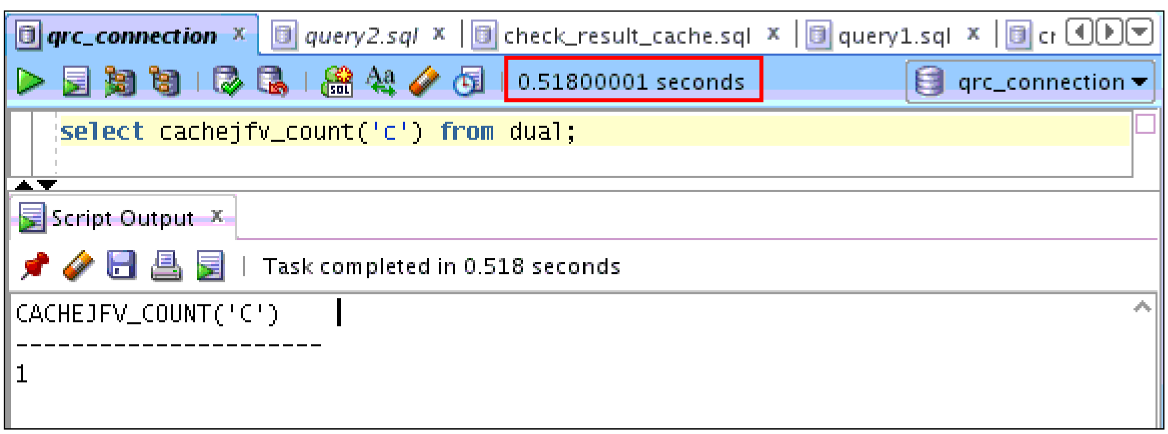
24. Call the new function with `'b'` as its argument again. What do you observe?

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



– This time the function executes almost instantaneously.

25. 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.

Practices for Lesson 9

Chapter 9

Overview of Practices for Lesson 9

Practices Overview

In these practices, you will explore the variations in the star transformation optimizer technique.

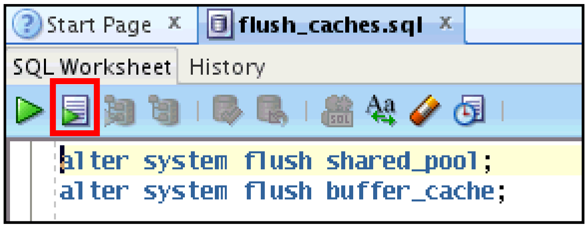
Practice 9-1: Star Schema Tuning

In this practice, you optimize a query to use star transformation and access the benefits of using this optimizer technique. The scripts for this practice are located in the `/home/oracle/solutions/Star_Schema_Tuning` directory.

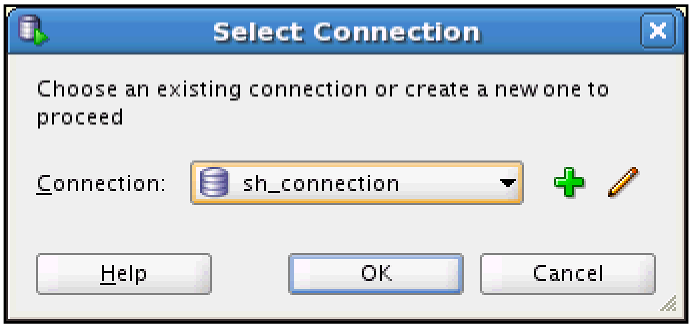
1. From a terminal session, connected as the `oracle` user, execute the `setup_star_schema_lab.sh` script located in your `/home/oracle/solutions/Star_Schema_Tuning` directory.

```
$ cd $HOME/solutions/Star_Schema_Tuning
$ ./setup_star_schema_lab.sh

…

SQL> SQL> SQL> SQL>
Grant succeeded.

SQL> SQL>
User altered.

SQL> SQL>
…
$

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


#!/bin/bash

cd /home/oracle/solutions/Star_Schema_Tuning

sqlplus / as sysdba <<FIN!

set echo on

grant dba to sh;

alter user sh identified by sh account unlock;

FIN!
```

2. From SQL Developer, open and execute the `flush_cache.sql` script to flush both the shared pool and the buffer cache to avoid caching issues as much as possible.
   a. Open the `/home/oracle/solutions/Star_Schema_Tuning/flush_caches.sql` script in SQL Developer and use the `sh_connection` to execute the script.



b. In the `flush_caches.sql` window, click the Run Script button or press F5.



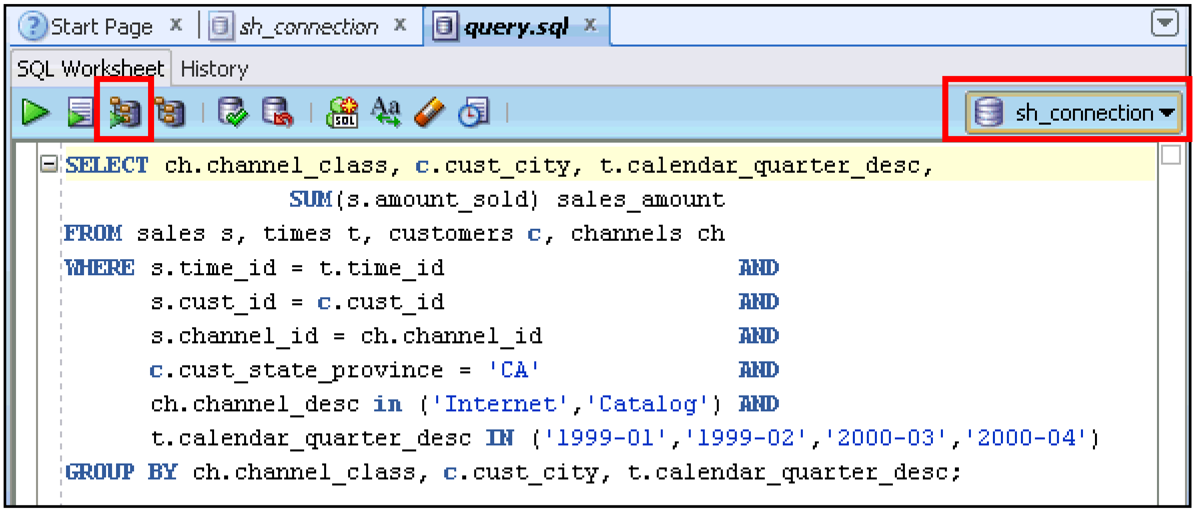
c. Select `sh_connection` and click OK

3. Open and Autotrace the execution of the query in the `query.sql` script. The query is listed here:

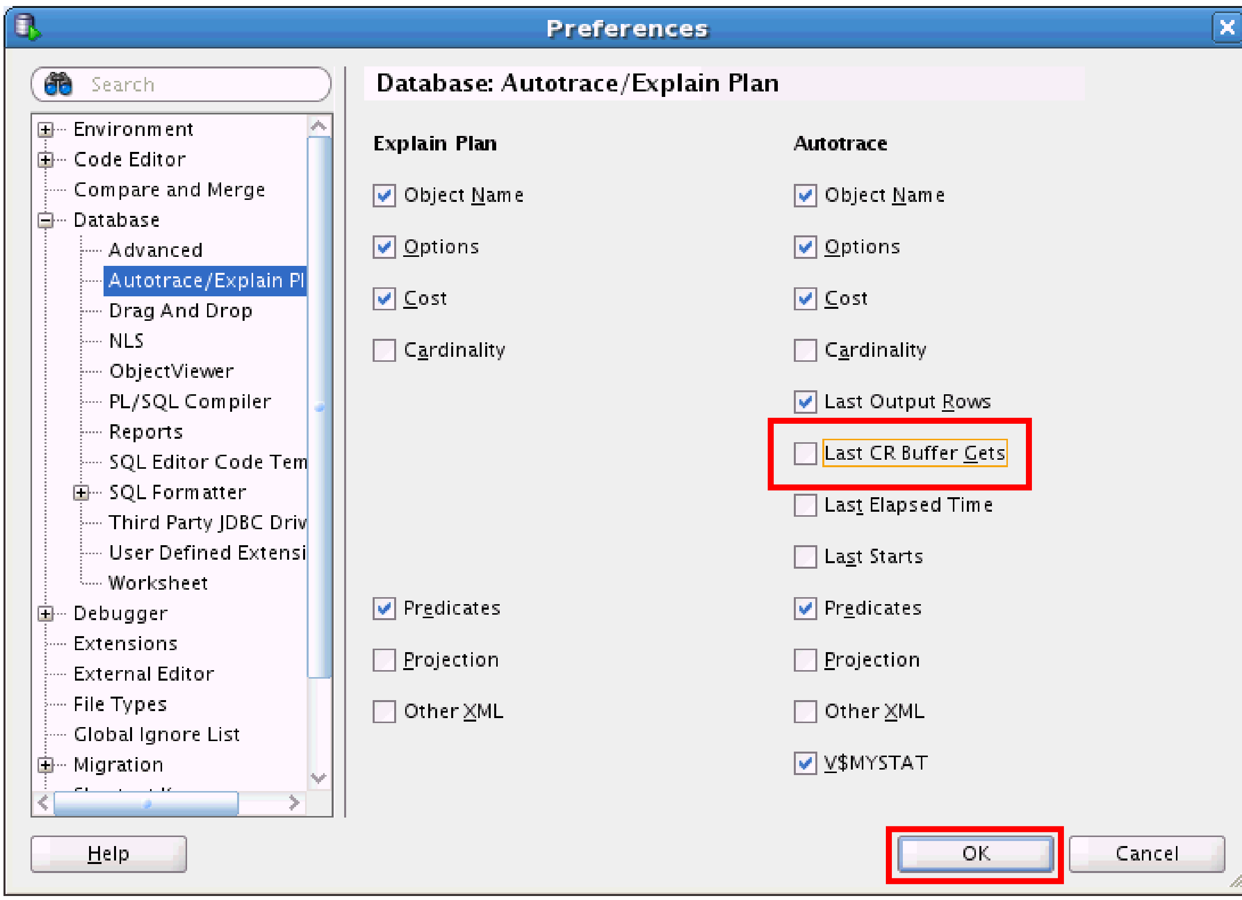
```
SELECT ch.channel_class, c.cust_city, t.calendar_quarter_desc,
                SUM(s.amount_sold) sales_amount
FROM sales s, times t, customers c, channels ch
WHERE s.time_id = t.time_id                     AND
      s.cust_id = c.cust_id                     AND
      s.channel_id = ch.channel_id              AND
      c.cust_state_province = 'CA'              AND
      ch.channel_desc in ('Internet','Catalog') AND
      t.calendar_quarter_desc IN ('1999-01','1999-02','2000-
03','2000-04')
GROUP BY ch.channel_class, c.cust_city, t.calendar_quarter_desc;
```

What are your conclusions?

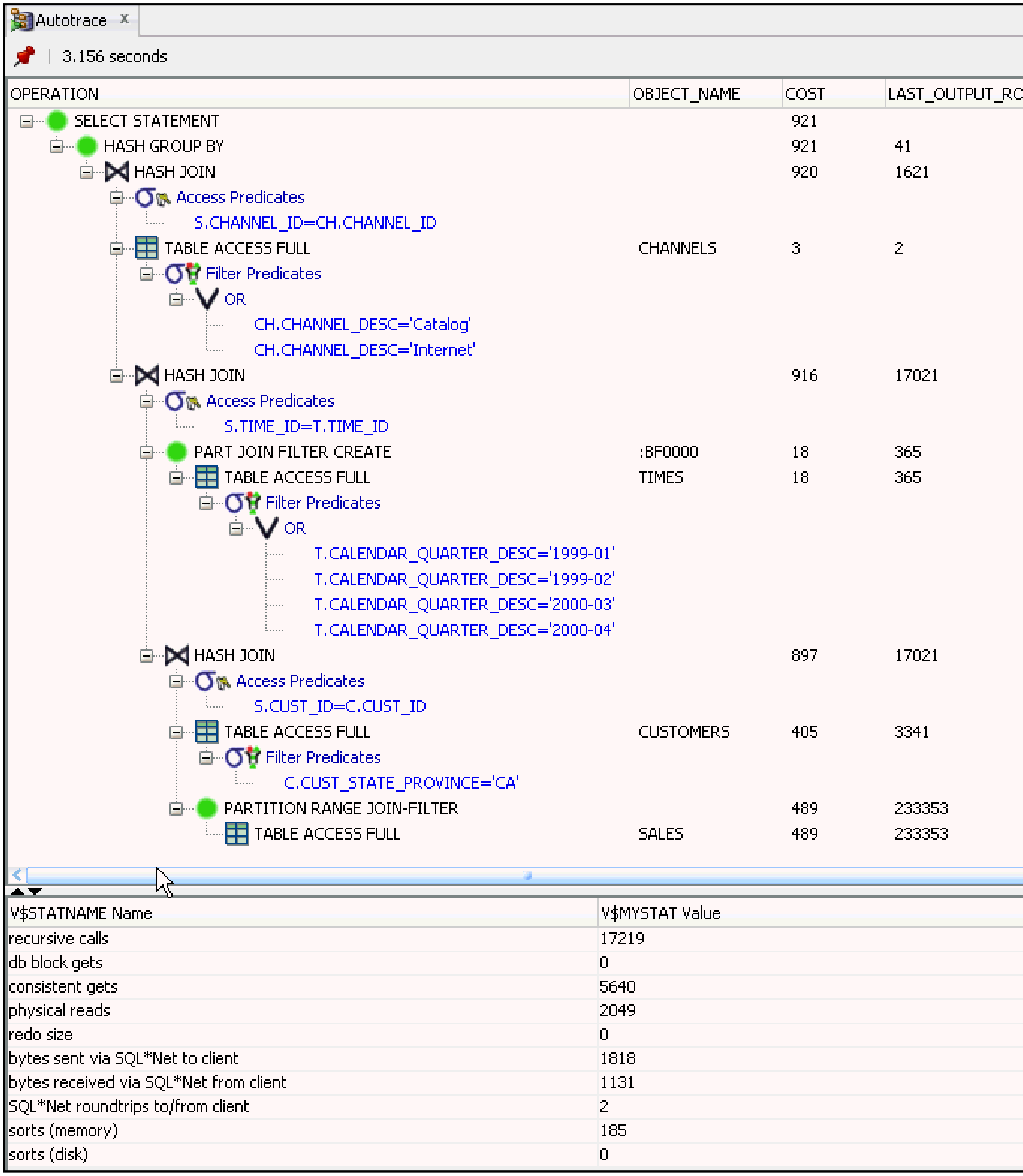
a. Open `/home/oracle/solutions/Star_Schema_Tuning/flush_cache.sql` in SQL Developer, select `sh_connection`, and click Autotrace.



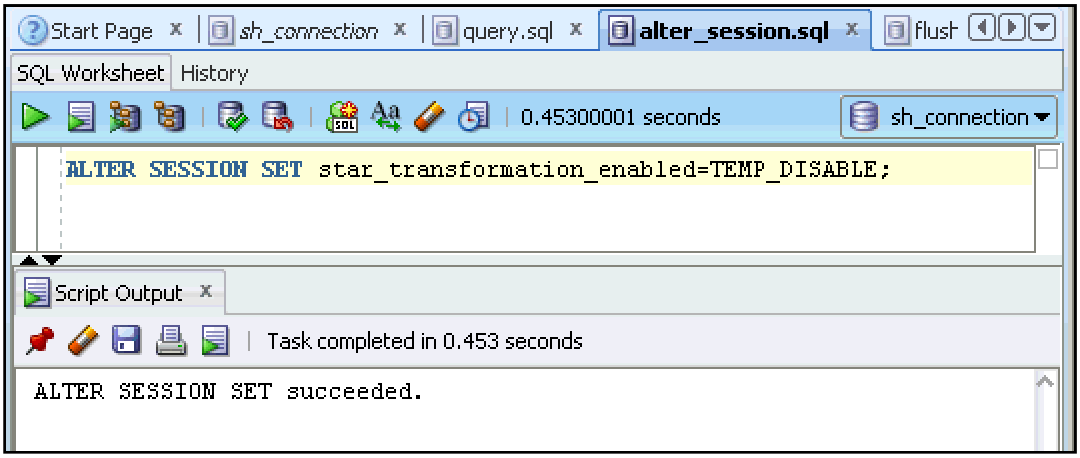
b. Change the Autotrace preferences to show the Last Output Rows and not show the Last CR Buffer Gets. Click OK.



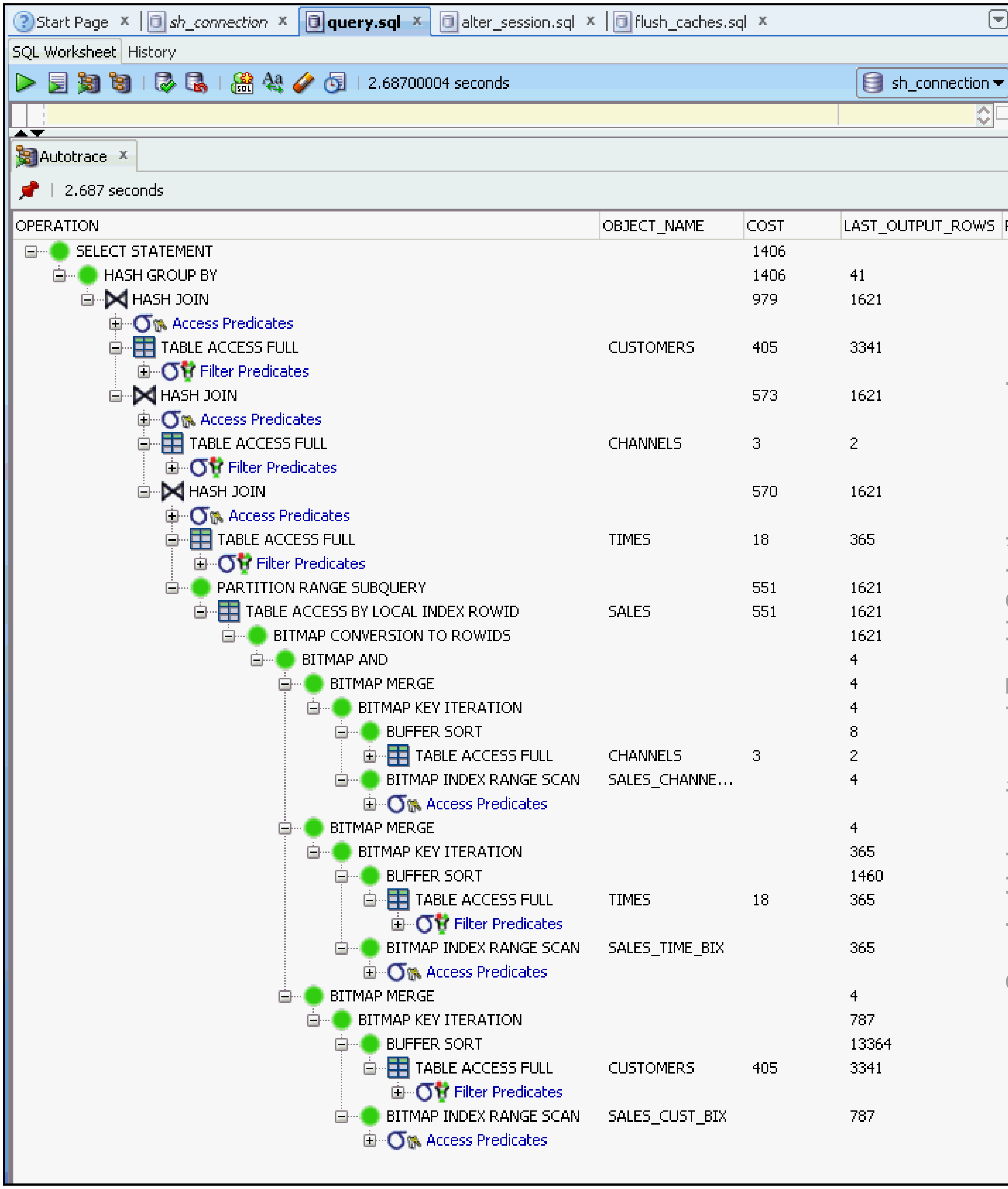
As you can see in the output of the execution plan, this query seems to use a large number of bytes to access the `SALES` table. Basically, the optimizer performs a full scan of this table. This might not be the best way to handle it.



4. Without modifying the `SH` schema, how can you improve the execution plan for this query in the previous step? Verify your solution and explain why it is probably a better solution.
   a. Execute the `flush_caches.sql` script again. **Hint:** Click the `flush_caches.sql` tab and press F5.
   b. Enable star transformation in your session. Open the `alter_session.sql` script. Press F5, and connect using `sh_connection`.

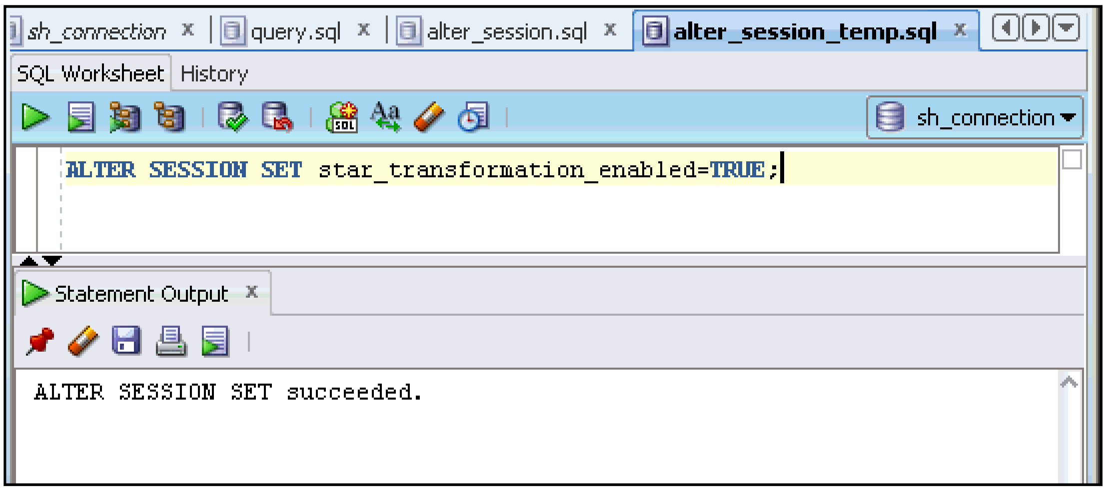


c. In this step, you do not want to use a temporary table for the star transformation. Looking at the previous execution plan, the optimizer estimates the data that is to be manipulated in megabytes. Using the star transformation as follows, the estimation is now expressed in kilobytes. That is why this new execution plan is probably a much better alternative. However, note that this time the `CUSTOMERS` table is accessed using full scan twice. If the table is larger, the impact is significant.

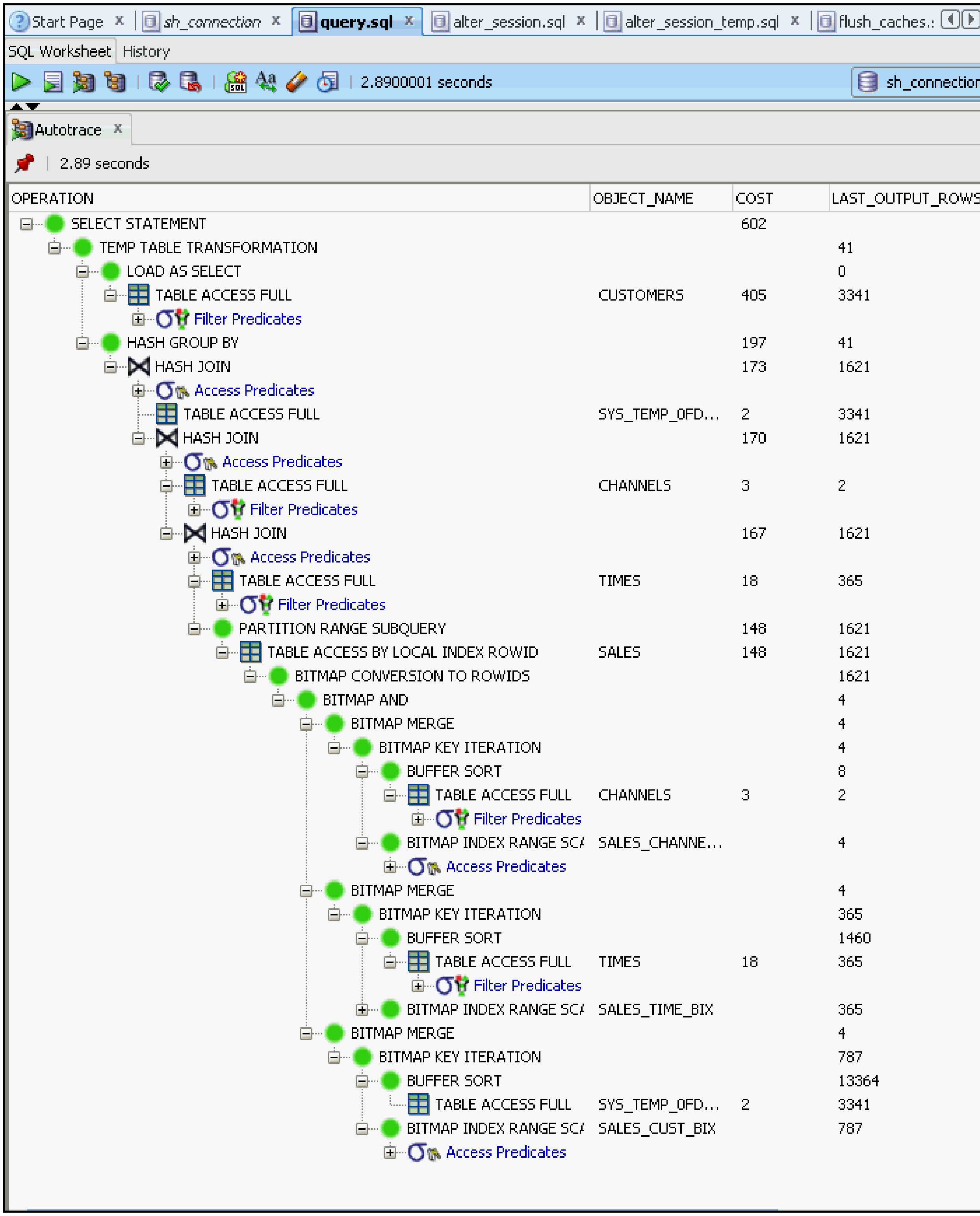

Start Page
sh_connection
query.sql
alter_session.sql
flush_caches.sql
SQL Worksheet
History
2.68700004 seconds
sh_connection
Autotrace
2.687 seconds
OPERATION | OBJECT_NAME | COST | LAST_OUTPUT_ROWS
SELECT STATEMENT | | 1406 |
HASH GROUP BY | | 1406 | 41
HASH JOIN | | 979 | 1621
Access Predicates
TABLE ACCESS FULL | CUSTOMERS | 405 | 3341
Filter Predicates
HASH JOIN | | 573 | 1621
Access Predicates
TABLE ACCESS FULL | CHANNELS | 3 | 2
Filter Predicates
HASH JOIN | | 570 | 1621
Access Predicates
TABLE ACCESS FULL | TIMES | 18 | 365
Filter Predicates
PARTITION RANGE SUBQUERY | | 551 | 1621
TABLE ACCESS BY LOCAL INDEX ROWID | SALES | 551 | 1621
BITMAP CONVERSION TO ROWIDS | | | 1621
BITMAP AND | | | 4
BITMAP MERGE | | | 4
BITMAP KEY ITERATION | | | 4
BUFFER SORT | | | 8
TABLE ACCESS FULL | CHANNELS | 3 | 2
BITMAP INDEX RANGE SCAN | SALES_CHANNE... | | 4
Access Predicates
BITMAP MERGE | | | 4
BITMAP KEY ITERATION | | | 365
BUFFER SORT | | | 1460
TABLE ACCESS FULL | TIMES | 18 | 365
Filter Predicates
BITMAP INDEX RANGE SCAN | SALES_TIME_BIX | | 365
Access Predicates
BITMAP MERGE | | | 4
BITMAP KEY ITERATION | | | 787
BUFFER SORT | | | 13364
TABLE ACCESS FULL | CUSTOMERS | 405 | 3341
Filter Predicates
BITMAP INDEX RANGE SCAN | SALES_CUST_BIX | | 787
Access Predicates

V$STATNAME Name	V$MYSTAT Value
recursive calls	17
db block gets	0
consistent gets	33341
physical reads	3455
redo size	0
bytes sent via SQL*Net to client	1818
bytes received via SQL*Net from client	1131
SQL*Net roundtrips to/from client	2
sorts (memory)	6
sorts (disk)	0

5. How would you enhance the previous optimization without changing the `SH` schema?
   a. Let the optimizer decide if it is better to use a temporary table. Set the `STAR_TRANSFORMATION_ENABLED` parameter to `TRUE`. Open the `alter_session_temp.sql` script, set the connection to `sh_connection`, and click Execute Statement or press Ctrl + Enter. Note that because there is only one statement in the script, the execute statement and the execute script are equivalent.



   b. Flush the caches again. Click the `flush_caches.sql` tab and click execute Script or press F5.
   c. Analyze the query again, Click the `query.sql` tab, and click Autotrace. Note that the Filter Predicate and Access Predicate lines have been collapsed, so the execution plan will fit.

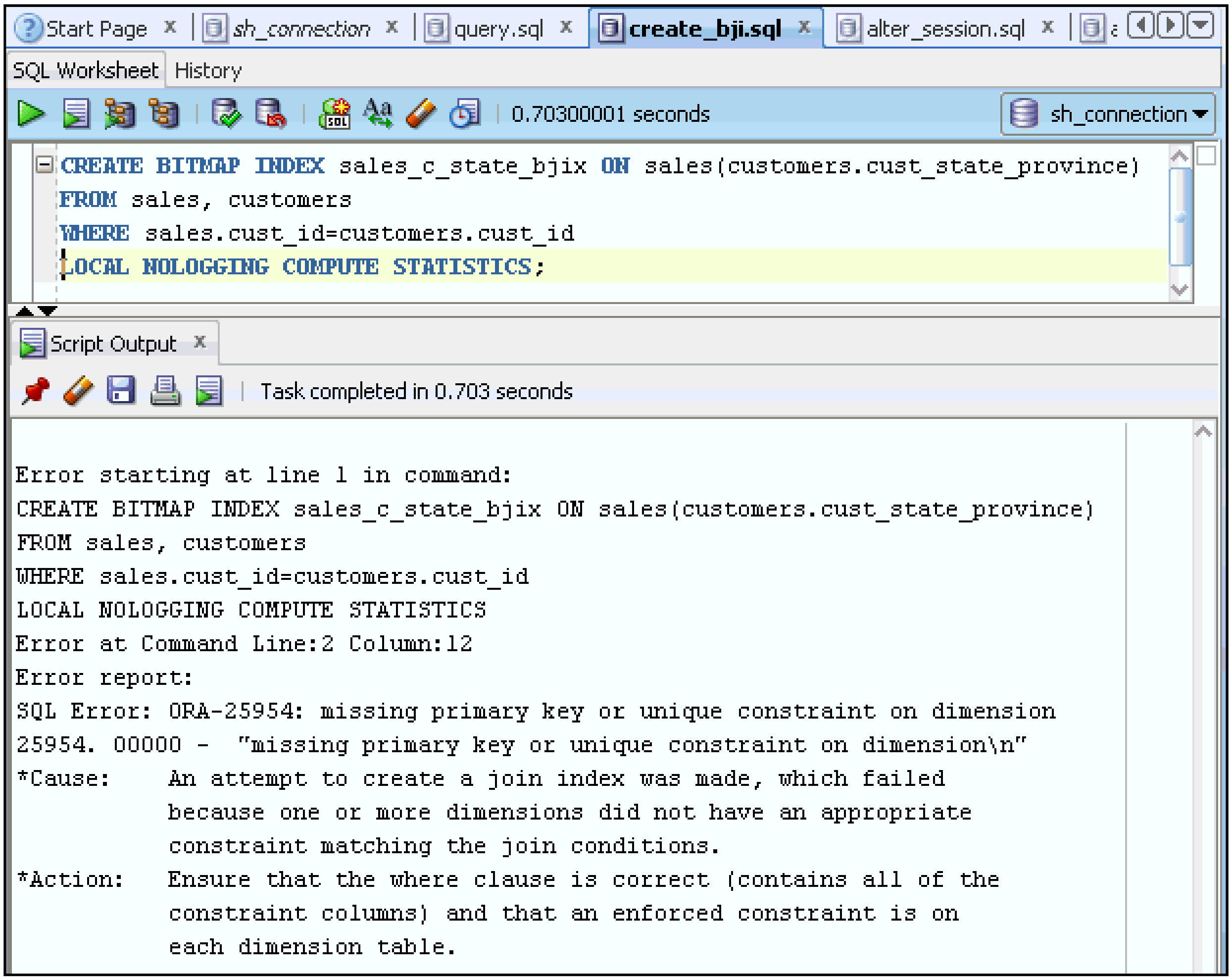


OPERATION	OBJECT_NAME	COST	LAST_OUTPUT_ROWS
SELECT STATEMENT		602	
TEMP TABLE TRANSFORMATION			41
LOAD AS SELECT			0
TABLE ACCESS FULL	CUSTOMERS	405	3341
Filter Predicates			
HASH GROUP BY		197	41
HASH JOIN		173	1621
Access Predicates			
TABLE ACCESS FULL	SYS_TEMP_0FD...	2	3341
HASH JOIN		170	1621
Access Predicates			
TABLE ACCESS FULL	CHANNELS	3	2
Filter Predicates			
HASH JOIN		167	1621
Access Predicates			
TABLE ACCESS FULL	TIMES	18	365
Filter Predicates			
PARTITION RANGE SUBQUERY		148	1621
TABLE ACCESS BY LOCAL INDEX ROWID	SALES	148	1621
BITMAP CONVERSION TO ROWIDS			1621
BITMAP AND			4
BITMAP MERGE			4
BITMAP KEY ITERATION			4
BUFFER SORT			8
TABLE ACCESS FULL	CHANNELS	3	2
Filter Predicates			
BITMAP INDEX RANGE SCA	SALES_CHANNE...		4
Access Predicates			
BITMAP MERGE			4
BITMAP KEY ITERATION			365
BUFFER SORT			1460
TABLE ACCESS FULL	TIMES	18	365
Filter Predicates			
BITMAP INDEX RANGE SCA	SALES_TIME_BIX		365
BITMAP MERGE			4
BITMAP KEY ITERATION			787
BUFFER SORT			13364
TABLE ACCESS FULL	SYS_TEMP_0FD...	2	3341
BITMAP INDEX RANGE SCA	SALES_CUST_BIX		787
Access Predicates			

6. How do you eliminate one table access on the CUSTOMERS table from the previous execution plan for the same SELECT statement?
   a. Create a bitmap join index between the SALES and CUSTOMERS tables.
7. Try to apply your finding. What happens and why?

a. Open the `create_bji.sql` script. Click execute script or press F5. Select `sh_connection`.

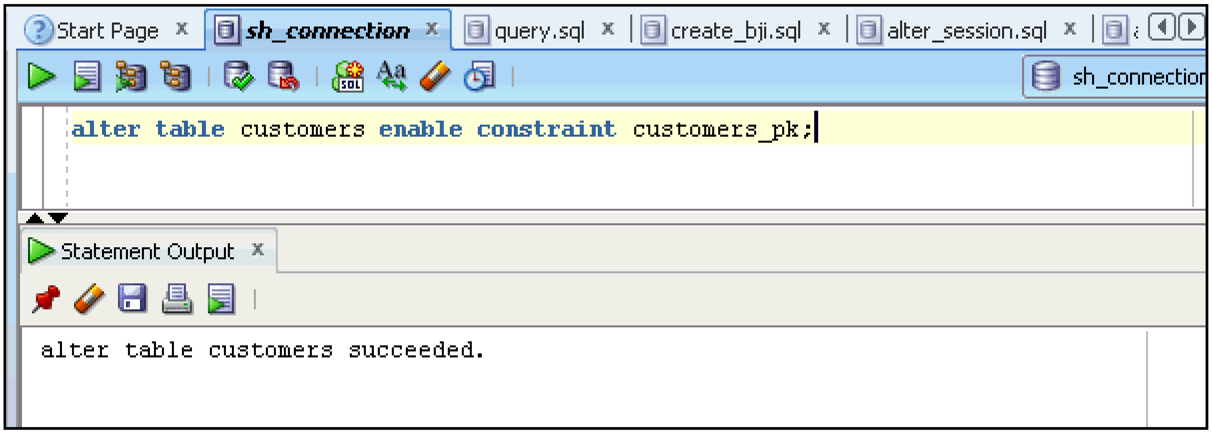
Because the `CUSTOMERS_PK` primary key constraint is not enforced, it is not possible to create a bitmap join index between the `SALES` and `CUSTOMERS` tables.



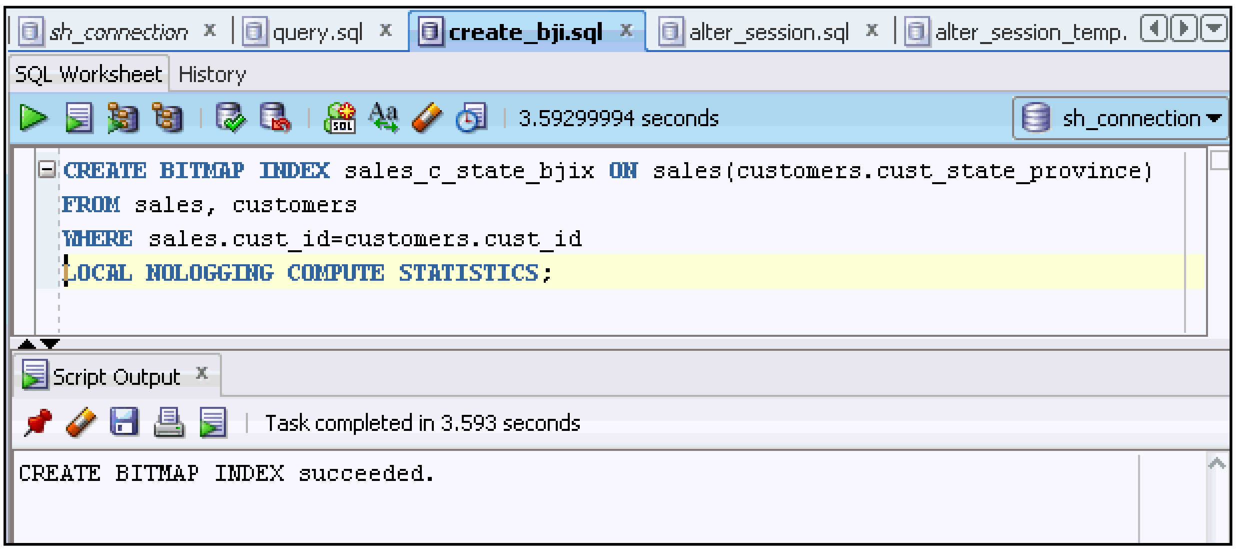
8. Fix the issue you found and check whether the bitmap join index does enhance performance for the query.

   a. You need to `ENABLE VALIDATE` the `CUSTOMERS_PK` constraint before you can create the bitmap join index. Enter the following command in the `sh_conection` worksheet and execute:

```
alter table customers enable constraint customers_pk;
```

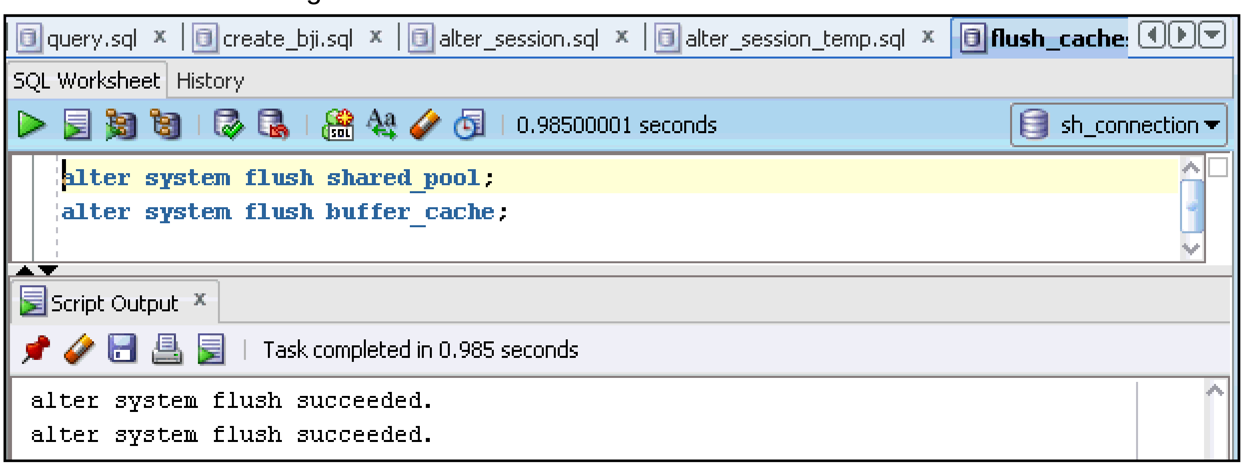


b. Create the bitmap join index. Execute `create_bji.sql`.

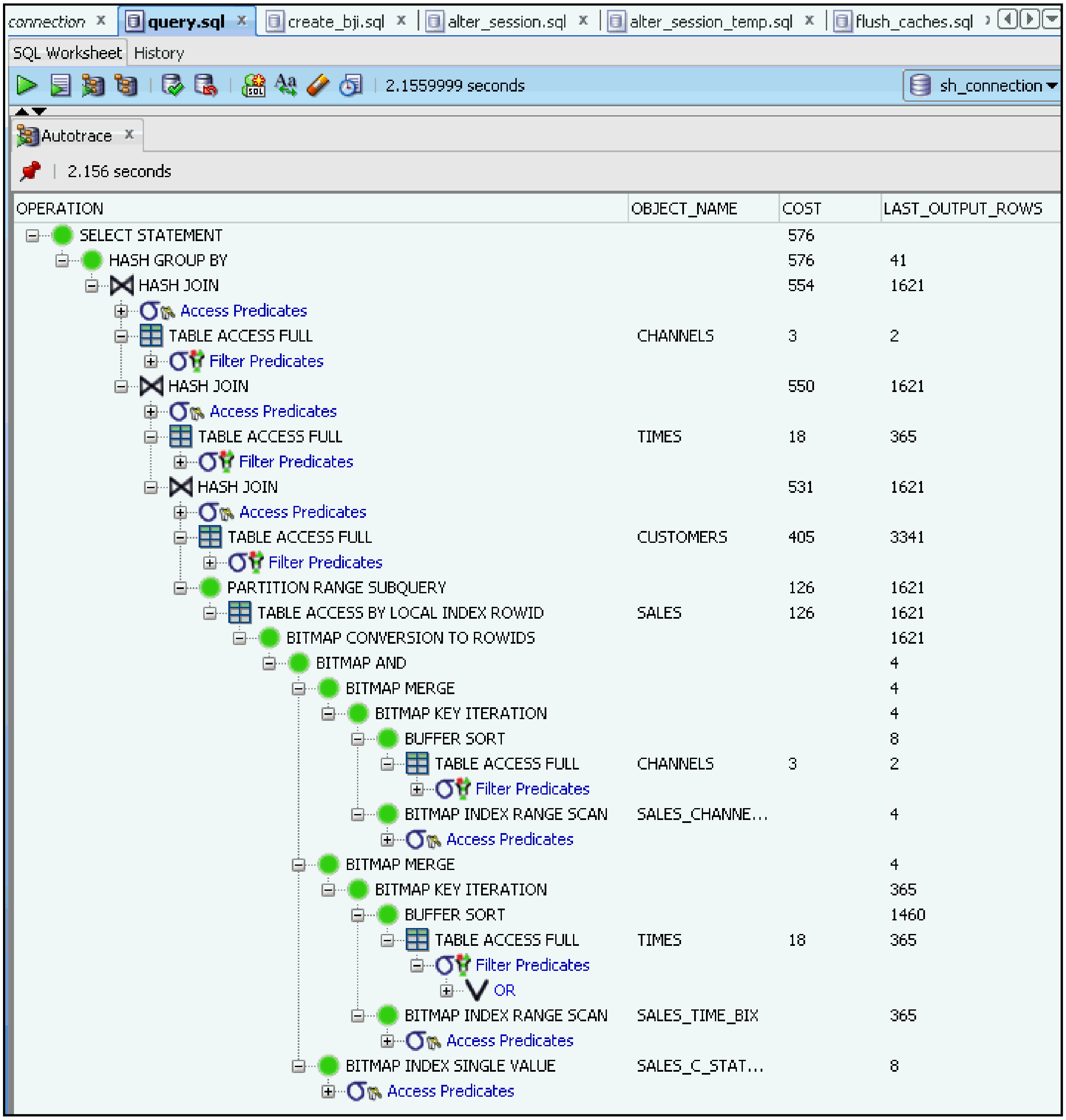


9. Verify that the bitmap join will help the performance of the query.

a. Flush the caches again.



b. Autotrace `query.sql`. **Note:** The access predicates and filter predicates have been collapsed to allow the execution plan to fit without scrolling



10. Determine how the system could dynamically determine which `SALES` partitions to access for the previous query.

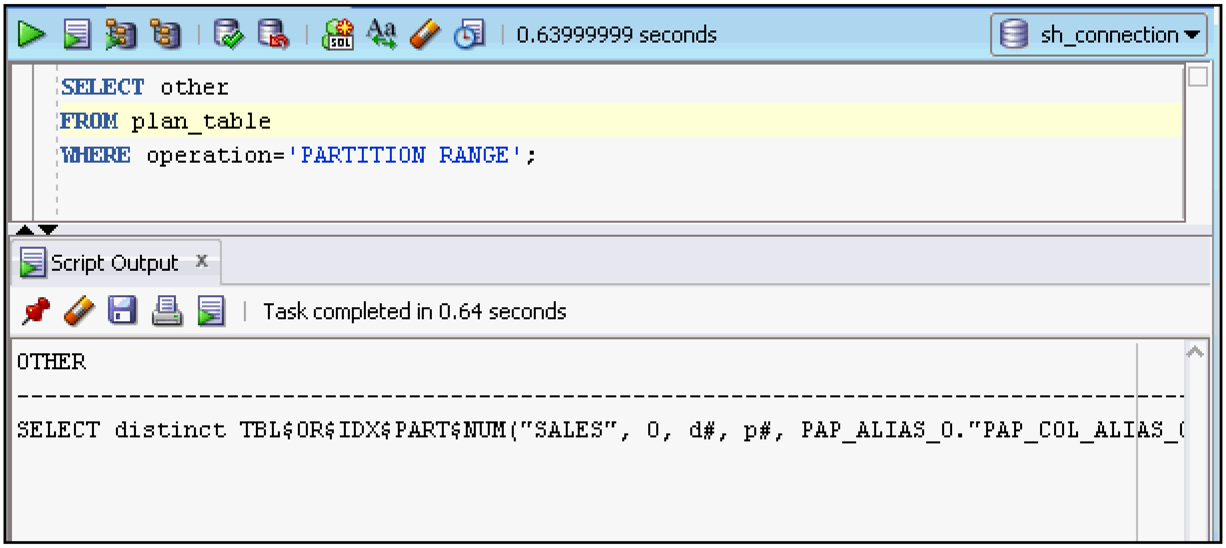
    a. Explain plan (F10) the `query.sql` script in the previous step.

    b. View the `OTHER` column of `PLAN_TABLE` for the `PARTITION RANGE` operation.

       Use the following query:

```
SELECT other
FROM plan_table
WHERE operation='PARTITION RANGE';
```

       You can enter this query in SQL Developer, and view the output by scrolling. The output from SQL Developer is reformatted below the screenshot.




```
 OTHER
----------------------------------------------------------------
SELECT distinct TBL$OR$IDX$PART$NUM("SALES", 0, d#, p#,
 PAP_ALIAS_0."PAP_COL_ALIAS_0")
FROM (SELECT /*+ SEMIJOIN_DRIVER */ "T"."TIME_ID"
"PAP_COL_ALIAS_0" FROM "TIMES" "T"
WHERE "T"."CALENDAR_QUARTER_DESC"='1999-01'
OR "T"."CALENDAR_QUARTER_DESC"='1999-02'
OR "T"."CALENDAR_QUARTER_DESC"='2000-03'
OR "T"."CALENDAR_QUARTER_DESC"='2000-04') PAP_ALIAS_0 ORDER BY 1
```

11. Clean up your environment by removing the index you created and returning the constraint to its original state.

```
$ ./cleanup_star_schema_lab.sh

SQL*Plus: Release 11.2.0.1.0 Production on Thu Jul 15 01:56:48
2010

Copyright (c) 1982, 2009, Oracle.  All rights reserved.

SQL> Connected.
SQL> SQL> SQL> SQL> SQL> SQL>
Index dropped.

SQL> SQL>
Table altered.


SQL> SQL> Disconnected …
$
```

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

sqlplus /nolog <<-FIN
connect sh/sh
set echo on

set timing off
set autotrace off

drop index sales_c_state_bjix;

alter table customers enable novalidate constraint customers_pk;

exit;
FIN
```

Practices for Lesson 10

Chapter 10

Overview of Practices for Lesson 10

Practices Overview

In these practices, you will examine the role system statistics play in optimizer choices and the effect object statistics have on the execution plans created by the optimizer.

Practice 10-1: Using System Statistics

In this practice, you manipulate system statistics and show that they are important for the optimizer to select the correct execution plans. All scripts used in this practice are located in the `$HOME/solutions/System_Stats` directory.

1. The `sysstats_setup.sh` script located in your `$HOME/solutions/System_Stats` directory was run as part of the class setup. This script creates a user called `JFV` and some tables used throughout this lab. The script also makes sure that object statistics are correctly gathered. The script is listed here:

```
#!/bin/bash

cd /home/oracle/solutions/System_Stats

sqlplus / as sysdba @sysstats_setup.sql

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

set echo on

connect / as sysdba;

drop user jfv cascade;

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

grant connect, resource, dba to jfv;


connect jfv/jfv

drop table t purge;

drop table z purge;

create table t(c number);

insert into t values (1);

commit;

insert into t select * from t;

/
/
/
```

```
/
/
/
/
/
/
/

commit;

insert into t select * from t;

/
/
/
/
/
/
/

commit;

create table z(d number);

begin
for i in 1..100 loop
    insert into z values (i);
end loop;
commit;
end;
/

create unique index iz on z(d);

execute dbms_stats.gather_table_stats('JFV','T',cascade=>true);

execute dbms_stats.gather_table_stats('JFV','Z',cascade=>true);

exit;
```

2. In a terminal window, execute the `sysstats_start.sh` script to flush the caches, and set the `CPUSPEEDNW` to 0 in the system statistics.

```
$ cd /home/oracle/solutions/System_Stats
$ ./sysstats_start.sh
```

```
…

SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> alter system flush buffer_cache;

System altered.

SQL>
SQL> execute dbms_stats.delete_system_stats;

PL/SQL procedure successfully completed.

SQL>
SQL> execute DBMS_STATS.SET_SYSTEM_STATS (pname => 'cpuspeednw',
pvalue => 0);

PL/SQL procedure successfully completed.

SQL>
SQL> select sname,pname,pval1 from aux_stats$;

SNAME                          PNAME                               PVAL1
------------------------------ ------------------------------ --------
SYSSTATS_INFO                  STATUS
SYSSTATS_INFO                  DSTART
SYSSTATS_INFO                  DSTOP
SYSSTATS_INFO                  FLAGS                                  1
SYSSTATS_MAIN                  CPUSPEEDNW                             0
SYSSTATS_MAIN                  IOSEEKTIM                             10
SYSSTATS_MAIN                  IOTFRSPEED                          4096
SYSSTATS_MAIN                  SREADTIM
SYSSTATS_MAIN                  MREADTIM
SYSSTATS_MAIN                  CPUSPEED
SYSSTATS_MAIN                  MBRC

SNAME                          PNAME
PVAL1
------------------------------ ------------------------------ --------
SYSSTATS_MAIN                  MAXTHR
SYSSTATS_MAIN                  SLAVETHR
```

```
13 rows selected.

SQL>
SQL> exit;
…
$

--------------------------------------------------------------
#!/bin/sh

sqlplus /nolog @sysstats_start.sql
--------------------------------------------------------------
connect / as sysdba;

alter system flush shared_pool;

alter system flush buffer_cache;

execute dbms_stats.delete_system_stats;

execute DBMS_STATS.SET_SYSTEM_STATS (pname => 'cpuspeednw', pvalue =>
0);

select sname,pname,pval1 from aux_stats$;

exit;
```

3. From your terminal session, connect as the `JFV` user in the SQL*Plus session. Then execute the `select_without_sysstats.sql` script which contains the following statement and determine how long it takes to execute:

```
select /* Without system stats */ count(*)
from t,z
where t.c=z.d;
```

```
$ sqlplus jfv/jfv

…

SQL> @select_without_sysstats
SQL>
SQL> set timing on
SQL>
SQL> select /* Without system stats */ count(*)
  2  from t,z
  3  where t.c=z.d;
```

```
  COUNT(*)
----------
    524288

Elapsed: 00:00:00.33
SQL>
SQL> set timing off
SQL>
SQL>
```

4. Determine the execution plan used to execute the previous statement. In addition, determine the optimizer's cost, CPU cost, and I/O cost for the previous execution. Use the `show_latest_exec_plan.sql` script. What do you observe?
   a. The optimizer does not use CPU costing. This is because system statistics were deleted during the first step of this lab. 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$sql_plan
  4  where address in (select address from v$sql
  5                    where sql_text like '%system stats%'
  6                    and sql_text not like '%connect%'));

OPERATIONS           OBJECT_NAME OPTIONS         COST_CPU_IO
-------------------- ----------- --------------- --------------------
SELECT STATEMENT                                 134 --  --
SORT                             AGGREGATE        --  --
NESTED LOOPS                                     134 --  -- 134
TABLE ACCESS         T           FULL            134 --  -- 134
INDEX                IZ          UNIQUE SCAN      --  --

SQL>
---------------------------------------------------------------


set echo on

select * from table(dbms_xplan.display_cursor);

col operations  format a20
col object_name format a11
col options     format a15
col cost_cpu_io format a30



select operation operations, object_name, options,
```

```
       cost||' -- '||cpu_cost||' -- '||io_cost cost_cpu_io
from (select * from v$sql_plan where address in (select address
                                                 from v$sql
                                                 where sql_text like
'%system stats%' and
                                                       sql_text not
like '%connect%'));
```

5. How can you ensure that the optimizer finds a better plan during future executions of the same statement? Implement your solution.
   a. Gather system statistics again. Because you do not have a real workload yet, you can gather system statistics in `NOWORKLOAD` mode.

```
SQL> connect / as sysdba;
Connected.
SQL> @gather_system_stats
SQL> set echo on
SQL>
SQL> execute DBMS_STATS.GATHER_SYSTEM_STATS(gathering_mode =>
'NOWORKLOAD');

PL/SQL procedure successfully completed.

SQL>
SQL> select sname,pname,pval1 from aux_stats$;

SNAME                          PNAME                             PVAL1
------------------------------ ------------------------------ --------
SYSSTATS_INFO                  STATUS
SYSSTATS_INFO                  DSTART
SYSSTATS_INFO                  DSTOP
SYSSTATS_INFO                  FLAGS                                 1
SYSSTATS_MAIN                  CPUSPEEDNW                     1520.786
SYSSTATS_MAIN                  IOSEEKTIM                        10.767
SYSSTATS_MAIN                  IOTFRSPEED                         4096
SYSSTATS_MAIN                  SREADTIM
SYSSTATS_MAIN                  MREADTIM
SYSSTATS_MAIN                  CPUSPEED
SYSSTATS_MAIN                  MBRC

SNAME                          PNAME
PVAL1
------------------------------ ------------------------------ --------
SYSSTATS_MAIN                  MAXTHR
SYSSTATS_MAIN                  SLAVETHR

13 rows selected.
```

```
SQL>
```

6. Before verifying your solution, you should flush the System Global Area (SGA) pools, such as the shared pool and the buffer cache. This is done to prevent the already loaded buffers or SQL plans from affecting the results.

```
SQL> @flush_sga
SQL>
SQL> set echo on
SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> alter system flush buffer_cache;

System altered.

SQL>
SQL>
```

7. Connected as the `JFV` user again, check your solution against the following query in the `select_with_sysstats.sql` script:

```
select /* With system stats */ count(*)
from t,z
where t.c=z.d;
```

What do you observe?

a. The optimizer can make a better decision because it was able to use meaningful system statistics. You can see that the execution time is now half the value it was previously, and the execution plan now includes CPU costing information.

```
SQL> connect jfv/jfv
Connected.
SQL> @select_with_sysstats
SQL> set timing on
SQL>
SQL> select /* With system stats */ count(*)
  2  from t,z
  3  where t.c=z.d;

  COUNT(*)
----------
    524288

Elapsed: 00:00:00.16
```

```
SQL>
SQL> set timing off
SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor);

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
SQL_ID  2x55txn3742by, child number 0
-------------------------------------
select /* With system stats */ count(*) from t,z where t.c=z.d

Plan hash value: 2407521827

-----------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time
|
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |       |       |   240 (100)|
|
|   1 |  SORT AGGREGATE     |      |     1 |     6 |            |
|

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
|*  2 |   HASH JOIN         |      |   524K|  3072K|   240   (4)|
00:00:04 |
|   3 |    INDEX FULL SCAN  | IZ   |   100 |   300 |     1   (0)|
00:00:01 |
|   4 |    TABLE ACCESS FULL| T    |   524K|  1536K|   235   (2)|
00:00:04 |
-----------------------------------------------------------------------

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


   2 - access("T"."C"="Z"."D")



21 rows selected.

SQL>
SQL> col operations     format a20
SQL> col object_name format a11
SQL> col options        format a15
```

```
SQL> col cost_cpu_io format a30
SQL>
SQL>
SQL> select operation operations, object_name, options,
  2         cost||' -- '||cpu_cost||' -- '||io_cost cost_cpu_io
  3  from (select * from v$sql_plan
  4  where address in (select address from v$sql
  5                    where sql_text like '%system stats%'
  6                    and sql_text not like '%connect%'));

OPERATIONS           OBJECT_NAME OPTIONS       COST_CPU_IO
-------------------- ----------- ------------- ---------------------
SELECT STATEMENT                               240 --  --
SORT                             AGGREGATE      --  --
HASH JOIN                                      240 -- 147046197 -- 232
INDEX                IZ          FULL SCAN     1 -- 27121 -- 1
TABLE ACCESS         T           FULL          235 -- 84867339 -- 231

SQL>
```

8. Exit from your SQL*Plus session and clean up your environment for this lab by executing the `systats_cleanup.sh` script.

```
SQL> exit

…

$ ./sysstats_cleanup.sh

…

SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> exit;

…

$

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

#!/bin/bash
```

```
cd /home/oracle/solutions/System_Stats

sqlplus / as sysdba @sysstats_cleanup.sql

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

set echo on

alter system flush shared_pool;

exit;
```

Practice 10-2: Automatic Statistics Gathering

In this practice, you manipulate object statistics to see the effect on the execution plans of your SQL statements. **Note:** All scripts needed for this lab can be found in your `$HOME/solutions/Automatic_Gather_Stats` directory.

1. The environment for this lab has been set up with the `ags_setup.sh` script. This script created a user called `AGS` that you use throughout this lab. The script also created a table called `EMP` and an index. The script is listed here:

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

#!/bin/bash

cd /home/oracle/solutions/Automatic_Gather_Stats

sqlplus / as sysdba @ags_setup.sql

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

set echo on

drop user ags cascade;

create user ags identified by ags;

grant dba to ags;

connect ags/ags

drop table emp purge;

create table emp
(
empno   number,
ename   varchar2(20),
phone   varchar2(20),
deptno  number
);


insert into emp
  with tdata as
       (select rownum empno
         from all_objects
         where rownum <= 1000)
  select rownum,
         dbms_random.string ('u', 20),
```

```
         dbms_random.string ('u', 20),
         case
                when rownum/100000 <= 0.001 then mod(rownum, 10)
                else 10
         end
    from tdata a, tdata b
   where rownum <= 100000;

commit;

create index emp_i1 on emp(deptno);

exit;
```

2. Execute the `ags_start.sql` script in SQL*Plus from the `$HOME/solutions/Automatic_Gather_Stats` directory.

```
$ cd $HOME/solution/Automatic_Gather_Stats
$ ./ags_start.sh

…
SQL>
SQL> connect / as sysdba
Connected…
SQL>
SQL> alter system flush shared_pool;

System altered.

SQL>
SQL> --
SQL> -- Turn off AUTOTASK
SQL> --
SQL>
SQL> alter system set "_enable_automatic_maintenance"=0;

System altered.

SQL>
SQL> exec dbms_stats.delete_schema_stats('AGS');

PL/SQL procedure successfully completed.

SQL>
```

```
SQL> exit;
Disconnected …
$
```

3. Connect as the ags user under a SQL*Plus session.

```
$ sqlplus ags/ags

…

SQL>
```

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

   Name: ags_connection

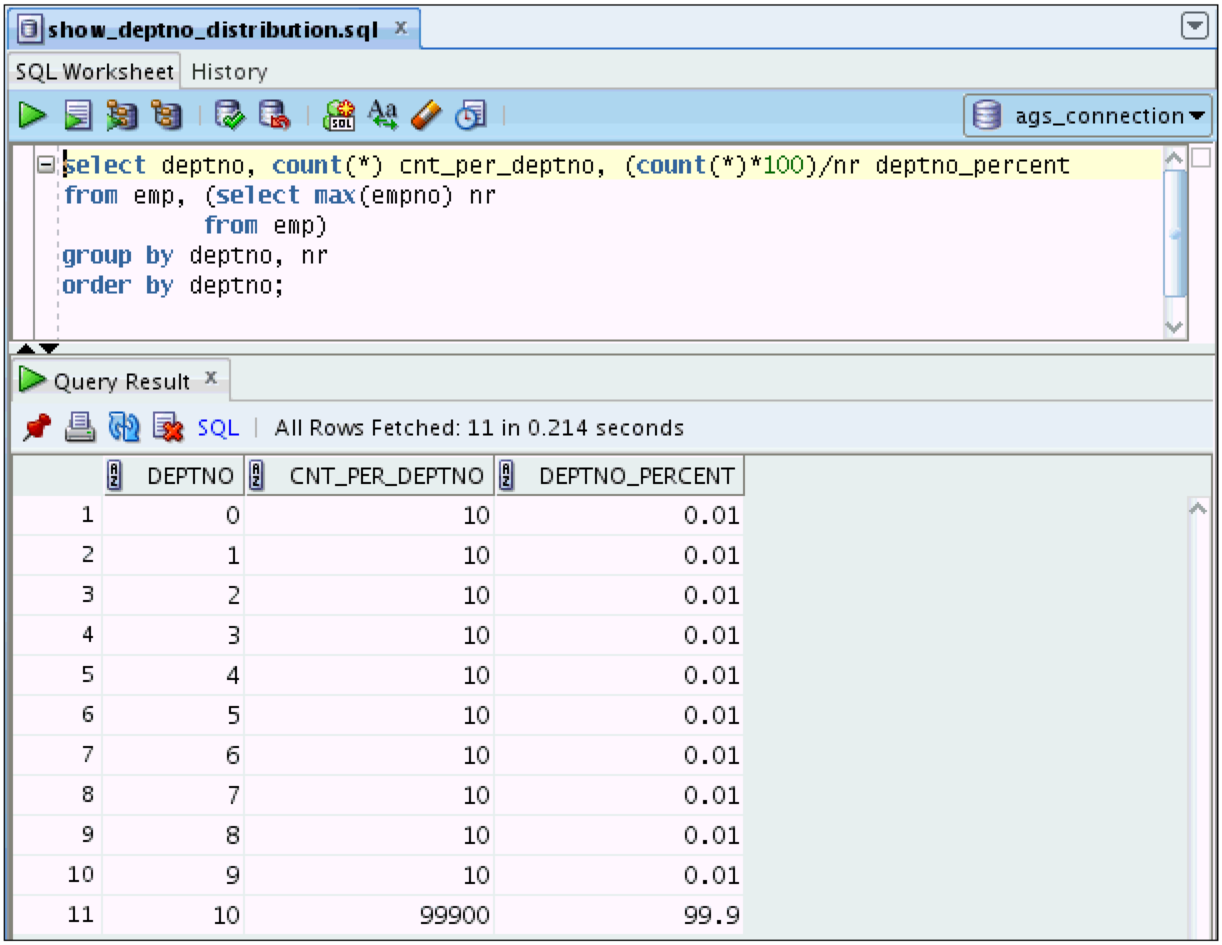
   User: ags

   Password: ags

   Service name: orcl.example.com

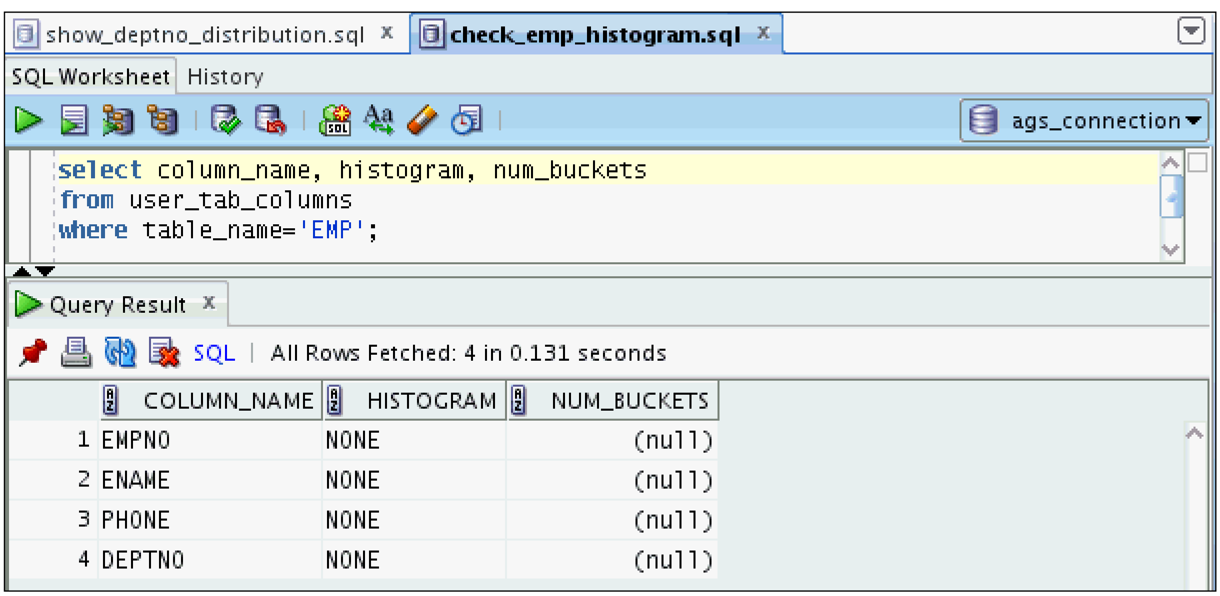
5. Using ags_connection, determine the distribution of the deptno values found in the EMP table. What do you observe?

   a. Open and execute the show_deptno_distribution.sql file in the $HOME/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.

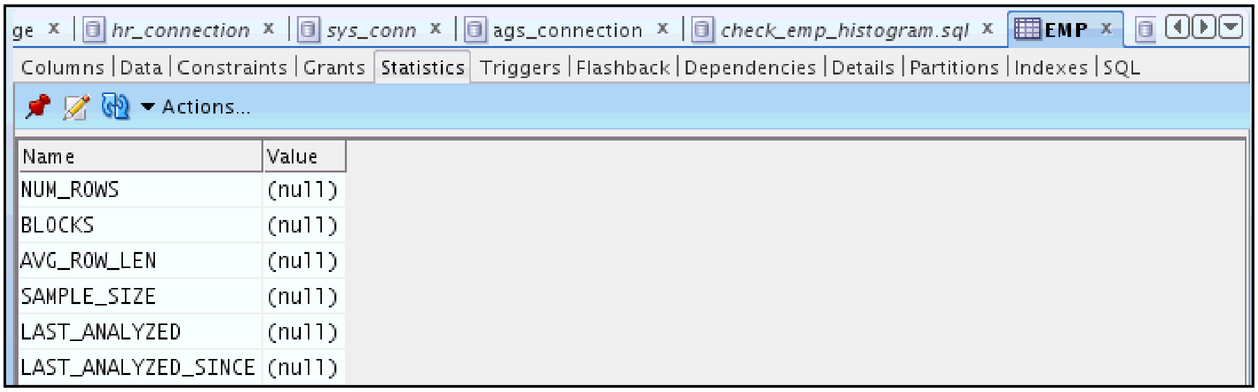


6. Determine if there are histograms available on any column of the `EMP` table. What do you observe?

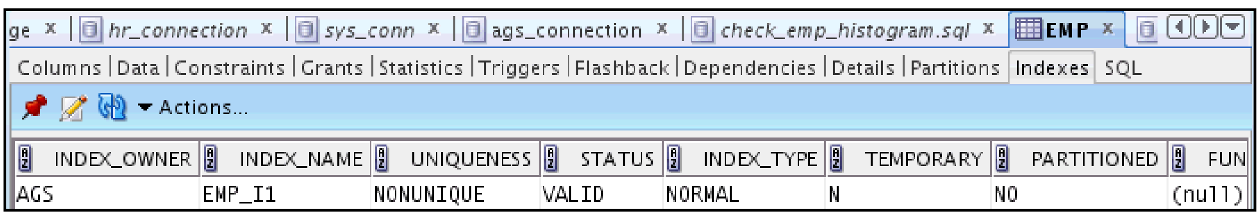
   a. Open the `check_emp_histogram.sql` file, and execute it in SQL Developer. Currently, there are no histograms defined on any column of the `EMP` table.



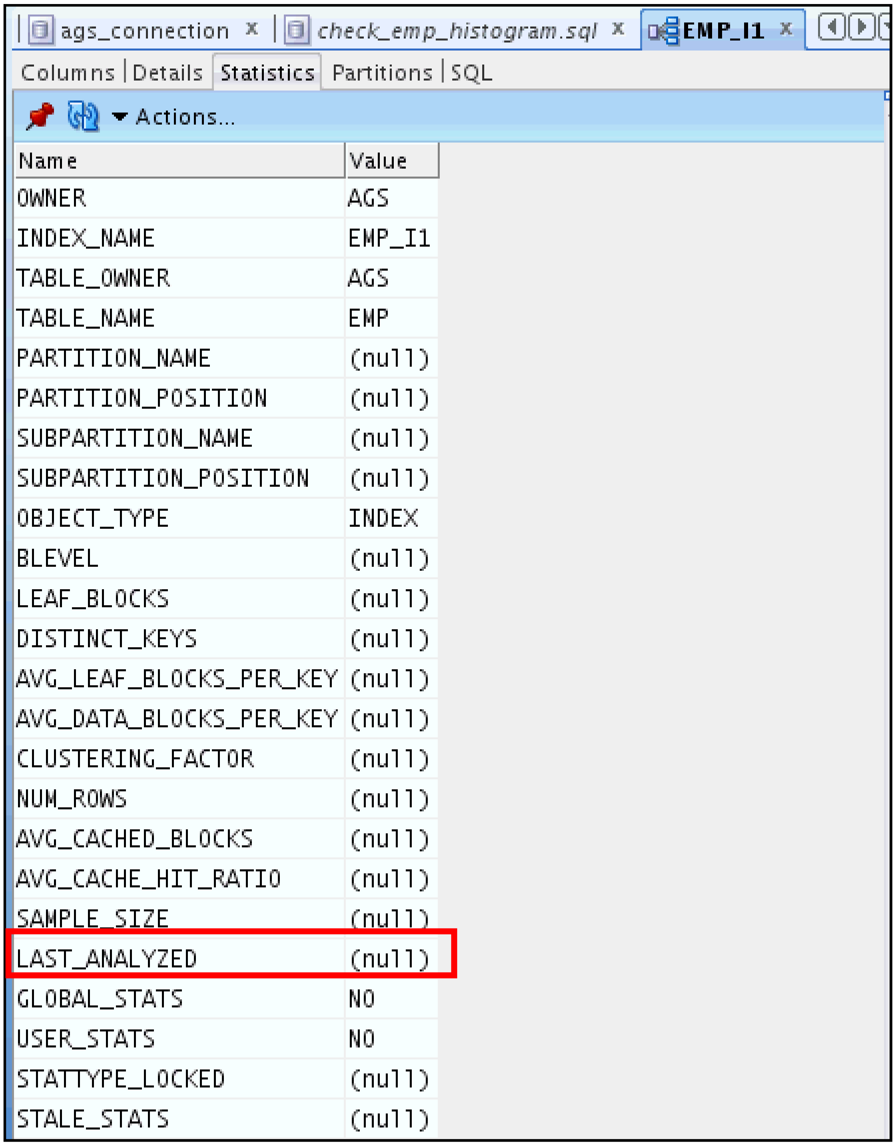
7. Determine if you currently have statistics gathered on your `EMP` table. What do you observe?
   a. In the SQL Developer navigator pane, click the Connections tab, then expand `ags_connection`, and then the Tables node and the `EMP` table node. Select the `EMP` table node.
   b. On the EMP tab, click 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 created on top of the `EMP` table. What do you observe?
   a. Click the Indexes subtab of the `EMP` pane.



   b. In the Connections navigator pane, under `ags_connection`, expand the indexes node, then select the EMP_I1 index. In the EMP_I1 pane, click the statistics subtab. Currently, `EMP_I1` has no statistics gathered.



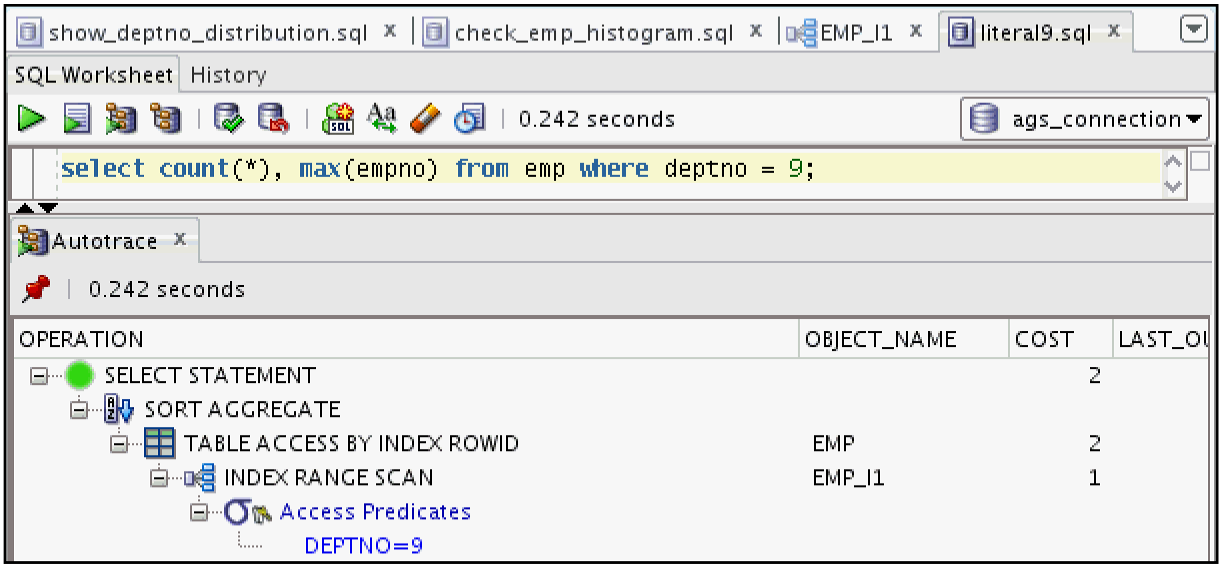
Name	Value
OWNER	AGS
INDEX_NAME	EMP_I1
TABLE_OWNER	AGS
TABLE_NAME	EMP
PARTITION_NAME	(null)
PARTITION_POSITION	(null)
SUBPARTITION_NAME	(null)
SUBPARTITION_POSITION	(null)
OBJECT_TYPE	INDEX
BLEVEL	(null)
LEAF_BLOCKS	(null)
DISTINCT_KEYS	(null)
AVG_LEAF_BLOCKS_PER_KEY	(null)
AVG_DATA_BLOCKS_PER_KEY	(null)
CLUSTERING_FACTOR	(null)
NUM_ROWS	(null)
AVG_CACHED_BLOCKS	(null)
AVG_CACHE_HIT_RATIO	(null)
SAMPLE_SIZE	(null)
LAST_ANALYZED	(null)
GLOBAL_STATS	NO
USER_STATS	NO
STATTYPE_LOCKED	(null)
STALE_STATS	(null)

9. Autotrace the following two statements and determine their execution plans:

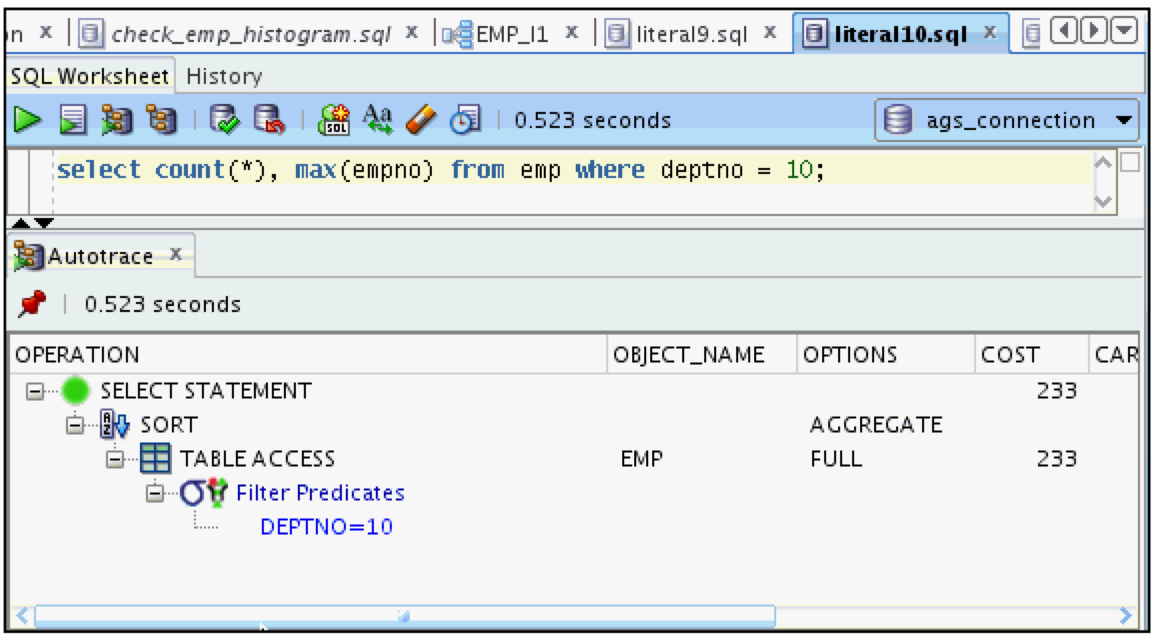
```
select count(*), max(empno) from emp where deptno = 9;
select count(*), max(empno) from emp where deptno = 10;
```

These statements are in the `literal9.sql` and `literal10.sql` files. What do you observe and why?

a. Open the `literal9.sql` file and Autotrace in `ags_connection`. You see that for `deptno = 9`, the optimizer decided to use the index.



b. Open the `literal10.sql` file and Autotrace in `ags_connection`. Notice that for `deptno = 10`, the optimizer decided to do `TABLE ACCESS FULL`. The optimizer determined the correct plans in both cases. This is because dynamic sampling was used because there were no statistics gathered on either object.

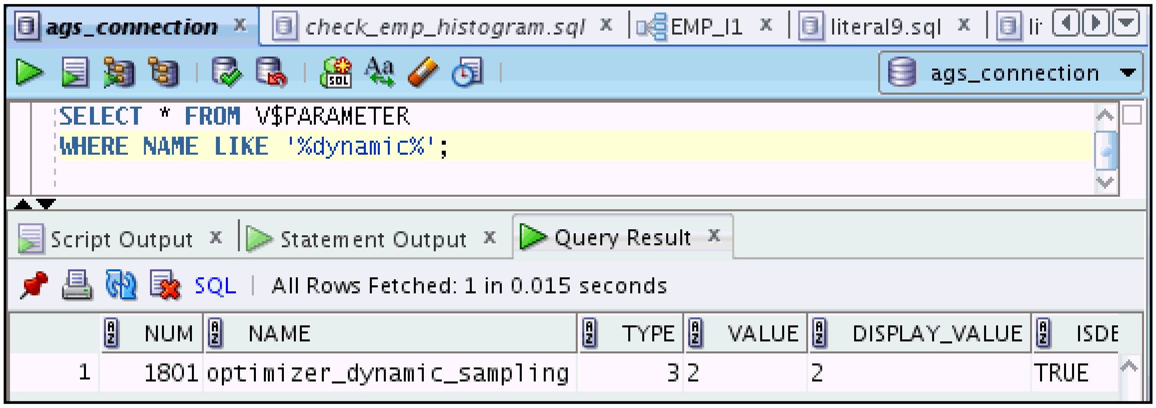


10. Confirm your assumption from the previous step.

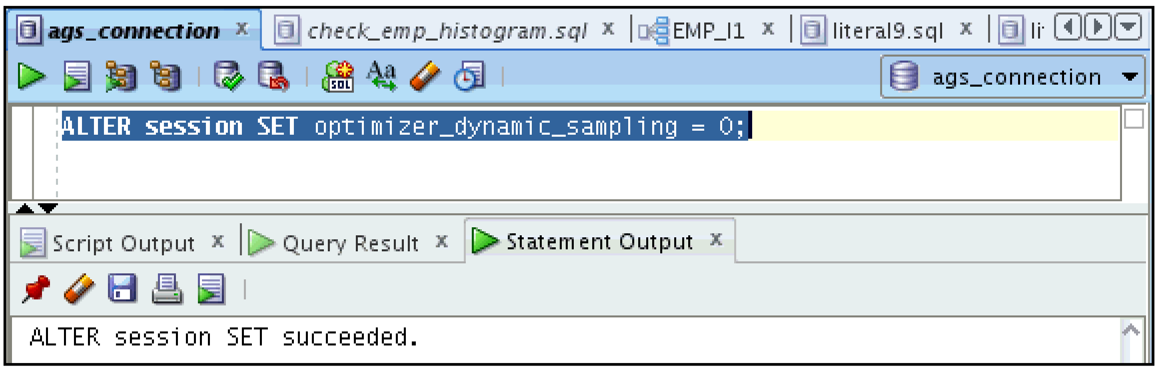
a. On the `ags_connection` tab, enter the following query and execute:

```
SELECT * FROM v$parameter WHERE NAME LIKE '%dynamic%';
```

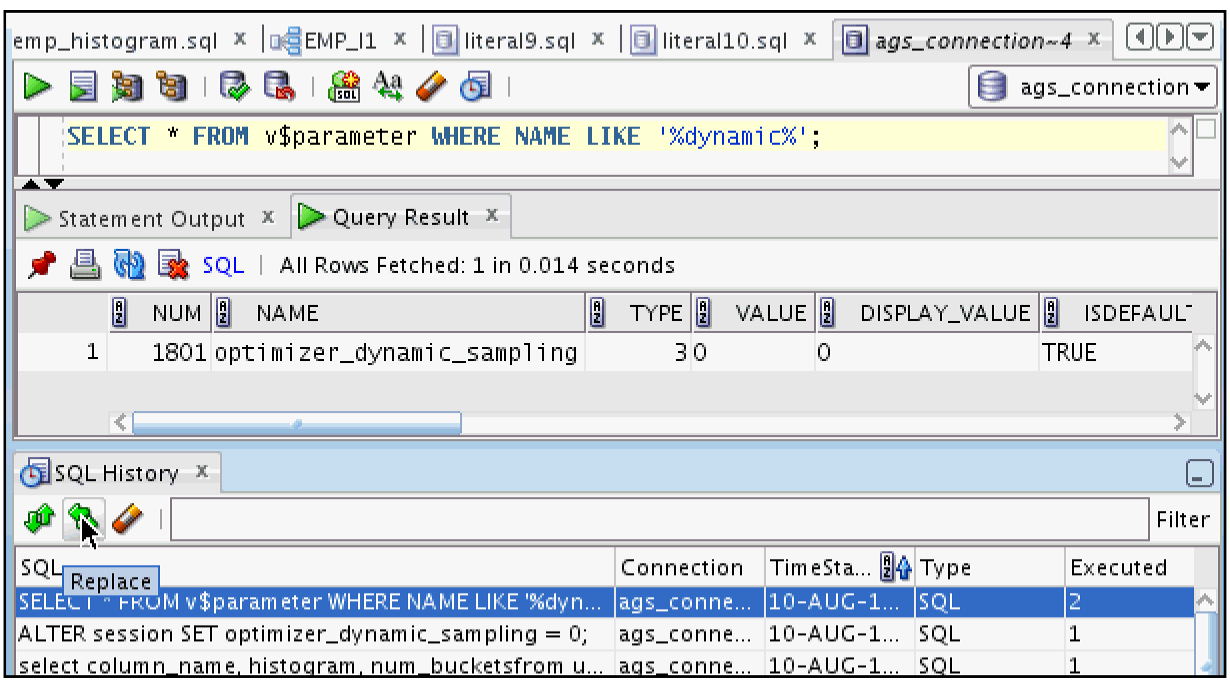
Note: In SQL Developer, there is only one session for a connection even though there may be multiple windows using that session.



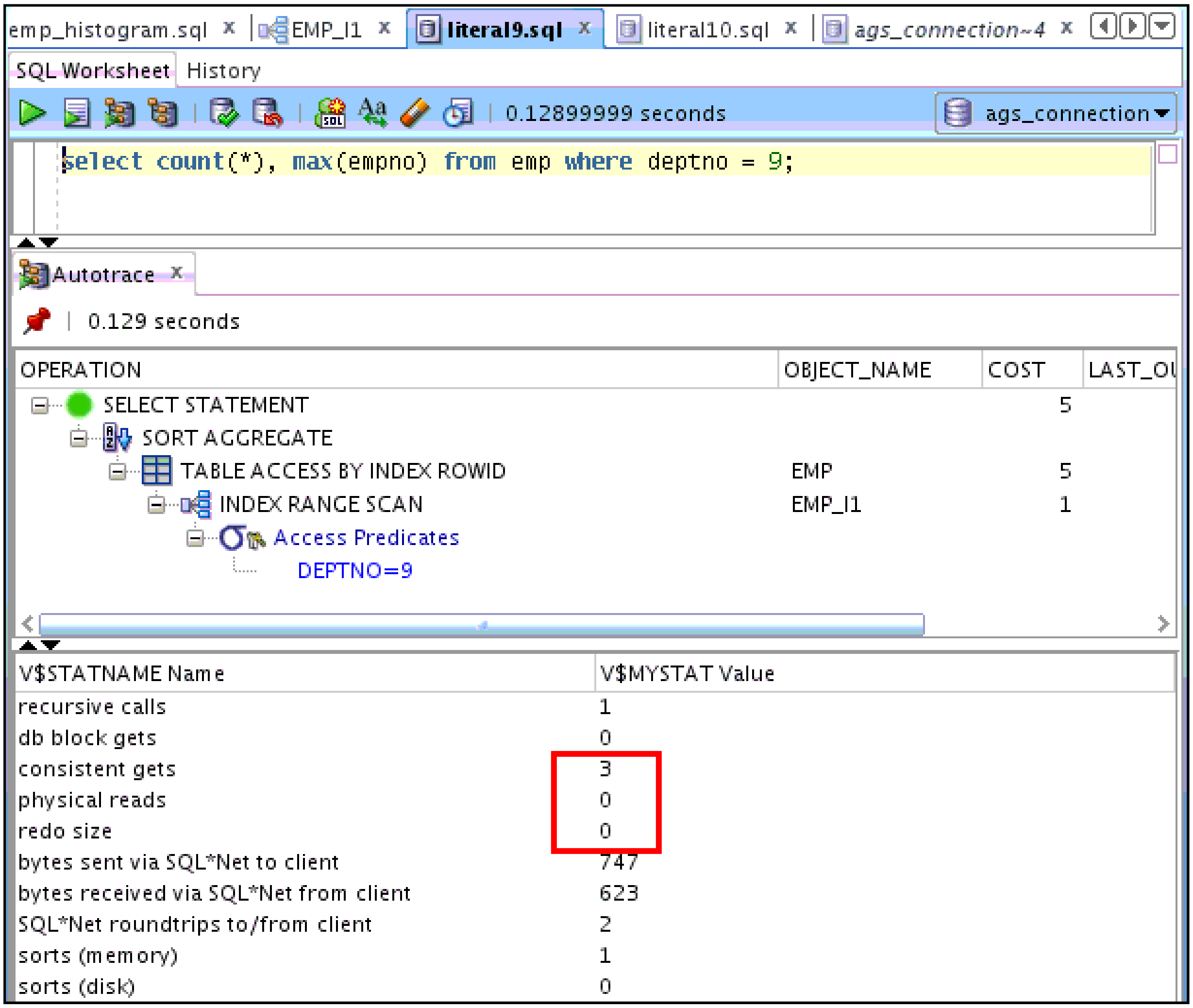
11. Turn off dynamic sampling.
    a. Set the parameter for your session to 0. Use the following command.
       `ALTER session SET optimizer_dynamic_sampling = 0;`

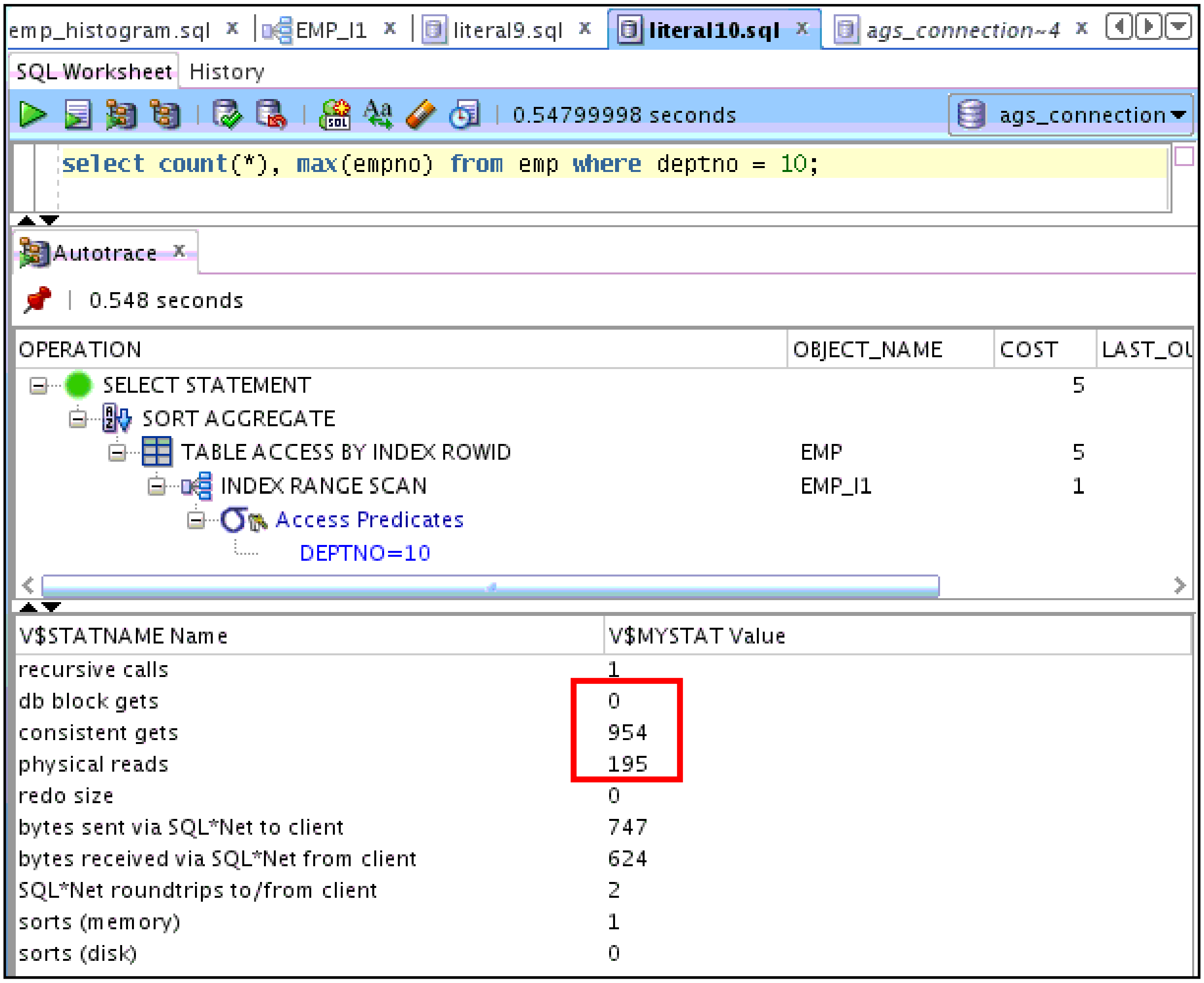


    b. Confirm that the parameter has changed. Open `SQL_HISTORY`, select the `select * from V$parameter`... command, click Replace and execute.



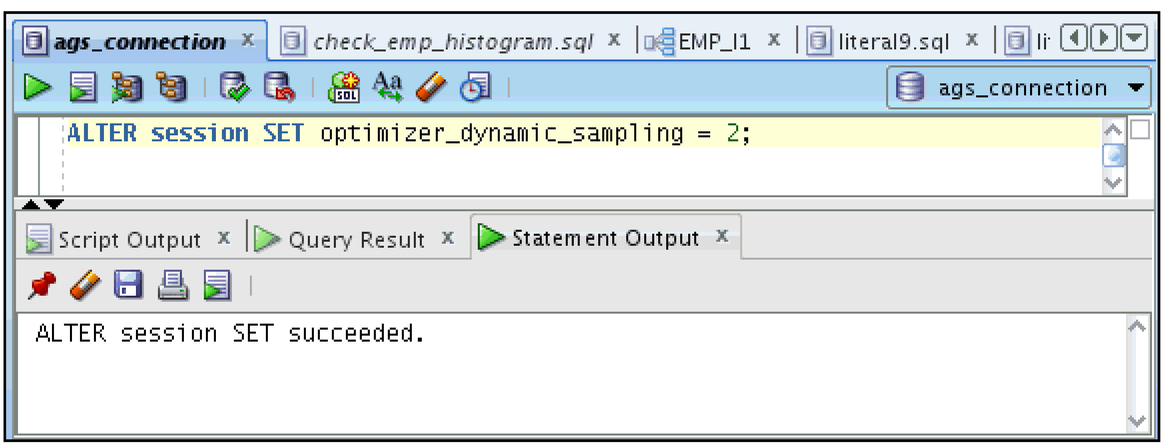
12. Autotrace both `literal9.sql` and `literal10.sql` again. What do you observe and why?
    a. Because dynamic sampling is not used, the optimizer cannot use any statistics. It uses default statistics that are not a good choice for the second statement. The cost is estimated to be the same, but when you compare the actual statistics you can see the difference.





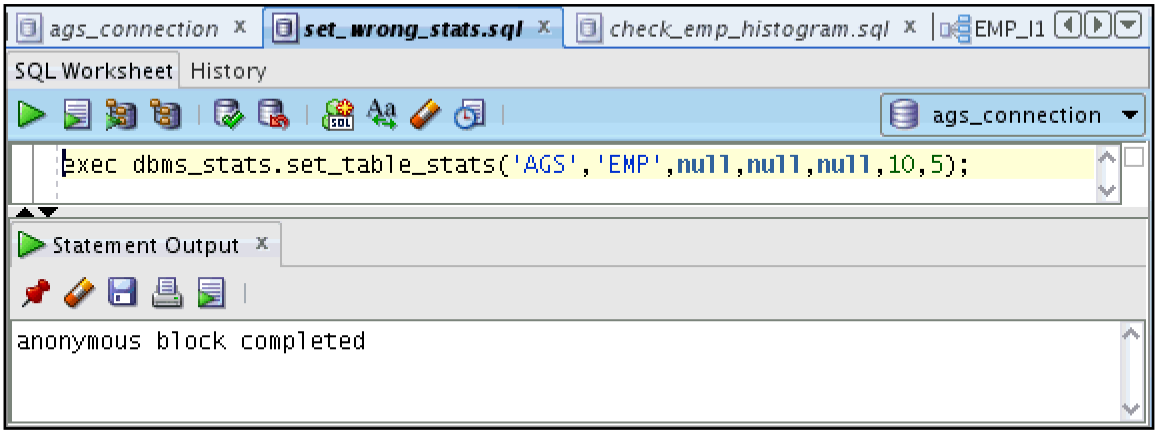
13. Reset dynamic sampling as it was at the beginning of this lab. Use the command:

```
ALTER session SET optimizer_dynamic_sampling = 2;
```



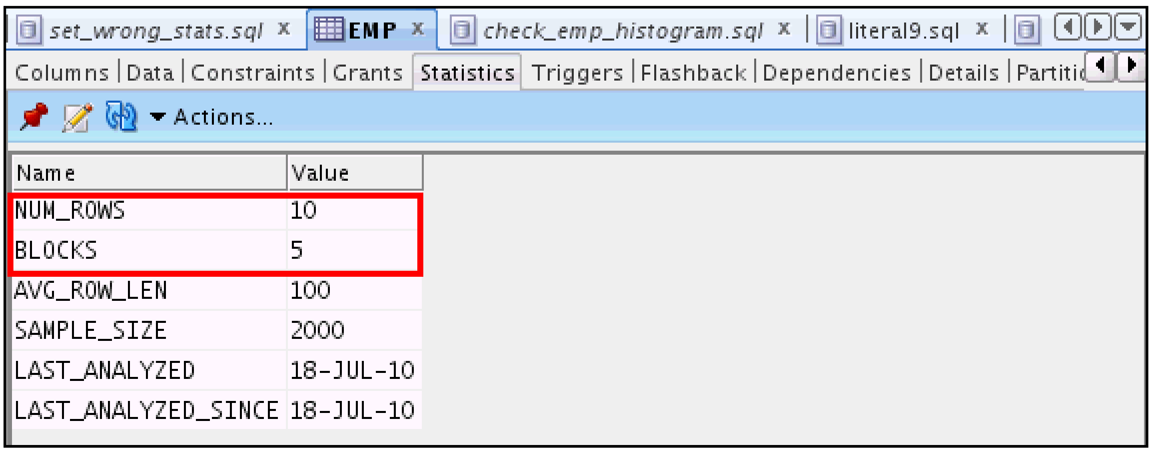
14. Set the following wrong statistic values for your `EMP` table:
    - Set its number of rows to 10.
    - Set its number of blocks to 5.

    Open the `set_wrong_stats.sql` file and execute.



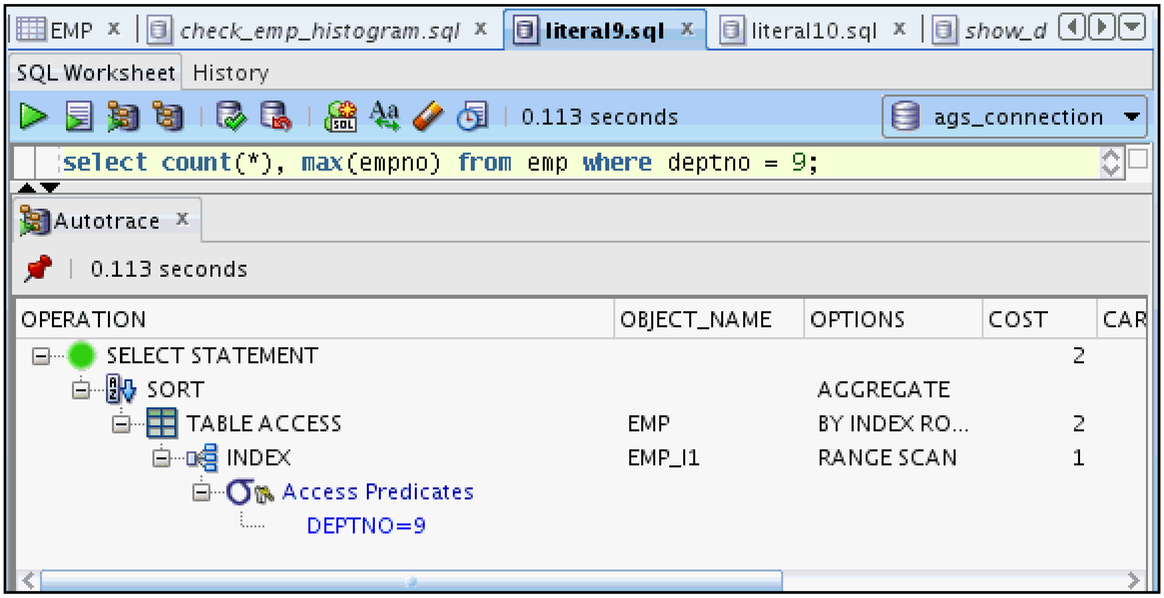
15. Verify that you modified the statistics of the `EMP` table correctly.

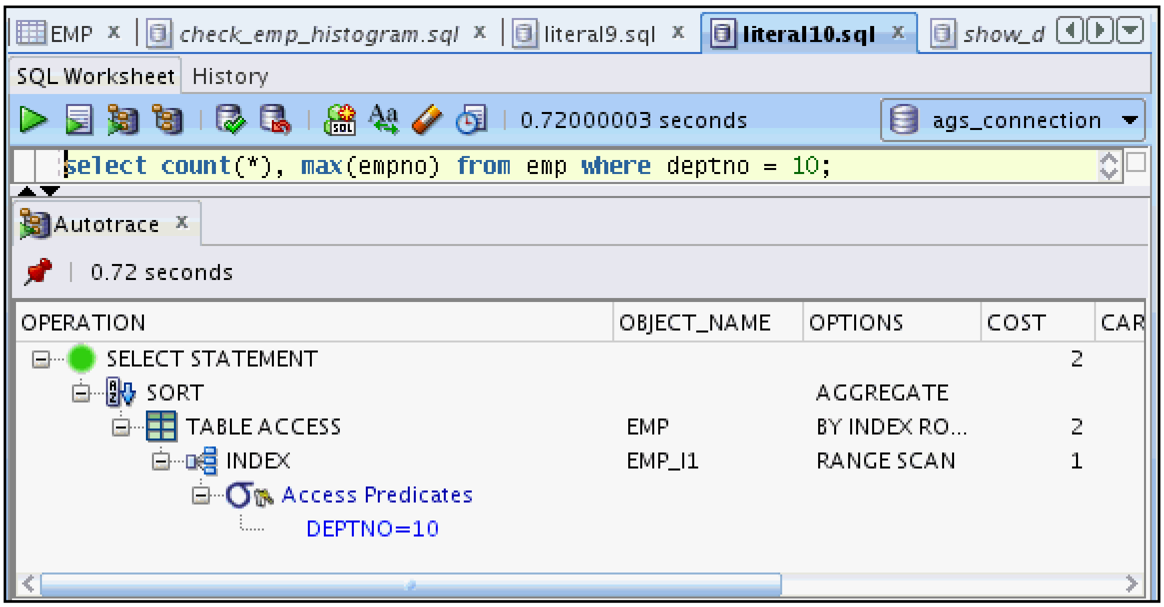
    a. In the navigator pane, select the EMP table under `ags_connection`. On the EMP tab, click the Statistics subtab.



16. Autotrace both `literal9.sql` and `literal10.sql` again. What do you observe and why?

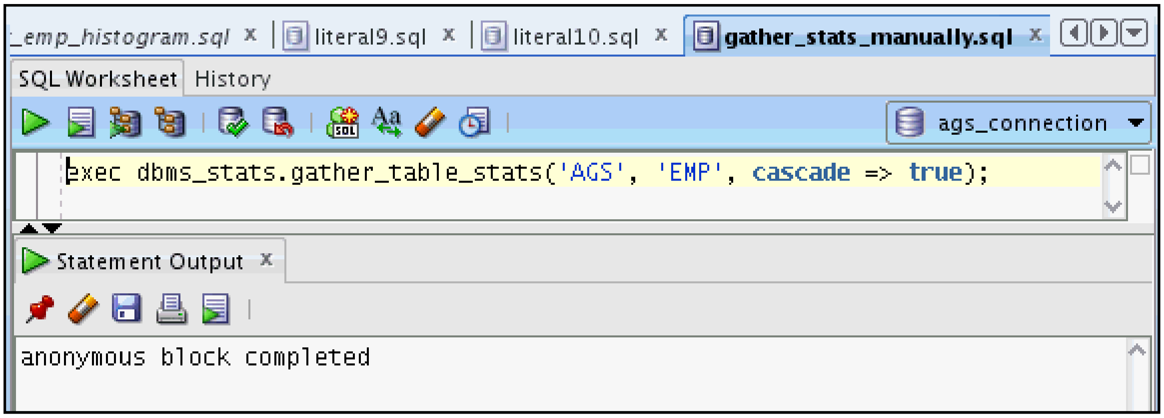
    a. Because there are statistics defined on the `EMP` table, the optimizer uses them, and not dynamic sampling. However, because the statistics are incorrect, the generated plans are also incorrect, at least for the second statement. This is noticeable in the execution time.



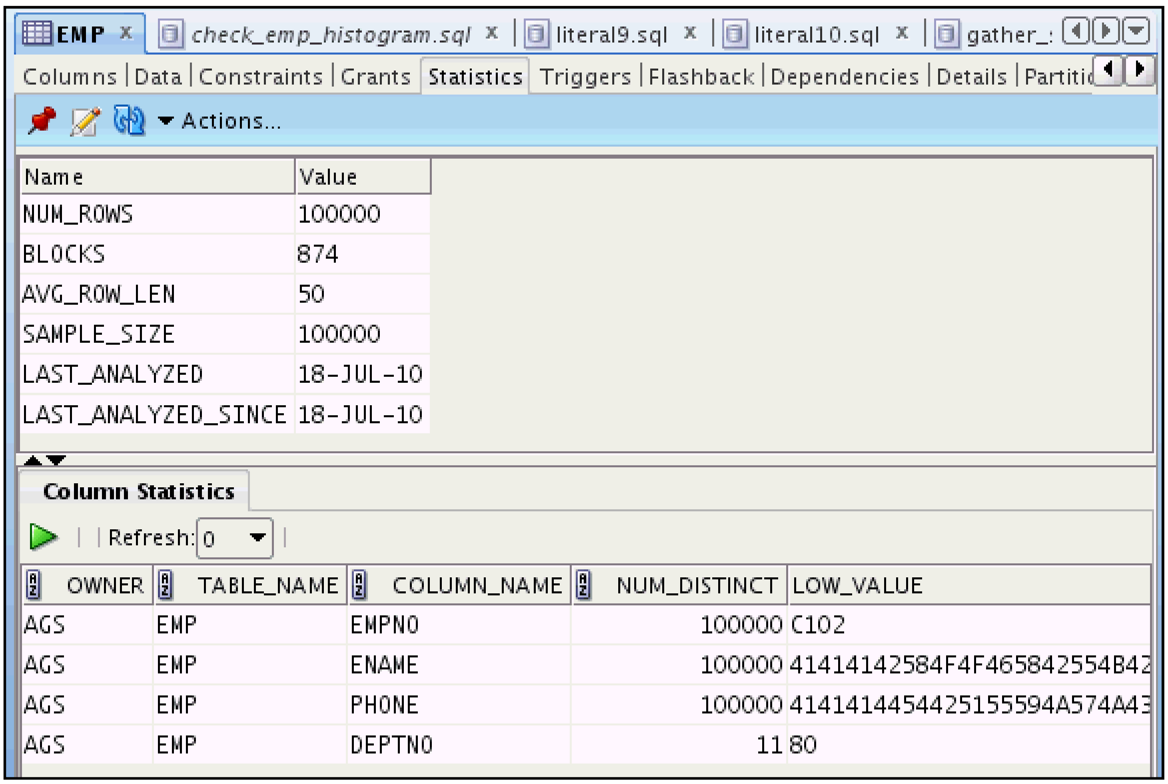


17. Make sure that you manually gather statistics on your `EMP` table and its corresponding index. Enter the following or execute the `gather_stats_manually.sql` script:

```
exec dbms_stats.gather_table_stats('AGS', 'EMP', cascade =>
true);
```

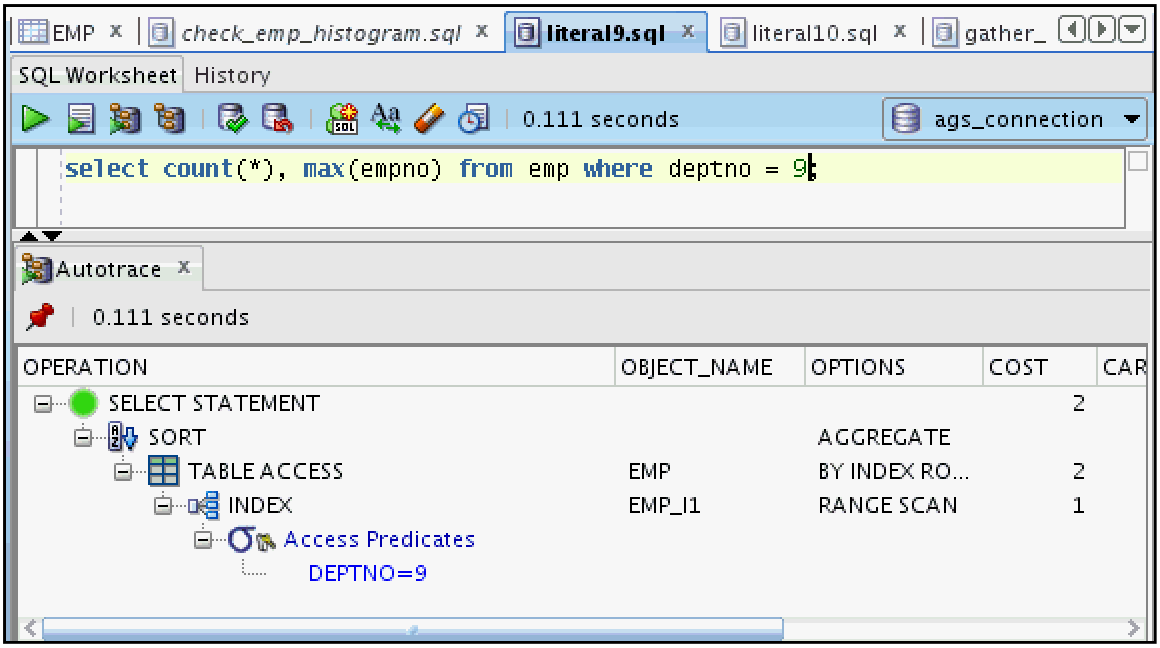


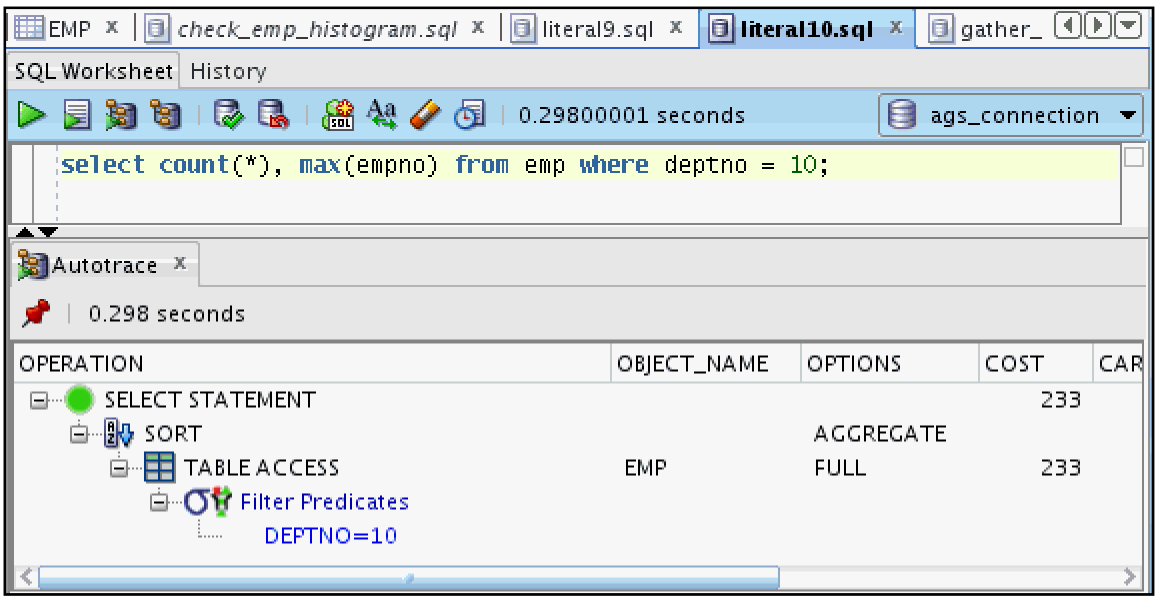
18. Make sure that statistics were gathered on your objects. Select the EMP table in the navigator pane. Click the Statistics subtab on the EMP tab. Click the Refresh button to ensure that the latest statistics are displayed. Notice that the Column Statistics are populated.



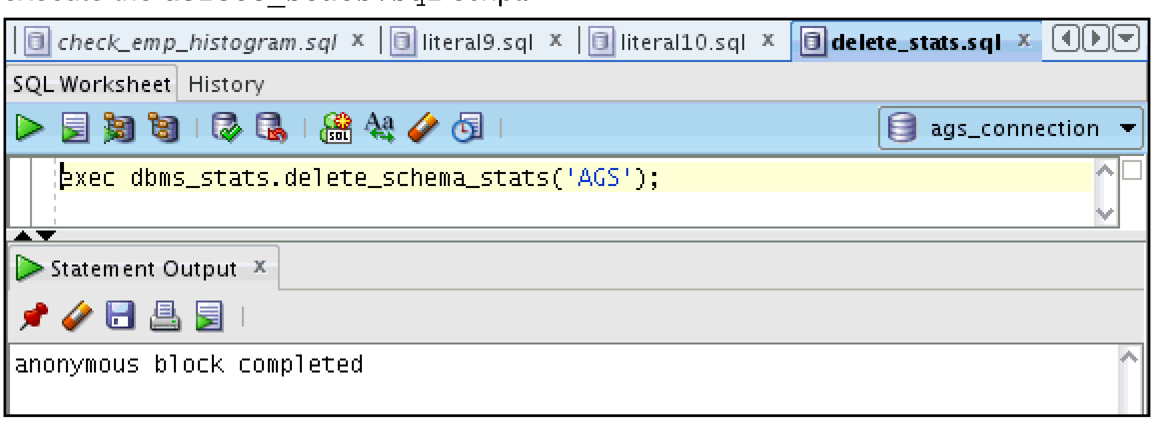
19. Autotrace both `literal9.sql` and `literal10.sql` again. What do you observe and why?

   a. Because statistics were correctly gathered on both objects, the optimizer is able to use the correct execution plans for both statements.



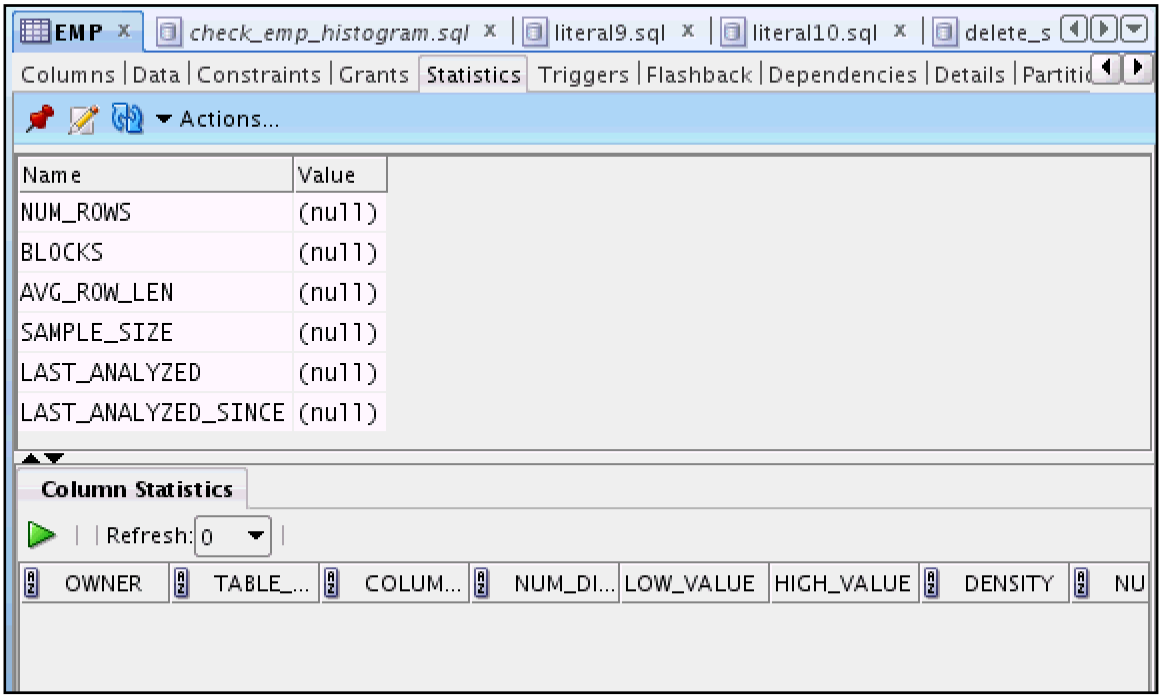


20. Delete all statistics previously generated on the `EMP` table and the `EMP_I1` index. Open and execute the `delete_stats.sql` script.

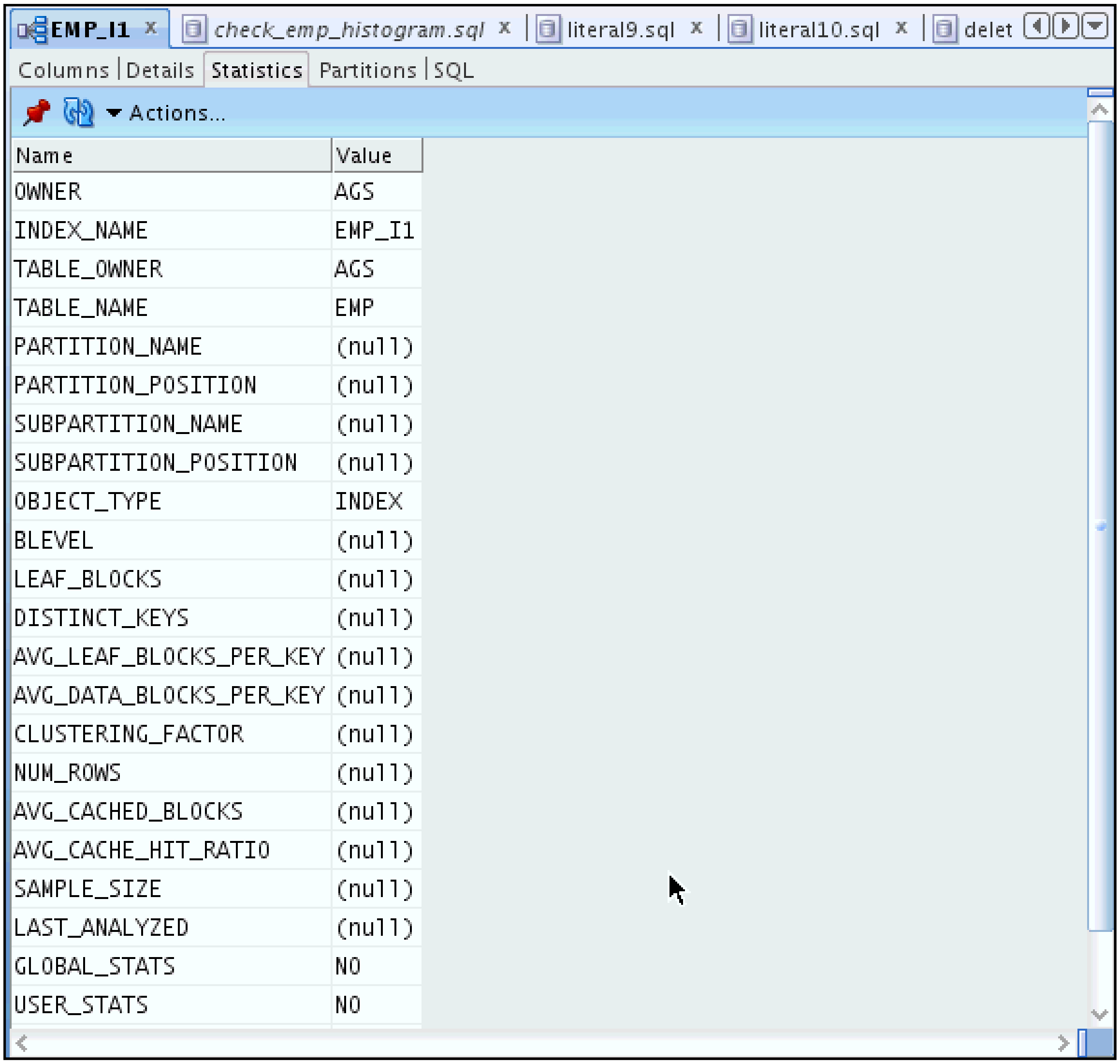


21. Verify that you no longer have statistics gathered on both objects.

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



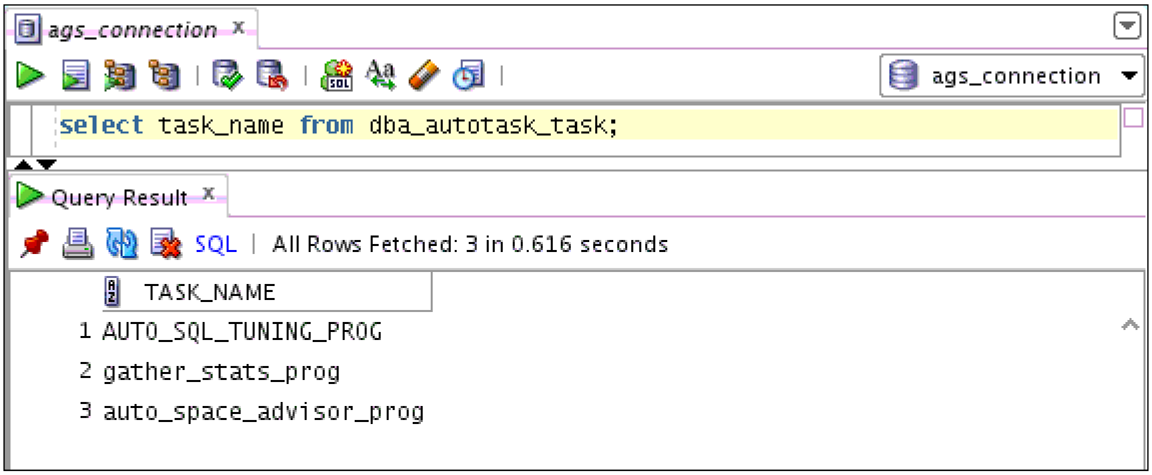
   b. In the navigator pane, select the EMP_I1 index, click the Statistics subtab on the EMP tab and refresh.



Name	Value
OWNER	AGS
INDEX_NAME	EMP_I1
TABLE_OWNER	AGS
TABLE_NAME	EMP
PARTITION_NAME	(null)
PARTITION_POSITION	(null)
SUBPARTITION_NAME	(null)
SUBPARTITION_POSITION	(null)
OBJECT_TYPE	INDEX
BLEVEL	(null)
LEAF_BLOCKS	(null)
DISTINCT_KEYS	(null)
AVG_LEAF_BLOCKS_PER_KEY	(null)
AVG_DATA_BLOCKS_PER_KEY	(null)
CLUSTERING_FACTOR	(null)
NUM_ROWS	(null)
AVG_CACHED_BLOCKS	(null)
AVG_CACHE_HIT_RATIO	(null)
SAMPLE_SIZE	(null)
LAST_ANALYZED	(null)
GLOBAL_STATS	NO
USER_STATS	NO

22. How would you determine the list of automated tasks that exist on your database?
    a. You can use Enterprise Manager Database Control by navigating to the Automated Maintenance Tasks page (Home > Server> Automated Maintenance Tasks). On the Automated Maintenance Tasks page, you can see the three automated tasks implemented by default on your database.
    b. Another possibility is to use the `DBA_AUTOTASK_TASK` table as shown by the following statement:

```
select task_name from dba_autotask_task;
```



23. You now want to observe the effects of the automatic statistics-gathering feature of your database. However, you do not want to wait until the database automatically opens the next maintenance window. From SQL Developer, open and execute the `run_ags_pl.sql` script. This script forces the execution of the automatic statistics-gathering task.

    a. If you do not already have a `sys_connection`, create a connection as a `sysdba` user.

       Name: `sys_connection`

       Username: `sys`

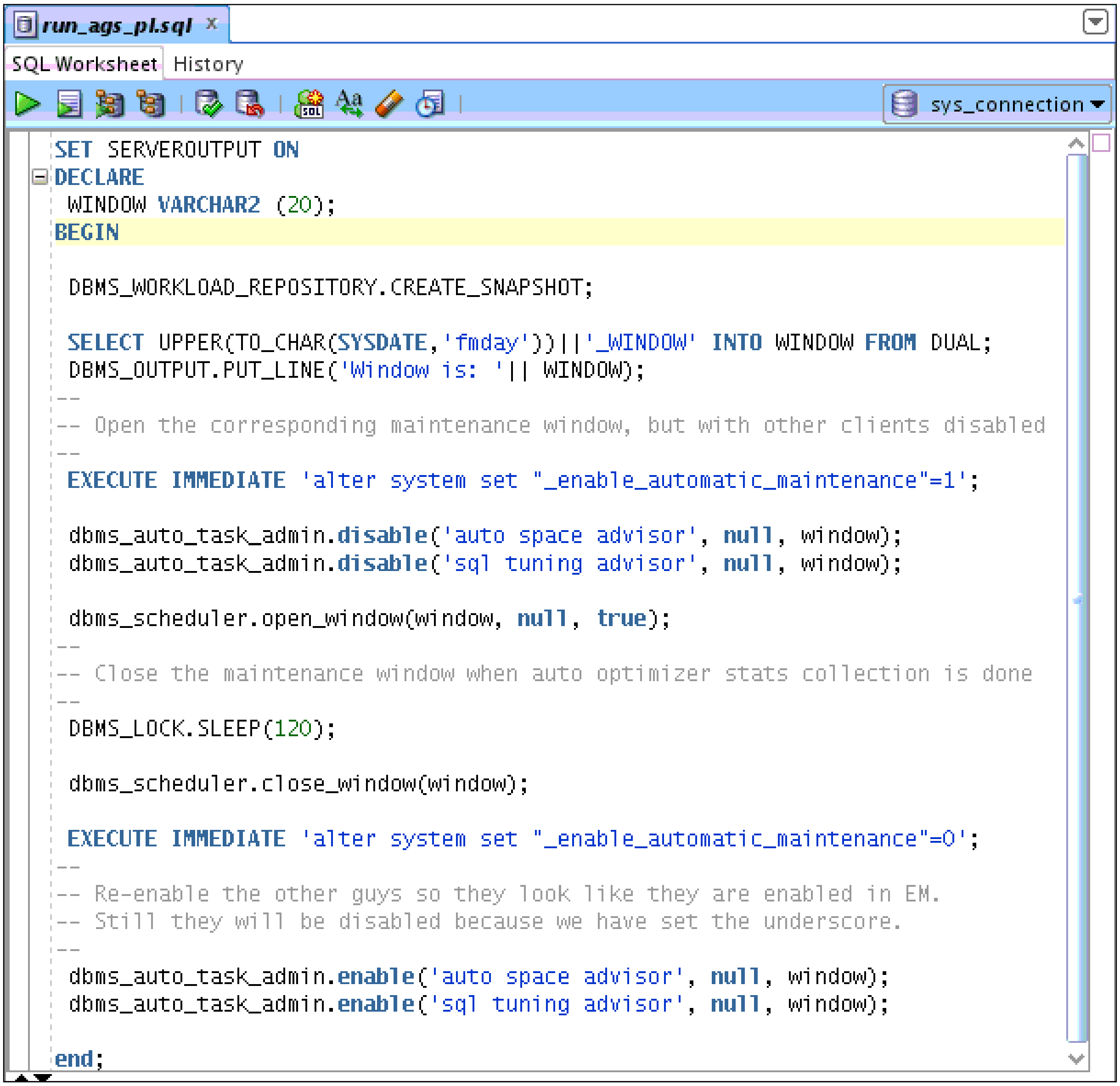
       Password: `oracle_4U`

       Role: `sysdba`

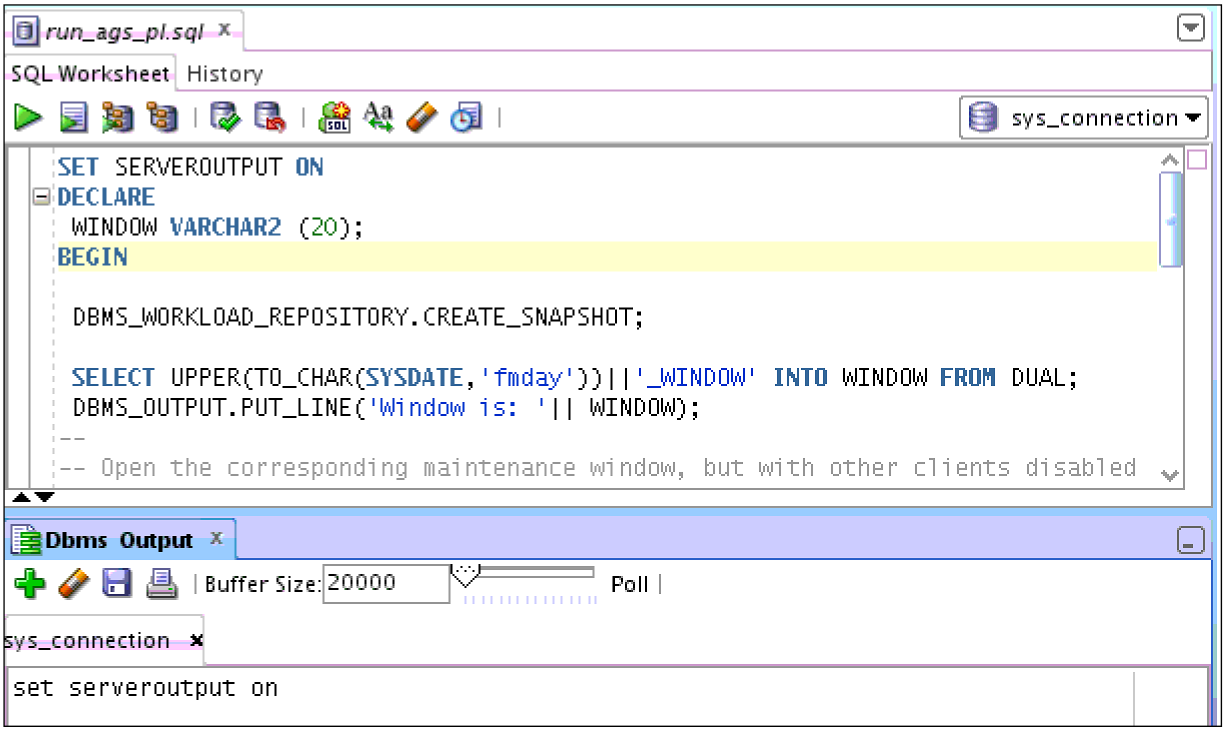
       SID: `orcl`

    b. Test and save the connection.

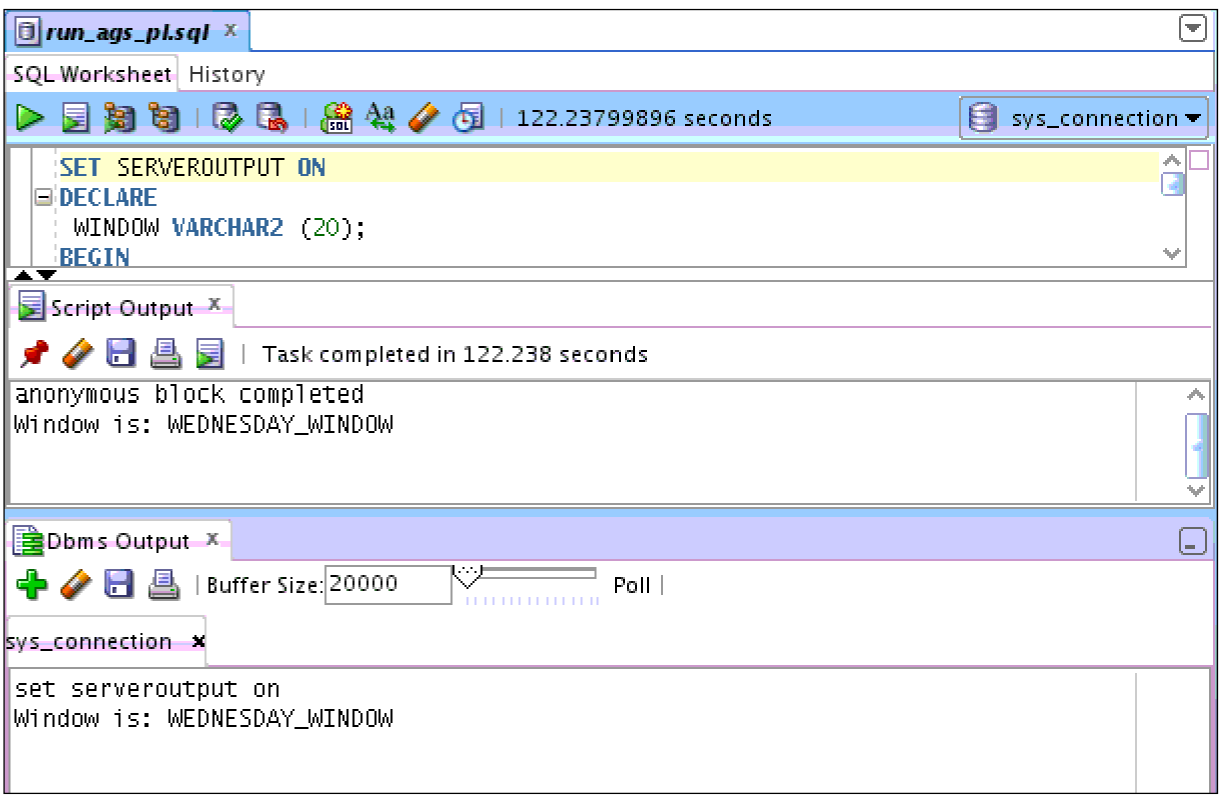
    c. Open the `run_ags_pl.sql` file.



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



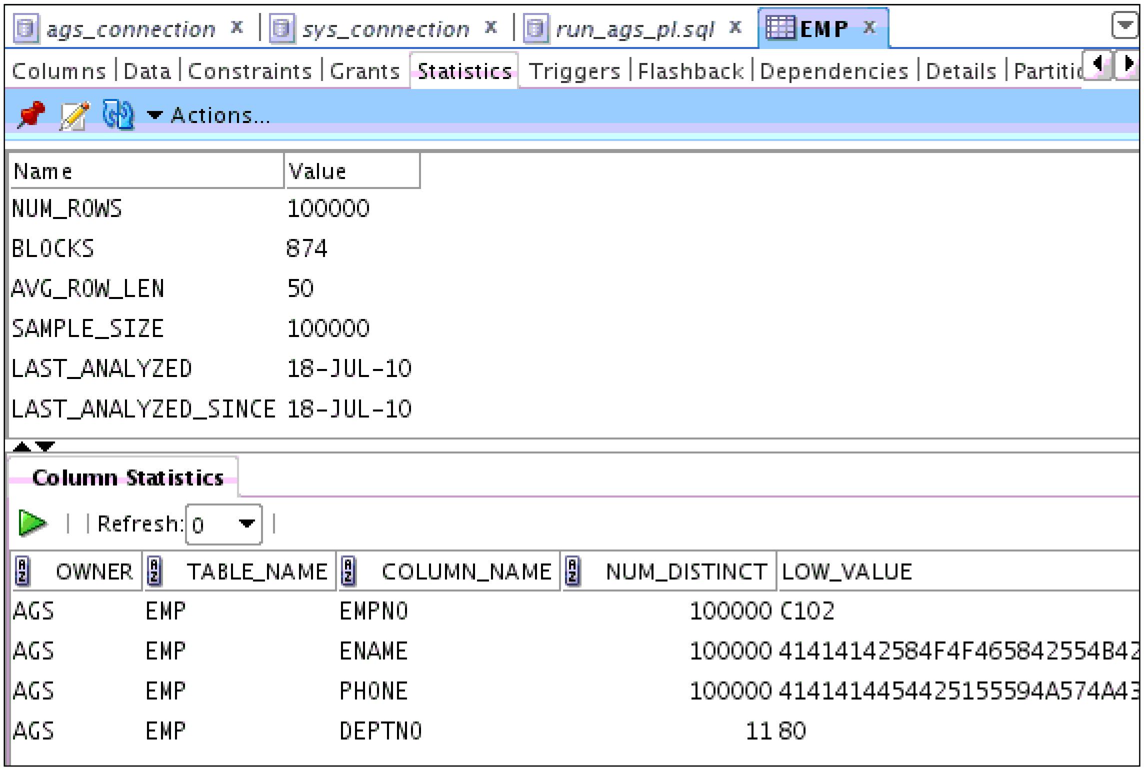
e. Execute the script (F5) as `sys`, use `sys_connection`. This script takes two minutes to run.



f. Close the Dbms Output pane.

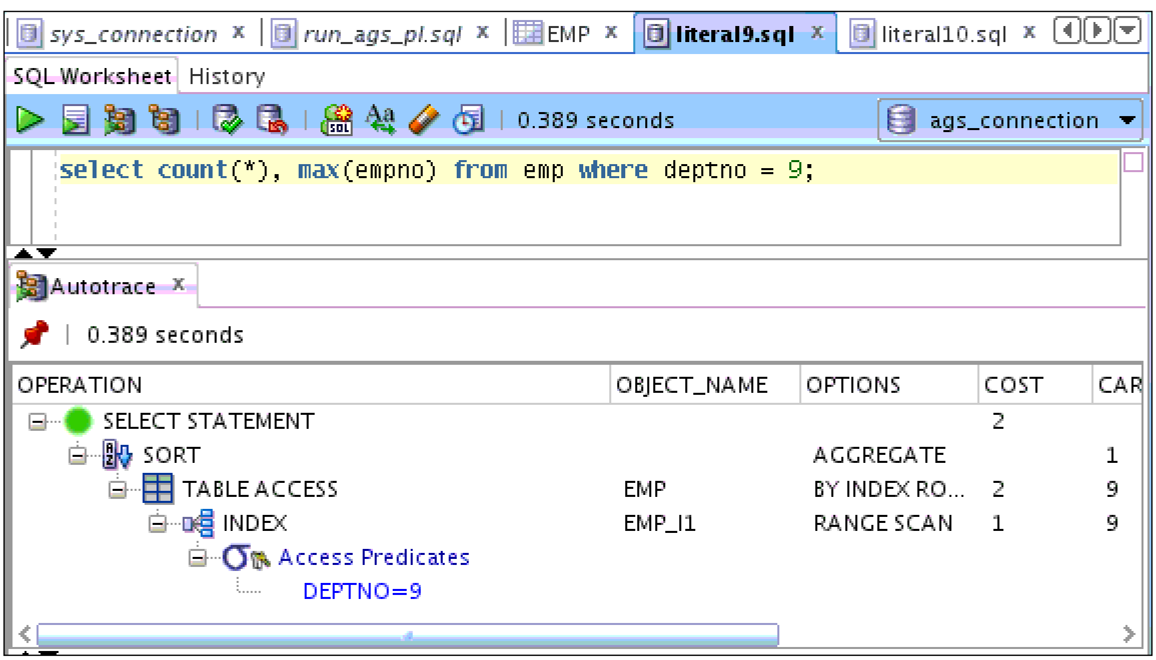
24. Navigate to the EMP table and view the statistics. Be sure to refresh the view. What do you observe and why?

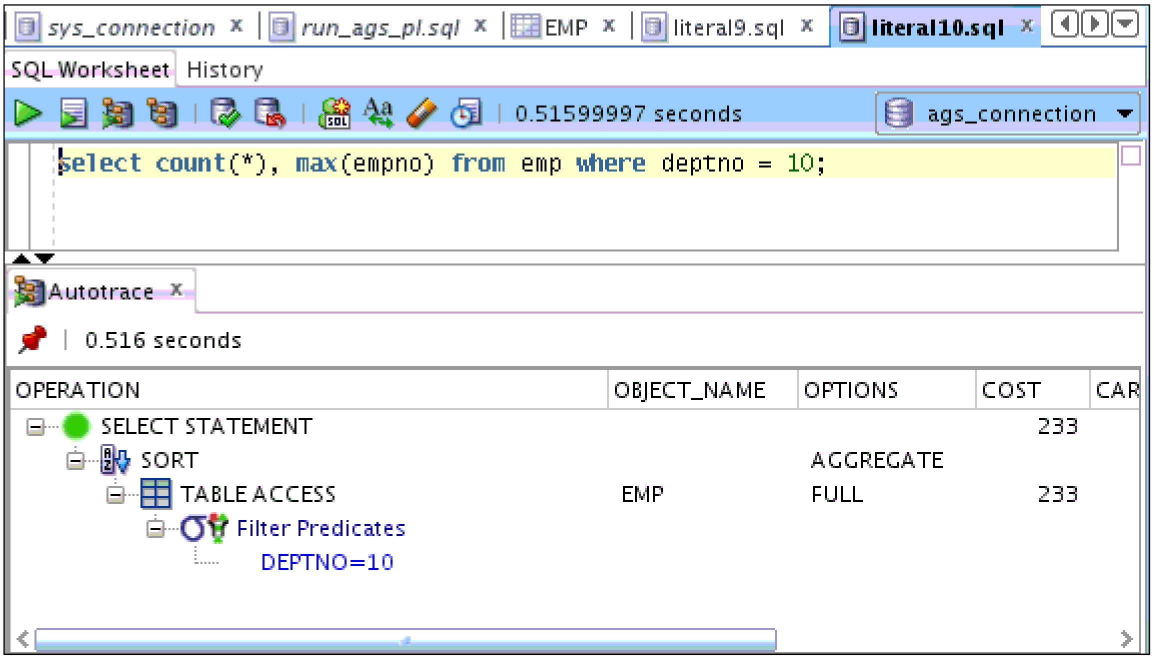
The statistics were automatically gathered by the database during the maintenance window. You can also see this directly from the Automated Maintenance Tasks page in Enterprise Manager. The important thing is that the database automatically gathered the right statistics and histograms. Depending on your environment, you may see different sample sizes.



25. Autotrace `literal9.sql` and `literal10.sql` again. What do you observe and why?

a. The optimizer can make the right decisions for both statements. This is because of the statistics that were automatically gathered by the database previously.





26. Close `ags_connection` and file tabs in SQL Developer.
27. From your terminal window, clean up your environment by executing the `ags_cleanup.sh` script.

```
$ ./ags_cleanup.sh
…
SQL>
SQL> alter system set "_enable_automatic_maintenance"=1;

System altered.

SQL>
SQL> exit;
Disconnected …
$
```

Practices for Lesson 11

Chapter 11

Overview of Practices for Lesson 11

Practices Overview

In these practices, you will examine the behavior of adaptive cursor sharing and the effect of various settings of the `CURSOR_SHARING` parameter on execution plans.

Practice 11-1: Understanding Adaptive Cursor Sharing

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

1. The `acs_setup.sh` script was executed as part of the class setup to set up the environment used for this lab. This script is in your `$HOME/solutions/Adaptive_Cursor_Sharing` directory and listed here:

```
set echo on

drop user acs cascade;

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

grant dba, connect to acs;

connect acs/acs

drop table emp purge;

create table emp
(
empno   number,
ename   varchar2(20),
phone   varchar2(20),
deptno  number
);


insert into emp
  with tdata as
       (select rownum empno
         from all_objects
         where rownum <= 1000)
  select rownum,
         dbms_random.string ('u', 20),
         dbms_random.string ('u', 20),
         case
                when rownum/100000 <= 0.001 then mod(rownum, 10)
                else 10
         end
    from tdata a, tdata b
   where rownum <= 100000;

create index emp_i1 on emp(deptno);

exec dbms_stats.gather_table_stats(null, 'EMP', METHOD_OPT => 'FOR
COLUMNS DEPTNO SIZE 10', CASCADE => TRUE);
```

```
alter system flush shared_pool;
exit;
```

2. Change directories to `$HOME/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 lab. 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/solutions/Adaptive_Cursor_Sharing
$ sqlplus acs/acs
…
SQL> alter system flush shared_pool;

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                         HEIGHT BALANCED          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 of them (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.

SQL>
```

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 lab 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
```

5. Determine the execution plan for the following statement:

```
select /*ACS_L9*/ count(*), max(empno)
from emp
where deptno = 9;
```

This statement is in the `select_deptno_literal_9.sql` file.

What do you notice and why?

a. The optimizer uses an index range scan because value 9 is very selective.

```
SQL> @select_deptno_literal_9
SQL> set echo on
SQL>
SQL> select /*ACS_L9*/ count(*), max(empno)
  2  from emp
  3  where deptno = 9;

  COUNT(*) MAX(EMPNO)
---------- ----------
        10         99

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

```
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------
SQL_ID  64ngy4j55d1z5, child number 0
-------------------------------------
select /*ACS_L9*/ count(*), max(empno) from emp where deptno = 9

Plan hash value: 3184478295

----------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost
(%CPU)| Time     |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     2
(100)|          |
|   1 |  SORT AGGREGATE              |        |     1 |    16 |
|          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP    |    14 |   224 |     2
(0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_I1 |    14 |       |     1
(0)| 00:00:01 |
----------------------------------------------------------------------


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

   3 - access("DEPTNO"=9)



20 rows selected.

SQL>
```

6. Determine the execution plan for the following statement:

```
select /*ACS_L10*/ count(*), max(empno)
from emp
where deptno = 10;
```

This statement is in the `select_deptno_literal_10.sql` file.

What do you notice and why?

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

```
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*ACS_L10*/ count(*), max(empno)
  2  from emp
```

```
  3  where deptno = 10;

  COUNT(*) MAX(EMPNO)
---------- ----------
     99900     100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
SQL_ID  3232j5gkp2u5h, child number 0
-------------------------------------
select /*ACS_L10*/ count(*), max(empno) from emp where deptno = 10

Plan hash value: 2083865914

-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time
----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   239 (100)|
|   1 |  SORT AGGREGATE    |      |     1 |    16 |            |
|*  2 |   TABLE ACCESS FULL| EMP  | 95000 |  1484K|   239   (1)|
00:00:03 |
----------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("DEPTNO"=10)


19 rows selected.

SQL>
```

7. Define a bind variable called `DEPTNO` in your SQL*Plus session, set it to value 9, and execute the following query, and determine its execution plan:

   ```
   select /*ACS_1*/ count(*), max(empno)
   from emp
   where deptno = :deptno;
   ```

   This statement is in the `select_deptno_bind.sql` file.

   What do you notice and why?

Because the optimizer uses bind peeking the first time you execute a statement with a bind variable, and because for this first execution, value 9 is used, the execution plan with index access is used.

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

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
  2  from emp
  3  where deptno = :deptno;

  COUNT(*) MAX(EMPNO)
---------- ----------
        10         99

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------
SQL_ID  272gr4hapc9w1, child number 0
-------------------------------------
select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295


----------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost
(%CPU)| Time     |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     2
(100)|          |
|   1 |  SORT AGGREGATE              |        |     1 |    16 |
|          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP    |    14 |   224 |     2
(0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_I1 |    14 |       |     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>
```

8. Determine the execution statistics in terms of child cursors, executions, and buffer gets for the previously executed statement. What do you observe?
   a. In `V$SQL`, only one child cursor exists, and it has been executed only once (first time ever in this case). Also, the number of buffer gets is small due to the efficient access path that was used.

```
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
  2  from v$sql
  3  where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS
------------ ---------- -----------
           0          1           3

SQL>
```

9. Perform steps 7 and 8 again, but this time using 10 as the bind value for `DEPTNO`. What do you observe and why?
   a. The execution plan is identical. The index path is used although value 10 is not selective. This is because bind peeking only operates 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    |    14 |   224 |     2
(0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_I1 |    14 |       |     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> @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 wipe out all cursor's information.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

11. Perform step 9 again. This time, set the bind variable to 10. What do you observe and why?
    a. The execution plan is a full table scan because you used the value 10 as your first bind value. You can see this in the peeked binds. There is only one child cursor that is created so far to handle your statement.

```
SQL> exec :deptno := 10

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
  2  from emp
  3  where deptno = :deptno;

  COUNT(*) MAX(EMPNO)
---------- ----------
     99900     100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));
```

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
SQL_ID  272gr4hapc9w1, child number 0
-------------------------------------
select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914
-----------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU)| Time
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |       |       |   239 (100)|
|   1 |  SORT AGGREGATE     |      |     1 |    16 |            |
|*  2 |   TABLE ACCESS FULL| EMP  | 95000 |  1484K|   239   (1)|
00:00:03 |
-----------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - :DEPTNO (NUMBER): 10

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("DEPTNO"=:DEPTNO)


24 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
  2  from v$sql
  3  where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS
------------ ---------- -----------
           0          1         856

SQL>
```

12. Perform step 9 again, but this time, use 9 as your bind value. What do you observe and why?
    a. Although value 9 is very selective, a full table scan is still used. This is because the second time you execute your statement, bind peeking is not done. So you continue to use the same child cursor.

```
SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
  2  from emp
  3  where deptno = :deptno;

  COUNT(*) MAX(EMPNO)
---------- ----------
        10         99

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------
SQL_ID  272gr4hapc9w1, child number 0
-------------------------------------
select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914

----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time
----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   239 (100)|
|   1 |  SORT AGGREGATE    |      |     1 |    16 |            |
|*  2 |   TABLE ACCESS FULL| EMP  | 95000 |  1484K|   239   (1)|
00:00:03 |
----------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - :DEPTNO (NUMBER): 10

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("DEPTNO"=:DEPTNO)
```

```
24 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
  2  from v$sql
  3  where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS
------------ ---------- -----------
           0          2        1661

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="11.2.0.1";

Session altered.

SQL> alter system flush shared_pool;

System altered.

SQL>
```

14. Perform step 12 again. What do you observe, and why?
    a. Because this is the first time you execute the statement, bind peeking is used, and because value 9 is very selective, the index path is used. Only one child cursor is used to handle this statement.

```
SQL> exec :deptno := 9

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
  2  from emp
  3  where deptno = :deptno;

  COUNT(*) MAX(EMPNO)
---------- ----------
```

```
        10         99

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
----------------------------------------------------------------------
SQL_ID  272gr4hapc9w1, child number 0
-------------------------------------
select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 3184478295
----------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost
(%CPU)| Time     |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     2
(100)|          |
|   1 |  SORT AGGREGATE              |        |     1 |    16 |
|          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP    |    14 |   224 |     2
(0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_I1 |    14 |       |     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          54

SQL>
```

15. Perform step 14 again, but this time using value 10 as your bind value. What do you observe and why?
    a. 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);

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    |    14 |   224 |     2
(0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_I1 |    14 |       |     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> @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        1008

SQL>
```

16. Perform step 15 again. What do you observe and why?
    a. Because you now use adaptive cursor sharing, the system realizes that you benefit from another child cursor for handling your statement. This time, a full table access path is used to better handle your statement.

```
SQL> exec :deptno := 10

PL/SQL procedure successfully completed.

SQL> @select_deptno_bind
SQL> set echo on
SQL>
SQL> select /*ACS_1*/ count(*), max(empno)
  2  from emp
  3  where deptno = :deptno;
```

```
  COUNT(*) MAX(EMPNO)
---------- ----------
     99900     100000

SQL>
SQL> @show_latest_exec_plan
SQL> set echo on
SQL>
SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL
+PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------
SQL_ID  272gr4hapc9w1, child number 0
-------------------------------------
select /*ACS_1*/ count(*), max(empno) from emp where deptno = :deptno

Plan hash value: 2083865914
-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time
|
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   239 (100)|
|
|   1 |  SORT AGGREGATE    |      |     1 |    16 |            |
|
|*  2 |   TABLE ACCESS FULL| EMP  | 95000 |  1484K|   239   (1)|
00:00:03 |
-----------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - :DEPTNO (NUMBER): 10

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("DEPTNO"=:DEPTNO)


24 rows selected.

SQL>
SQL> @show_latest_exec_stats
SQL> set echo on
SQL>
SQL> select child_number, executions, buffer_gets
  2  from v$sql
```

```
  3  where sql_text like 'select /*ACS_1%';

CHILD_NUMBER EXECUTIONS BUFFER_GETS
------------ ---------- -----------
           0          1         805
           1          2        1008

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: Understanding CURSOR_SHARING

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

1. You can find all the necessary scripts for this lab in your $HOME/solutions/Cursor_Sharing directory. The environment for this lab 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 used throughout this lab. The script is listed here:

```
#!/bin/bash

cd /home/oracle/solutions/Cursor_Sharing

sqlplus / as sysdba @cs_setup.sql

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

set echo on

drop user cs cascade;

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

grant dba, connect to cs;

connect cs/cs

drop table emp purge;

create table emp
(
empno   number,
ename   varchar2(20),
phone   varchar2(20),
deptno  number
);


insert into emp
  with tdata as
       (select rownum empno
         from all_objects
         where rownum <= 1000)
  select rownum,
         dbms_random.string ('u', 20),
         dbms_random.string ('u', 20),
         case
```

```
                 when rownum/100000 <= 0.001 then mod(rownum, 10)
                 else 10
          end
     from tdata a, tdata b
    where rownum <= 100000;

create index emp_i1 on emp(deptno);

execute dbms_stats.gather_table_stats(null, 'EMP', cascade => true);

alter system flush shared_pool;

connect / as sysdba

shutdown immediate;

startup;

exit;
```

2. In a terminal session, change directory to the `$HOME/solutions/Cursor_Sharing` directory. Then connect as the `CS` user in a SQL*Plus session, and stay connected to that session until the end of this lab. For formatting reasons, after you have connected in the SQL*Plus session, execute the following command:

   `set linesize 200 pagesize 1000`

```
$ cd $HOME/solutions/Cursor_Sharing
$ sqlplus cs/cs
…
Connected to: …
SQL> set linesize 200 pagesize 1000
SQL>
```

3. Check the existence of histograms on the columns of the `EMP` table using the `check_emp_histogram.sql` script, and then determine the data distribution in the `DEPTNO` column of the `EMP` table with the statement in the `show_deptno_distribution.sql` file. What do you observe?
   a. Currently, there are no histograms created on the columns of the `EMP` table. Also, it is clear that you have data skew in the `DEPTNO` column. Value 10 repeats most of the time (99.9%), whereas all other values only repeat 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, and 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>
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 a5
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
-------- ---------- ---------- ---------------- --------------- ------
4CDF79D8 3434097775          0 SELECT STATEMENT
4CDF79D8 3434097775          0 SORT             AGGREGATE
4CDF79D8 3434097775          0 TABLE ACCESS     BY INDEX ROWID  EMP
4CDF79D8 3434097775          0 INDEX            RANGE SCAN      EMP_I1
SQL>
```

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                         HEIGHT BALANCED        10

SQL>
```

8. Before you continue, ensure that you flush your shared pool.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

9. Perform step 6 again. What do you observe and why?
   a. Although you captured histogram for the DEPTNO column that shows data skew, the system continues to share only one child cursor to handle both statements. This behavior is due to the FORCE option for the CURSOR_SHARING initialization parameter.

```

SQL> @select_deptno_literal_9
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 9;

  COUNT(*) MAX(EMPNO)
---------- ----------
        10         99

SQL>
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 10;

  COUNT(*) MAX(EMPNO)
---------- ----------
     99900     100000

SQL>
SQL> @show_latest_cursors
SQL> set echo on
SQL>
SQL> col sql_text format a70
SQL>
SQL> select sql_text,hash_value
  2  from v$sql
  3  where sql_text like '%select /*CS%';
```

```
SQL_TEXT
HASH_VALUE
--------------------------------------------------------------------------------
----------
select /*CS*/ count(*), max(empno) from emp where deptno = :"SYS_B_0"
3434097775


SQL>
```

10. Before you continue, ensure that you flush your shared pool.

```
SQL> alter system flush shared_pool;

System altered.

SQL>
```

11. How would you ensure that you now use more than one child cursor to handle both statements? Implement your solution, and check it.
    a. By setting `CURSOR_SHARING` to `SIMILAR` for your session, the system is able to see that you benefit from using two different child cursors to handle both statements because they lend themselves to very different execution plans.

```
SQL> alter session set cursor_sharing = similar;

Session altered.

SQL> @select_deptno_literal_9
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 9;

  COUNT(*) MAX(EMPNO)
---------- ----------
        10         99

SQL>
SQL> @select_deptno_literal_10
SQL> set echo on
SQL>
SQL> select /*CS*/ count(*), max(empno) from emp where deptno = 10;

  COUNT(*) MAX(EMPNO)
---------- ----------
     99900     100000

SQL>
```

```
SQL> @show_latest_cursors
SQL> set echo on
SQL>
SQL> col sql_text format a70
SQL>
SQL> select sql_text,hash_value
  2  from v$sql
  3  where sql_text like '%select /*CS%';

SQL_TEXT
HASH_VALUE
----------------------------------------------------------------------
----------
select /*CS*/ count(*), max(empno) from emp where deptno = :"SYS_B_0"
3434097775
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 object_name format a5
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
-------- ---------- ---------- ---------------- --------------- ------
4CDF79D8 3434097775          1 SELECT STATEMENT
4CDF79D8 3434097775          1 SORT             AGGREGATE
4CDF79D8 3434097775          1 TABLE ACCESS     FULL            EMP
4CDF79D8 3434097775          0 SELECT STATEMENT
4CDF79D8 3434097775          0 SORT             AGGREGATE
4CDF79D8 3434097775          0 TABLE ACCESS     BY INDEX ROWID  EMP
4CDF79D8 3434097775          0 INDEX            RANGE SCAN      EMP_I1

7 rows selected.

SQL>
```

```
```

12. Flush the shared pool script to clean up your environment for this lab. Then exit your SQL*Plus session.

```
SQL> alter system flush shared_pool;

System altered.

SQL> exit
Disconnected …

$
```

Practices for Lesson 12

Chapter 12

Overview of Practices for Lesson 12

Practices Overview

In these practices, you will use the SQL Tuning Advisor to tune a high load SQL statement.

Practice 12-1: Proactively Tuning High-Load SQL Statements

In this practice, you use the SQL Tuning Advisor to tune problematic SQL statements.

1. Connect as `SYSDBA` through Database Control and navigate to the Performance tab of the Database Control Home page. On the Performance tabbed page, make sure that the View Data field is set to Real Time: 15 second Refresh. After this is done, open a terminal emulator window connected as the `oracle` user. When this is done, change your current directory to your lab directory: `cd $HOME/solutions/SQL_Tuning_Advisor`. Then enter the following command from the OS prompt: `./setup_dina.sh`.

```
$ cd $HOME/solutions/SQL_Tuning_Advisor
$ ./setup_dina.sh

PL/SQL procedure successfully completed.


Grant succeeded.


Session altered.


PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.


User altered.


User altered.


Index dropped.

drop index sales_time_idx
           *
ERROR at line 1:
ORA-01418: specified index does not exist



Index created.

$
```

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

#!/bin/bash

cd /home/oracle/solutions/SQL_Tuning_Advisor

sqlplus -s /NOLOG <<EOF

set echo on

connect / as sysdba

exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();

grant dba to SH;

-- event to allow setting very short Flushing interval

alter session set events '13508 trace name context forever, level 1';


-- change INTERVAL setting to 2 minutes
-- change RETENTION setting to 6 hours (total of 180 snapshots)
execute dbms_workload_repository.modify_snapshot_settings(interval =>
2,retention => 360);


-- play with ADDM sensitiveness
exec
dbms_advisor.set_default_task_parameter('ADDM','DB_ACTIVITY_MIN',30);


alter user sh account unlock;
alter user sh identified by sh;

connect sh/sh

drop index sales_time_bix;
drop index sales_time_idx;
create index sales_time_idx on sales(time_id) compute statistics;

EOF
```

2. After this is executed, execute the `start_dinas.sh` script by using the following command: `./start_dinas.sh`.

   This script starts the workload used for this lab.

```
$ ./start_dinas.sh
```

```
Started stream with pid=30479
Started stream with pid=30480
$

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

#!/bin/bash

cd /home/oracle/solutions/SQL_Tuning_Advisor

STREAM_NUM=0
MAX_STREAM=6
PIDLST=""

while [ $STREAM_NUM -lt $MAX_STREAM ]; do

  # one more
  let STREAM_NUM="STREAM_NUM+1"

  # start one more stream
  sqlplus -S sh/sh @dina.sql &

  # remember PID
  PIDLST="$! $PIDLST"

  echo "Started stream with pid=$!"

done

#
# Save PID List
#
echo $PIDLST > /tmp/dina_pids

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

DECLARE
n number;
BEGIN
for i in 1..1000 loop
select /*+ ORDERED USE_NL(c) FULL(c) FULL(s)*/ count(*) into n
from sales s, customers c
where c.cust_id = s.cust_id and CUST_FIRST_NAME='Dina'
order by time_id;
DBMS_LOCK.SLEEP(1);
end loop;
```

```
END;
/
```

3. When the `start_dinas.sh` script completes, observe the Performance tabbed page for around 10 minutes (5 snapshot icons). What are your conclusions?
   a. You should see that the workload activity goes up very quickly. Because the CPU used by the workload is very close to the maximum CPU available on your system, there must be an issue with this workload. Because the most important area corresponding to a wait class is the CPU Wait class, the issue must be associated to that class. Note that the snapshot interval is now around two minutes.
4. Fix the problem.
   a. The fastest way to determine the problem is by looking at an Automatic Database Diagnostic Monitor (ADDM) report analysis executed during the problematic period. Then by following its analysis, ADDM should guide you through the process of fixing the problem.
   b. Using the Database Control Home page, there are two ways to identify the correct ADDM analysis task:
      1) If the time corresponding to the problematic time period corresponds with the latest ADDM run detected by Database Control, you should find the link corresponding to the correct performance analysis directly in the Diagnostic Summary section of the Database Control Home page. Note that you should wait around 8 to 10 minutes before the Diagnostic Summary section is refreshed with the correct ADDM analysis. If you are in this case, click the link corresponding to the number of findings right next to the ADDM Findings row. This takes you to the corresponding Automatic Database Diagnostic Monitor (ADDM) page.
      2) If not, you should open the Advisor Central page and search for the correct ADDM task. This is how you can retrieve the task from the Advisor Central page:
         a) On the Database Control Home page, click the Advisor Central link.
         b) On the Advisor Central page, in the search section, select ADDM from the Advisory Type drop-down list, and Last 24 Hours from the Advisor Runs drop-down list.
         c) After this is done, click Go.
         d) Then select the ADDM task corresponding to the time of the problematic period.
         e) This takes you to the corresponding Automatic Database Diagnostic Monitor (ADDM) page.
   c. On the Automatic Database Diagnostic Monitor (ADDM) page, you should see two main findings: Top SQL Statements and CPU Usage. The Top SQL finding should be close to 100%. If it is not, ensure that what you see is the latest ADDM analysis.
   d. Click the “Top SQL Statements” link.
   e. On the “Performance Finding Details: Top SQL Statements” page, click Show All Details link. You should see something similar to the following: Run SQL Tuning Advisor on the SQL statement with SQL_ID "5mxdwvuf9j3vp." Next to this recommendation, click the Run Advisor Now button.
   f. Wait on the Processing: SQL Tuning Advisor Task SQL_TUNING_… page for a while.

g. You are automatically directed to the “Recommendations for SQL ID:5mxdwvuf9j3vp” page from where you should see two recommendations: one for a SQL Profile and one for an index creation.

h. You can investigate the consequence of implementing the recommended profile by comparing execution plans before and after profile implementation. You can do so by clicking the “Compare explain plan” eyeglass icon for the corresponding SQL Profile row. Because the potential benefit of using the proposed SQL profile is very high (see both computed Costs), you implement the SQL profile.

i. On the Compare Explain Plans page, click the “Recommendations for SQL ID:5mxdwvuf9j3vp” locator link.

j. Back to the “Recommendations for SQL ID:5mxdwvuf9j3vp” page, ensure that the SQL Profile row is selected, and click Implement.

k. On the Confirmation page, click Yes.

l. On the “Recommendations for SQL ID:5mxdwvuf9j3vp” page, you should see the following message at its top: “Confirmation: The recommended SQL Profile has been created successfully”.

m. Click Return.

n. Click the Database Instance: orcl locator link.

o. Back to the Database Home Page, click the Performance tab.

5. After following the SQL Tuning Advisor recommendation to implement a SQL Profile, how can you quickly verify that the problem is solved?

   a. On the Performance tabbed page, you expect to see a dramatic drop for the CPU Wait class in the Average Active Sessions graph. However, when you view the graph, you see that it has not changed.

6. How do you interpret the result you see, and how would you ensure that the new profile is taken into account?

   a. A profile is taken into account the next time you execute the corresponding statement. Because the SQL statement for which you created a profile takes a long time to execute, you could wait for a long time to see the benefit. To quickly see the benefit, stop and restart your workload. This executes the same SQL statements again, and the profile should be used automatically this time.

```
$ ./stop_dinas.sh
Killing stream with pid=30486
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed


Killing stream with pid=30485
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed


$
```

```
$ ./start_dinas.sh
Started stream with pid=31731
Started stream with pid=31732
$

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

#!/bin/bash

cd /home/oracle/solutions/SQL_Tuning_Advisor

PIDLST=`cat /tmp/dina_pids`

#
# Kill all these processes
#
for PID in $PIDLST; do
  echo "Killing stream with pid=$PID"
  sqlplus / as sysdba @kill_dina.sql $PID >> /tmp/stop_dina.log 2>&1
  sqlplus /nolog @/tmp/drop_dina.sql >> /tmp/stop_dina.log 2>&1
  kill -9 $PID >> /tmp/stop_dina.log 2>&1
done

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


set head off
set timing off
set feedback off;
set pagesize 0
set verify off

spool /tmp/drop_dina.sql;

select 'connect / as sysdba;' from dual;

select 'alter system kill session ''' || sid || ',' || serial# ||
''';'
from v$session
where process=&1;

select 'exit;' from dual;

spool off
```

```
exit;
```

b. Go back to the Performance tabbed page.

c. On the Performance page, you should now see the benefit of the SQL Profile. The CPU Wait category is much reduced in the Average Active Sessions graph. This may take three or four updates to be visible.

7. How would you make sure that the SQL Profile was implemented?

a. On the Average Active Sessions graph, click the CPU Wait category in the caption.

b. On the Active Sessions Waiting: CPU Wait page, you should see that your statement is waiting. You can see that from the `ACTIVITY(%)` column.

c. Click the 5mxdwvuf9j3vp SQL Id link in the Top SQL table.

d. On the "SQL Details: 5mxdwvuf9j3vp" page, click the Plan Control tab in the Details section.

e. You should see that this statement is associated to a SQL Profile that was manually implemented. It is part of the DEFAULT category and is ENABLED.

8. Clean up your environment by executing the following commands from your command-line window:

`./stop_dinas.sh`

`./cleanup_dina.sh` -- takes ~ 4 minutes

```
$ ./stop_dinas.sh
Killing stream with pid=31740
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed

Killing stream with pid=31735
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed

Killing stream with pid=31734
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed

Killing stream with pid=31733
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed

Killing stream with pid=31732
DECLARE
```

```
*
ERROR at line 1:
ORA-00028: your session has been killed

Killing stream with pid=31731
DECLARE
*
ERROR at line 1:
ORA-00028: your session has been killed

$


$ ./cleanup_dina.sh

specify password for SH as parameter 1:

specify default tablespace for SH as parameter 2:

specify temporary tablespace for SH as parameter 3:

specify password for SYS as parameter 4:

specify directory path for the data files as parameter 5:

writeable directory path for the log files as parameter 6:

specify version as parameter 7:


Session altered.


User dropped.

old   1: CREATE USER sh IDENTIFIED BY &pass
new   1: CREATE USER sh IDENTIFIED BY sh

User created.

old   1: ALTER USER sh DEFAULT TABLESPACE &tbs
new   1: ALTER USER sh DEFAULT TABLESPACE example
old   2:  QUOTA UNLIMITED ON &tbs
new   2:  QUOTA UNLIMITED ON example

User altered.
```

```
old   1: ALTER USER sh TEMPORARY TABLESPACE &ttbs
new   1: ALTER USER sh TEMPORARY TABLESPACE temp

User altered.


Grant succeeded.


Grant succeeded.


...


Grant succeeded.


Grant succeeded.


PL/SQL procedure successfully completed.

Connected.

Grant succeeded.

old   1: CREATE OR REPLACE DIRECTORY data_file_dir AS '&data_dir'
new   1: CREATE OR REPLACE DIRECTORY data_file_dir AS
'/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/'

Directory created.

old   1: CREATE OR REPLACE DIRECTORY log_file_dir AS '&log_dir'
new   1: CREATE OR REPLACE DIRECTORY log_file_dir AS '/home/oracle/'

Directory created.


Grant succeeded.


Grant succeeded.
```

```
Grant succeeded.

Connected.

Session altered.


Session altered.


Table created.


Table created.


...


Table created.


Creating constraints ...

Table altered.


...


Table altered.


specify password for SH as parameter 1:

specify path for data files as parameter 2:

specify path for log files as parameter 3:

specify version as parameter 4:

Looking for indexes that could slow down load ...

no rows selected
```

```
loading TIMES using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/time
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/time
_v3.dat
/home/oracle/time_v3.log

…

Save data point reached - logical record count 1000.

Load completed - logical record count 1826.


loading COUNTRIES using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/coun
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/coun
_v3.dat
/home/oracle/coun_v3.log

…

Load completed - logical record count 23.


loading CUSTOMERS using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/cust
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/cust
1v3.dat
/home/oracle/cust1v3.log

...

Save data point reached - logical record count 10000.
Save data point reached - logical record count 20000.
Save data point reached - logical record count 30000.
Save data point reached - logical record count 40000.
Save data point reached - logical record count 50000.

Load completed - logical record count 55500.


loading PRODUCTS  using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/prod
_v3.ctl
```

```
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/prod
1v3.dat
/home/oracle/prod1v3.log

...


Load completed - logical record count 72.


loading PROMOTIONS  using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/prom
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/prom
1v3.dat
/home/oracle/prom1v3.log

…

Save data point reached - logical record count 10.
Save data point reached - logical record count 20.
...
Save data point reached - logical record count 500.

Load completed - logical record count 503.


loading CHANNELS using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/chan
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/chan
_v3.dat
/home/oracle/chan_v3.log

…

Load completed - logical record count 5.


loading SALES  using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/sale
_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/sale
1v3.dat
/home/oracle/sale1v3.log

…
```

```
Save data point reached - logical record count 100000.
...
Save data point reached - logical record count 900000.

Load completed - logical record count 916039.


loading COSTS using external table


Table created.


82112 rows created.


loading additonal SALES using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/dmsal_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/dmsal_v3.dat
/home/oracle/dmsal_v3.log

…

Save data point reached - logical record count 100.
...
Save data point reached - logical record count 2800.

Load completed - logical record count 2804.


loading SUPPLEMENTARY DEMOGRAPHICS using:
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/dem_v3.ctl
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/dem1v3.dat
/home/oracle/dem1v3.log

...

Save data point reached - logical record count 10.
...
Save data point reached - logical record count 4500.

Load completed - logical record count 4500.
```

```
Commit complete.


Enabling constraints ...

Table altered.


...


Table altered.


Creating additional indexes ...

Index created.


...


Index created.


Create dimensions ...

Dimension created.


Commit complete.


PL/SQL procedure successfully completed.


no rows selected


Dimension created.


PL/SQL procedure successfully completed.
```

```
no rows selected


Dimension created.


PL/SQL procedure successfully completed.


no rows selected


Dimension created.


PL/SQL procedure successfully completed.


no rows selected


Dimension created.


PL/SQL procedure successfully completed.


no rows selected

Creating MVs as tables ...


View created.


Table created.


Table created.


Index created.


Index created.
```

```
Index created.


Index created.

Creating materialized views ...


Materialized view created.


Materialized view created.


Creating comments ...

Comment created.


...

<<<<< 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.

SQL> SQL> Disconnected …

$


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

```
#!/bin/bash

cd /home/oracle/solutions/SQL_Tuning_Advisor

#
# Cleanup ADDM snapshot settings
#
sqlplus -s /NOLOG <<EOF >> /tmp/cleanup_dina.log 2>&1

  connect / as sysdba

  rem -- change INTERVAL setting to 30 minute
  execute dbms_workload_repository.modify_snapshot_settings(interval
=> 60);

  rem -- change ADDM sensitiveness back to normal
  exec
dbms_advisor.set_default_task_parameter('ADDM','DB_ACTIVITY_MIN',300);

  connect sh/sh

  drop index sales_time_idx;

  create bitmap index sales_time_bix
  on sales(time_id)
  tablespace example
  local nologging compute statistics;


EOF

#
# Cleanup sql profile
#
sqlplus -s /NOLOG <<EOF > /tmp/cleanup_dina.log 2>&1

  connect / as sysdba

  set head off
  set timing off
  set feedback off;
  set pagesize 0

spool /tmp/drop_dyn.sql;

select q'#connect / as sysdba;#' from dual;
```

```
select q'#execute dbms_sqltune.drop_sql_profile('#' || name || q'#')
;#'
from dba_sql_profiles ;

select q'#execute dbms_advisor.delete_task('#' || task_name || q'#')
;#'
from user_advisor_tasks
where  CREATED > SYSDATE-(1/24);

select q'#connect system/oracle;#' from dual;

select q'#execute dbms_advisor.delete_task('#' || task_name || q'#')
;#'
from user_advisor_tasks
where  CREATED > SYSDATE-(1/24);

spool off

@/tmp/drop_dyn.sql

EOF



cp /home/oracle/solutions/SQL_Access_Advisor/sh/*
$ORACLE_HOME/demo/schema/sales_history

cd /home/oracle/solutions/SQL_Access_Advisor/sh

sqlplus -s /NOLOG <<EOF

set echo on

connect / as sysdba

@sh_main sh example temp oracle
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/
/home/oracle/ v3

EOF
```

Practices for Lesson 13

Chapter 13

Practices for Lesson 13

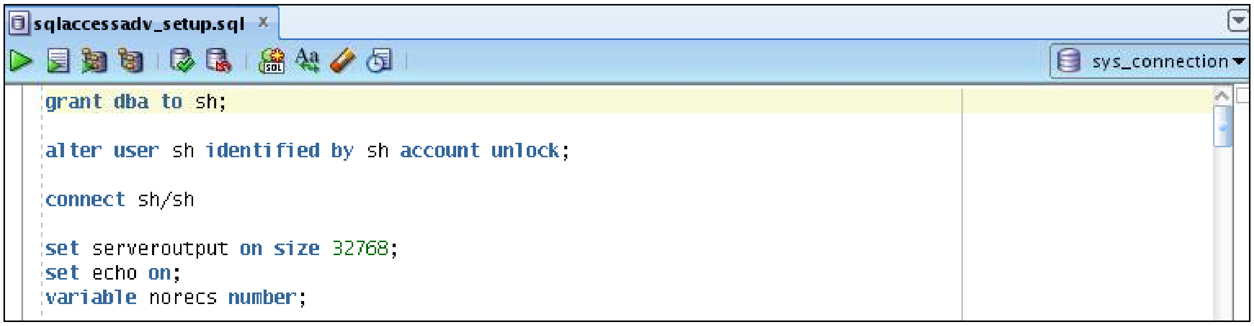
Practices Overview

In the following practices, you will use SQL Access Advisor to get recommendations and SQL Performance Analyzer to confirm the benefit of those recommendations.

Practice 13-1: Using SQL Access Advisor

The following scenario illustrates the types of recommendations that can be made by SQL Access Advisor. The scenario also uses the SQL Performance Analyzer to prove that recommendations made by SQL Access Advisor are good.

1. From SQL Developer, connected as `sys_connection`, execute the `$HOME/solutions/SQL_Access_Advisor/sqlaccessadv_setup.sql` script. This script generates the necessary data that you use throughout this lab. In particular, it generates the SQL Tuning Set that is used to represent the workload you want to analyze.




```
grant dba succeeded.
 alter user sh succeeded.
Connected
variable norecs number

Rem Clean up

declare
    name varchar2(30);
    cursor name_cur1 is
      select task_name from user_advisor_templates
        where task_name like '%SQLACCESS%';
  begin
    -------------------------------------------------------------------
---------
    --  Get rid of templates, tasks and workloads.
    -------------------------------------------------------------------
---------

    open name_cur1;

    loop
      fetch name_cur1 into name;
      exit when name_cur1%NOTFOUND;


dbms_advisor.update_task_attributes(name,null,null,'FALSE','FALSE');
      dbms_advisor.delete_task(name);
    end loop;
```

```
    close name_cur1;
  end;

anonymous block completed
Rem make a temp table

DROP TABLE temp_table purge



Error starting at line 44 in command:
DROP TABLE temp_table purge
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
CREATE TABLE temp_table AS SELECT * FROM SYS.WRI$_ADV_SQLW_STMTS WHERE
NULL IS NOT NULL

CREATE TABLE succeeded.
Rem create a large number of pseudo-random (repeatable) queries in the
temporary table

alter system flush shared_pool

 alter system flush succeeded.
execute dbms_sqltune.drop_sqlset('SQLSET_MY_SQLACCESS_WORKLOAD')



Error starting at line 53 in command:
execute dbms_sqltune.drop_sqlset('SQLSET_MY_SQLACCESS_WORKLOAD')
Error report:
ORA-13754: "SQL Tuning Set" "SQLSET_MY_SQLACCESS_WORKLOAD" does not
exist for user "SH".
ORA-06512: at "SYS.DBMS_SQLTUNE_INTERNAL", line 13171
ORA-06512: at "SYS.DBMS_SQLTUNE", line 4409
ORA-06512: at line 1
13754. 00000 -  "\"SQL Tuning Set\" \"%s\" does not exist for user
\"%s\"."
*Cause:    The user attempted to access a SQL Tuning Set that does not
exist.
*Action:   Check the speelling of the SQL Tuning Set name and retry
           the operation.
drop table tempjfv purge



Error starting at line 55 in command:
```

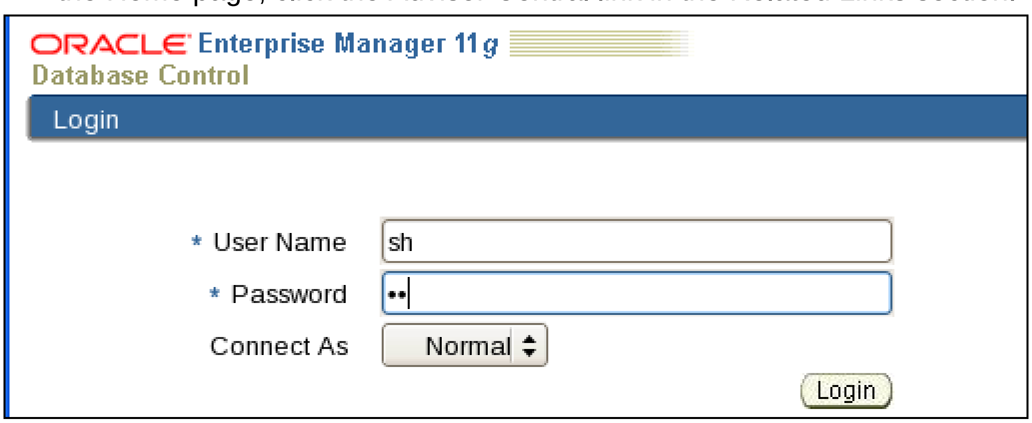
```
drop table tempjfv purge
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
create table tempjfv (c number, d varchar2(1000))

create table succeeded.
begin
for i in 1..20000 loop
  insert into tempjfv values(-
i,'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
end loop;
commit;
end;

anonymous block completed
drop table customersjfv purge



Error starting at line 68 in command:
drop table customersjfv purge
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
create table customersjfv as select * from customers

create table succeeded.
DECLARE
   sql_stmt   varchar2(2000);
   sqlsetname  VARCHAR2(30);
   sqlsetcur   dbms_sqltune.sqlset_cursor;
   refid    NUMBER;
 k NUMBER := 0;
 num_queries NUMBER := 500;
BEGIN
```

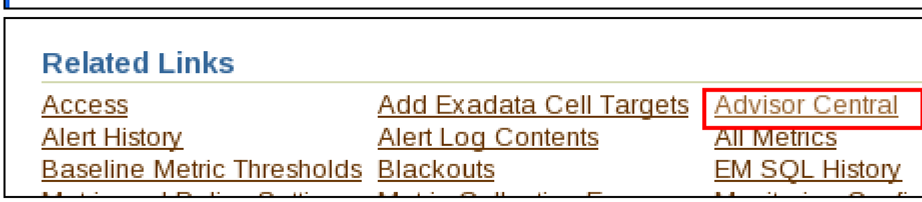
```
sql_stmt := 'SELECT /* QueryJFV 2 */ ch.channel_class, c.cust_city,
t.calendar_quarter_desc, SUM(s.amount_sold) sales_amount FROM sh.sales
s, sh.times t, sh.customers c, sh.channels ch WHERE s.time_id =
t.time_id AND s.cust_id = c.cust_id AND s.channel_id = ch.channel_id
AND c.cust_state_province = ''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';
insert into temp_table values(1,1,NULL,0,'SH','Access
Advisor','Workload',0,0,0,0,1,100,2,to_date('02-FEB-
2007'),3,0,sql_stmt,1);


sql_stmt := 'SELECT /* QueryJFV 3 */ 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-
03'',''1999-04'') GROUP BY ch.channel_class, c.cust_city,
t.calendar_quarter_desc';


insert into temp_table values(1,1,NULL,0,'SH','Access
Advisor','Workload',0,0,0,0,1,100,2,to_date('02-FEB-
2007'),3,0,sql_stmt,1);


sql_stmt := 'SELECT /* QueryJFV 4 */ c.country_id, c.cust_city,
c.cust_last_name FROM sh.customers c WHERE c.country_id in (52790,
52798) ORDER BY c.country_id, c.cust_city, c.cust_last_name';


insert into temp_table values(1,1,NULL,0,'SH','Access
Advisor','Workload',0,0,0,0,1,100,2,to_date('02-FEB-
2007'),3,0,sql_stmt,1);


sql_stmt := 'select /* func_indx */ count(*) from tempjfv where
abs(c)=5';


insert into temp_table values(1,1,NULL,0,'SH','Access
Advisor','Workload',0,0,0,0,1,100,2,to_date('02-FEB-
2007'),3,0,sql_stmt,1);


sql_stmt := 'SELECT /* QueryJFV 5 */ * FROM sh.customersjfv WHERE
cust_state_province = ''CA''';


insert into temp_table values(1,1,NULL,0,'SH','Access
Advisor','Workload',0,0,0,0,1,100,2,to_date('02-FEB-
2007'),3,0,sql_stmt,1);



sqlsetname := 'SQLSET_MY_SQLACCESS_WORKLOAD';
  dbms_sqltune.create_sqlset(sqlsetname, 'Generated STS');


   OPEN sqlsetcur FOR
```

```
     SELECT
          SQLSET_ROW(null,null, sql_text, null, null, username,
module,
                     action, elapsed_time, cpu_time, buffer_gets,
disk_reads,
                     0,rows_processed, 0, executions, 0,
optimizer_cost, null,
                     priority, command_type,
                     to_char(last_execution_date,'yyyy-mm-
dd/hh24:mi:ss'),
                     0,0,NULL,0,NULL,NULL
                     )
    FROM temp_table;

   dbms_sqltune.load_sqlset(sqlsetname, sqlsetcur);
END;

anonymous block completed
SELECT COUNT(*) FROM
TABLE(DBMS_SQLTUNE.SELECT_SQLSET('SQLSET_MY_SQLACCESS_WORKLOAD'))

COUNT(*)
----------------------
5

Rem Cleanup anything left behind

execute dbms_advisor.delete_task('%')

anonymous block completed
execute dbms_advisor.delete_sqlwkld('%')

anonymous block completed
EXECUTE DBMS_STATS.UNLOCK_SCHEMA_STATS('SH')

anonymous block completed
execute dbms_stats.gather_schema_stats(ownname => 'SH',
estimate_percent=> DBMS_STATS.AUTO_SAMPLE_SIZE, method_opt => 'FOR ALL
COLUMNS SIZE  AUTO', degree => 4 )

anonymous block completed
select distinct last_analyzed from dba_tab_statistics where owner='SH'

LAST_ANALYZED
-------------------------


11-AUG-10
```

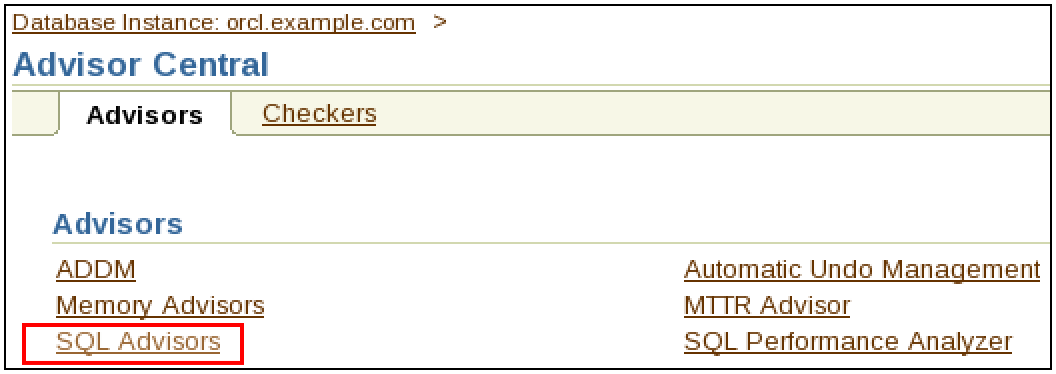
```
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
11-AUG-10
17 rows selected

REM EXECUTE DBMS_STATS.LOCK_SCHEMA_STATS('SH')

Connection created by CONNECT script command disconnected
```

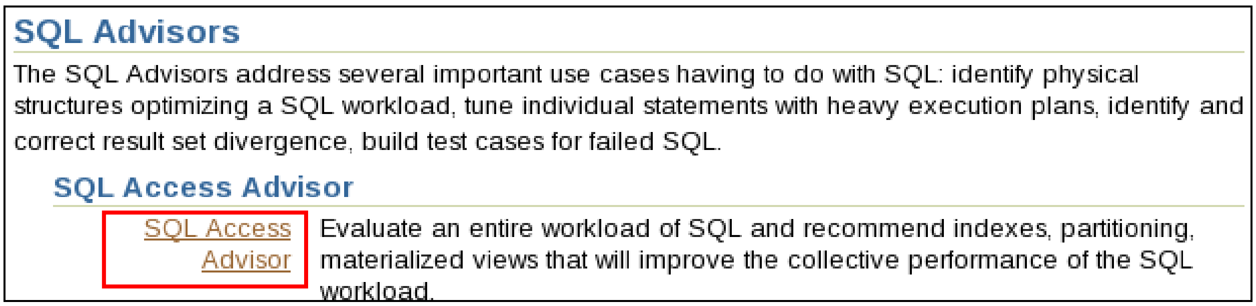
2. Using Enterprise Manager, create a SQL Access Advisor tuning task based on the captured workload held in the SH.SQLSET_MY_ACCESS_WORKLOAD SQL tuning set using the SQLACCESS_WAREHOUSE template.
   a. Connect to Enterprise Manager Database Control as the `sh` user (password: `sh`). On the Home page, click the Advisor Central link in the Related Links section.



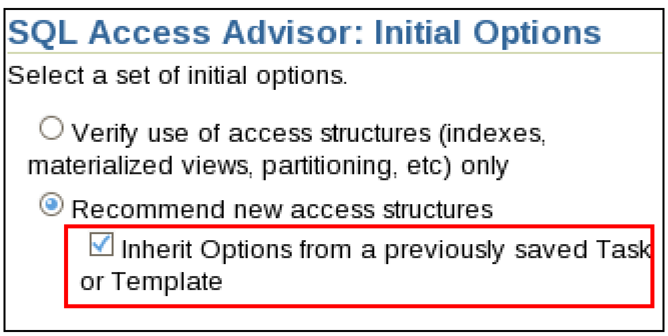


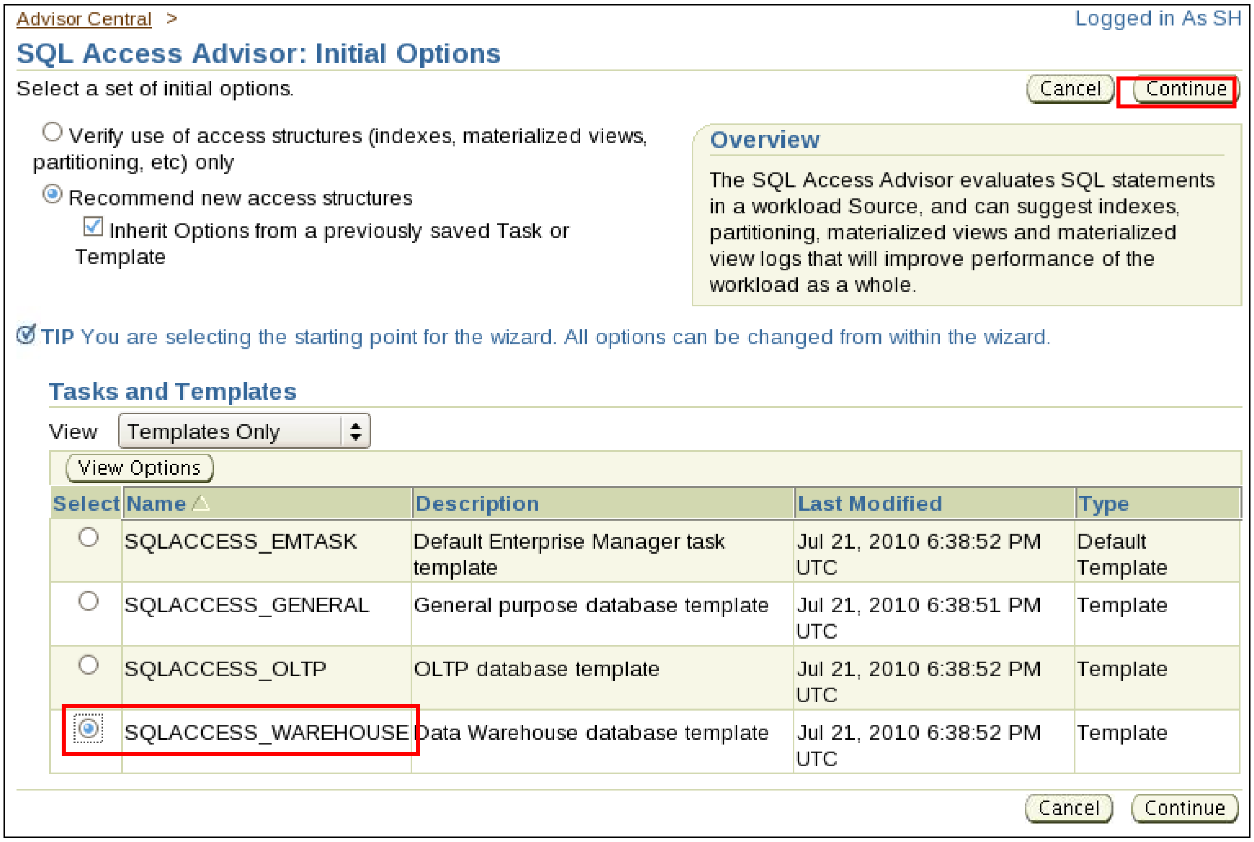
   b. On the Advisor Central page, click the SQL Advisors link. Then on the SQL Advisors page, click the SQL Access Advisor link.



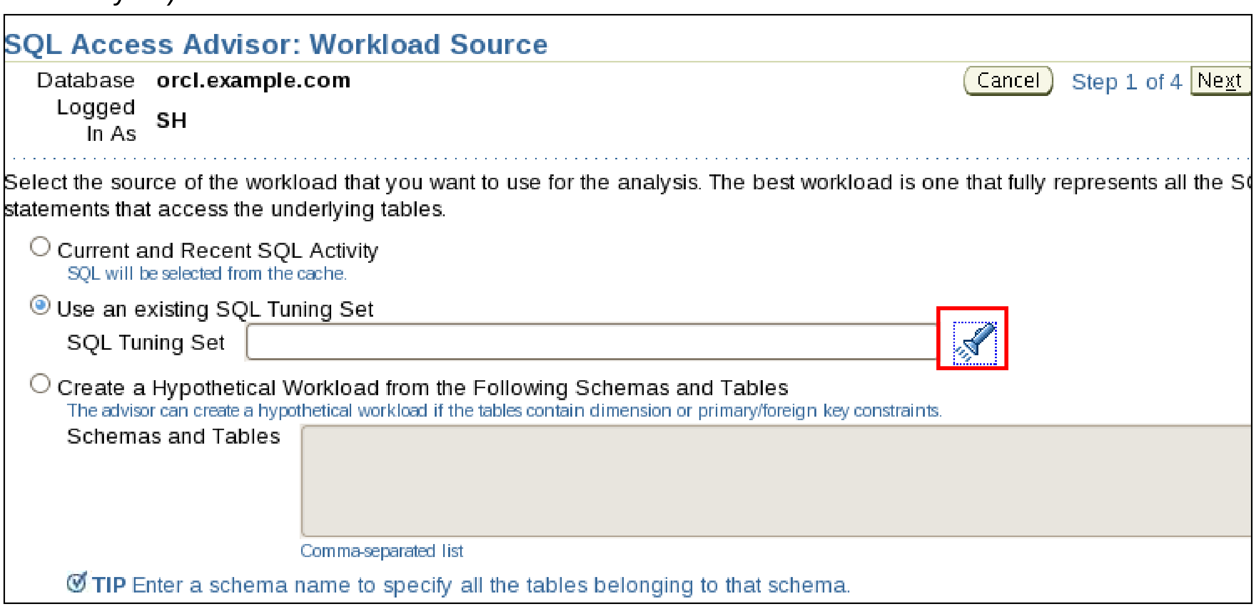


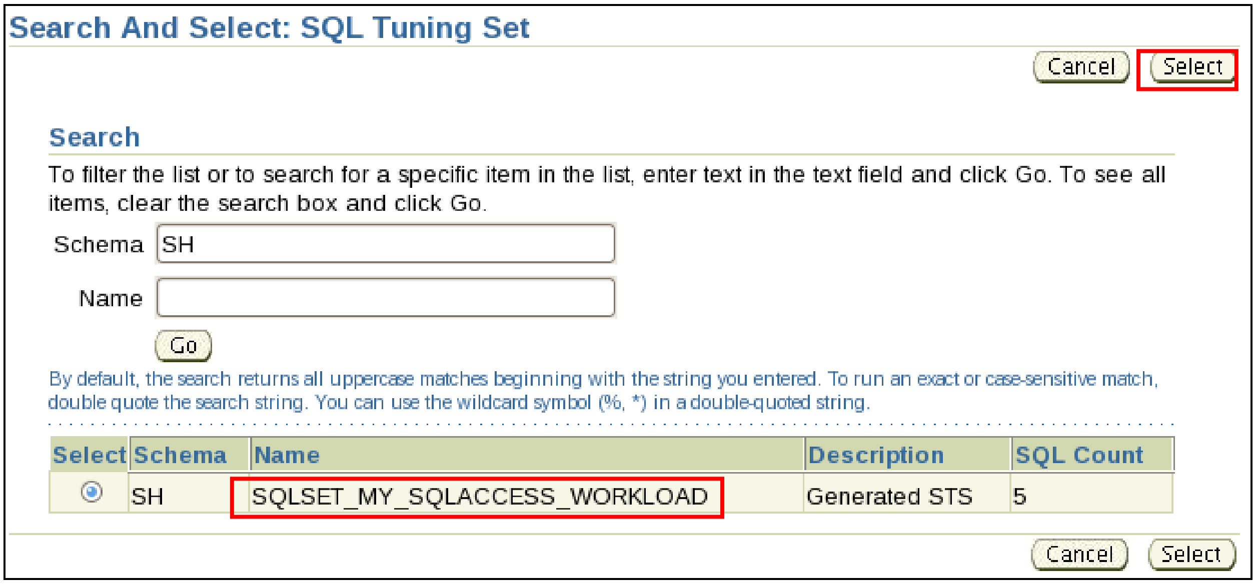
c. On the Initial Options page, select "Inherit Options from a previously saved Task or Template," and then select the SQLACCESS_WAREHOUSE template. After this is done, click Continue.

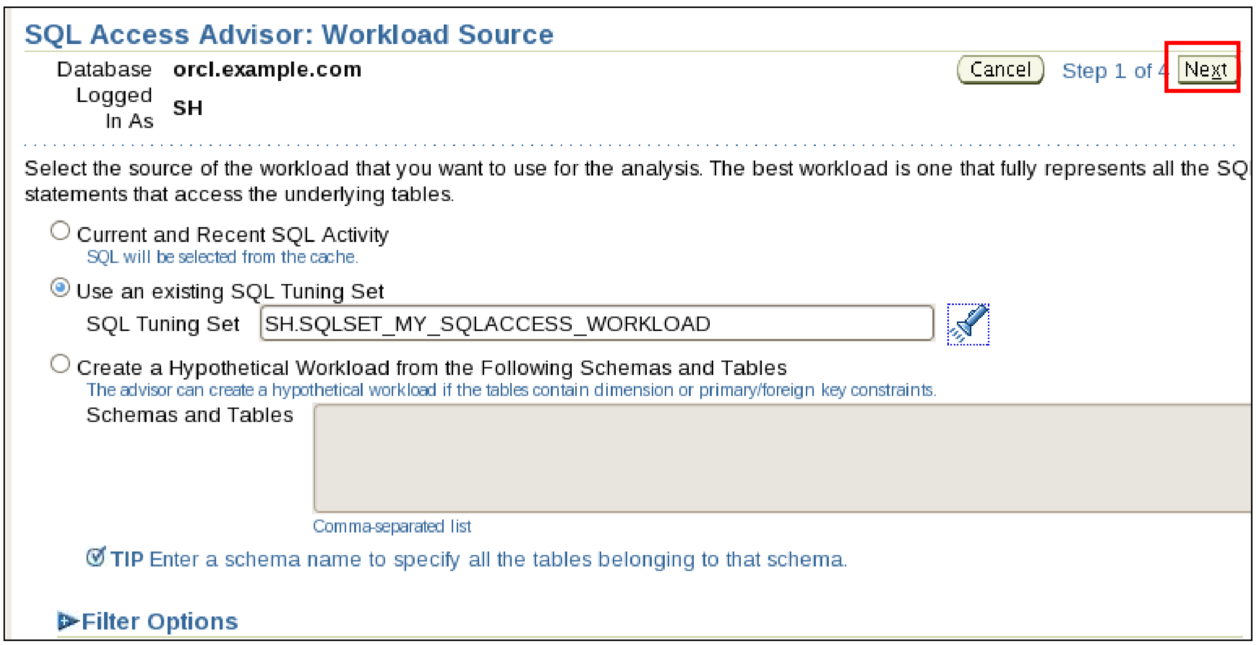




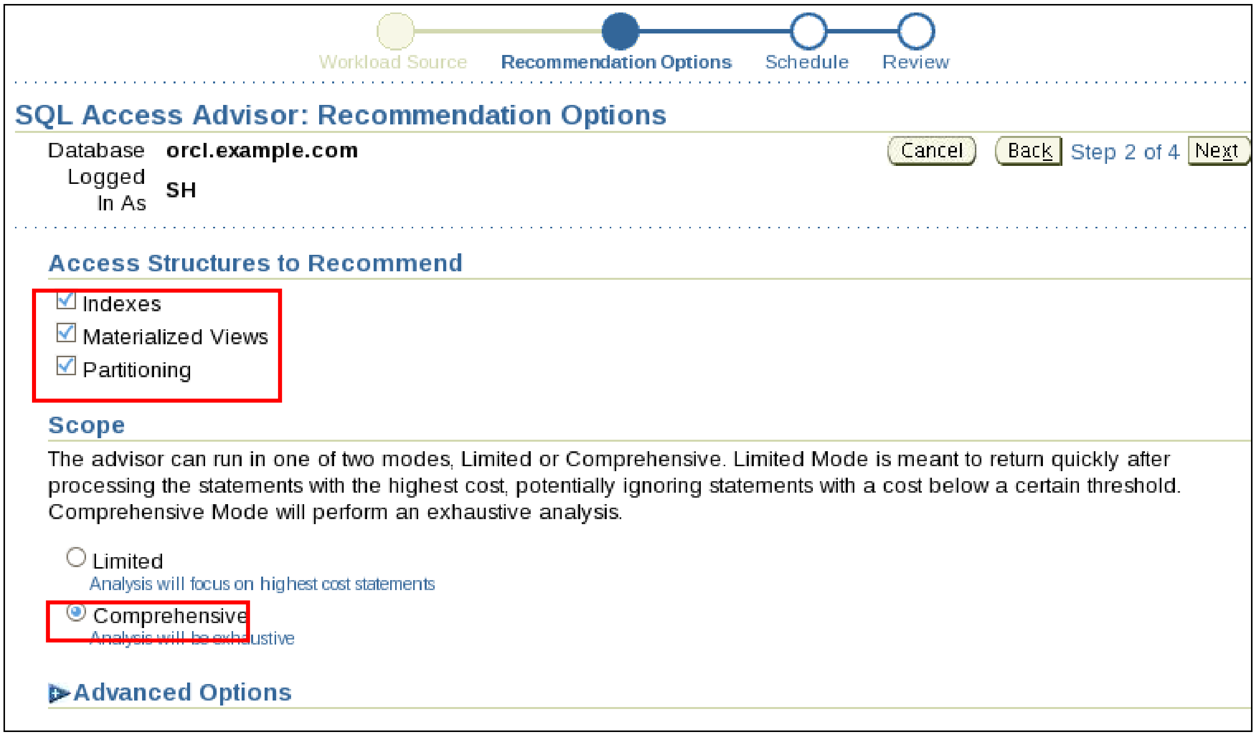
d. On the Workload Source page, select "Use an existing SQL Tuning Set" and enter `SH.SQLSET_MY_SQLACCESS_WORKLOAD` in the SQL Tuning Set field. (This SQL Tuning Set was generated earlier. It represents a warehouse workload that you want to analyze.) Click Next.



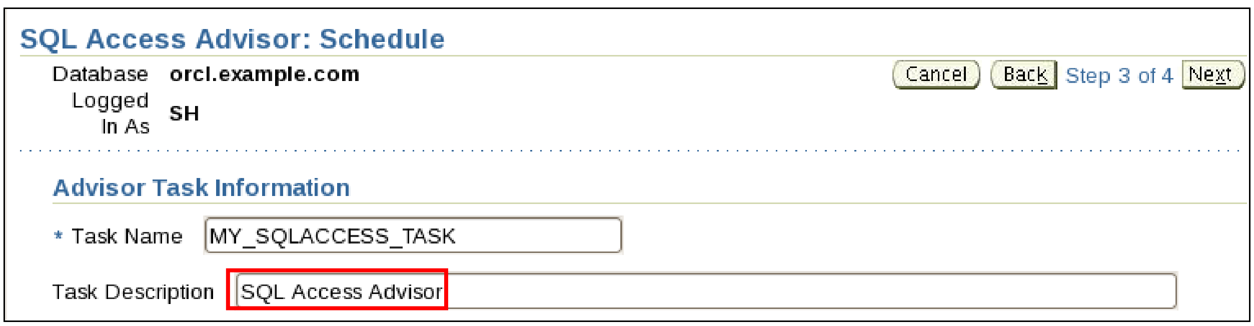




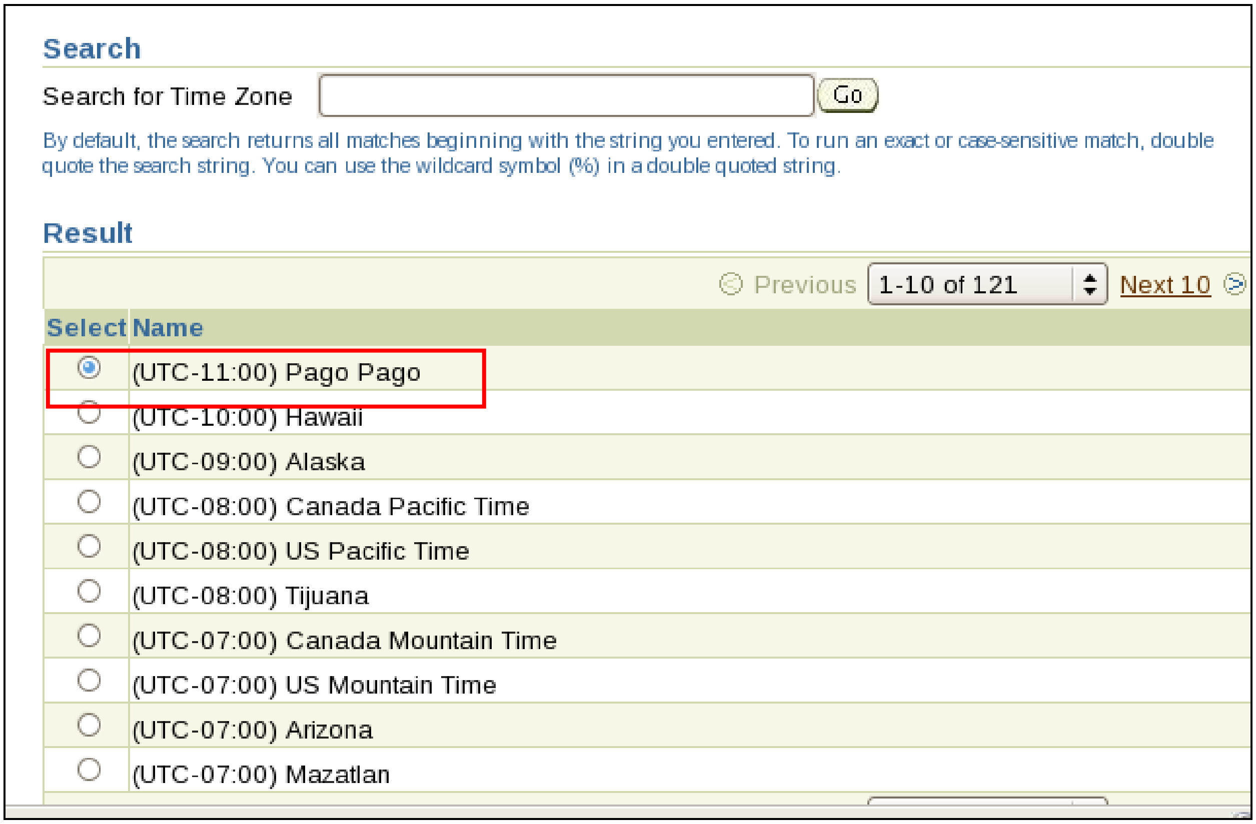
e. On the Recommendation Options page, ensure that all possible access structures are selected, and that Comprehensive is selected. After this is done, click Next.

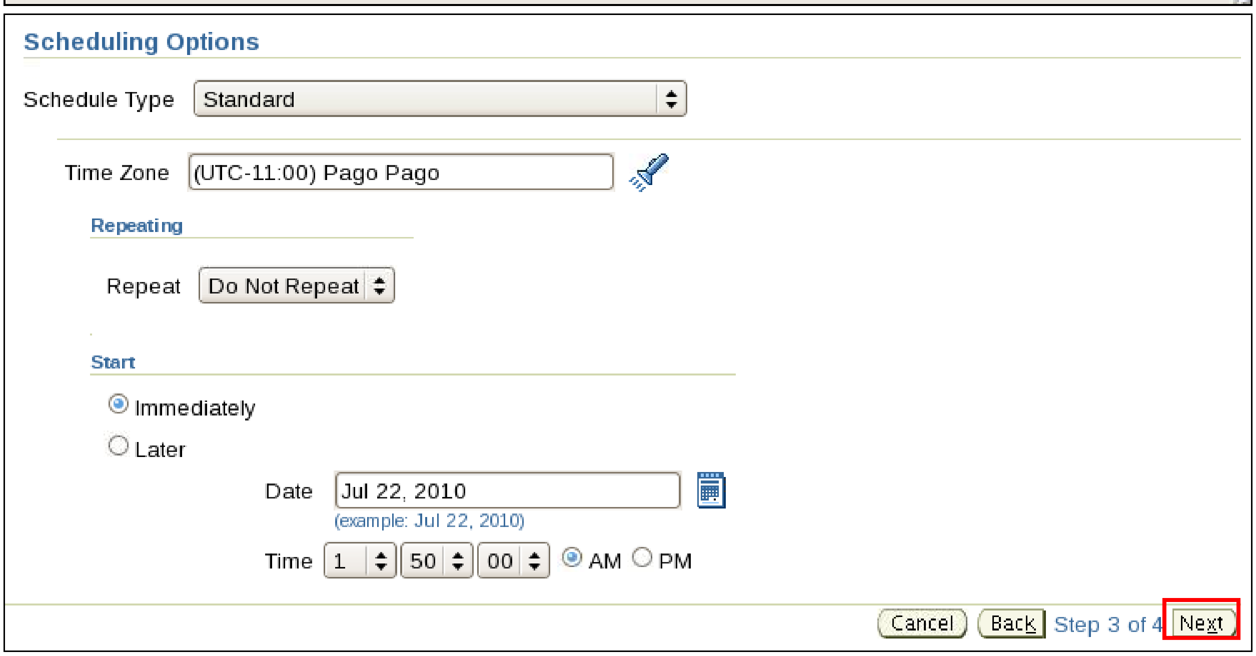


f. On the Schedule page, enter `MY_SQLACCESS_TASK` in the Task Name field.

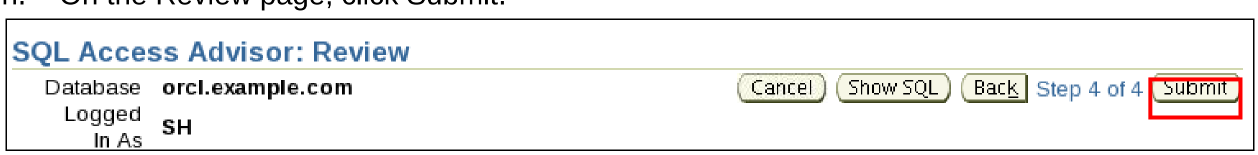


g. Select the first Time Zone from the provided list. (Click the search icon.) After this is done, click Next.





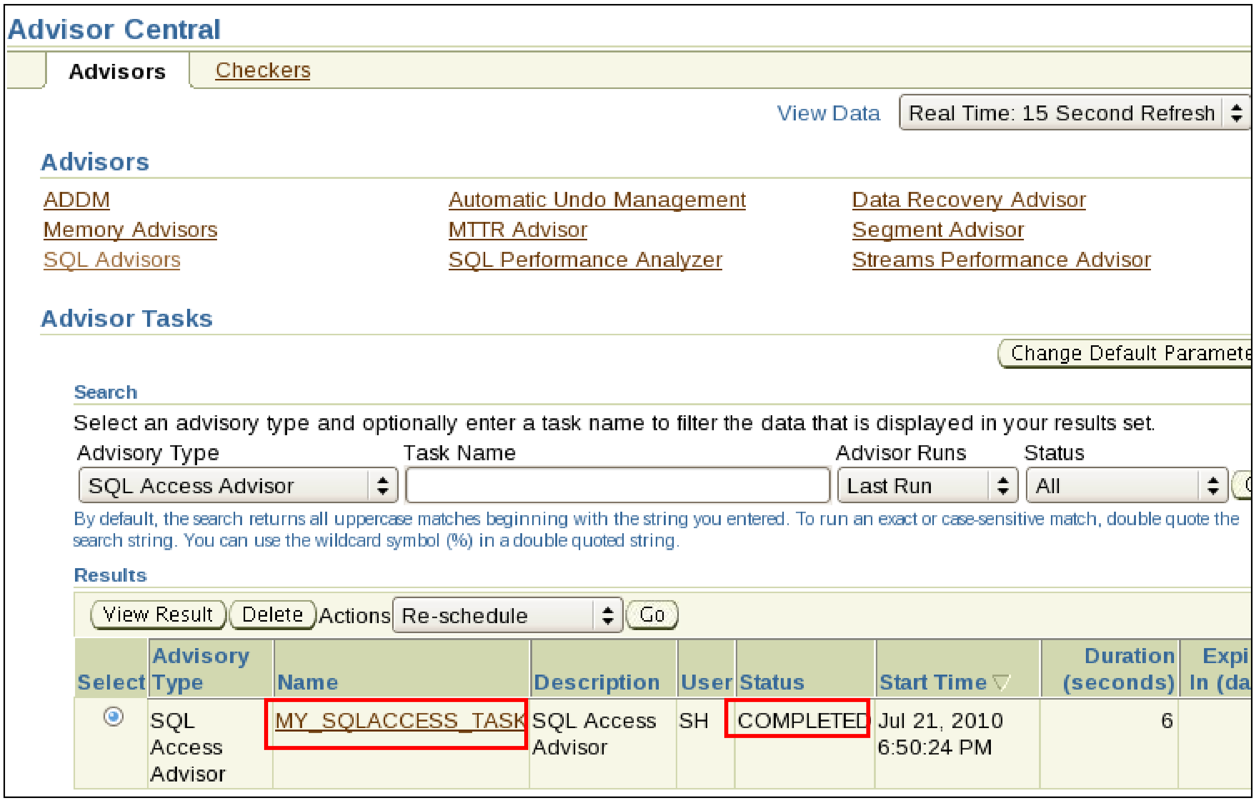
h. On the Review page, click Submit.



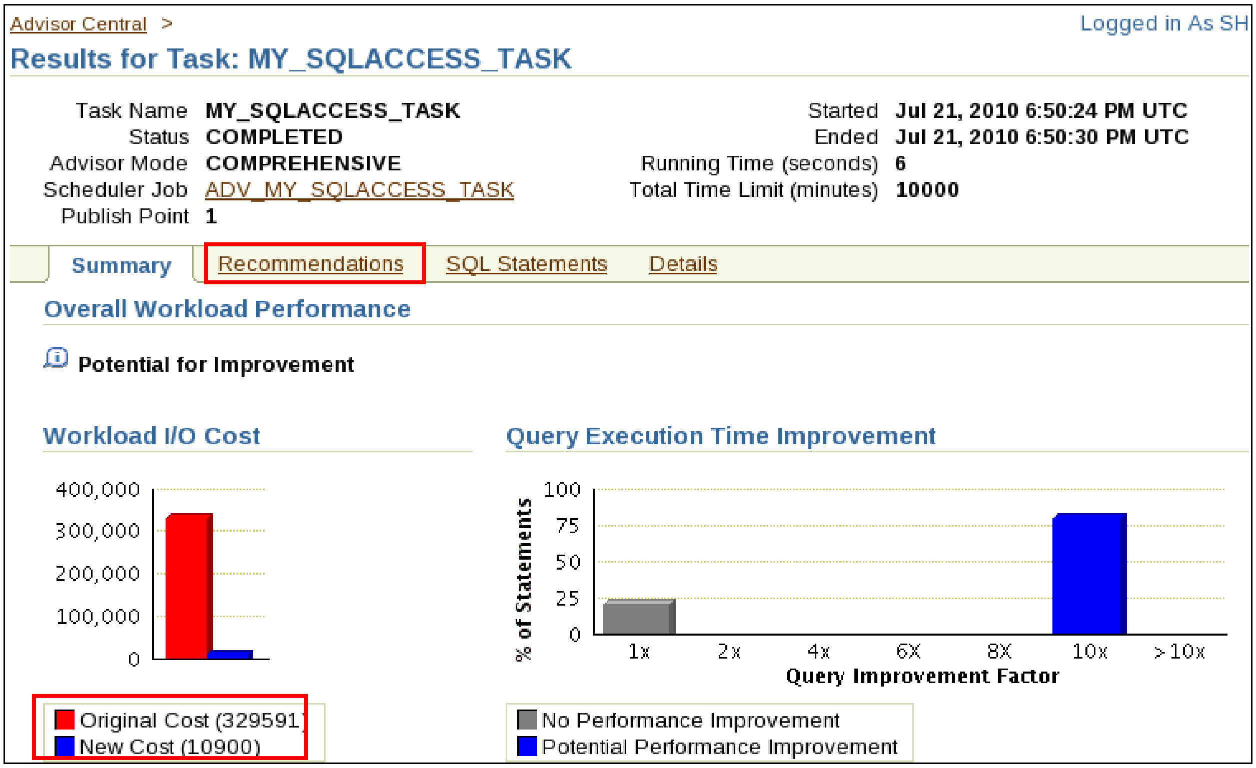
i. Back to the Advisor Central page, click Refresh until the Status of your task is COMPLETED.

3. After this is done, investigate the proposed recommendations:

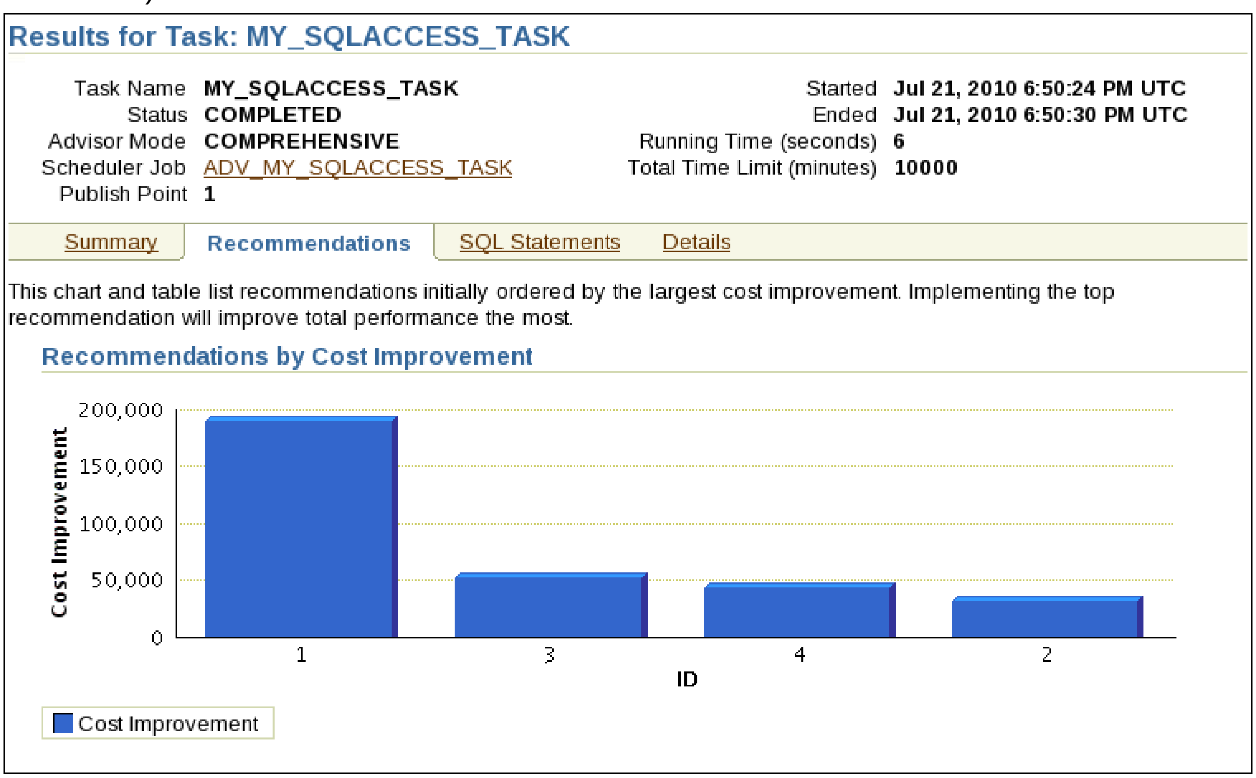
a. On the Advisor Central page, click the MY_SQLACCESS_TASK link in the Results table. The task should have COMPLETED as the status.



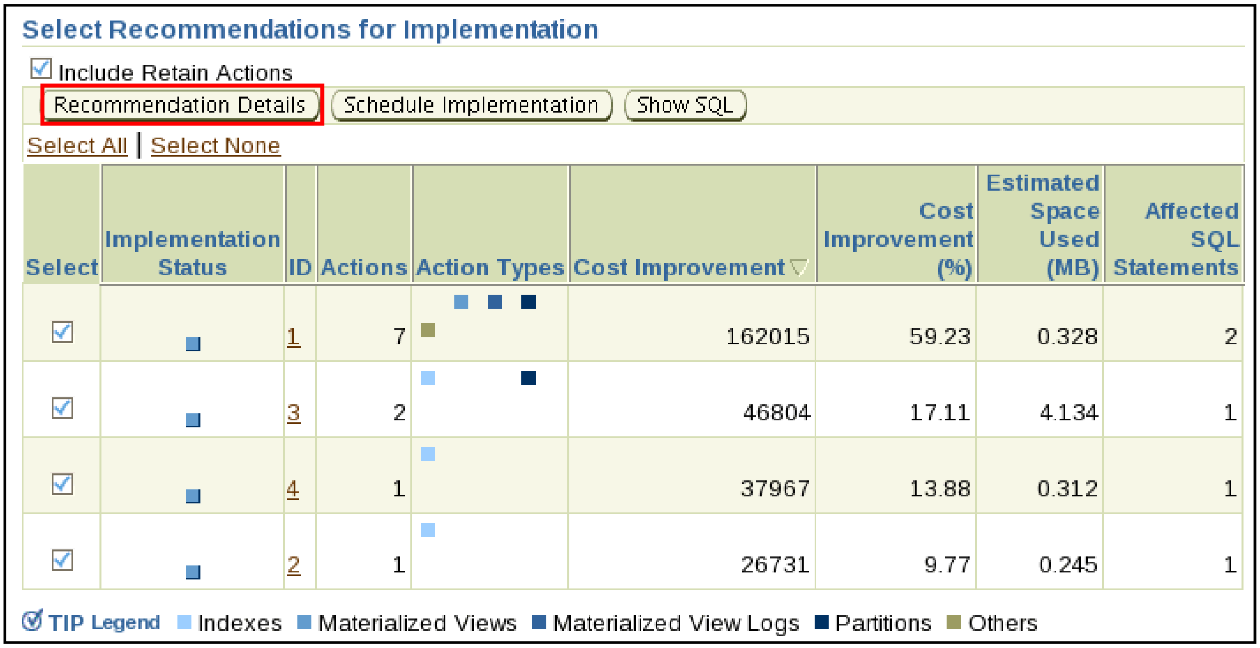
b. This takes you to the Results page. From this page, you can see the potential benefit of implementing the SQL Access Advisor recommendations on the workload. There should be a huge difference between the original and the new costs. Click the Recommendation subtab.



c. On the Recommendations subtab, you can see the high-level overview of the recommendations. Basically, all possible types of recommendations were generated for this workload (Indexes, Materialized Views, Materialized View Logs, Partitions, and Others).



d. Ensure that all recommendations are selected, and click the Recommendation Details button.



Select Recommendations for Implementation

☑ Include Retain Actions

Recommendation Details | Schedule Implementation | Show SQL

Select All | Select None

Select	Implementation Status	ID	Actions	Action Types	Cost Improvement ▽	Cost Improvement (%)	Estimated Space Used (MB)	Affected SQL Statements
☑		1	7		162015	59.23	0.328	2
☑		3	2		46804	17.11	4.134	1
☑		4	1		37967	13.88	0.312	1
☑		2	1		26731	9.77	0.245	1

TIP Legend Indexes Materialized Views Materialized View Logs Partitions Others

e. This takes you to the Details page, where you can see more details about each of the recommendations, as well as the corresponding SQL statements from the workload that are affected by these recommendations. You should see the following recommendations:

- Partition CUSTOMERS table.
- Create four materialized view log.
- Create a materialized view.
- Create one bitmap index.
- Create a function-based index.
- Create a B*-tree index on multiple columns.

Implementation Status	Recommendation IDs	Action	Object Name	Object Attributes	Indexed Co
■	1,3	PARTITION_TABLE	CUSTOMERS		
■	1	CREATE_MATERIALIZED_VIEW_LOG			
■	1	CREATE_MATERIALIZED_VIEW_LOG			
■	1	CREATE_MATERIALIZED_VIEW_LOG			
■	1	CREATE_MATERIALIZED_VIEW_LOG			
■	1	CREATE_MATERIALIZED_VIEW	MV$$_01E10000	General Match	
■	1	GATHER_TABLE_STATISTICS	MV$$_01E10000		
■	2	CREATE_INDEX	CUSTOMERSJFV_IDX$$_01E10000	BITMAP	CUST_STA
■	4	CREATE_INDEX	TEMPJFV_IDX$$_01E10001	BTREE,FUNCTION-BASED	ABS("C"
■	3	CREATE_INDEX	CUSTOMERS_IDX$$_01E10002	BTREE	COUNTRY CUST_CIT CUST_LAS

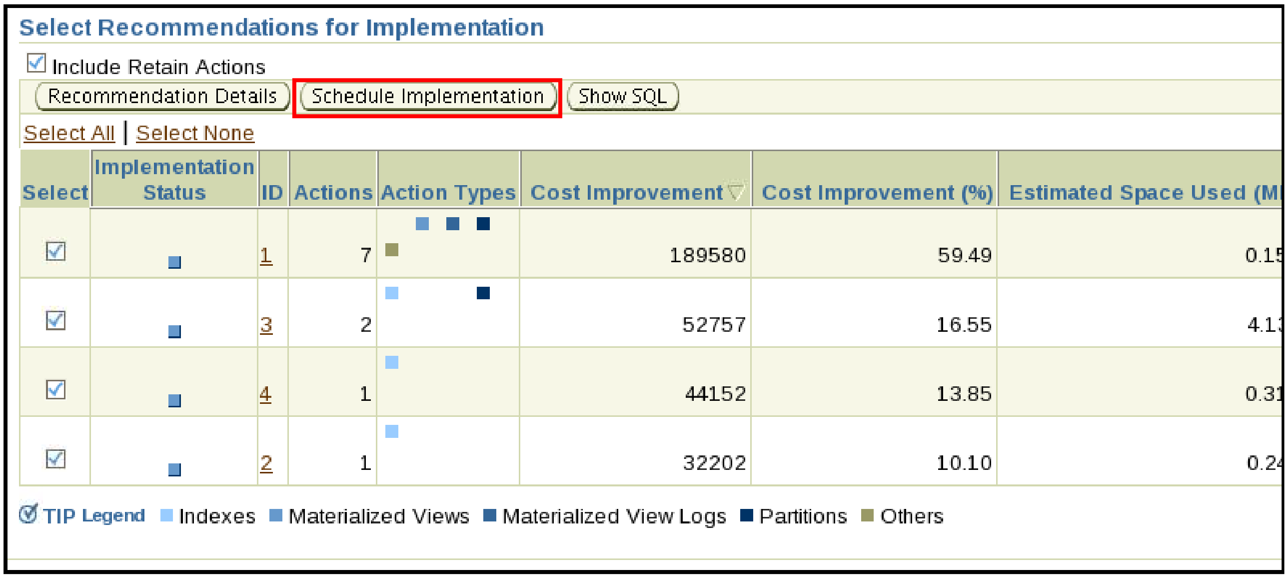
SQL Affected by Recommendations

Statement ID	Statement	Recommendation ID	Original Cost	New Cost	Cost Improvement	Cost Improvement (%)	Execution Count
2	SELECT /* QueryJFV 3 */ 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.c...	1	82411	1400	81011	98.30	100
1	SELECT /* QueryJFV 2 */ 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.c...	1	82404	1400	81004	98.30	100
3	SELECT /* QueryJFV 4 */ c.country_id, c.cust_city, c.cust_last_name FROM sh.customers c WHERE c.country_id in (52790, 52798) ORDER BY c.country_id, c.cust_city, c.cust_last_name	3	47404	600	46804	98.73	100
4	select /* func_indx */ count(*) from tempjfv where abs(c)=5	4	38167	200	37967	99.48	100
5	SELECT /* QueryJFV 5 */ * FROM sh.customersjfv WHERE cust_state_province = 'CA'	2	34831	8100	26731	76.74	100

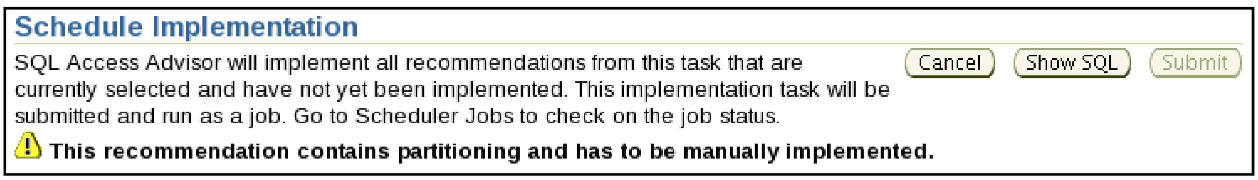
f. Click OK.

4. Try to implement the generated recommendations. What happens, and why?

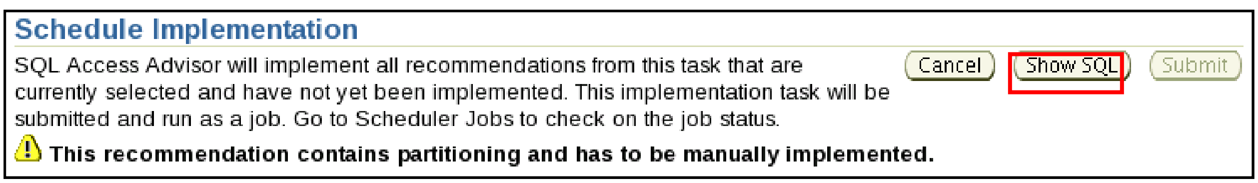
a. Back to the Recommendations subtab, click the Schedule Implementation button.



b. On the Schedule Implementation page, a warning is displayed indicating that the wizard will not try to implement its recommendations because some of them are very important changes that should be looked at closely by the administrator.



c. Click the Show SQL button to look at the script you could use to implement all recommendations. In fact, you already created this script and you will use it later in this lab. After you review the script, click Done.



Show SQL

Done

```
Rem SQL Access Advisor: Version 11.2.0.1.0 - Production
Rem
Rem Username: SH
Rem Task: MY_SQLACCESS_TASK
Rem Execution date:
Rem

Rem
Rem Repartitioning table "SH"."CUSTOMERS"
Rem

SET SERVEROUTPUT ON
SET ECHO ON

Rem
Rem Creating new partitioned table
Rem
CREATE TABLE "SH"."CUSTOMERS1"
(    "CUST_ID" NUMBER,
    "CUST_FIRST_NAME" VARCHAR2(20),
    "CUST_LAST_NAME" VARCHAR2(40),
    "CUST_GENDER" CHAR(1),
    "CUST_YEAR_OF_BIRTH" NUMBER(4,0),
```

d. Back on the Schedule Implementation page, click Cancel.

Schedule Implementation

SQL Access Advisor will implement all recommendations from this task that are currently selected and have not yet been implemented. This implementation task will be submitted and run as a job. Go to Scheduler Jobs to check on the job status.

Cancel | Show SQL | Submit

This recommendation contains partitioning and has to be manually implemented.

e. Click the Advisor Central locator link at the top of the "Results for Task" page.

f. On the Advisor Central page, select the SQL Access Advisor MY_SQLACCESS_TASK task and click Delete.

Results

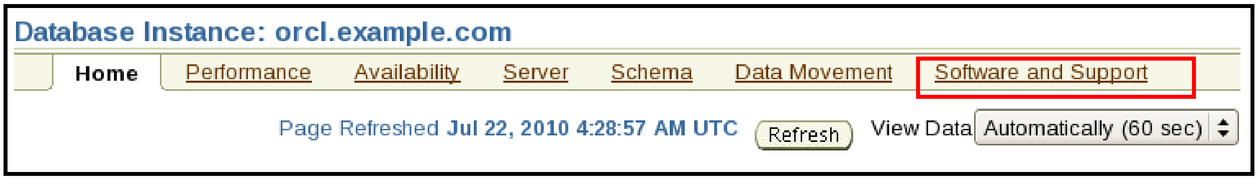
View Result | Delete | Actions Re-schedule | Go

Select	Advisory Type	Name	Description	User	Status	Start Time	Duration (seconds)	Expires In (days)
◉	SQL Access Advisor	MY_SQLACCESS_TASK	SQL Access Advisor	SH	COMPLETED	Jul 21, 2010 6:50:24 PM	6	30

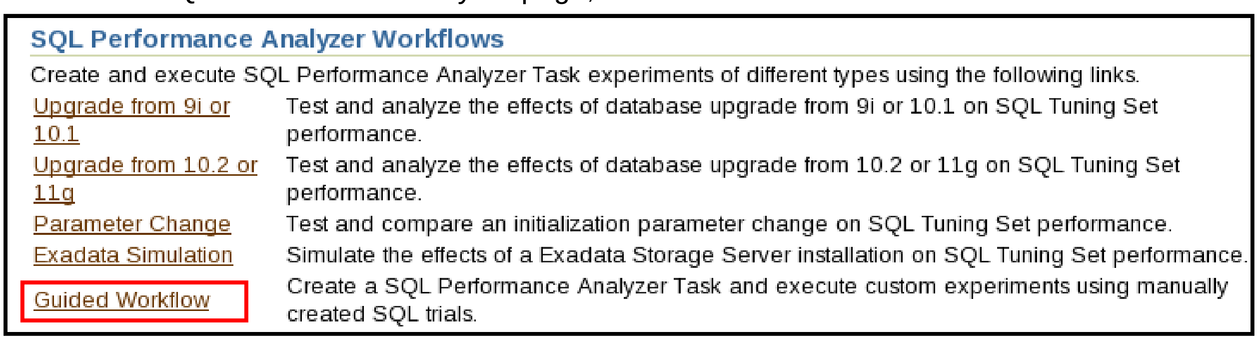
g. On the Information page, click Yes.

5. Use Enterprise Manager SQL Performance Analyzer to verify the performance improvement, if you implement the recommendations produced by SQL Access Advisor.

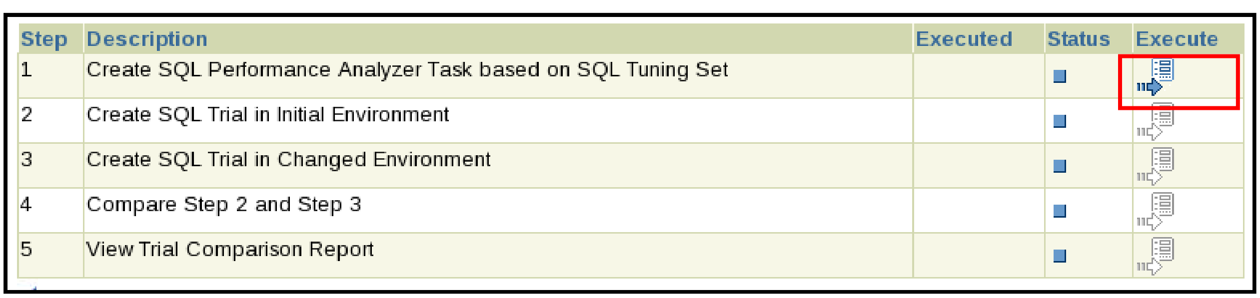
   a. Click the Database tab at the top-right corner, and then the "Software and Support" tab. On the "Software and Support" tabbed page, click the SQL Performance Analyzer link. You want to prove that implementing the recommendations is beneficial.



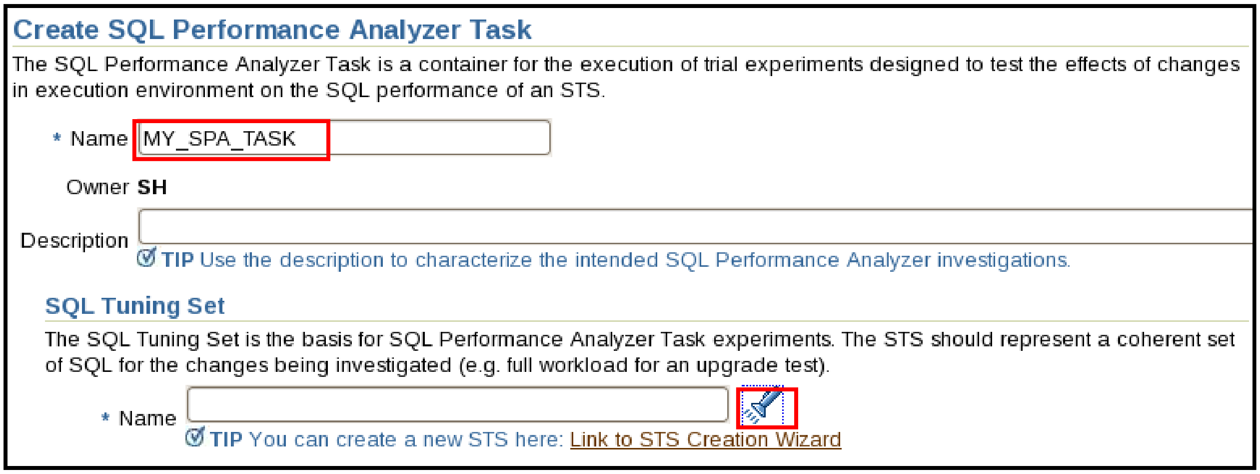
b. On the SQL Performance Analyzer page, click the Guided Workflow link.

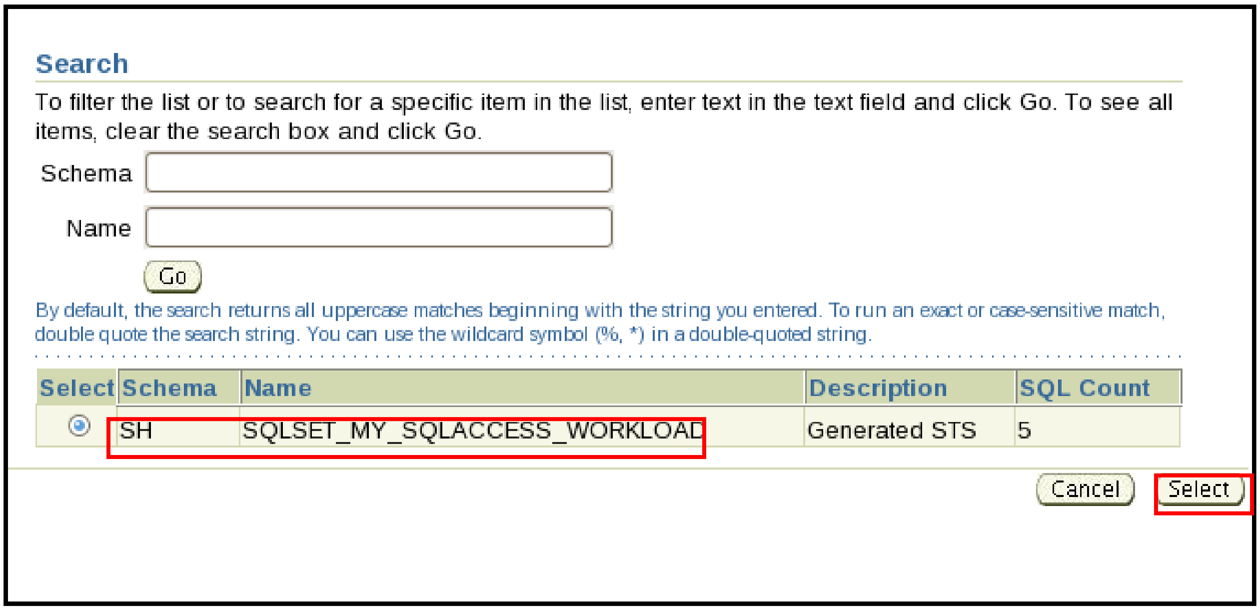


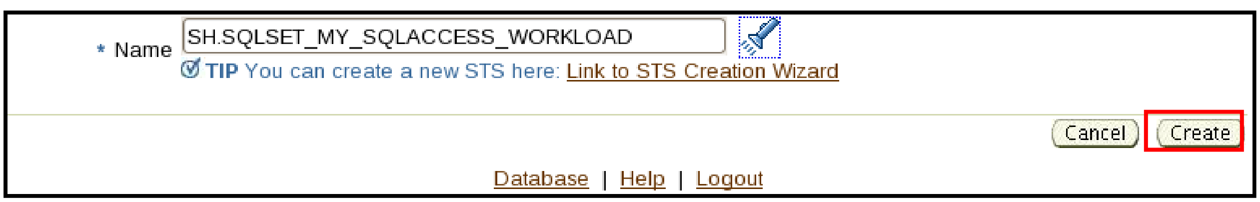
c. On the Guided Workflow page, click the Execute icon on the line corresponding to step 1.



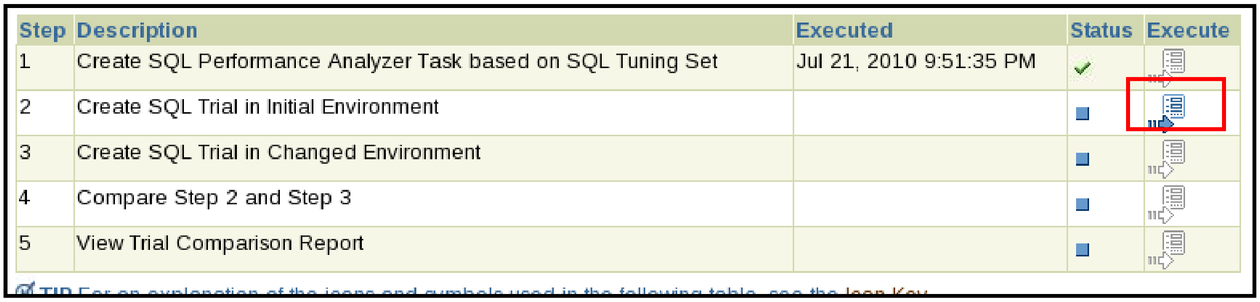
d. On the Create SQL Performance Analyzer Task page, enter `MY_SPA_TASK` in the SQL Performance Analyzer Task Name field. Then enter `SH.SQLSET_MY_SQLACCESS_WORKLOAD` in the SQL Tuning Set Name field. After this is done, click Create.



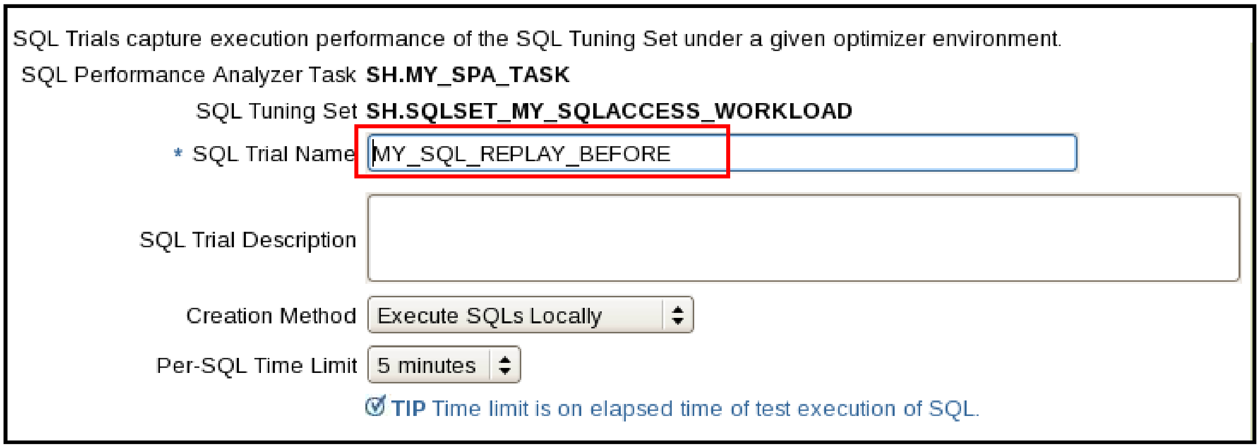


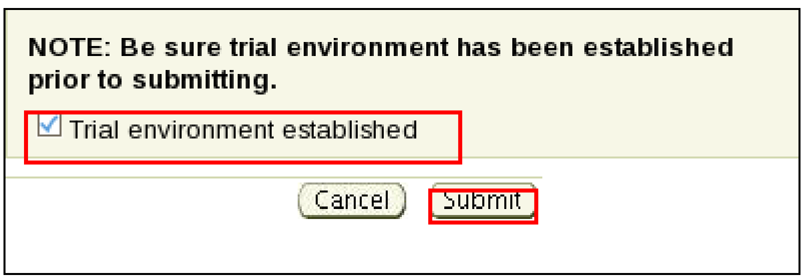


e. Back to the Guided Workflow page, click the Execute icon for step 2.



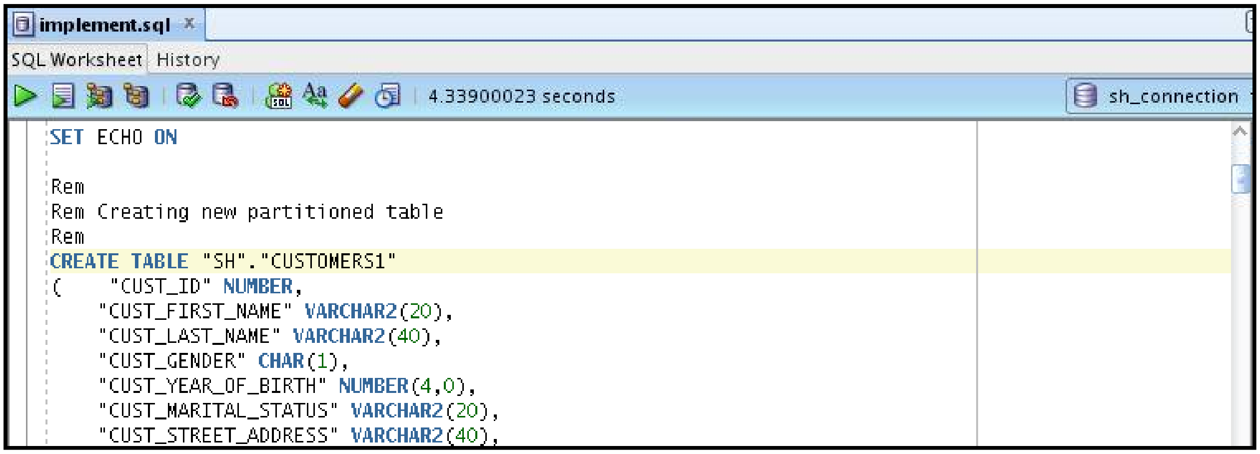
f. On the Create Replay Trial page, enter `MY_SQL_REPLAY_BEFORE` in the SQL Trial Name field, and ensure that you select the "Trial environment established" check box (on the right side of the page). Then click Submit.



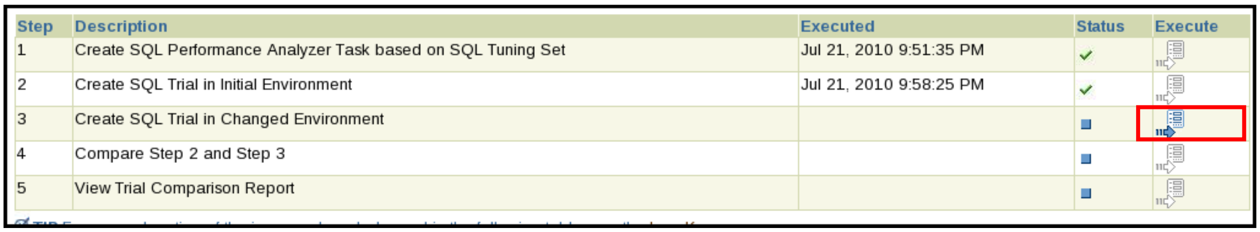


g. Wait on the Guided Workflow page until step 2 is completed.

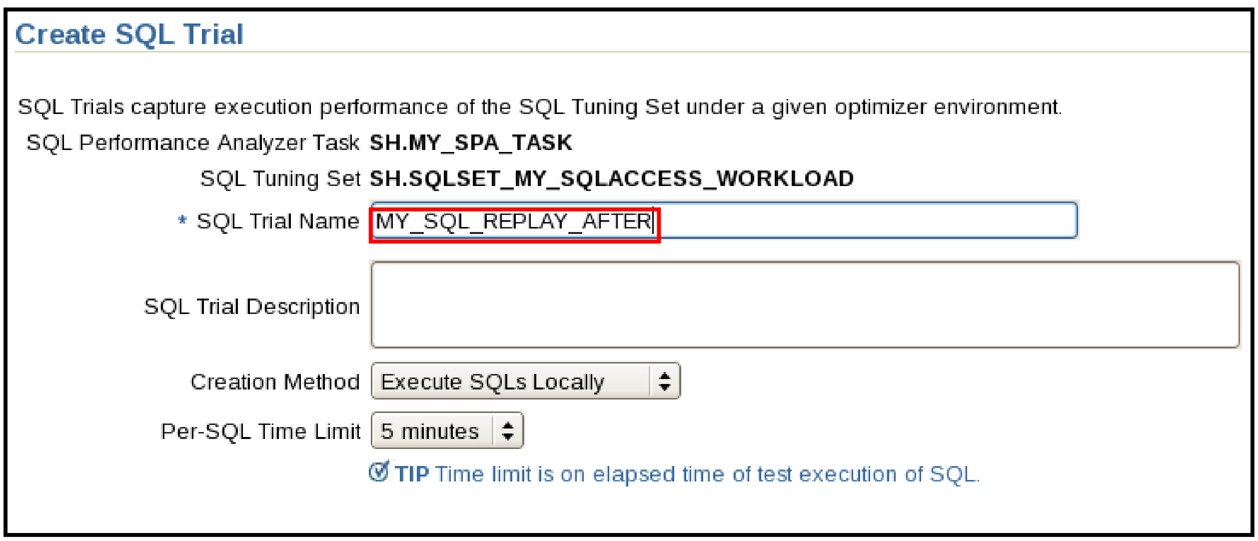
h. From SQL Developer, using `sh_connection`, execute the `/home/oracle/solutions/SQL_Access_Advisor/implement.sql` script. This script is a precreated script corresponding to the recommendations previously generated by your SQL Access Advisor session.

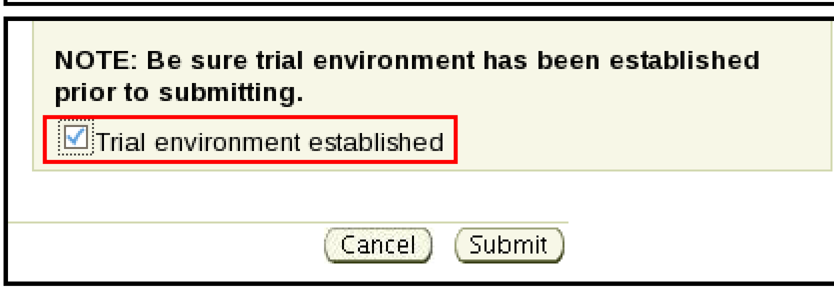


i. Back to your Guided Workflow page, click the Execute icon corresponding to step 3.



j. On the Create Replay Trial page, enter `MY_SQL_REPLAY_AFTER` in the SQL Trial Name field. Ensure that you select the "Trial environment established" check box, and click Submit.



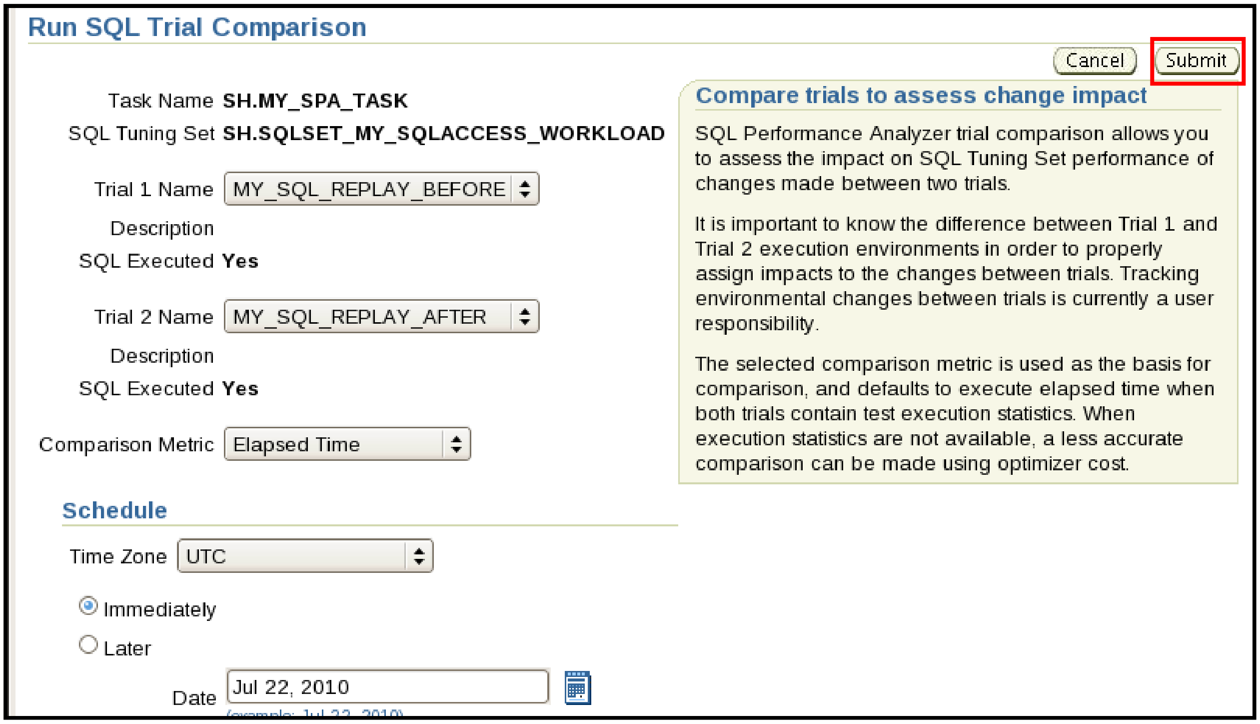


k. Wait until step 3 is completed.

l. Back to your Guided Workflow Enterprise Manager page, click the Execute icon corresponding to step 4.

Step	Description	Executed	Status	Execute
1	Create SQL Performance Analyzer Task based on SQL Tuning Set	Jul 21, 2010 9:51:35 PM		
2	Create SQL Trial in Initial Environment	Jul 21, 2010 9:58:25 PM		
3	Create SQL Trial in Changed Environment	Jul 22, 2010 5:07:29 AM		
4	Compare Step 2 and Step 3			
5	View Trial Comparison Report			

m. On the Run SQL Trial Comparison page, ensure that you create a comparison between MY_SQL_REPLAY_BEFORE and MY_SQL_REPLAY_AFTER. Click Submit.



n. Wait until step 4 is completed.

o. Back to your Guided Workflow Enterprise Manager page, click the Execute icon corresponding to step 5.

Step	Description	Executed	Status	Execute
1	Create SQL Performance Analyzer Task based on SQL Tuning Set	Jul 21, 2010 9:51:35 PM		
2	Create SQL Trial in Initial Environment	Jul 21, 2010 9:58:25 PM		
3	Create SQL Trial in Changed Environment	Jul 21, 2010 10:07:29 PM		
4	Compare Step 2 and Step 3	Jul 21, 2010 10:10:02 PM		
5	View Trial Comparison Report			

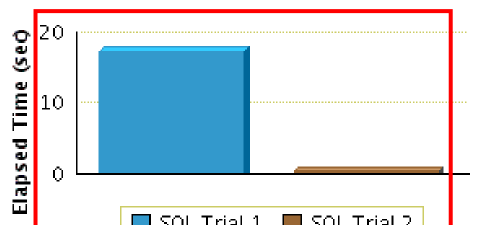
p. On the SQL Performance Analyzer Task Result page, you can clearly see that the second trial is much faster than the original one. You should see that all five SQL statements are improved in the second trial due to a changed execution plan.

SQL Performance Analyzer Task Report: SH.MY_SPA_TASK

SQL Tuning Set Name	SQLSET_MY_SQLACCESS_WORKLOAD	SQL Trial 1	**MY_SQL_REPLAY_BEFORE**
STS Owner	**SH**	SQL Trial 2	**MY_SQL_REPLAY_AFTER**
Total SQL Statements	**5**	Comparison Metric	**Elapsed Time**
SQL Statements With Errors	0	SQL Statements With Timeout	**0**
SQL Statements Unsupported	**0**		

Global Statistics

Projected Workload Elapsed Time

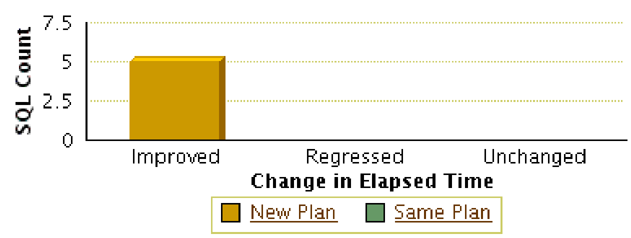


Improvement Impact 97%

Regression Impact 0%

Overall Impact 97%

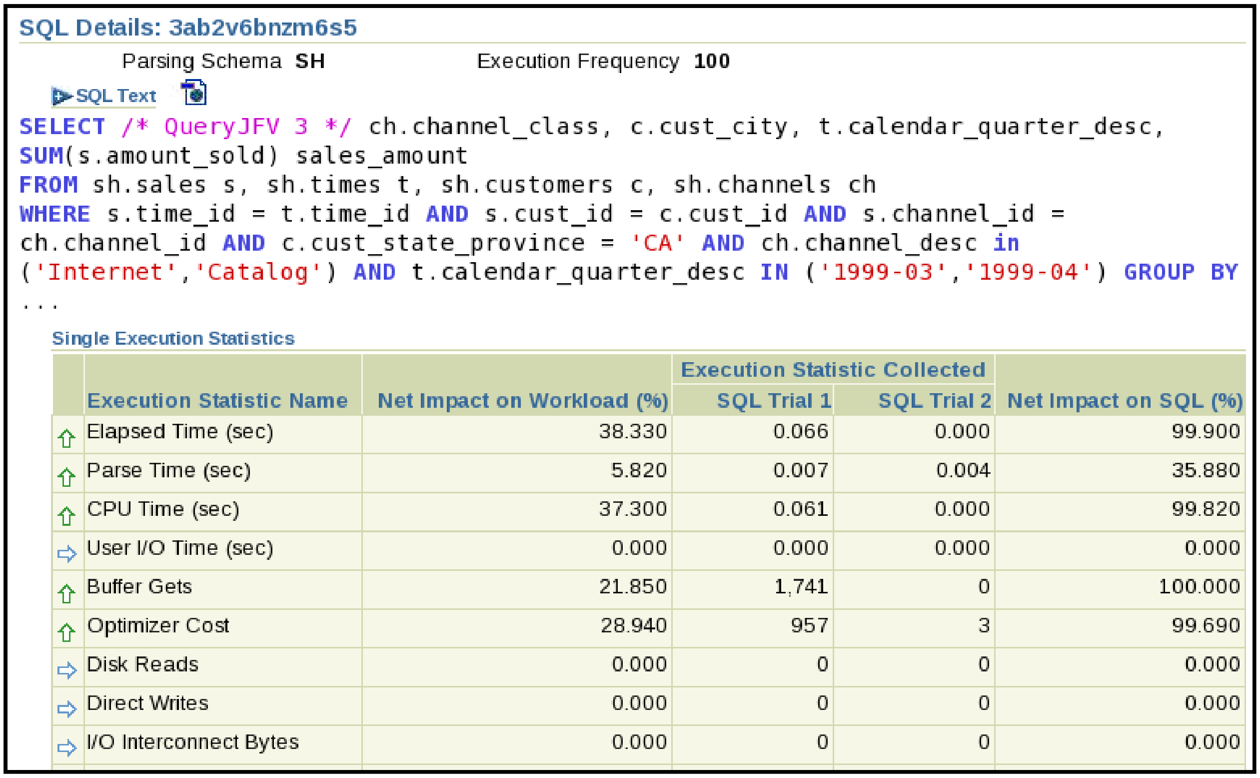
SQL Statement Count



q. To get more details, analyze the differences in the execution plan for all five statements. You can do so directly from the SQL Performance Analyzer Task Result page by clicking each SQL ID in the Top 10 table. Each time you click a SQL ID, you can see in the SQL Details section all statistics differences between the two trials as well as the differences in execution plans.

Top 10 SQL Statements Based on Impact on Workload

	SQL ID	Net Impact on Workload (%)	Elapsed Time (sec) SQL Trial 1	Elapsed Time (sec) SQL Trial 2	Net Impact on SQL (%)	New Plan
⇧	3ab2v6bnzm6s5	38.330	0.066	0.000	99.900	Y
⇧	277zymdg9vv44	33.370	0.057	0.000	99.870	Y
⇧	dvwqfgm3prdu4	16.010	0.028	0.000	99.860	Y
⇧	0fz7yspwpwjd1	5.310	0.014	0.005	66.410	Y
⇧	3d3agf621cgcj	4.150	0.007	0.000	99.370	Y



SQL Details: 3ab2v6bnzm6s5

Parsing Schema **SH** Execution Frequency **100**

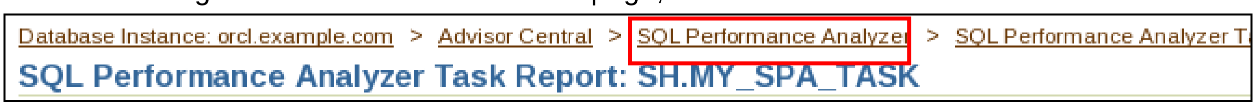
SQL Text

```
SELECT /* QueryJFV 3 */ 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 ('1999-03','1999-04') GROUP BY
...
```

Single Execution Statistics

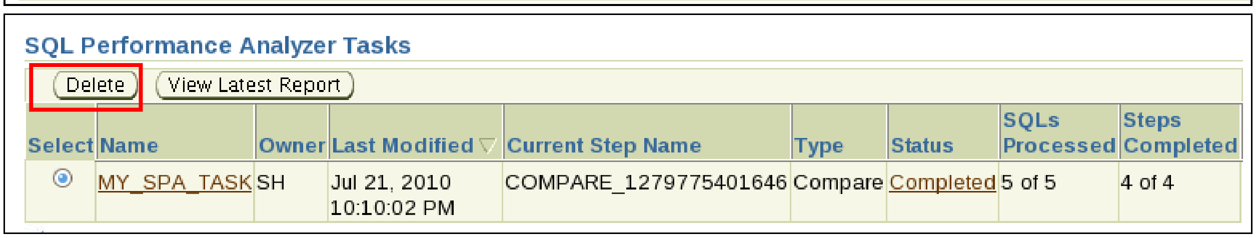
			Execution Statistic Collected		
	Execution Statistic Name	Net Impact on Workload (%)	SQL Trial 1	SQL Trial 2	Net Impact on SQL (%)
⇧	Elapsed Time (sec)	38.330	0.066	0.000	99.900
⇧	Parse Time (sec)	5.820	0.007	0.004	35.880
⇧	CPU Time (sec)	37.300	0.061	0.000	99.820
⇨	User I/O Time (sec)	0.000	0.000	0.000	0.000
⇧	Buffer Gets	21.850	1,741	0	100.000
⇧	Optimizer Cost	28.940	957	3	99.690
⇨	Disk Reads	0.000	0	0	0.000
⇨	Direct Writes	0.000	0	0	0.000
⇨	I/O Interconnect Bytes	0.000	0	0	0.000

r. After this is done, go back to the SQL Performance Analyzer page (Home > Software and Support > SQL Performance Analyzer), and delete MY_SPA_TASK by selecting it and clicking Delete. On the Confirmation page, click Delete.



Database Instance: orcl.example.com > Advisor Central > SQL Performance Analyzer > SQL Performance Analyzer T

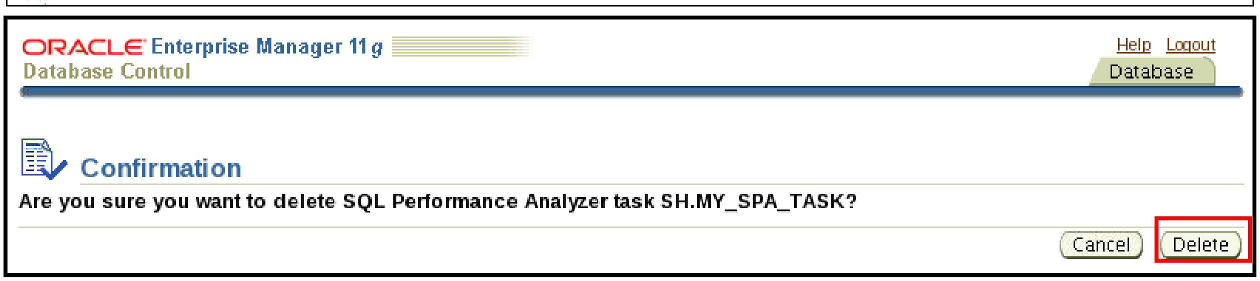
SQL Performance Analyzer Task Report: SH.MY_SPA_TASK



SQL Performance Analyzer Tasks

Delete | View Latest Report

Select	Name	Owner	Last Modified ▽	Current Step Name	Type	Status	SQLs Processed	Steps Completed
◉	MY_SPA_TASK	SH	Jul 21, 2010 10:10:02 PM	COMPARE_1279775401646	Compare	Completed	5 of 5	4 of 4

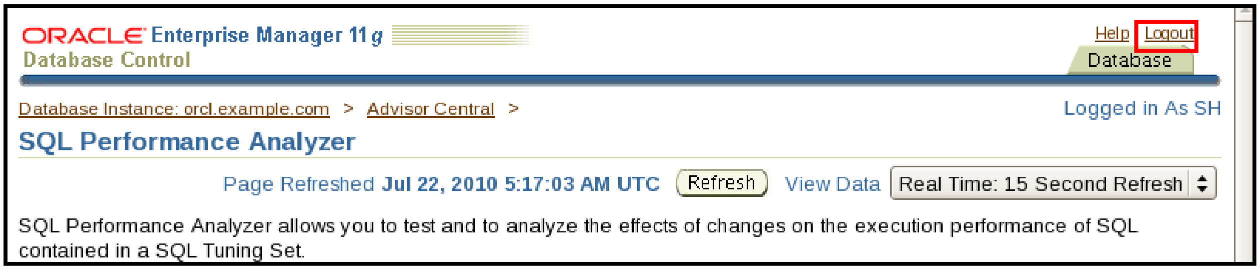


Confirmation

Are you sure you want to delete SQL Performance Analyzer task SH.MY_SPA_TASK?

Cancel | Delete

s. Log out from Enterprise Manager. **This is important. The `sh_cleanup.sh` script will fail if you do not log out.**



6. From SQL Developer, execute the `revert.sql` script using `sys_connection`.

Output

```
line 3: SQLPLUS Command Skipped: SET NUMWIDTH 10
line 4: SQLPLUS Command Skipped: SET LINESIZE 8000
line 5: SQLPLUS Command Skipped: SET TRIMSPOOL ON
line 6: SQLPLUS Command Skipped: SET TAB OFF
line 7: SQLPLUS Command Skipped: SET PAGESIZE 100
line 8: SQLPLUS Command Skipped: SET LONG 1000
grant dba succeeded.
 alter user sh succeeded.
Connected
variable norecs number

Rem Clean up

declare
    name varchar2(30);
    cursor name_cur1 is
      select task_name from user_advisor_templates
        where task_name like '%SQLACCESS%';
  begin
    -----------------------------------------------------------------
      Get rid of templates, tasks and workloads.
    -----------------------------------------------------------------

    open name_cur1;

    loop
```

```
      fetch name_cur1 into name;
      exit when name_cur1%NOTFOUND;


dbms_advisor.update_task_attributes(name,null,null,'FALSE','FALSE');
      dbms_advisor.delete_task(name);
    end loop;

    close name_cur1;
  end;

anonymous block completed
Rem make a temp table

DROP TABLE temp_table purge

 DROP TABLE temp_table succeeded.
alter system flush shared_pool

 alter system flush succeeded.
drop table tempjfv purge

 drop table tempjfv succeeded.
drop table customersjfv purge

 drop table customersjfv succeeded.
execute dbms_advisor.delete_task('%')

anonymous block completed
execute dbms_advisor.delete_sqlwkld('%')

anonymous block completed
execute dbms_sqltune.drop_sqlset('SQLSET_MY_SQLACCESS_WORKLOAD')

anonymous block completed
EXECUTE DBMS_STATS.UNLOCK_SCHEMA_STATS('SH')

anonymous block completed
DROP MATERIALIZED VIEW LOG ON "SH"."CUSTOMERS"

 DROP MATERIALIZED VIEW succeeded.
DROP MATERIALIZED VIEW LOG ON "SH"."CHANNELS"

 DROP MATERIALIZED VIEW succeeded.
DROP MATERIALIZED VIEW LOG ON "SH"."TIMES"
```

```
 DROP MATERIALIZED VIEW succeeded.
DROP MATERIALIZED VIEW LOG ON "SH"."SALES"

 DROP MATERIALIZED VIEW succeeded.
DROP MATERIALIZED VIEW "SH"."MV_01DF0000"

 DROP MATERIALIZED VIEW succeeded.
DROP INDEX "SH"."CUSTOMERS_IDX_01DF0002"

 DROP INDEX "SH"."CUSTOMERS_IDX_01DF0002" succeeded.
DROP TABLE "SH"."CUSTOMERS" PURGE

 DROP TABLE "SH"."CUSTOMERS" succeeded.
DROP TABLE "SH"."CUSTOMERS11" CASCADE CONSTRAINTS PURGE

 DROP TABLE "SH"."CUSTOMERS11" succeeded.
```

7. In SQL Developer, close all panes connected as `sh_connection`, and disconnect `sh_connection` in the Connections pane.
8. Close all SQL*Plus sessions that might be connected using the `SH` user.
9. From a terminal session, execute the following commands to return to the situation you were in before you started the lab. If the “User dropped” feedback 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 connected as the `SH` user, and then execute this script again.

```
$ ./sh_cleanup.sh
…
Connected to:…


SQL>
SQL> @sh_main sh example temp oracle_4U
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_histor
y/ /home/oracle/ v3
SQL> Rem
SQL> Rem $Header: sh_main.sql 06-mar-2008.15:00:45 cbauwens Exp
$
SQL> Rem
SQL> Rem sh_main.sql
SQL> Rem
SQL> Rem Copyright (c) 2001, 2008, Oracle. All rights reserved.
SQL> Rem
SQL> Rem    NAME
SQL> Rem          sh_main.sql - Main schema creation and load
script
SQL> Rem
SQL> Rem    DESCRIPTION
```

```
SQL> Rem         SH is the Sales History schema of the Oracle
Sample
SQL> Rem        Schemas
SQL> Rem
SQL> Rem    NOTES
SQL> Rem        CAUTION: use absolute pathnames as parameters 5
and 6.
SQL> Rem        Example (UNIX) echo
$ORACLE_HOME/demo/schema/sales_history
SQL> Rem        Please make sure that parameters 5 and 6 are
specified
SQL> Rem        INCLUDING the trailing directory delimiter,
since the
SQL> Rem        directory parameters and the filenames are
concatenated
SQL> Rem        without adding any delimiters.
SQL> Rem        Run this as SYS or SYSTEM
SQL> Rem
SQL> Rem    MODIFIED   (MM/DD/YY)
SQL> Rem        cbauwens        03/06/08 - NLS settings for load
SQL> Rem        cbauwens        07/10/07 - NLS fix bug 5684394
SQL> Rem        glyon   06/28/07 - grant CWM_USER role, if it
exists
SQL> Rem        cbauwens        02/23/05 - 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:


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 14

Chapter 14

Overview of Practices for Lesson 14

Practices Overview

In this practice, you will use the Automatic SQL Tuning task to tune a small workload.

Practice 14-1: Using Automatic SQL Tuning

In this practice, you manually launch Automatic SQL Tuning to automatically tune a small application workload. You then investigate the outcomes and configuration possibilities.

1. Log in to Enterprise Manager. Use the URL https://localhost:1158/em to start EM in your browser if you have not already done so. Log in as `SYS` with the password `oracle_4U`, and connect as `SYSDBA`.
2. In Enterprise Manager, on the Server page, click Automated Maintenance Tasks, check that Status is set to Enabled, and click Configure. Click the Configure button next to Automatic SQL Tuning. Select Yes for "Automatic Implementation of SQL Profiles." Then click Apply. Execute the `ast_setup.sh` script from a terminal window connected as the `oracle` user. This turns off automatic maintenance tasks, and drops any existing profiles on queries executed by the `AST` user. The `AST` user used throughout this practice was created in the classroom setup script.

```
$ cd $HOME/solutions/Automatic_SQL_Tuning

$ ./ast_setup.sh

…
Connected to:…
SQL> SQL> SQL> SQL>
System altered.

SQL> SQL> SQL> SQL> SQL> SQL>
System altered.

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

SQL> SQL> SQL> SQL> SQL> SQL>   2    3    4    5    6    7    8    9
PL/SQL procedure successfully completed.

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

SQL> SQL> Disconnected …
$

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


#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

sqlplus / as sysdba <<FIN!
```

```
set echo on

alter system flush shared_pool;

--
-- Turn off AUTOTASK
--

alter system set "_enable_automatic_maintenance"=0;

--
-- Clear out old executions of auto-sqltune
--

exec dbms_sqltune.reset_tuning_task('SYS_AUTO_SQL_TUNING_TASK');

--
-- Drop any profiles on AST queries
--

declare
  cursor prof_names is
    select name from dba_sql_profiles where sql_text like '%AST%';
begin
  for prof_rec in prof_names loop
    dbms_sqltune.drop_sql_profile(prof_rec.name);
  end loop;
end;
/

-- Set AST to automatically implement profiles

exec DBMS_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER(
'ACCEPT_SQL_PROFILES', 'TRUE');

FIN!
```

3. In preparation for the practice, you should execute a workload. Execute the `run_workload_stream.sh` script. This script executes, multiple times, a query that is not correctly optimized. The query in question uses hints that force the optimizer to pick a suboptimal execution plan. The script executes for approximately 30 seconds.

```
$ ./run_workload_stream.sh
Mon Jul 26 05:37:37 GMT 2010

…

SQL> SQL> SQL> SQL>
```

```
no rows selected

SQL>
no rows selected

SQL>
no rows selected

...

SQL>
no rows selected

SQL>
no rows selected

SQL>
no rows selected

SQL>
no rows selected

SQL>
no rows selected

SQL> SQL> Disconnected …

Mon Jul 26 05:38:02 GMT 2010
$

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

#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

date

sqlplus ast/ast <<FIN!

set echo on

select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ 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;
```

```
…

select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ 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;

FIN!

date
```

4. Automatic SQL Tuning is implemented using an automated task that runs during maintenance windows. However, you do not wait for the next maintenance window to open. Instead, you force the opening of your next maintenance window now. This automatically triggers the Automatic SQL Tuning task. Execute the `run_ast.sh` script to open your next maintenance window now. The execution of the script may take up to 6 minutes.

```
$ ./run_ast.sh
Mon Jul 26 05:39:56 GMT 2010
…

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

SQL> SQL> SQL> SQL>   2    3    4
PL/SQL procedure successfully completed.

SQL> SQL>
WINDOW
----------------------------------------------------------------------
----------
TUESDAY_WINDOW

SQL> SQL> SQL> SQL> SQL> SQL>   2
System altered.

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

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

SQL> SQL>
PL/SQL procedure successfully completed.

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

SQL> SQL>   2    3    4    5    6    7    8    9   10   11   12   13
14   15   16   17   18   19   20   21   22
```

```
PL/SQL procedure successfully completed.

SQL> SQL>   2
System altered.

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

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

SQL> SQL> SQL> Disconnected …

Mon Jul 26 05:47:00 GMT 2010
$

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

#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

date

sqlplus / as sysdba <<FIN!

set echo on

exec dbms_workload_repository.create_snapshot;

variable window varchar2(20);

begin
select upper(to_char(sysdate,'fmday'))||'_WINDOW' into :window from
dual;
end;
/

print window;

--
-- Open the corresponding maintenance window, but with other clients
disabled
--

alter system set "_enable_automatic_maintenance"=1
```

```
/

exec dbms_auto_task_admin.disable( -
  'auto optimizer stats collection', null, :window);

exec dbms_auto_task_admin.disable( -
  'auto space advisor', null, :window);

exec dbms_scheduler.open_window(:window, null, true);

--
-- Close the maintenance window when sqltune is done
--

exec dbms_lock.sleep(60);

declare
  running number;
begin

  loop
    select count(*)
    into   running
    from   dba_advisor_executions
    where  task_name = 'SYS_AUTO_SQL_TUNING_TASK' and
           status = 'EXECUTING';

    if (running = 0) then
      exit;
    end if;

    dbms_lock.sleep(60);
  end loop;

  dbms_scheduler.close_window(:window);

end;
/

alter system set "_enable_automatic_maintenance"=0
/

--
-- Re-enable the other guys so they look like they are enabled in EM.
-- Still they will be disabled because we have set the underscore.
--
```

```
exec dbms_auto_task_admin.enable( -
  'auto optimizer stats collection', null, :window);

exec dbms_auto_task_admin.enable( -
  'auto space advisor', null, :window);


FIN!

date
```

5. Execute the `run_workload_stream.sh` script again. What do you observe?
   a. You should see that the execution time for `run_workload_stream.sh` is much faster than the original execution. This is probably due to the fact that Automatic SQL Tuning implemented a profile for your statement automatically.

```
$ ./run_workload_stream.sh
Mon Jul 26 05:48:13 GMT 2010

…

SQL> SQL> SQL> SQL>
no rows selected

SQL>
no rows selected

SQL>
no rows selected

...

SQL>
no rows selected

SQL>
no rows selected

SQL>
no rows selected

SQL> SQL> Disconnected …

Mon Jul 26 05:48:13 GMT 2010
$
```

```
------------------------------------------------------------
#!/bin/bash

cd /home/oracle/solutions/AST

…

date

sqlplus ast/ast <<FIN!

set echo on

select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ 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;

…

select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ 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;

FIN!

date
```

6. Force the creation of an Automatic Workload Repository (AWR) snapshot.

```
./create_snapshot.sh

…

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

SQL> SQL> Disconnected …

$

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

#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

sqlplus / as sysdba <<FIN!
```

```
set echo on

exec dbms_workload_repository.create_snapshot;

FIN!
```

7. How would you confirm that a SQL Profile was automatically implemented?
   a. In Oracle Enterprise Manager, locate the Automatic SQL Tuning summary page under **Server > Automated Maintenance Tasks > Automatic SQL Tuning**. The task has already run in one maintenance window and has results ready to be viewed.
   b. On the Automatic SQL Tuning Result Summary page, view the tuning results.
   c. Look at the graphs on the Summary page.
   d. Focus on understanding the pie chart and the bar graph next to it. You should be able to get a feeling for the general finding breakdown, as well as the number of SQL profiles implemented by the task.
   e. Click View Report to see a detailed SQL-level report. Find the SQL that ran in the `AST` schema. Note the green check mark, which indicates that the profile was implemented.
   f. Select the SQL corresponding to the profile that was implemented, and then click View Recommendations.
   g. Click the Compare Explain Plans eyeglass icon for the SQL Profile entry.
   h. View the old and new explain plans for the query.
   i. Then click the "Recommendations for SQL_ID" locator link to return to the previous screen.
   j. Investigate a SQL profile. While still on the "Recommendations for SQL_ID" page, click the SQL text to go to the SQL Details page for this SQL.
   k. This takes you to the Tuning History tab. Note the link to `SYS_AUTO_SQL_TUNING_TASK` that is there to show that the SQL was tuned by this tuning task.
   l. Look at the Plan Control subpage and note that a profile was created automatically for this SQL. The `AUTO` type means it was automatically created.
   m. Click the Statistics tab to take a look at the execution history for this SQL.
   n. Depending on the speed of your machine, you may not see two hash values. If that is the case, ignore this step and the following one. Select Real Time: Manual Refresh from the View Data and then each of possible two Plan Hash Values from the corresponding drop-down list. Choose one after the other and wait for the page to refresh each time.
   o. Depending on the speed of your environment, you should see one statement with a relatively high elapsed time per execution, and one with very low elapsed time per execution. This shows the improved plan. If you select All from the Plan Hash Values drop-down list, you might not be able to see the execution corresponding to the statement after tuning in the Summary graph. This might be because the workload was too short to capture.
8. Generate a text report for more in-depth information. From the command line, execute the `get_task_report.sh` script. What do you observe?

a. Note the first queries that fetch execution name and object number from the advisor schema, followed by the final query that gets the text report. In the text report, look for the section about the SQL profile finding and peruse the Validation Results section. This shows you the execution statistics observed during test-execute and allows you to get a better idea about the quality of the profile. You can also use the `report_auto_tuning_task` API to get reports that span multiple executions of the task.

```
$ ./get_task_report.sh

…

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL>
Session altered.

SQL> SQL>   2    3    4
EXECUTION_NAME                 STATUS      EXECUTION_START
------------------------------ ----------- -------------------
EXEC_142                       COMPLETED   07/26/2010 04:56:20

SQL> SQL> SQL> SQL>   2    3    4    5    6    7
PL/SQL procedure successfully completed.

SQL> SQL>
LAST_EXEC
----------------------------------------------------------------------
EXEC_142

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL>   2    3    4    5    6    7
8    9   10
PL/SQL procedure successfully completed.

SQL> SQL>
    OBJ_ID
----------
         5

SQL> SQL> SQL> SQL> SQL> SQL>   2    3  GENERAL INFORMATION SECTION
----------------------------------------------------------------------
---------
Tuning Task Name                        : SYS_AUTO_SQL_TUNING_TASK
Tuning Task Owner                       : SYS
Workload Type                           : Automatic High-Load SQL
Workload
Scope                                   : COMPREHENSIVE
Global Time Limit(seconds)              : 3600
Per-SQL Time Limit(seconds)             : 1200
Completion Status                       : COMPLETED
```

```
Started at                              : 08/12/2010 08:16:20
Completed at                            : 08/12/2010 08:16:30
Number of Candidate SQLs                : 7
Cumulative Elapsed Time of SQL (s)      : 28

-------------------------------------------------------------------------------
---------
Object ID  : 3
Schema Name: AST
SQL ID     : by9m5m597zh19
SQL Text   : select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ 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

-------------------------------------------------------------------------------
---------
FINDINGS SECTION (2 findings)
-------------------------------------------------------------------------------
---------

1- SQL Profile Finding (see explain plans section below)
--------------------------------------------------------
  A potentially better execution plan was found for this statement.
  SQL profile "SYS_SQLPROF_012a655f13c40001" was created automatically
for
  this statement.

  Recommendation (estimated benefit: 98.47%)
  ------------------------------------------
  - An automatically-created SQL profile is present on the system.
    Name:   SYS_SQLPROF_012a655f13c40001
    Status: ENABLED

  Validation results
  ------------------
  The SQL profile was tested by executing both its plan and the
original plan
  and measuring their respective execution statistics. A plan may have
been
  only partially executed if the other could be run to completion in
less time.

                           Original Plan  With SQL Profile  % Improved
                           -------------  ----------------  ----------
  Completion Status:            COMPLETE          COMPLETE
  Elapsed Time(us):               180549               202      99.88 %
```

```
  CPU Time(us):                      180572                 200      99.88 %
  User I/O Time(us):                      0                   0
  Buffer Gets:                         2541                  39      98.46 %
  Physical Read Requests:                 0                   0
  Physical Write Requests:                0                   0
  Physical Read Bytes:                    0                   0
  Physical Write Bytes:                   0                   0
  Rows Processed:                         0                   0
  Fetches:                                0                   0
  Executions:                             1                   1

  Notes
  -----
  1. The original plan was first executed to warm the buffer cache.
  2. Statistics for original plan were averaged over next 4
executions.
  3. The SQL profile plan was first executed to warm the buffer cache.
  4. Statistics for the SQL profile plan were averaged over next 9
executions.

2- Index Finding (see explain plans section below)
--------------------------------------------------
  The execution plan of this statement can be improved by creating one
or more
  indices.

  Recommendation (estimated benefit: 90.97%)
  ------------------------------------------
  - Consider running the Access Advisor to improve the physical schema
design
    or creating the recommended index.
    create index SH.IDX$$_00010001 on SH.SALES("CUST_ID");

  Rationale
  ---------
    Creating the recommended indices significantly improves the
execution plan
    of this statement. However, it might be preferable to run "Access
Advisor"
    using a representative SQL workload as opposed to a single
statement. This
    will allow to get comprehensive index recommendations which takes
into
    account index maintenance overhead and additional space
consumption.


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

```
EXPLAIN PLANS SECTION
-----------------------------------------------------------------------
---------


1- Original With Adjusted Cost
------------------------------
Plan hash value: 4005616876


-----------------------------------------------------------------------
----------
-------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost
(%CPU)| Time
  | Pstart| Pstop |
-----------------------------------------------------------------------
----------
-------------------
|   0 | SELECT STATEMENT      |           |     1 |    13 |   770
(2)| 00:00:1
3 |       |       |
|   1 |  HASH GROUP BY        |           |     1 |    13 |   770
(2)| 00:00:1
3 |       |       |
|   2 |   NESTED LOOPS        |           |     1 |    13 |   769
(2)| 00:00:1
3 |       |       |
|*  3 |    TABLE ACCESS FULL  | CUSTOMERS |     1 |     5 |   348
(1)| 00:00:0
6 |       |       |
|   4 |    PARTITION RANGE ALL|           |     1 |     8 |   422
(3)| 00:00:0
7 |     1 |    28 |
|*  5 |     TABLE ACCESS FULL | SALES     |     1 |     8 |   422
(3)| 00:00:0
7 |     1 |    28 |
-----------------------------------------------------------------------
----------
-------------------


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


   3 - filter("C"."CUST_ID"<2)
   5 - filter("S"."CUST_ID"<2 AND "S"."CUST_ID"="C"."CUST_ID")


2- Using SQL Profile
--------------------
Plan hash value: 3070788227
```

```
---------------------------------------------------------------------
----------
-------------------------------------
| Id  | Operation                              | Name           | Rows
| Bytes |
Cost (%CPU)| Time      | Pstart| Pstop |
---------------------------------------------------------------------
----------
-------------------------------------
|   0 | SELECT STATEMENT                       |                |     1
|    13 |
   55   (2)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY                         |                |     1
|    13 |
   55   (2)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                         |                |     1
|    13 |
   54   (0)| 00:00:01 |       |       |
|   3 |    PARTITION RANGE ALL                 |                |     1
|     8 |
   54   (0)| 00:00:01 |     1 |    28 |
|   4 |     TABLE ACCESS BY LOCAL INDEX ROWID| SALES          |     1
|     8 |
   54   (0)| 00:00:01 |     1 |    28 |
|   5 |      BITMAP CONVERSION TO ROWIDS     |                |
|       |
           |          |       |       |
|*  6 |       BITMAP INDEX RANGE SCAN        | SALES_CUST_BIX |
|       |
           |          |     1 |    28 |
|*  7 |    INDEX UNIQUE SCAN                 | CUSTOMERS_PK   |     1
|     5 |
    0   (0)| 00:00:01 |       |       |
---------------------------------------------------------------------
----------
-------------------------------------

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


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


3- Using New Indices
--------------------
```

```
Plan hash value: 1871796534

----------------------------------------------------------------------
----------
------------------------------------
| Id  | Operation                           | Name            | Rows
| Bytes |
Cost (%CPU)| Time      | Pstart| Pstop |
----------------------------------------------------------------------
----------
------------------------------------
|   0 | SELECT STATEMENT                    |                 |     1
|    13 |
    5   (0)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY NOSORT               |                 |     1
|    13 |
    5   (0)| 00:00:01 |       |       |
|   2 |   NESTED LOOPS                      |                 |
|       |
           |          |       |       |
|   3 |    NESTED LOOPS                     |                 |     1
|    13 |
    5   (0)| 00:00:01 |       |       |
|*  4 |     INDEX RANGE SCAN                | CUSTOMERS_PK    |     1
|     5 |
    2   (0)| 00:00:01 |       |       |
|*  5 |     INDEX RANGE SCAN                | IDX$$_00010001 |     1
|       |
    2   (0)| 00:00:01 |       |       |
|   6 |    TABLE ACCESS BY GLOBAL INDEX ROWID| SALES          |     1
|     8 |
    3   (0)| 00:00:01 | ROWID | ROWID |
----------------------------------------------------------------------
------------------------------------

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

   4 - access("C"."CUST_ID"<2)
   5 - access("S"."CUST_ID"="C"."CUST_ID")
       filter("S"."CUST_ID"<2)

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



SQL> SQL> Disconnected …
```

```
$

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

#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

sqlplus / as sysdba <<FIN!

set echo on
set long 1000000000
set longchunksize 1000

--
-- Check the execution names
--
alter session set nls_date_format = 'MM/DD/YYYY HH24:MI:SS';

select execution_name, status, execution_start
from   dba_advisor_executions
where  task_name = 'SYS_AUTO_SQL_TUNING_TASK'
order by execution_start;

variable last_exec varchar2(30);

begin
  select max(execution_name) keep (dense_rank last order by
execution_start)
  into   :last_exec
  from   dba_advisor_executions
  where  task_name = 'SYS_AUTO_SQL_TUNING_TASK';
end;
/

print :last_exec

--
-- Find the object ID for query AST with sql_id by9m5m597zh19
--

variable obj_id number;

begin
  select object_id
  into   :obj_id
```

```
  from   dba_advisor_objects
  where  task_name = 'SYS_AUTO_SQL_TUNING_TASK' and
         execution_name = :last_exec and
         type = 'SQL' and
         attr1 = 'by9m5m597zh19';
end;
/

print :obj_id

--
-- Get a text report to drill down on this one query
--
set pagesize 0
select dbms_sqltune.report_auto_tuning_task(
  :last_exec, :last_exec, 'TEXT', 'TYPICAL', 'ALL', :obj_id)
from dual;

FIN!
```

9. Investigate how to configure Automatic SQL Tuning using Enterprise Manager.
   a. Back in EM, go to the Automated Maintenance Tasks page.
   b. The chart here shows times in the past when each client was executed, and times in the future when they are scheduled to run again.
   c. Modify the begin point and interval of the graph with the widgets at the upper right to show a week of activity starting on the Sunday before today.
   d. Click the Configure button.
   e. This brings you to the Automated Maintenance Tasks Configuration page.
   f. From this page, you can disable individual clients and change which windows they run in.
   g. Disable the Automatic SQL Tuning client entirely, click Apply, and then click the locator link to return to the last page.
   h. Note that no light blue bars appear for Automatic SQL Tuning in the future.
   i. Return to the configuration page, enable the task again, and click Apply to enable Automatic SQL Tuning.
   j. Click Configure beside Automatic SQL Tuning on the Automated Maintenance Tasks Configuration page.
   k. This takes you to the page where you can configure the task itself. There are fine-grained controls here, such as one that allows the task to run but not implement profiles, and one that allows you to control the maximum number of profiles created per run.
10. Investigate how to configure Automatic SQL Tuning using PL/SQL. From your terminal session, execute the `manual_config.sh` script. What does it do?
   a. Note the first action. You changed the total time limit for the task. Instead of running for an unlimited amount of time (still bound by the maintenance window boundaries), it now runs for a maximum of one hour. The `execute_tuning_task` API call runs the

task immediately in the foreground. Use this to run the task yourself whenever you want.

```
$ ./manual_config.sh

…

Connected to: …
SQL> SQL> Connected.
SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL>   2    3    4
PARAMETER_VALUE
----------------------------------------------------------------------
3600

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

SQL> SQL>   2    3    4
PARAMETER_VALUE
----------------------------------------------------------------------
1800

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

SQL> SQL> Disconnected …

$
--------------------------------------------------------------

#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

…

sqlplus / as sysdba <<FIN!

connect / as sysdba

set echo on

--
-- Configure the task to run for at most 30 minutes.  The value of the
-- TIME_LIMIT parameter determines the total time allowed for a task
execution.
```

```
--

select parameter_value
from   dba_advisor_parameters
where  task_name = 'SYS_AUTO_SQL_TUNING_TASK' and
       parameter_name = 'TIME_LIMIT';

exec dbms_sqltune.set_tuning_task_parameter( -
  'SYS_AUTO_SQL_TUNING_TASK', 'TIME_LIMIT', 1800);

select parameter_value
from   dba_advisor_parameters
where  task_name = 'SYS_AUTO_SQL_TUNING_TASK' and
       parameter_name = 'TIME_LIMIT';

--
-- Run the task immediately
--

exec dbms_sqltune.execute_tuning_task('SYS_AUTO_SQL_TUNING_TASK');

FIN!
```

Note: In your case, the task executes quickly because the workload to take into account is really small. However, you could use the `interrupt_task.sh` script from another session to stop the task, should it last too long.

```
$ cat interrupt_task.sh
#!/bin/bash

cd /home/oracle/solutions/Automatic_SQL_Tuning

sqlplus / as sysdba <<FIN!

connect / as sysdba

set echo on

--
-- Interrupt the task
--

exec dbms_sqltune.interrupt_tuning_task('SYS_AUTO_SQL_TUNING_TASK');

FIN!

$
```

11. Ensure that you disable automatic implementation of SQL profiles to clean up your environment.
    a. On the EM Server page, click Automated Maintenance Tasks.
    b. Check that Status is set to Enabled, and click Configure.
    c. Click the Configure button next to Automatic SQL Tuning.
    d. Select No for “Automatic Implementation of SQL Profiles.”
    e. Then click Apply.

Practices for Lesson 15

Chapter 15

Overview of Practices for Lesson 15

Practices Overview

SQL Plan Management (SPM) is an Oracle Database 11*g* 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 15-1: Using SQL Performance Management (SPM)

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

1. Before you start this practice, change directories to the `$HOME/solutions/SPM` directory. Flush the shared pool to be sure that there is no interference from previous practices.

```
$ cd /home/oracle/solutions/SPM
$ sqlplus / as sysdba

SQL> alter system flush shared_pool;

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 turn on automatic plan capture so that the SPM repository is automatically populated for any repeatable SQL statement. Turn on automatic plan capture by setting the `optimizer_capture_sql_plan_baselines` initialization parameter to `TRUE` in your SQL*Plus session, connected as the `SPM` user. After you have connected in your session, do not disconnect.

```
$ sqlplus spm/spm
…
SQL> show parameter baseline

NAME                                 TYPE        VALUE
------------------------------------ ----------- -------------
optimizer_capture_sql_plan_baselines boolean     FALSE
optimizer_use_sql_plan_baselines     boolean     TRUE

SQL> alter session set optimizer_capture_sql_plan_baselines =
TRUE;

Session altered.

SQL>
```

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

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

Use the `query1.sql` script to execute the query.

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

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

no rows selected

SQL>
```

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

```
-- Should not return any rows

SQL> @check_baselines.sql
SQL> set echo on
SQL> set pagesize 30
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2         origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*LOAD_AUTO*/%';

no rows selected
```

5. Execute the `query1.sql` script again.

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

no rows selected

SQL>
```

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

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

```
-- You should see one accepted entry
```

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

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



SQL>
```

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

   This causes the optimizer to create a different plan for the SQL statement execution.

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

Session altered.

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

no rows selected

SQL>
```

   a. Because the SQL statement will have a new plan, another plan baseline is automatically captured. You can confirm this by checking the plan baseline again.

```

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

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

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
8.0622E+18 SYS_SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40
order by prod_id
SYS_SQL_PLAN_8dfc352f11df68d0  AUTO-CAPTURE   YES NO  NO  YES

8.0622E+18 SYS_SQL_6fe28d438dfc352f
select /*LOAD_AUTO*/ * from sh.sales where quantity_sold > 40
order by prod_id
SYS_SQL_PLAN_8dfc352f54bc8843  AUTO-CAPTURE   YES YES NO  YES
```

b. Now you see two plan baselines for your query, but notice that the new plan has not been accepted. This new plan will have to be validated before it is acceptable as a good plan to use.

8. Reset the optimizer mode to the default values and disable automatic capture of plan baselines.

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

Session altered.

SQL> alter session set optimizer_capture_sql_plan_baselines =
FALSE;

Session altered.
```

9. Purge the plan baselines and confirm that the SQL plan baseline is empty. Use `purge_auto_baseline.sql`.

```
SQL> @purge_auto_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
```

```
SQL> exec :cnt :=
dbms_spm.drop_sql_plan_baseline('SYS_SQL_6fe28d438dfc352f');

PL/SQL procedure successfully completed.

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

no rows selected

SQL>
```

Loading Plans from SQL Tuning Sets

10. Still connected to your SQL*Plus session, check the execution plan for the following SQL statement, and then execute it. (Use `explain_query2.sql` and `query2.sql`.)

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

```
SQL> @explain_query2.sql

-- You should see a Full Table Scan

SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*LOAD_STS*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3803407550

--------------------------------------
| Id  | Operation            | Name  |
```

```
--------------------------------------
|   0 | SELECT STATEMENT     |       |
|   1 |  SORT ORDER BY       |       |
|   2 |   PARTITION RANGE ALL|       |
|   3 |    TABLE ACCESS FULL | SALES |
--------------------------------------


10 rows selected.

SQL> @query2.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_STS*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

no rows selected
```

11. Alter the optimizer mode to use `FIRST_ROWS` optimization, check the execution plan, and execute the query.

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

Session altered.

-- You should see a different plan
-- (Table Access By Local Index)

SQL> @explain_query2.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*LOAD_STS*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null,null,'basic'));
PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 899219946
--------------------------------------------------------------
| Id  | Operation                             | Name
```

```
--------------------------------------------------------------
|   0 | SELECT STATEMENT                    |
|   1 |  SORT ORDER BY                      |
|   2 |   PARTITION RANGE ALL               |
|   3 |    TABLE ACCESS BY LOCAL INDEX ROWID| SALES
|   4 |     BITMAP CONVERSION TO ROWIDS     |
|   5 |      BITMAP INDEX FULL SCAN         | SALES_PROMO_BIX

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

12 rows selected.

SQL>
SQL> @query2.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_STS*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

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

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

Session altered.
```

13. Verify that there are no baseline plans for your statement. (Use the `check_baselines2.sql` script.)

```
SQL> @check_baselines2.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2       origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*LOAD_STS*/%';

no rows selected

SQL>
```

14. Create a SQL tuning set that captures the `SELECT` statements that contain the `LOAD_STS` hint. These statements are in the cursor cache. The STS should be called `SPM_STS` and owned by the `SPM` user. Use the `catchup_sts.sql` script to capture these statements.

```
SQL> @catchup_sts.sql
SQL> set echo on
SQL>
SQL> exec sys.dbms_sqltune.create_sqlset(-
> sqlset_name  => 'SPM_STS', sqlset_owner => 'SPM');

PL/SQL procedure successfully completed.

SQL>
SQL> DECLARE
  2     stscur   dbms_sqltune.sqlset_cursor;
  3  BEGIN
  4     OPEN stscur FOR
  5       SELECT VALUE(P)
  6       FROM   TABLE(dbms_sqltune.select_cursor_cache(
  7              'sql_text like ''select /*LOAD_STS*/%''',
  8              null, null, null, null, null, null, 'ALL')) P;
  9
10     -- populate the sqlset
11     dbms_sqltune.load_sqlset(sqlset_name     => 'SPM_STS',
12                              populate_cursor => stscur,
13                              sqlset_owner    => 'SPM');
14  END;
15  /

PL/SQL procedure successfully completed.

SQL>
```

15. Verify which SQL statements are in `SPM_STS`. (Use the `check_sts.sql` script.)

    `SPM_STS` has your SQL statement with two different plans.

```
SQL> @check_sts.sql
SQL> set echo on
SQL>
SQL> select sql_text from dba_sqlset_statements
  2  where sqlset_name='SPM_STS';

SQL_TEXT
--------------------------------------------------------------
```

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

select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id

SQL>
```

16. Populate the plan baseline repository with the plans found in `SPM_STS`. Use the `populate_baseline.sql` script for that:

```
SQL> @populate_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_sqlset( -
>            sqlset_name  => 'SPM_STS', -
>            basic_filter => 'sql_text like ''select
/*LOAD_STS*/%''');

PL/SQL procedure successfully completed.

SQL>
```

17. Confirm that the plan baselines are loaded and note the value in the origin column. What do you observe? (Use `check_baselines2.sql`.)

    You should see `MANUAL-LOAD` because you manually loaded these plans. Also note that this time, both plans are accepted.

```
SQL> @check_baselines2.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2        origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*LOAD_STS*/%';

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
---------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
1.2134E+19 SYS_SQL_a8632bd857a4a25e
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id
```

```
SYS_SQL_PLAN_57a4a25e11df68d0  MANUAL-LOAD    YES YES NO  YES

1.2134E+19 SYS_SQL_a8632bd857a4a25e
select /*LOAD_STS*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_57a4a25e54bc8843  MANUAL-LOAD    YES YES NO  YES


SQL>
```

18. Purge the plan baselines and drop `SPM_STS`. Use the `purge_sts_baseline.sql` script.

```
SQL> @purge_sts_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>                 'SYS_SQL_a8632bd857a4a25e');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

       CNT
----------
         2

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

no rows selected

SQL>
SQL> exec sys.dbms_sqltune.drop_sqlset(-
>                 sqlset_name  => 'SPM_STS', -
>                 sqlset_owner => 'SPM');

PL/SQL procedure successfully completed.
```

```
SQL>
```

Loading Plans from the Cursor Cache

19. Now, you see how to directly load plan baselines from the cursor cache. Before you begin, you need some SQL statements. Still connected to your SQL*Plus session, check the execution plan for the following SQL statement, and then execute it. (Use the `explain_query3.sql` and `query3.sql` scripts.)

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

```
SQL> @explain_query3.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*LOAD_CC*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3803407550

--------------------------------------
| Id  | Operation            | Name  |
--------------------------------------
|   0 | SELECT STATEMENT     |       |
|   1 |  SORT ORDER BY       |       |
|   2 |   PARTITION RANGE ALL|       |
|   3 |    TABLE ACCESS FULL | SALES |
--------------------------------------

10 rows selected.

SQL>
SQL> @query3.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_CC*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;
```

```
no rows selected

SQL>
```

20. Now, change the optimizer mode to use `FIRST_ROWS` optimization and execute the same script as in the previous step. What do you observe?
    a. You should see a different execution plan.

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

Session altered.

SQL> @explain_query3.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*LOAD_CC*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 899219946

--------------------------------------------------------------
| Id  | Operation                              | Name          |
--------------------------------------------------------------
|   0 | SELECT STATEMENT                       |               |
|   1 |  SORT ORDER BY                         |               |
|   2 |   PARTITION RANGE ALL                  |               |
|   3 |    TABLE ACCESS BY LOCAL INDEX ROWID| SALES            |
|   4 |     BITMAP CONVERSION TO ROWIDS        |               |
|   5 |      BITMAP INDEX FULL SCAN            | SALES_PROMO_BIX|

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


12 rows selected.
```

```
SQL>
SQL> @query3.sql
SQL> set echo on
SQL>
SQL> select /*LOAD_CC*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

21. Reset the optimizer mode to `ALL_ROWS`.

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

Session altered.
```

22. Now that the cursor cache is populated, you must get the SQL ID for your SQL statement. Use the SQL ID to filter the content of the cursor cache and load the baselines with these two plans. Use the `load_cc_baseline.sql` script.

```

SQL> @load_cc_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> variable sqlid varchar2(20);
SQL>
SQL> begin
  2   select distinct sql_id into :sqlid from v$sql
  3    where sql_text like 'select /*LOAD_CC*/%';
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL>
SQL> print sqlid;

SQLID
--------------------------------------------------------------
6qc9wxhgzzz8g

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
```

```
>                  sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>
```

23. Confirm that the baselines were loaded. (Use `check_baselines3.sql`.)

```
SQL> @check_baselines3.sql
SQL> set echo on
SQL> 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 SYS_SQL_f6cb7f742ef93547
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_2ef9354711df68d0  MANUAL-LOAD    YES YES NO  YES

1.7783E+19 SYS_SQL_f6cb7f742ef93547
select /*LOAD_CC*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_2ef9354754bc8843  MANUAL-LOAD    YES YES NO  YES

-- You should see two accepted baselines
```

24. Purge the plan baselines.

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

```
>                  'SYS_SQL_f6cb7f742ef93547');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

       CNT
----------
         2

SQL>
SQL> REM Check that plan baselines were purged:
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2           origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*LOAD_CC*/%';

no rows selected

SQL>
```

Optimizer Plan Selection

25. Now that you know how to capture plans, look at how the optimizer selects which plan baselines to use. First, you create two baselines for a statement and show them being used. Then you disable one of the baselines and see the second baseline being used. Finally, you disable both baselines and show that now the optimizer falls back to the default behavior of a cost-based plan. Start by executing the same query twice, with different plans. Determine the execution of the following statement, and then execute it. (Use the `explain_query4.sql` and `query4.sql` scripts.)

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

```
SQL> @explain_query4.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*SPM_USE*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.
```

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3803407550

--------------------------------------
| Id  | Operation            | Name  |
--------------------------------------
|   0 | SELECT STATEMENT     |       |
|   1 |  SORT ORDER BY       |       |
|   2 |   PARTITION RANGE ALL|       |
|   3 |    TABLE ACCESS FULL | SALES |
--------------------------------------

10 rows selected.

SQL>
SQL> @query4.sql
SQL> set echo on
SQL>
SQL> select /*SPM_USE*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

26. Change the optimizer mode to use `FIRST_ROWS` optimization and execute the same script as in the previous step. What do you observe?

    You should see a different execution plan.

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

Session altered.

SQL> @explain_query4.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*SPM_USE*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;
```

```
Explained.

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 899219946
--------------------------------------------------------------
| Id  | Operation                            | Name           |
--------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                |
|   1 |  SORT ORDER BY                       |                |
|   2 |   PARTITION RANGE ALL                |                |
|   3 |    TABLE ACCESS BY LOCAL INDEX ROWID| SALES          |
|   4 |     BITMAP CONVERSION TO ROWIDS      |                |
|   5 |      BITMAP INDEX FULL SCAN          | SALES_PROMO_BIX|

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


12 rows selected.

SQL>
SQL> @query4.sql
SQL> set echo on
SQL>
SQL> select /*SPM_USE*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

no rows selected

SQL>
```

27. Reset the optimizer mode to `ALL_ROWS`.

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


Session altered.
```

28. Populate the baseline with the two plans for your statement directly from the cursor cache. Use the `load_use_baseline.sql` script for that. Then verify that baselines were loaded. What do you observe?

You should see both plan baselines loaded. Note that both plans have been marked acceptable. This is because both plans were present in the cursor cache at the time of the load, and because these plans have been manually loaded, it is assumed that both plans have acceptable performance.

```
SQL> @load_use_baseline.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> variable sqlid varchar2(20);
SQL>
SQL> begin
  2   select distinct sql_id into :sqlid from v$sql
  3     where sql_text like 'select /*SPM_USE*/%';
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL>
SQL> print sqlid;

SQLID
--------------------------------------------------------------
2pma6tcaczdc8

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
>                 sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

       CNT
----------
         2

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

```
SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45511df68d0  MANUAL-LOAD    YES YES NO  YES

7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45554bc8843  MANUAL-LOAD    YES YES NO  YES

SQL>
```

29. Determine the baseline and the execution plan used to execute the following query:

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

What do you observe?

The note at the end of the explain output tells you that the system is using a baseline. From the execution plan, you can see that you are using the first baseline, a full-table scan. Use the `explain_query4_note.sql` script.

```
SQL> @explain_query4_note.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*SPM_USE*/ * from sh.sales
  3  where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null,null,'basic
+note'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3803407550
```

```
--------------------------------------
| Id  | Operation            | Name  |
--------------------------------------
|   0 | SELECT STATEMENT     |       |
|   1 |  SORT ORDER BY       |       |
|   2 |   PARTITION RANGE ALL|       |
|   3 |    TABLE ACCESS FULL | SALES |
--------------------------------------


PLAN_TABLE_OUTPUT
----------------------------------------------------------------
Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_00ece45554bc8843" used for
this statement

14 rows selected.

SQL>
```

30. Disable that plan baseline, and check whether the system uses the other plan baseline when executing the statement again. Use the `check_baseline_used.sql` script for that. What do you observe?

    Now from the execution plan, you see that you are using an index scan instead of a full-table scan. So this is the second baseline.

```
SQL> @check_baseline_used.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> select sql_handle,plan_name
  2  from dba_sql_plan_baselines
  3  where sql_text like 'select /*SPM_USE*/%';

SQL_HANDLE                     PLAN_NAME
------------------------------ ------------------------------
SYS_SQL_0a9d872600ece455       SYS_SQL_PLAN_00ece45511df68d0
SYS_SQL_0a9d872600ece455       SYS_SQL_PLAN_00ece45554bc8843

SQL>

-- disable baseline ----
```

```
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline( -
>       sql_handle        => 'SYS_SQL_0a9d872600ece455', -
>       plan_name         => 'SYS_SQL_PLAN_00ece45554bc8843',-
>       attribute_name    => 'ENABLED', -
>       attribute_value   => 'NO');

PL/SQL procedure successfully completed.

SQL>

-- Check the baselines Should see one disabled --------

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

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
----------------------------------------------------------------
----------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45511df68d0  MANUAL-LOAD    YES YES NO  YES

7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45554bc8843  MANUAL-LOAD    NO  YES NO  YES



SQL>
SQL>
SQL> explain plan for select /*SPM_USE*/ * from sh.sales
  2          where quantity_sold > 40 order by prod_id;

Explained.

SQL>
```

```
SQL> select * from table(dbms_xplan.display(null, null, 'basic
+note'));

PLAN_TABLE_OUTPUT
----------------------------------------------------------------
----------------
Plan hash value: 899219946

-------------------------------------------------------------
| Id  | Operation                           | Name           |-
------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                |
|   1 |  SORT ORDER BY                      |                |
|   2 |   PARTITION RANGE ALL               |                |
|   3 |    TABLE ACCESS BY LOCAL INDEX ROWID| SALES          |
|   4 |     BITMAP CONVERSION TO ROWIDS     |                |
|   5 |      BITMAP INDEX FULL SCAN         | SALES_PROMO_BIX|

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


Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_00ece45511df68d0" used for
this statement


16 rows selected.


SQL>
```

31. Disable the other plan baseline and check whether the system falls back to the cost-based approach when executing the explain plan for the statement. Use the `check_baseline_used2.sql` script.

    You know that the optimizer has gone back to the default cost-based approach because there is no note at the end of the plan stating that a baseline was used.

```
SQL> @check_baseline_used2.sql
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.alter_sql_plan_baseline( -
>           sql_handle        => 'SYS_SQL_0a9d872600ece455', -
>           plan_name         => 'SYS_SQL_PLAN_00ece45511df68d0',
-
>           attribute_name    => 'ENABLED', -
```

```
>          attribute_value   => 'NO');

PL/SQL procedure successfully completed.

SQL>

-- Both baselines are disabled ---
SQL> select signature, sql_handle, sql_text, plan_name,
  2         origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*SPM_USE*/%';

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45511df68d0  MANUAL-LOAD    NO  YES NO  YES

7.6492E+17 SYS_SQL_0a9d872600ece455
select /*SPM_USE*/ * from sh.sales
where quantity_sold > 40 order by prod_id
SYS_SQL_PLAN_00ece45554bc8843  MANUAL-LOAD    NO  YES NO  YES



SQL>
SQL>
SQL> explain plan for select /*SPM_USE*/ * from sh.sales
  2  where quantity_sold > 40 order by prod_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null, null, 'basic
+note'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3803407550
```

```
--------------------------------------
| Id  | Operation              | Name  |
--------------------------------------
|   0 | SELECT STATEMENT       |       |
|   1 |  SORT ORDER BY         |       |
|   2 |   PARTITION RANGE ALL|       |
|   3 |    TABLE ACCESS FULL | SALES |
--------------------------------------

10 rows selected.

SQL>
```

32. Drop the plan baselines and check whether they are purged. Use the `purge_use_baseline.sql` script.

```
SQL> @purge_use_baseline
SQL> set echo on
SQL>
SQL> variable cnt number;
SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>               'SYS_SQL_0a9d872600ece455');

PL/SQL procedure successfully completed.

SQL>
SQL> print cnt;

       CNT
----------
         2

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

no rows selected

SQL>
```

33. One of the methods used to enable plan evolution (or plan verification) is Automatic SQL Tuning that is run as an automated task in a maintenance window. Automatic SQL Tuning targets only the high-load SQL statements; for them, it automatically implements actions such as making a successfully verified plan an accepted plan. Here, you manually trigger SQL tuning to find a better plan for a given SQL statement. First, determine the execution plan of the following statement:

```
select /*+ USE_NL(s c) FULL(s) FULL(c) */
c.cust_id, sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
```

Some optimizer hints to the statements have been added to ensure that you get a less than optimal plan at first. Use the `check_evolve_plan.sql` script for that. What do you observe?

As you can see, the execution plan is being forced by the hints to perform two full-table scans, followed by a nest loop join.

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

Explained.

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

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 4005616876
--------------------------------------------------------------
| Id  | Operation             | Name      | Rows   | Bytes | Cost
(%CPU)| Time   | Pstart| Pstop |
--------------------------------------------------------------
PLAN_TABLE_OUTPUT
--------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |      1 |    39 |
893   (2)| 00:00:11 |       |       |
```

```
|   1 |  HASH GROUP BY        |           |     1 |    39 |
893   (2)| 00:00:11 |       |       |
|   2 |   NESTED LOOPS        |           |     1 |    31 |
892   (2)| 00:00:11 |       |       |
|*  3 |    TABLE ACCESS FULL  | CUSTOMERS |     1 |     5 |
405   (1)| 00:00:05 |       |       |
|   4 |    PARTITION RANGE ALL|           |     1 |     8 |
488   (2)| 00:00:06 |     1 |    28 |
|*  5 |     TABLE ACCESS FULL | SALES     |     1 |     8 |
488   (2)| 00:00:06 |     1 |    28 |
--------------------------------------------------------------
PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("C"."CUST_ID"<2)
   5 - filter("S"."CUST_ID"<2 AND "S"."CUST_ID"="C"."CUST_ID")


18 rows selected.


SQL>
```

34. Now execute the statement so that you can get the plan in the cursor cache and load the corresponding plan baseline. Use the `load_evolve_baseline.sql` script for that. What do you observe?

    You see that the current plan is both enabled and accepted, but not fixed.

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

no rows selected

SQL>
SQL> begin
  2   select sql_id into :sqlid from v$sql
```

```
  3   where sql_text like 'select /*+ USE_NL(s c) FULL(s)
FULL(c) */%';
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL>
SQL> exec :cnt := dbms_spm.load_plans_from_cursor_cache(-
>                 sql_id => :sqlid);

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2         origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like 'select /*+ USE_NL(s c) FULL(s) FULL(c)
*/%';

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */
       c.cust_id, sum(s.quantity_sold)
SQL_PLAN_1j8gg2m0ryqvpdc5f94e5 MANUAL-LOAD    YES YES NO  YES

SQL>
```

35. Manually create and execute a SQL tuning task to tune your statement. Use the `tune_evolve_sql.sql` script.

```
SQL> @tune_evolve_sql.sql
SQL> set echo on
SQL>
SQL> variable sqltext varchar2(4000);
SQL>
SQL> BEGIN
  2   :sqltext := q'# select /*+ USE_NL(s c) FULL(s) FULL(c)*/
  3                     c.cust_id, sum(s.quantity_sold)
```

```
  4                       from sh.sales s, sh.customers c
  5                       where s.cust_id = c.cust_id
  6                             and c.cust_id < 2
  7                       group by c.cust_id
  8                     #';
  9  END;
 10  /

PL/SQL procedure successfully completed.

SQL>
SQL> variable spmtune   varchar2(30);
SQL>
SQL> exec :spmtune := dbms_sqltune.create_tuning_task(-
>                          sql_text => :sqltext);

PL/SQL procedure successfully completed.

SQL>
SQL> exec dbms_sqltune.execute_tuning_task(:spmtune);

PL/SQL procedure successfully completed.

SQL>
```

36. Now that the tuning task has been completed, run the report and see what recommendations have been made for your statement. What do you observe?

    There are two recommendations: a SQL profile or a new index creation. Use the `report_evolve_tuning.sql` script.

```
SQL> @report_evolve_tuning.sql
SQL> set echo on
SQL>
SQL> set long 10000
SQL>
SQL> select dbms_sqltune.report_tuning_task(:spmtune,'TEXT')
  2   from dual;

--  Report is not included here -
--  For a sample report see the sql_tuning_report.lst file -

SQL>
```

37. Accept the SQL profile proposed by the SQL Tuning Advisor. Use the `accept_evolve_baseline.sql` script to accept the recommended SQL profile. What happens?

    Accepting the profile causes a new SQL profile and plan baseline to be created for your statement. Now you see two baselines for your statement. Both of them are enabled and accepted.

    Note: One is `MANUAL-LOAD` and the other is `MANUAL-SQLTUNE`.

```
SQL> @accept_evolve_baseline.sql
SQL> set echo on
SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2   origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like
  5    'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75dc5f94e5  MANUAL-LOAD    YES YES NO  YES



SQL>
SQL> exec dbms_sqltune.accept_sql_profile(-
>         task_name => :spmtune,-
>          name => 'SPM_SQL_PROF');

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, category, name, sql_text
  2  from dba_sql_profiles where name like 'SPM%';

SIGNATURE CATEGORY                       NAME
```

```
---------- ------------------------------ -------------------
SQL_TEXT
--------------------------------------------------------------
1.7750E+18 DEFAULT                        SPM_SQL_PROF
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id


SQL> select signature, sql_handle, sql_text, plan_name,
  2  origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like
  5     'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';

SIGNATURE SQL_HANDLE
---------- ------------------------------
SQL_TEXT
--------------------------------------------------------------
PLAN_NAME                      ORIGIN         ENA ACC FIX AUT
------------------------------ -------------- --- --- --- ---
1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75a10c1dcf  MANUAL-SQLTUNE YES YES NO  YES

1.7750E+18 SYS_SQL_18a1ef14c17f5b75
select /*+ USE_NL(s c) FULL(s) FULL(c) */ c.cust_id,
sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2
group by c.cust_id
SYS_SQL_PLAN_c17f5b75dc5f94e5  MANUAL-LOAD    YES YES NO  YES

SQL>
```

38. Determine the plan used for your statement when executing it. What do you observe?

    The next time that you execute the query, it uses the plan baseline and the SQL profile. Use the `explain_query5.sql` script.

```
SQL> @explain_query5.sql
SQL> set echo on
SQL>
SQL> explain plan for
  2  select /*+ USE_NL(s c) FULL(s) FULL(c) */
  3          c.cust_id, sum(s.quantity_sold)
  4  from sh.sales s, sh.customers c
  5  where s.cust_id = c.cust_id and c.cust_id < 2
  6  group by c.cust_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(null,
  2                          null, 'basic +note'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------
Plan hash value: 3070788227
--------------------------------------------------------------
| Id  | Operation                               | Name          |
--------------------------------------------------------------
|   0 | SELECT STATEMENT                        |               |
|   1 |  HASH GROUP BY                          |               |
|   2 |   NESTED LOOPS                          |               |
|   3 |    PARTITION RANGE ALL                  |               |
|   4 |     TABLE ACCESS BY LOCAL INDEX ROWID| SALES         |
|   5 |      BITMAP CONVERSION TO ROWIDS        |               |
|   6 |       BITMAP INDEX RANGE SCAN           | SALES_CUST_BIX|
|   7 |    INDEX UNIQUE SCAN                    | CUSTOMERS_PK  |
--------------------------------------------------------------


Note
-----
   - SQL profile "SPM_SQL_PROF" used for this statement
   - SQL plan baseline "SQL_PLAN_1j8gg2m0ryqvpa10c1dcf" used for
this statement

19 rows selected.

SQL>
```

39. Execute the `cleanup_spm.sql` script to purge your environment for this practice.

```
SQL> @cleanup_spm.sql
SQL> set echo on
SQL>
SQL> exec dbms_sqltune.drop_sql_profile(-
>         name => 'SPM_SQL_PROF');

PL/SQL procedure successfully completed.

SQL>
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline(-
>                'SYS_SQL_18a1ef14c17f5b75');

PL/SQL procedure successfully completed.

SQL>
SQL> select signature, sql_handle, sql_text, plan_name,
  2           origin, enabled, accepted, fixed, autopurge
  3  from dba_sql_plan_baselines
  4  where sql_text like
  5          'select /*+ USE_NL(s c) FULL(s) FULL(c) */%';

no rows selected

SQL>
SQL> select signature, category, name, sql_text
  2  from dba_sql_profiles where name like 'SPM%';

no rows selected

SQL> alter system set optimizer_use_sql_plan_baselines=FALSE;

System altered.

SQL> exit
```

Practices for Lesson B

Chapter 16

Overview of Practices for Lesson B

Practices Overview

In these optional practices, you will use various hints to influence the optimizer to choose a particular access method or path.

Practice B-1: Using Hints

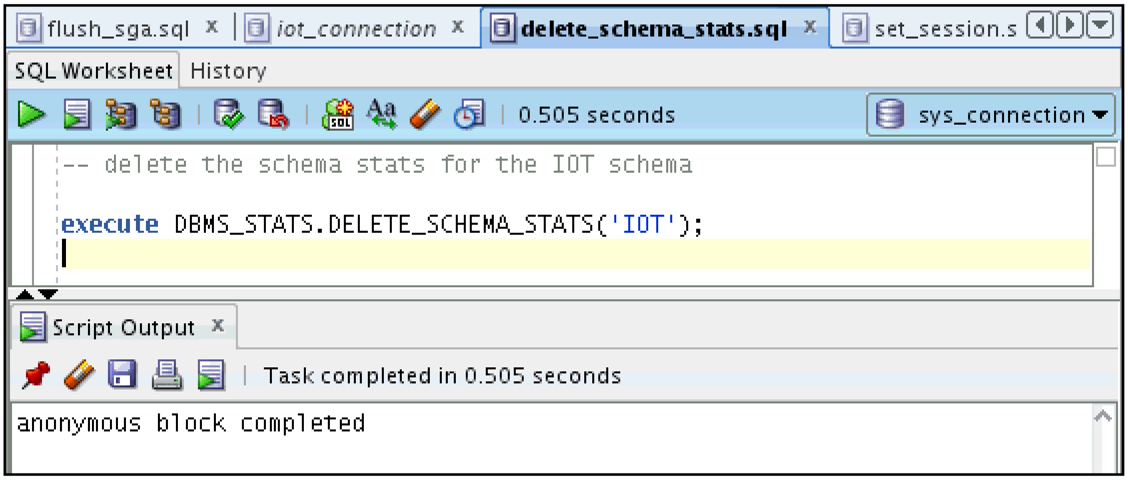
In this practice, you study five different hint cases. They are all independent from each other.
Note: You can find all the necessary scripts used for this lab in your `$HOME/solutions/Hints` directory.

1. **Case 1: Index Organized Table and No_INDEX hint:** The `iot_setup.sh` script has been executed as part of the class setup. This script creates an index-organized table. It is listed here:

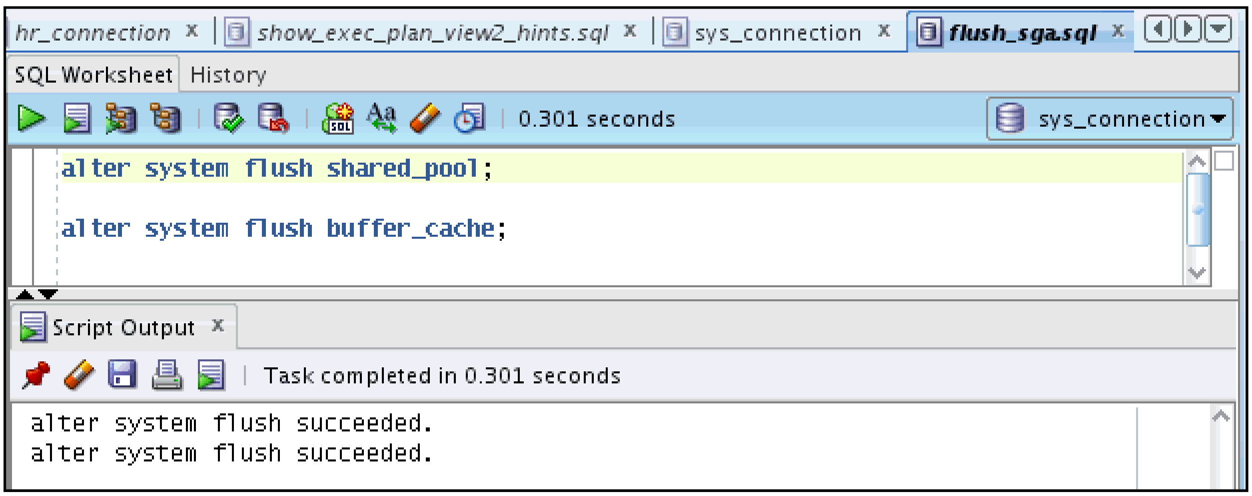
```
set echo on

drop user iot cascade;

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

grant connect, resource, dba to iot;

connect iot/iot

drop table iottab purge;

CREATE TABLE IOTTAB (
   OBJECT_ID      NUMBER(14, 0) NOT NULL ENABLE
 , OBJECT_ID_ATT  NUMBER(14, 0) NOT NULL ENABLE
 , OBJECT_ID_CAT  NUMBER(14, 0) NOT NULL ENABLE
 , BEGIN          DATE NOT NULL ENABLE
 , END            DATE NOT NULL ENABLE
 , STATUS         NUMBER
 , COMM           VARCHAR2(32) NOT NULL ENABLE
 , CONSTRAINT IOTTAB_PK
      PRIMARY KEY (OBJECT_ID
                 , OBJECT_ID_ATT
                 , OBJECT_ID_CAT
                 , BEGIN
                 , END) ENABLE )
ORGANIZATION INDEX PCTTHRESHOLD 50 ;

CREATE INDEX OBJECT_ID_ATT_INDX ON IOTTAB (OBJECT_ID_ATT);

-- load data

begin
for i in 400001..500000 loop
  insert into iottab values(i,mod(i,428),mod(i,20),sysdate-
mod(i,100),sysdate+mod(i,100),mod(i,3),'aaaaaaaaaaaaaaaaaaaaaaaaa'||i
);
end loop;
```

```
commit;
end;
/

begin
for i in 100001..200000 loop
  insert into iottab values(i,mod(i,428),mod(i,20),sysdate-
mod(i,100),sysdate+mod(i,100),mod(i,3),'aaaaaaaaaaaaaaaaaaaaaaaaaa'||i
);
end loop;
commit;
end;
/

begin
for i in 300001..400000 loop
  insert into iottab values(i,mod(i,428),mod(i,20),sysdate-
mod(i,100),sysdate+mod(i,100),mod(i,3),'aaaaaaaaaaaaaaaaaaaaaaaaaa'||i
);
end loop;
commit;
end;
/

begin
for i in 500001..600000 loop
  insert into iottab values(i,mod(i,428),mod(i,20),sysdate-
mod(i,100),sysdate+mod(i,100),mod(i,3),'aaaaaaaaaaaaaaaaaaaaaaaaaa'||i
);
end loop;
commit;
end;
/

begin
for i in 1..100000 loop
  insert into iottab values(i,mod(i,428),mod(i,20),sysdate-
mod(i,100),sysdate+mod(i,100),mod(i,3),'aaaaaaaaaaaaaaaaaaaaaaaaaa'||i
);
end loop;
commit;
end;
/

exit;
```

2. Delete the statistics from the IOT schema. Open and execute the `delete_schema_stats.sql` script using the `sys_connection`.



3. Using `sys_connection`, in SQL Developer, open and execute the `flush_sga.sql` script in `/home/oracle/solutions/Hints`.



4. Create a connection in SQL Developer for the `IOT` user called `iot_connection` and start a SQL Worksheet.

   Name: `iot_connection`

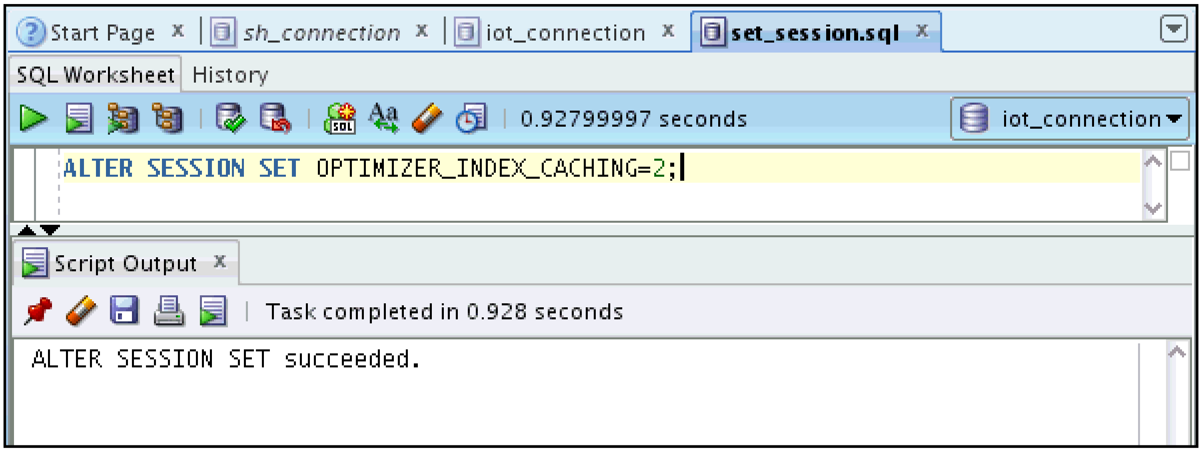
   Username: `iot`

   Password: `iot`

   Service name: `orcl.example.com`

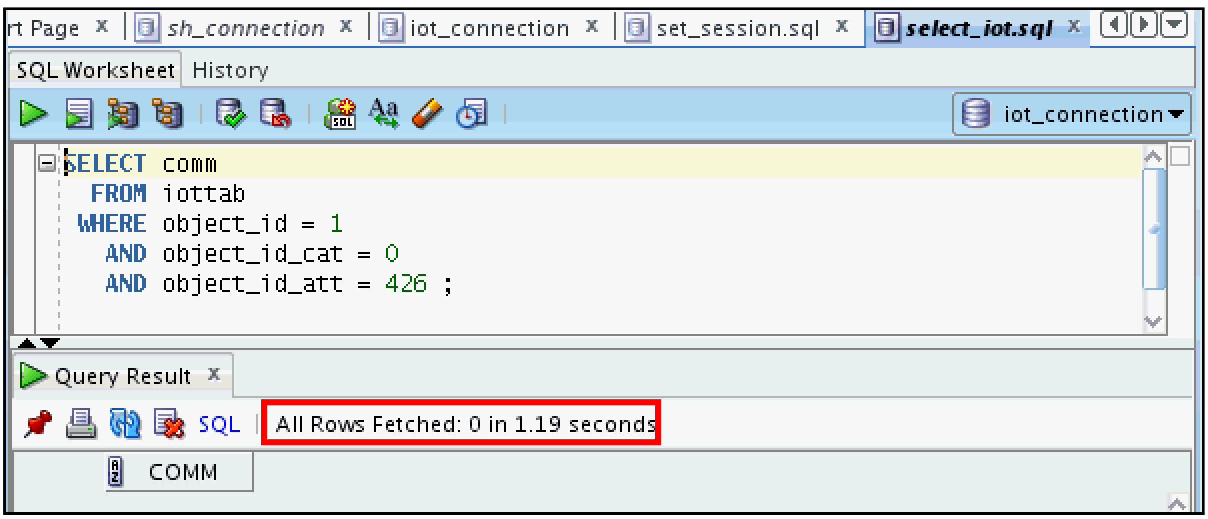
   Save password.

5. Use the file navigator, open, and execute the `set_session.sql` script in the `iot_connection`. This script sets the `OPTIMIZER_INDEX_CACHING` parameter for the duration of this case.



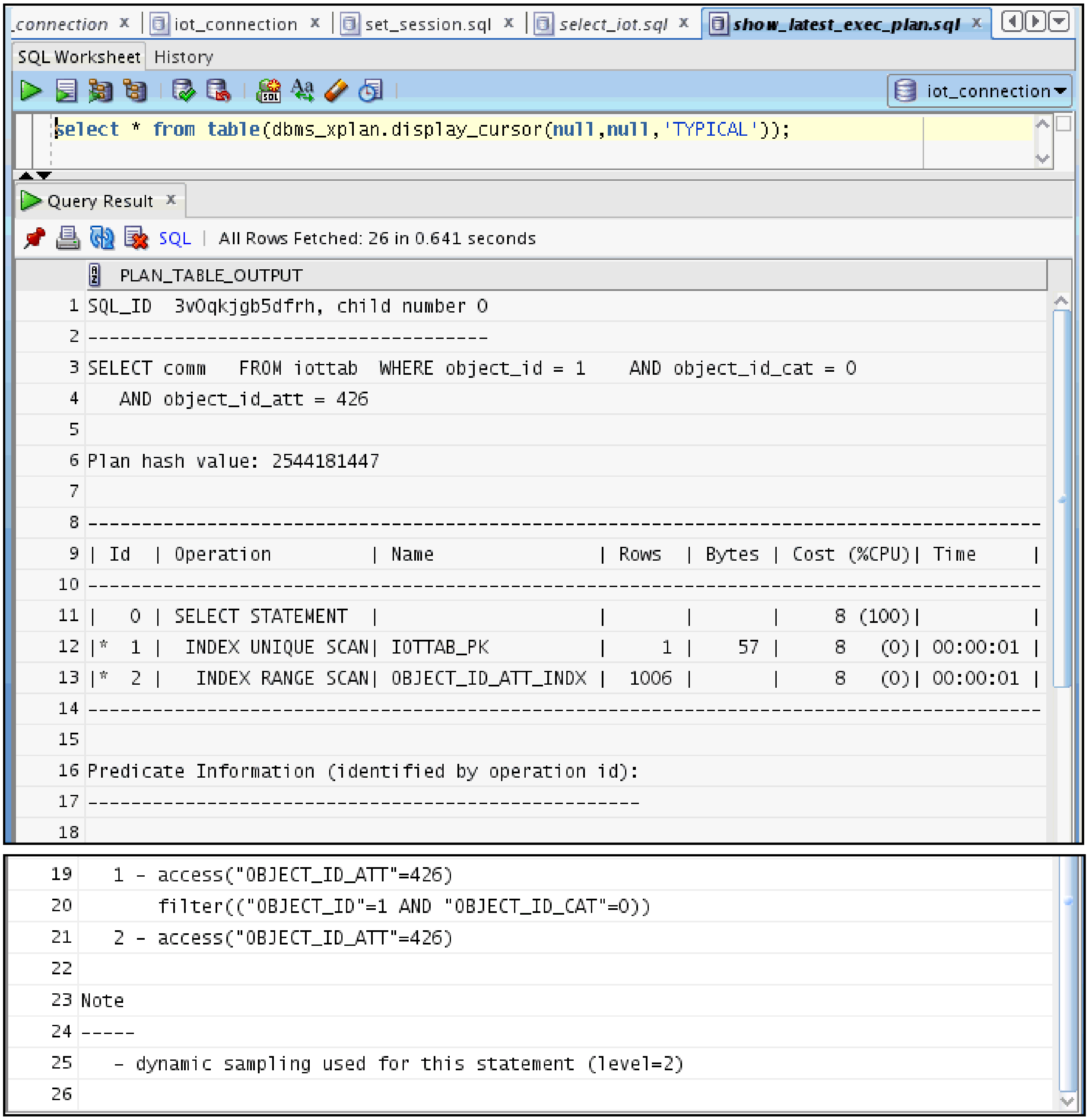
6. Open the `select_iot.sql` file, execute the following query in `iot_connection`: (Note the time it takes to execute.)

```
SELECT comm.
  FROM iottab
 WHERE object_id = 1
   AND object_id_cat = 0
   AND object_id_att = 426 ;
```

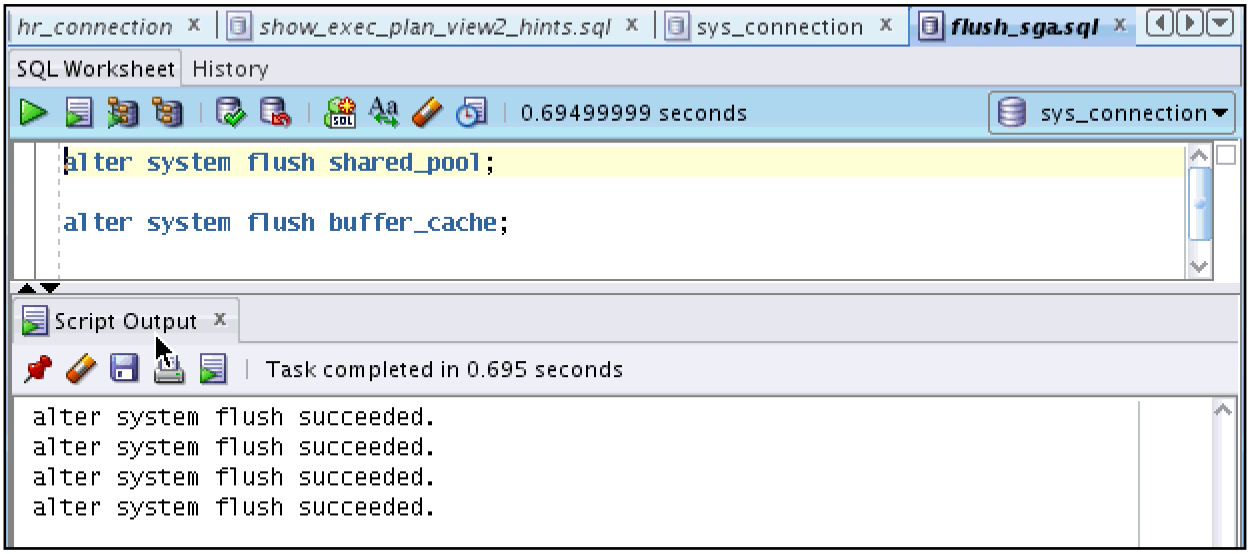


7. Use the `DBMS_XPLAN` package to display the execution plan associated with the statement you executed in the previous step. What do you observe?

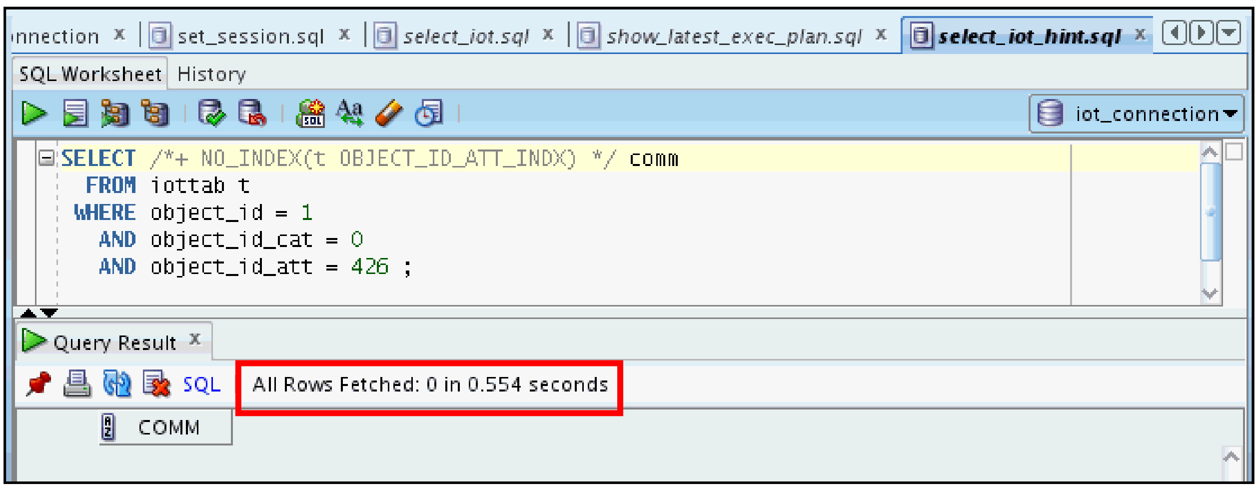
   a. It is strange to see that the optimizer chooses a plan that accesses the secondary index while the query references all columns of the primary key.



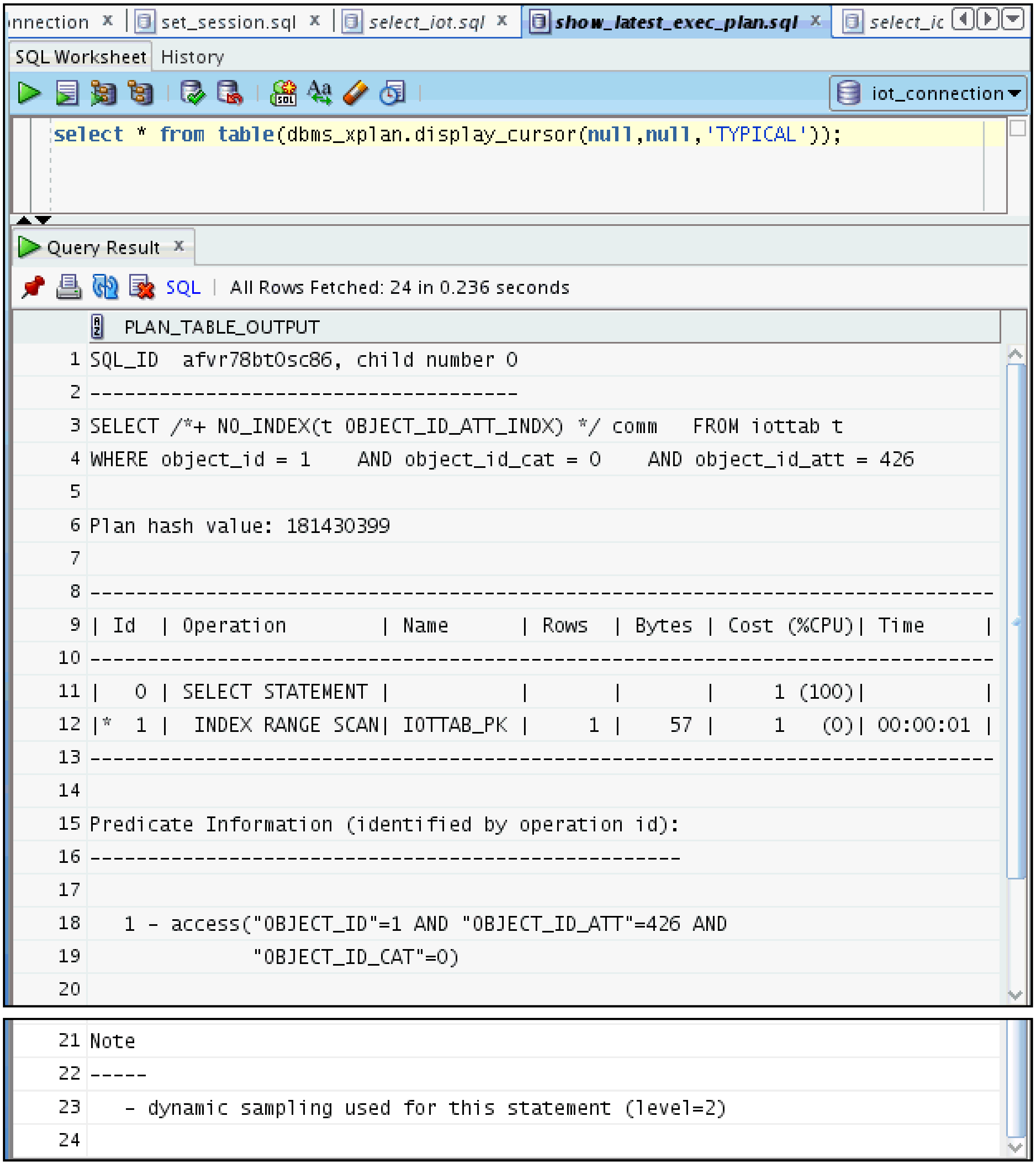
8. Before trying to fix this issue, make sure you flush the content of your SGA by executing the `flush_sga.sql` script again.



9. Using hints only, how would you fix the issue raised at step 4? Implement your solution and check if it works.
   a. The idea is to use a hint to prevent the optimizer from using the secondary index. You will use the `NO_INDEX` hint. Open and execute `select_iot_hint.sql` in SQL Developer. Note the elapsed time.



   b. Execute the `show_latest_exec_plan.sql` script again.



10. **Case 2: Global Hint with Views:** You study a second case of hint utilization to specify hints in lower query blocks. The `HR` user has been unlocked and granted the DBA privilege in the class setup as shown.

```
alter user hr identified by hr account unlock;

grant dba to hr

exit;
```

11. In SQL Developer, create a connection for the `HR` user if it does not exist.

    Connection name `hr_connection`

    Username: `hr`

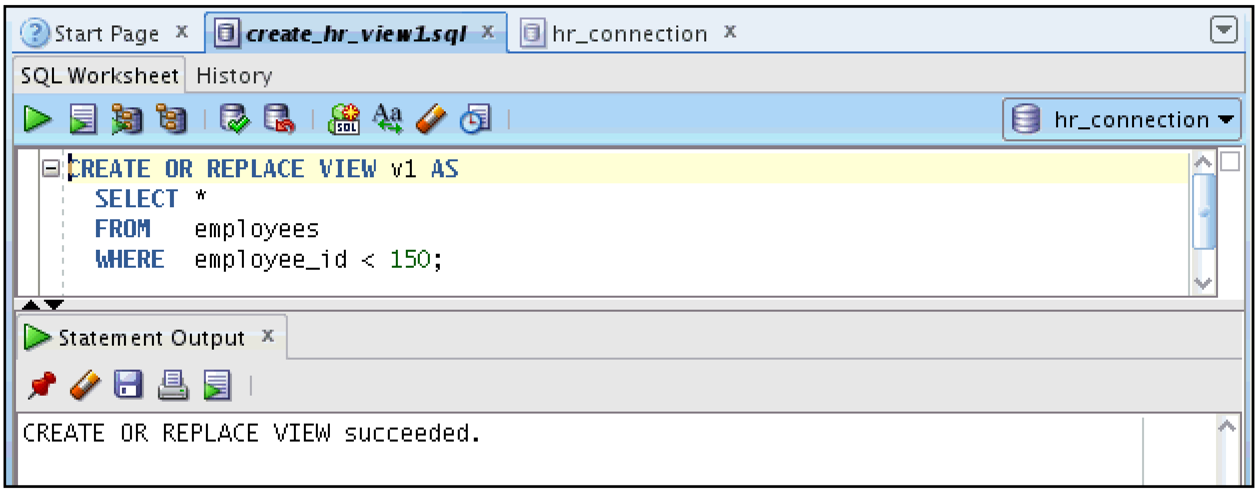
    Password: `hr`

    SID: `orcl`

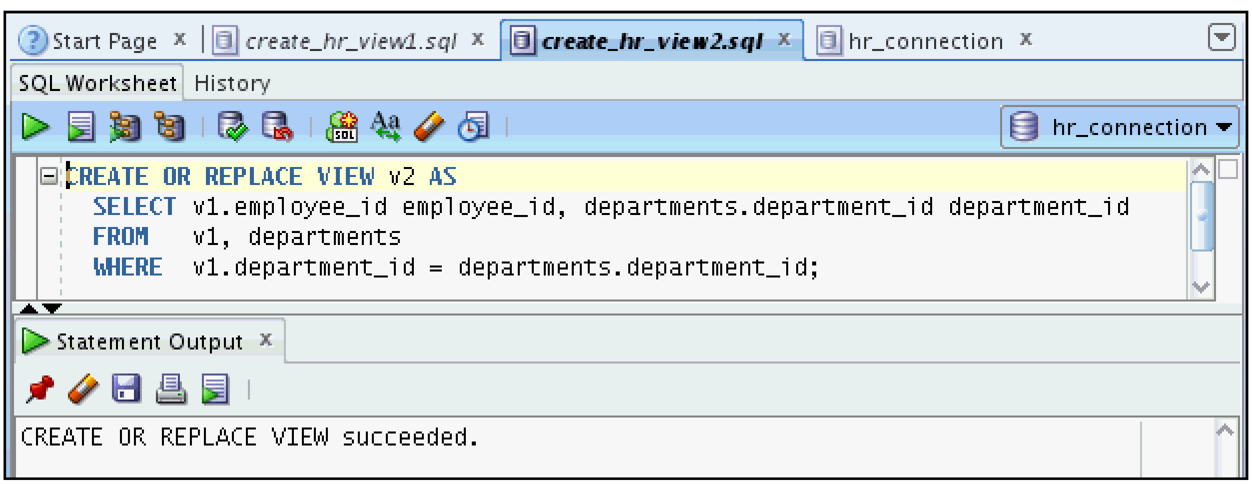
    Connect using the `hr_connection`.

12. Open and execute the `create_hr_view1.sql` script that creates a view called `V1` on top of the `EMPLOYEES` table, using `hr_connection`. After this is done, execute the `create_hr_view2.sql` script that creates a view `V2` on top of `V1`.

    a. Using the File Navigator, open and execute the `create_hr_view1.sql` script that creates a view called `V1` on top of the `EMPLOYEES` table.

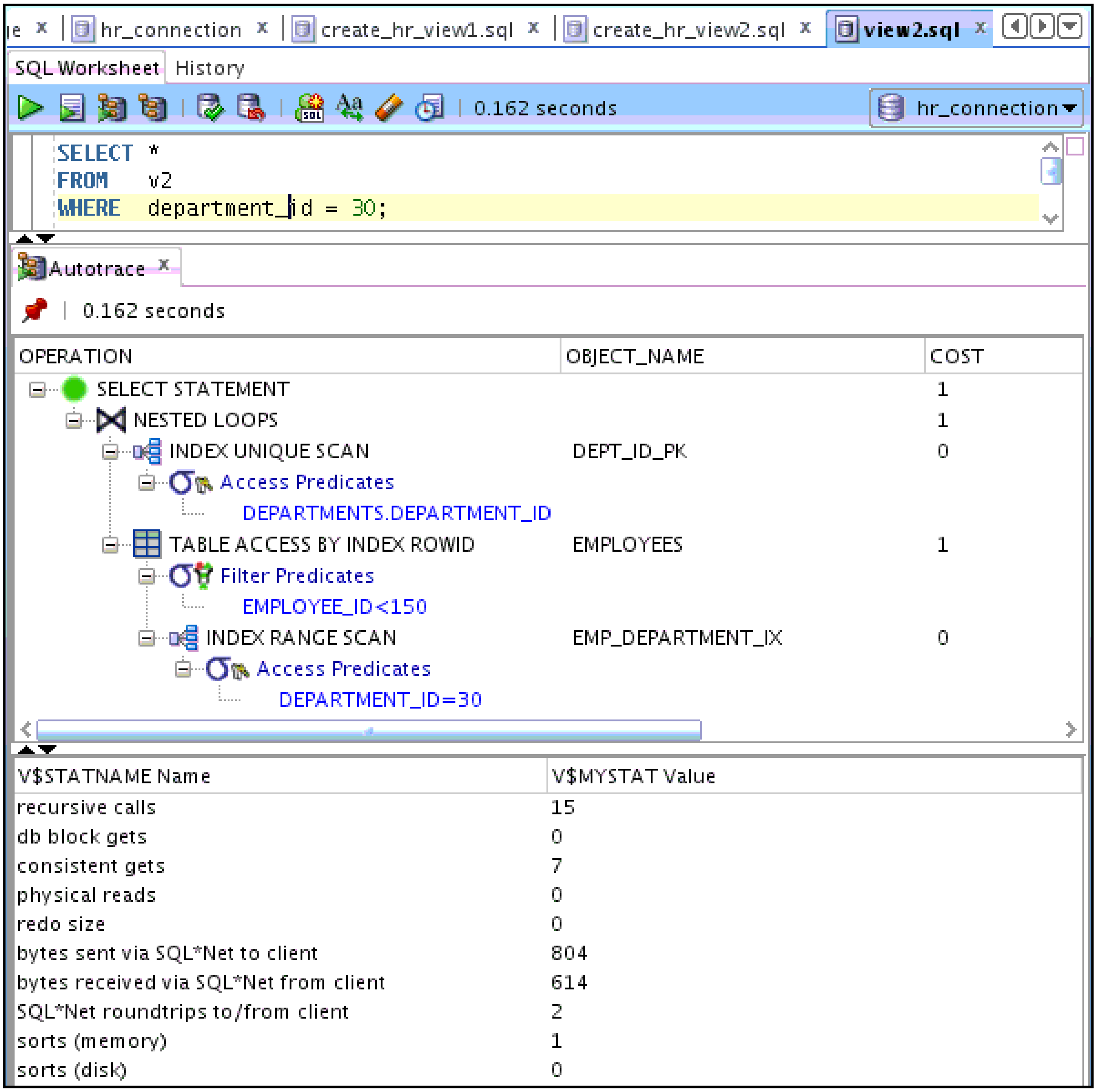


    b. Using the File Navigator, open and execute the `create_hr_view2.sql` script that creates a view called `V2` on top of `V1`



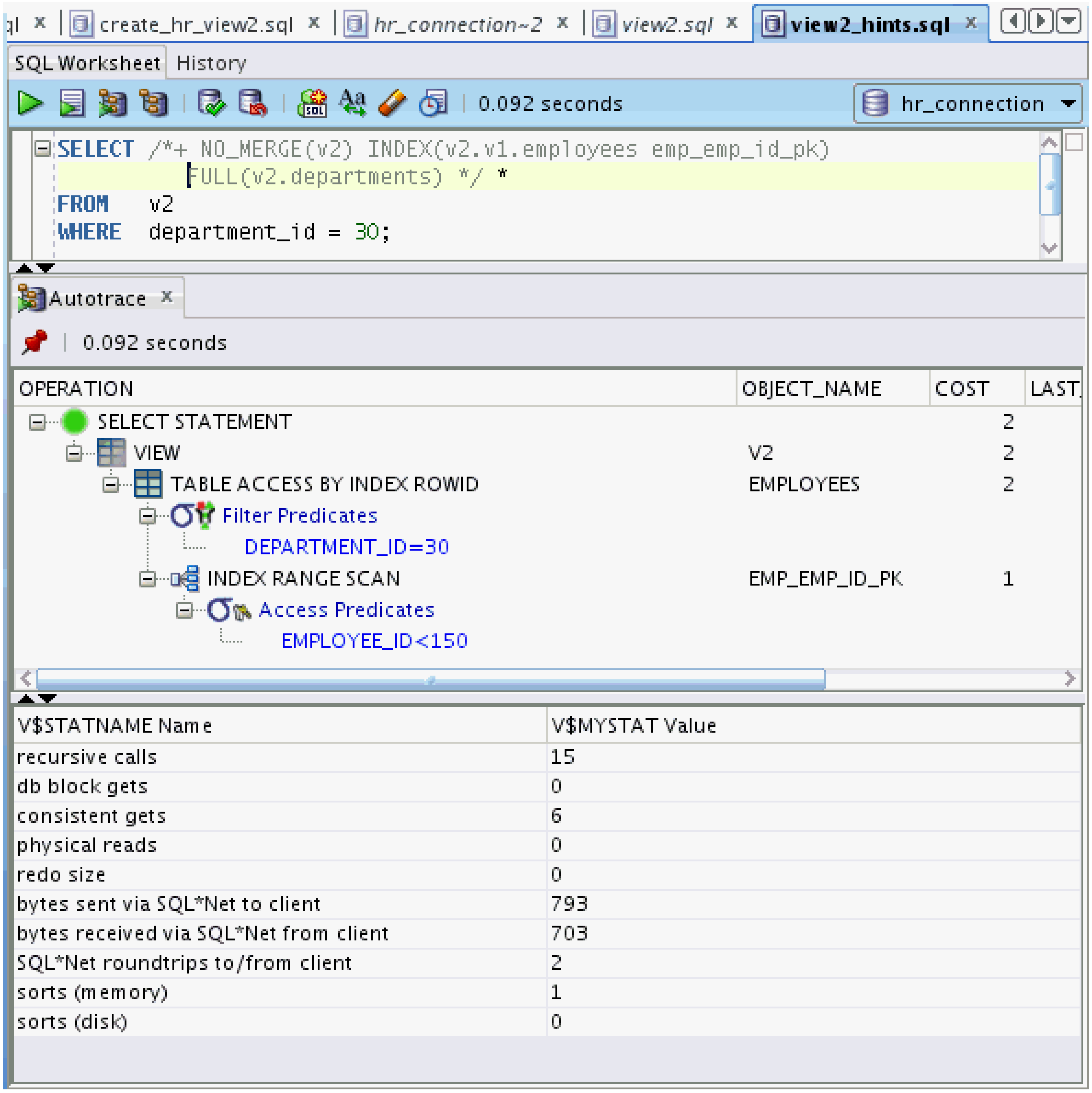
13. Open and Autotrace the query in the `view2.sql` script using `hr_connection`. Determine the execution plan used to process the following query:

```
SELECT * FROM v2 WHERE department_id = 30;
```



14. How do you force the query from step 10 to do a full table scan of the `departments` table and a range scan of the `emp_emp_id_pk` index?

You have to use extended hint syntax to be able to specify hints that apply to tables and indexes that appear in views. The `NO_MERGE` hint is used to make sure that the view is not merged into the surrounding query blocks. Open and Autotrace the `view2_hints.sql` script in an `hr_connection`.



15. In a terminal session, clean up your environment by executing the `hr_hint_cleanup.sh` script.

```
$ cd $HOME/solutions/Hints
$ ./hr_hint_cleanup.sh


…


SQL>
SQL> revoke dba from hr;
revoke dba from hr
*
ERROR at line 1:
ORA-01951: ROLE 'DBA' not granted to 'HR'
```

```
SQL>
SQL> drop view hr.v1;

View dropped.

SQL>
SQL> drop view hr.v2;

View dropped.

SQL>
SQL> exit;

…

$
```

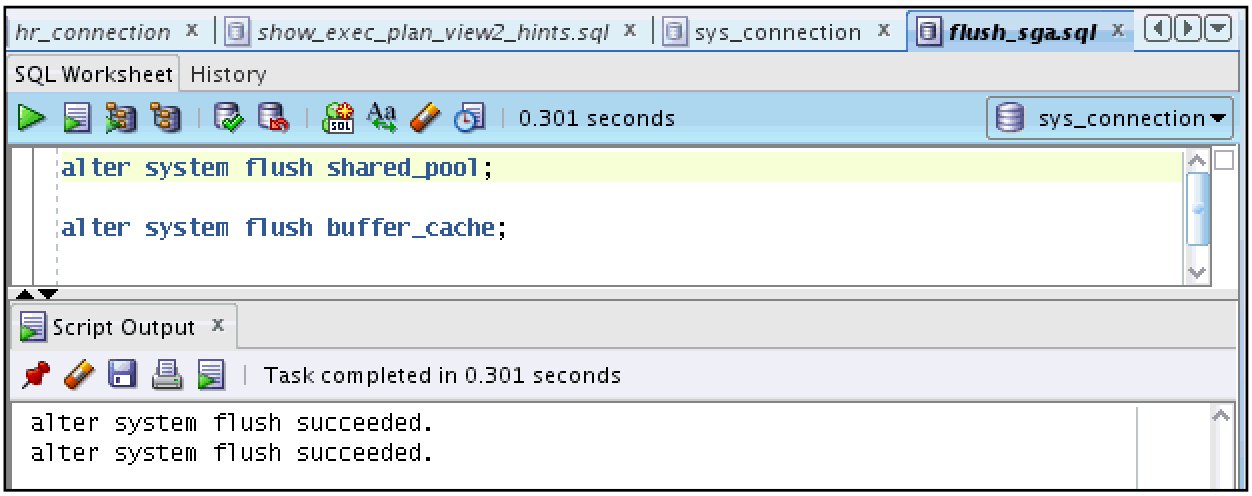
16. **Case 3: Join Hints:** In this third case, you investigate how to influence the choice the optimizer uses for joins. From your terminal session, execute the `sh_hint_setup.sh` script. This script deletes the statistics from the `SH` schema objects and sets the statistics for the `sales` table to very poor values.

```
$ ./sh_hint_setup.sh
…
SQL>
SQL> alter user sh identified by sh account unlock;

User altered.

SQL>
SQL> grant dba to sh;

Grant succeeded.

SQL>
SQL> connect sh/sh
Connected.
SQL>
SQL> exec dbms_stats.delete_schema_stats('SH');

PL/SQL procedure successfully completed.

SQL>
SQL> exec
dbms_stats.set_table_stats('SH','SALES',null,null,null,10,5);

PL/SQL procedure successfully completed.
```

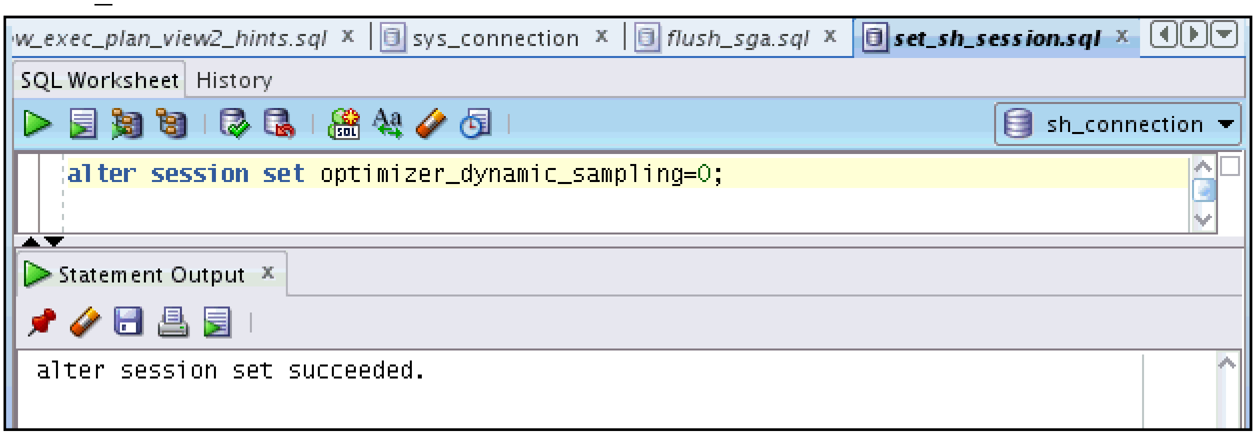
```
SQL>
SQL> exit;

...

$
```

17. In SQL Developer, using `sys_connection`, flush your SGA content using the `flush_sga.sql` script. And then set some important session parameters for `sh_connection` using the `set_sh_session.sql` script.

    a. Flush the SGA. Open and execute the `flush_sga.sql` script using `sys_connection`.

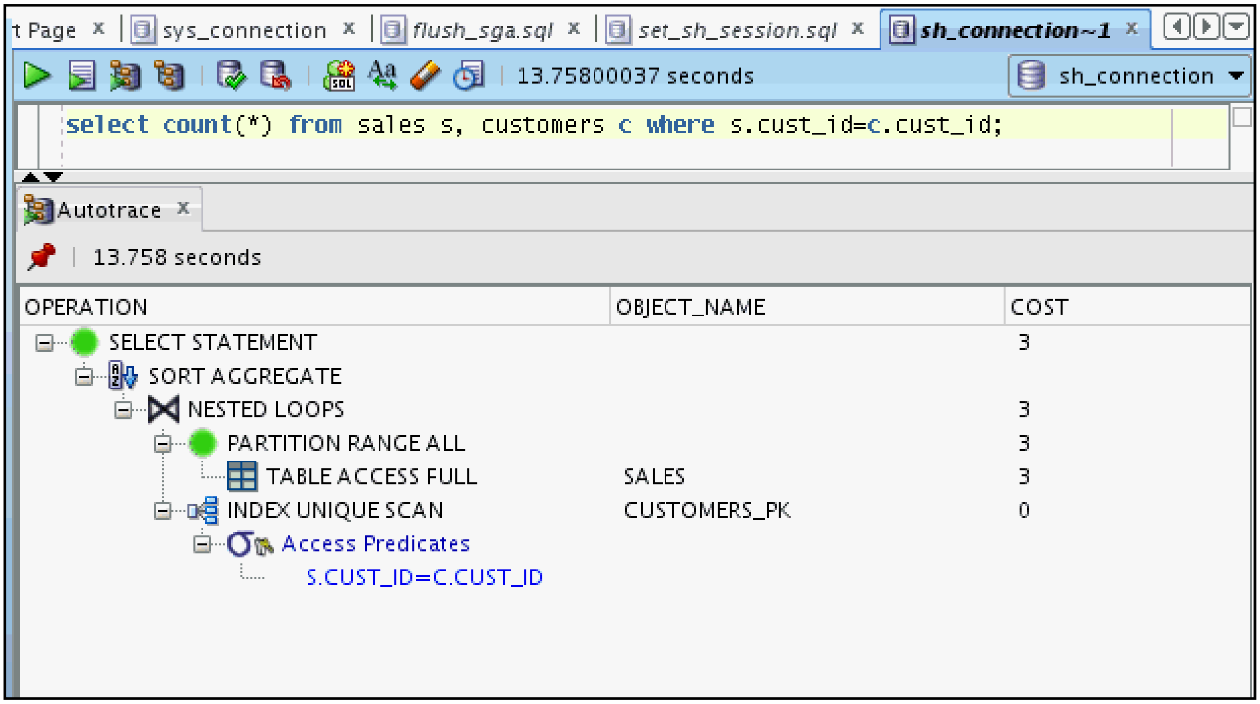


    b. Set session parameters. Open and execute the `set_sh_session.sql` script using `sh_connection`.

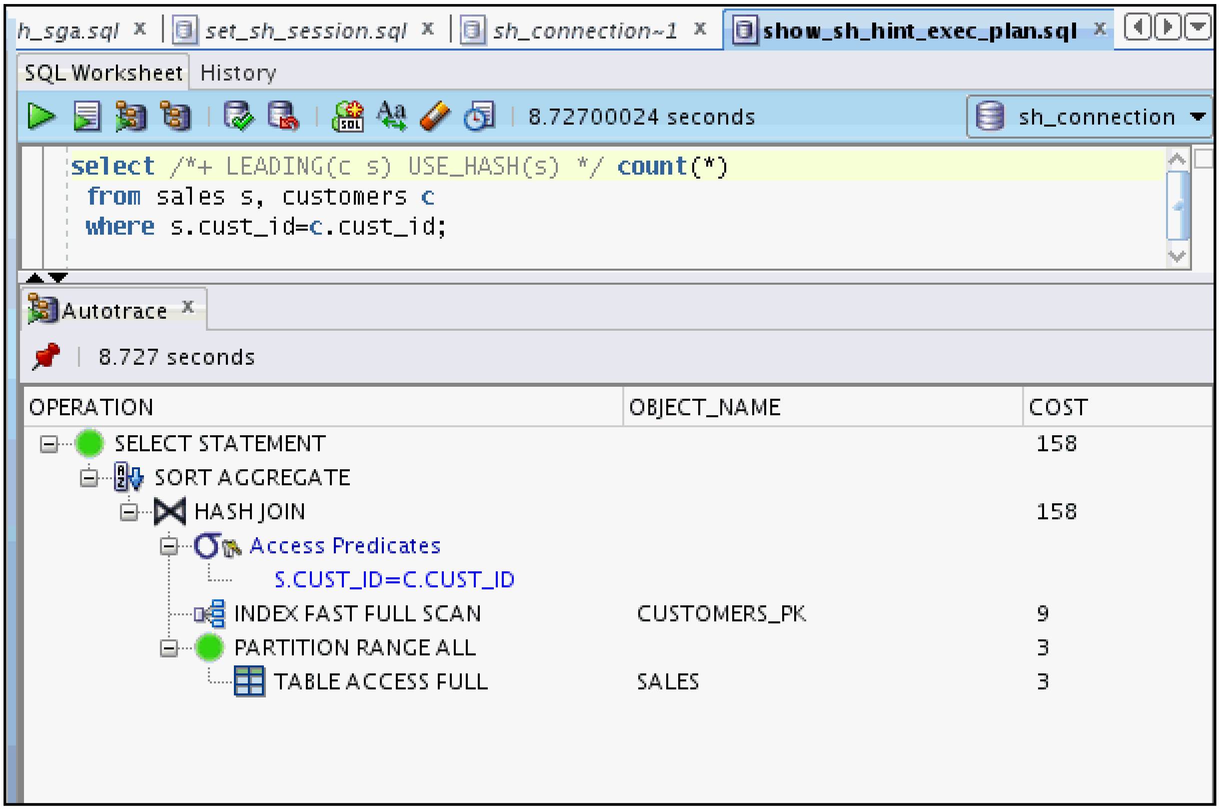


18. From your `sh_connection`, determine the execution plan of the following query: (You can open and autotrace `show_sh_exec_plan.sql`.)

```
select count(*) from sales s, customers c
where s.cust_id=c.cust_id;
```



19. Before you continue, ensure that you flush the content of your SGA to avoid caching issues later. Execute the `flush_sga.sql` script again.
20. Using only hints, how would you enhance the performance of the same query you executed in step 17? Make sure that you verify your implementation.
    a. In step 17, the optimizer chose a `NESTED LOOPS` join. You can try to force a hash join instead, using the `LEADING` and `USE_HASH` hints. Open and autotrace the `show_sh_hint_exec_plan.sql` script.



21. In a terminal session, clean up your environment by executing the `sh_hint_cleanup.sh` script.

```
$ ./sh_hint_cleanup.sh
…
Connected to: …

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> Connected.
SQL> SQL>
User altered.

SQL> SQL>
PL/SQL procedure successfully completed.

SQL> SQL> SQL> Disconnected …
$
```

22. **Case 4: Index Hints:** In this case, you study the influence of the `INDEX` hints. From your terminal session, execute the `sh_hint_index_setup.sh` script to set up the environment for this lab.

```
$ ./sh_hint_index_setup.sh
…

SQL>
SQL> alter user sh identified by sh account unlock;
```

```
User altered.

SQL>
SQL> grant dba to sh;

Grant succeeded.

SQL>
SQL> exit;
Disconnected …
$
```

23. From your terminal session, connect to the SQL*Plus session as the `SH` user. After you are connected, execute the following scripts to further set up your environment for this lab. Ensure that you stay connected to your session throughout this case.

```
$ sqlplus sh/sh

…

SQL> @drop_index_customers
SQL>
SQL> @dait
SQL>
SQL> drop index CUSTOMERS_YOB_BIX;

Index dropped.

SQL> drop index CUSTOMERS_MARITAL_BIX;

Index dropped.

SQL> drop index CUSTOMERS_GENDER_BIX;

Index dropped.

SQL>
SQL>
SQL> @create_cust_indexes
SQL> set echo on
SQL>
SQL> CREATE INDEX CUST_CUST_GENDER_idx
  2  ON CUSTOMERS(CUST_GENDER)
  3  NOLOGGING COMPUTE STATISTICS;

Index created.
```

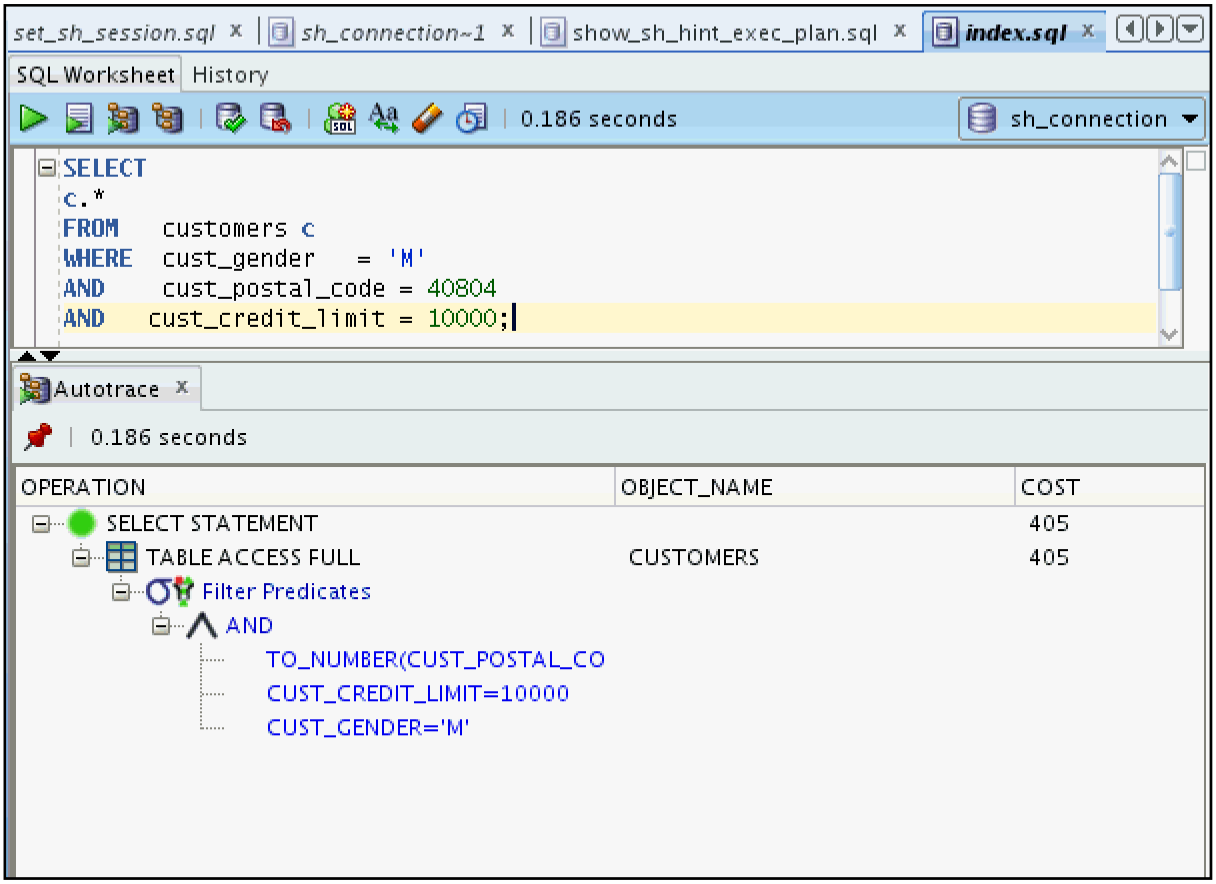
```
SQL>
SQL> CREATE INDEX CUST_CUST_POSTAL_CODE_idx
  2  ON CUSTOMERS(CUST_POSTAL_CODE)
  3  NOLOGGING COMPUTE STATISTICS;

Index created.

SQL>
SQL> CREATE INDEX CUST_CUST_CREDIT_LIMIT_idx
  2  ON CUSTOMERS(CUST_CREDIT_LIMIT)
  3  NOLOGGING COMPUTE STATISTICS;

Index created.

SQL>

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

REM    drop all indexes on CUSTOMERS table
REM    does not touch indexes associated with constraints
REM    ==================================================

set    termout off
store  set sqlplus_settings replace
save   buffer.sql replace
set    heading off verify off autotrace off feedback off

spool  dait.sql

SELECT 'drop index '||i.index_name||';'
FROM   user_indexes i
WHERE  i.table_name = 'CUSTOMERS'
AND    NOT EXISTS
      (SELECT 'x'
       FROM   user_constraints c
       WHERE  c.index_name = i.index_name
       AND    c.table_name = i.table_name
       AND    c.status = 'ENABLED');

spool  off

get    buffer.sql nolist
@sqlplus_settings
set    termout on
```

```
set echo on

@dait

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


set echo on

CREATE INDEX CUST_CUST_GENDER_idx
ON CUSTOMERS(CUST_GENDER)
NOLOGGING COMPUTE STATISTICS;

CREATE INDEX CUST_CUST_POSTAL_CODE_idx
ON CUSTOMERS(CUST_POSTAL_CODE)
NOLOGGING COMPUTE STATISTICS;

CREATE INDEX CUST_CUST_CREDIT_LIMIT_idx
ON CUSTOMERS(CUST_CREDIT_LIMIT)
NOLOGGING COMPUTE STATISTICS;
```

24. Use SQL Developer Autotrace to determine the execution plan for the following query:

```
SELECT
c.*
FROM   customers c
WHERE  cust_gender   = 'M'
AND    cust_postal_code = 40804
AND   cust_credit_limit = 10000;
```

   a. You can find this query in the `index.sql` script. Autotrace using `sh_connection`.



25. Try to get a better execution plan using the `INDEX` hint for the same query you investigated in step 23. Which index is best suited?

   a. Open and autotrace the `index_hint.sql` script inserting each index for `&indexname` in the script. The three indexes are:

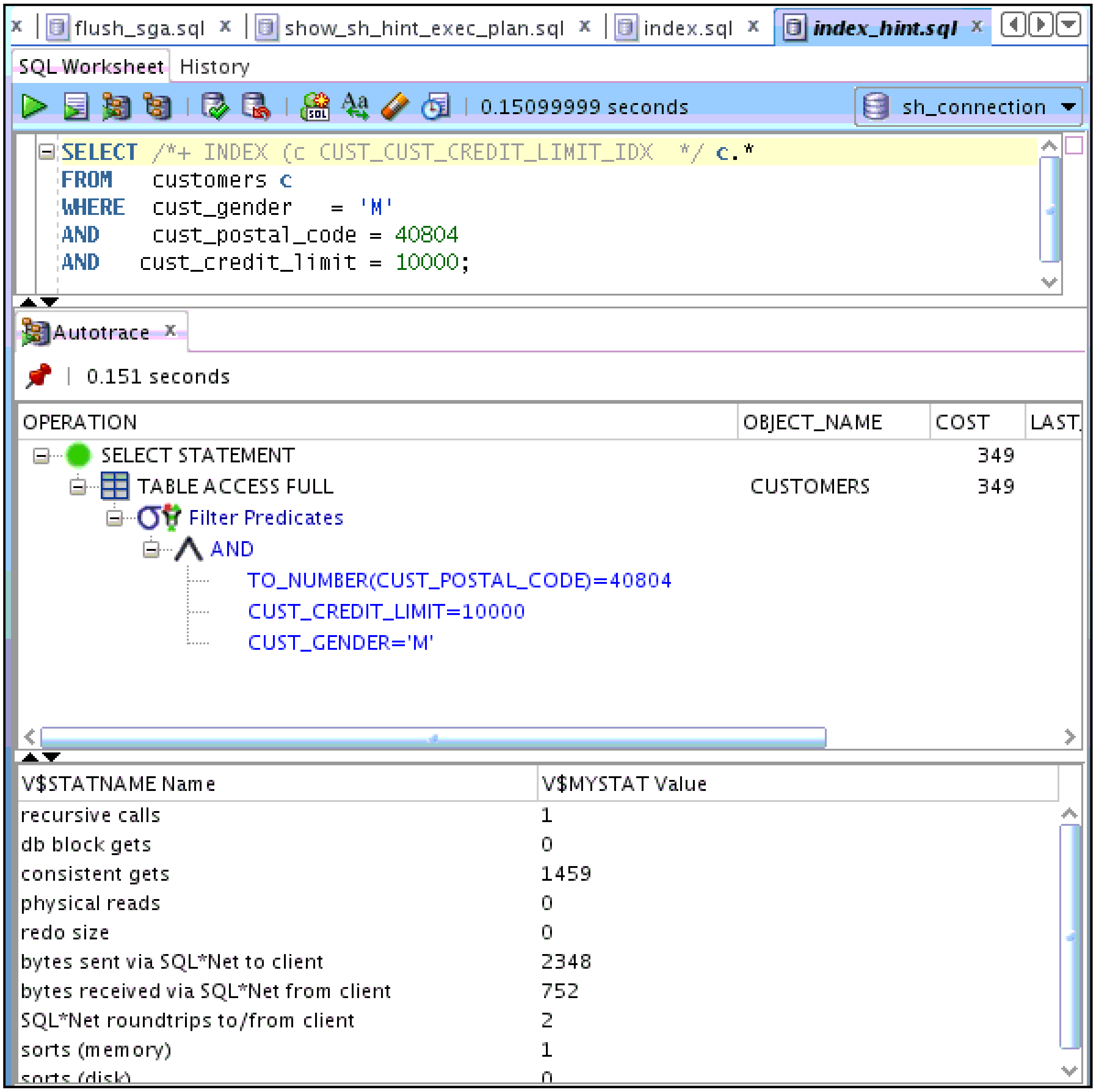
      `CUST_CUST_CREDIT_LIMIT_IDX`

      `CUST_CUST_GENDER_IDX`

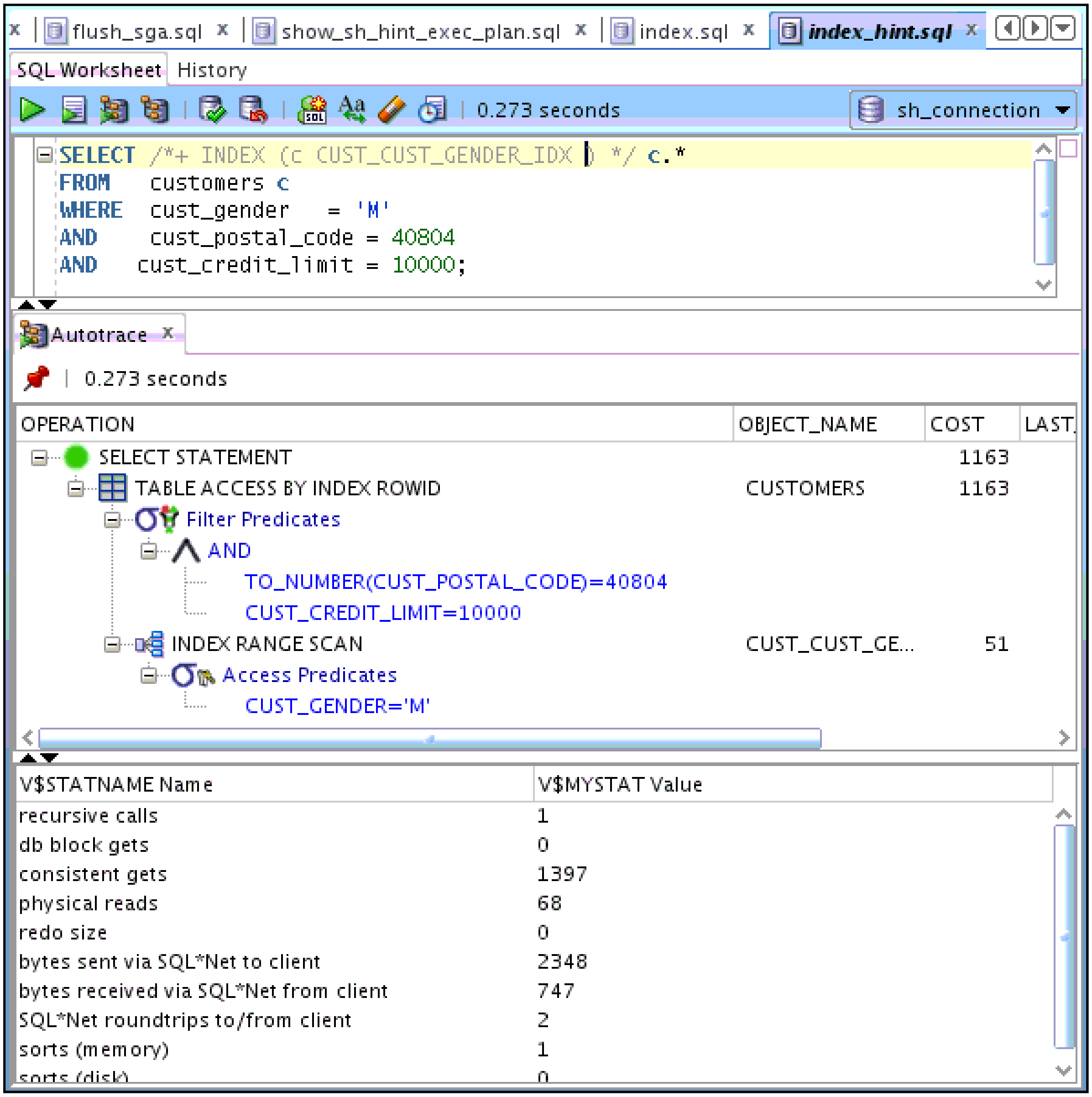
      `CUST_CUST_POSTAL_CODE_IDX`

      Note: Type over the string `&indexname` with the name of the index above, and then autotrace.

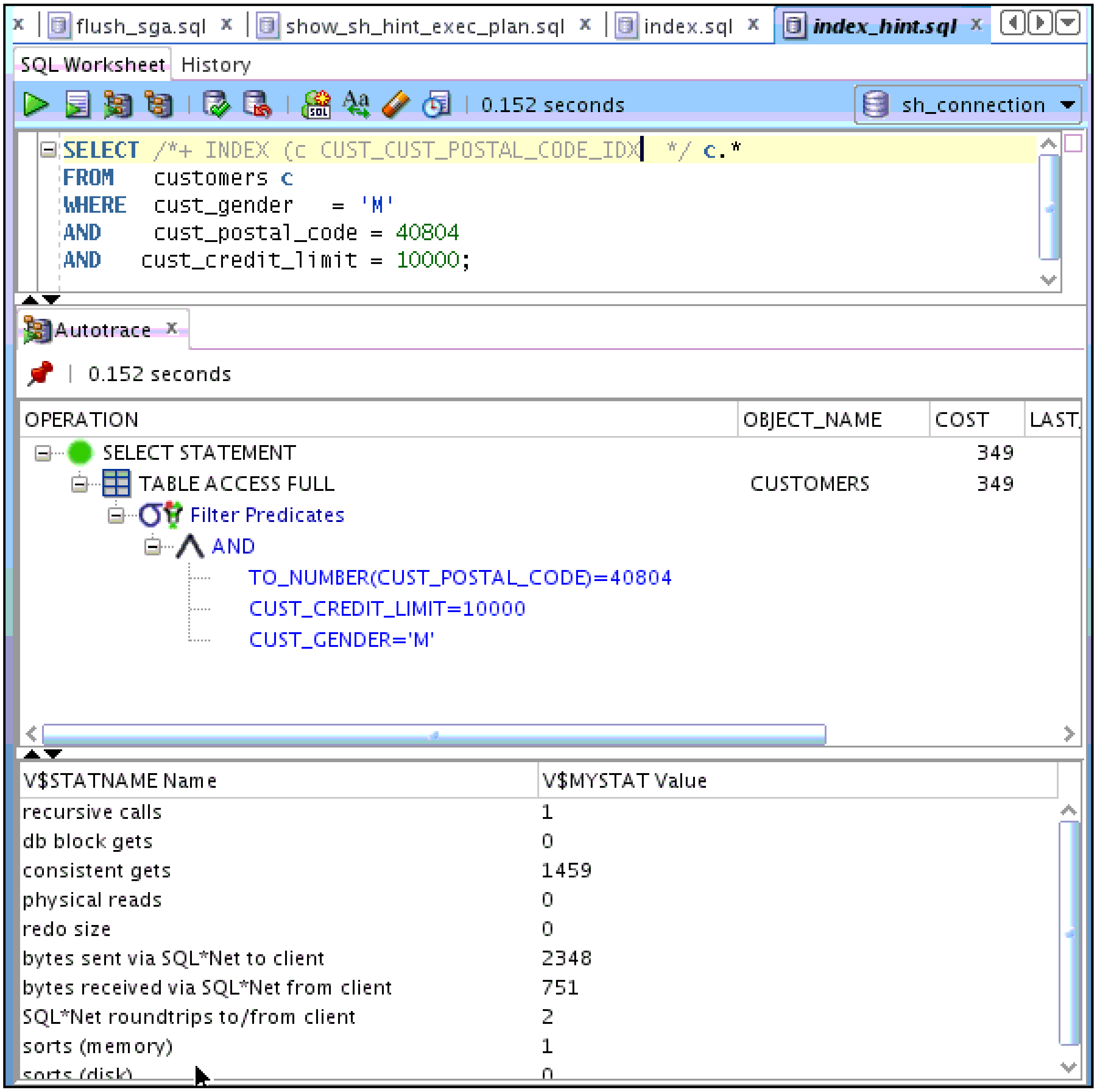
      1) The autotrace for `CUST_CUST_CREDIT_LIMIT_IDX` is:



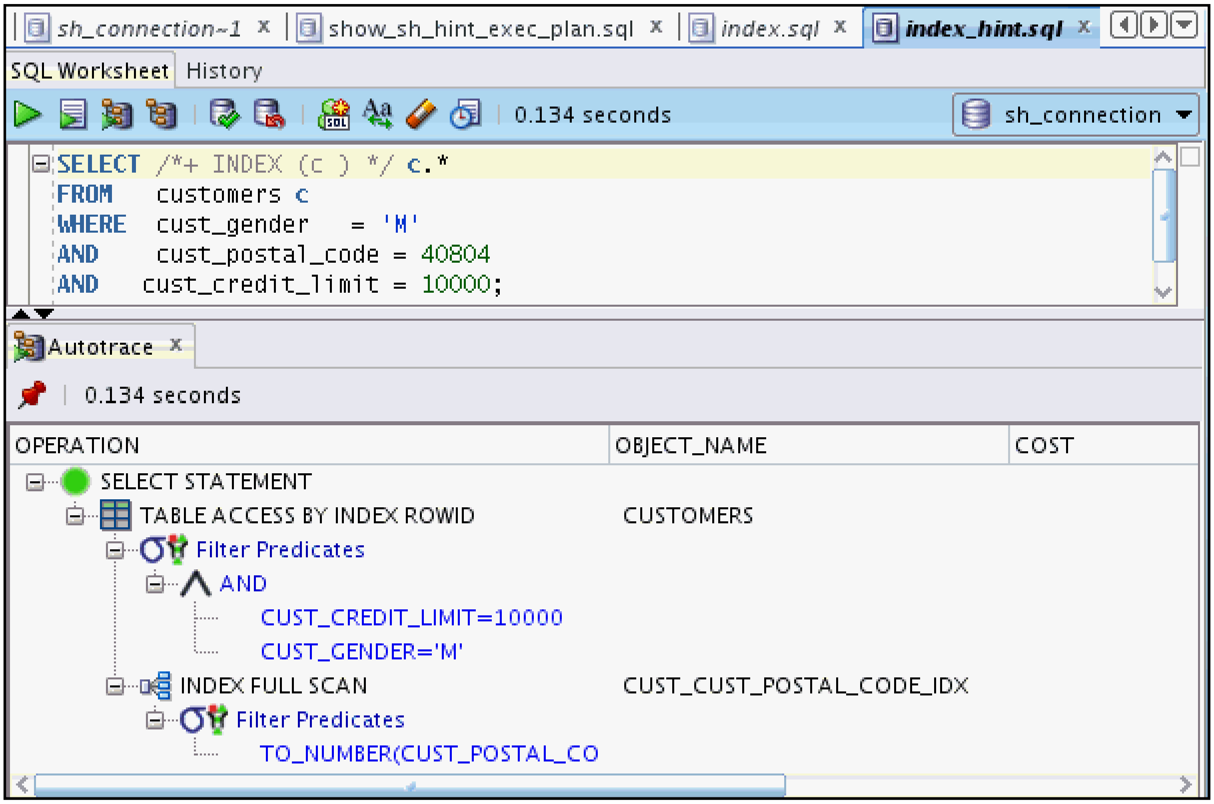
2) The autotrace for `CUST_CUST_GENDER_IDX` is:



3) The autotrace for `CUST_CUST_POSTAL_CODE_IDX` is



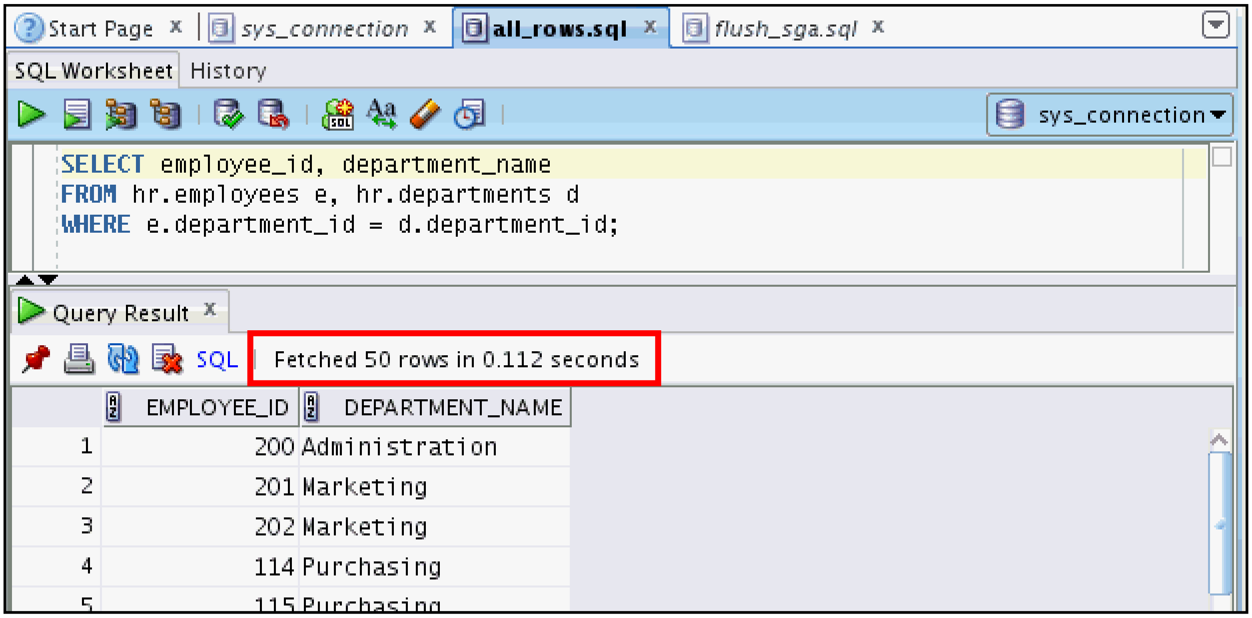
b. The `CUST_CUST_POSTAL_CODE_IDX` index is the best one for this query. Note that using the `INDEX` hint without specifying any index leads the optimizer to use the `CUST_CUST_POSTAL_CODE_IDX` index.

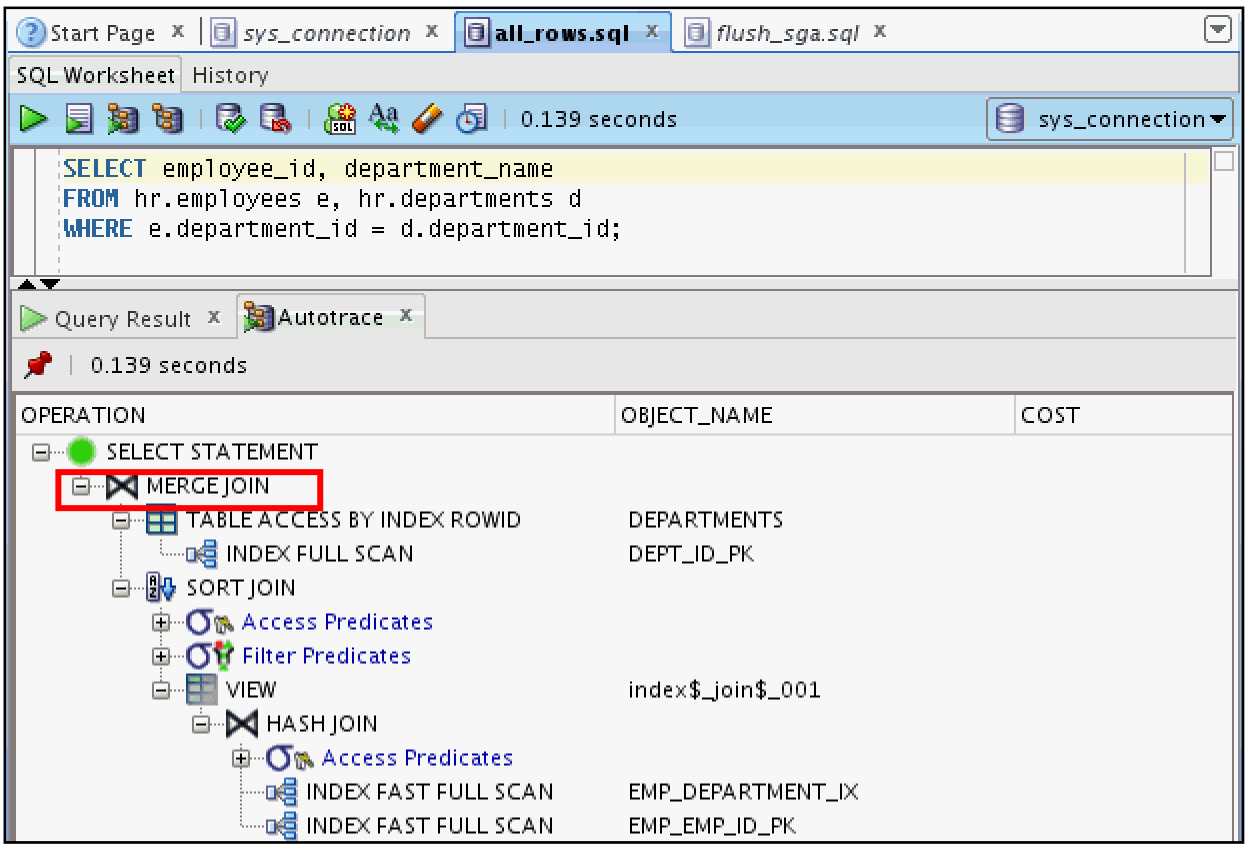


26. **Case 5: First Rows Hint:** In this case, you retrieve the first rows of your query as fast as possible. Using `sys_connection`, execute the statement and then autotrace the following query. This query is in the `all_rows.sql` script. Based on the fact that you want to retrieve the first 10 rows as fast as possible, what do you observe?

```
SELECT employee_id, department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id = d.department_id;
```

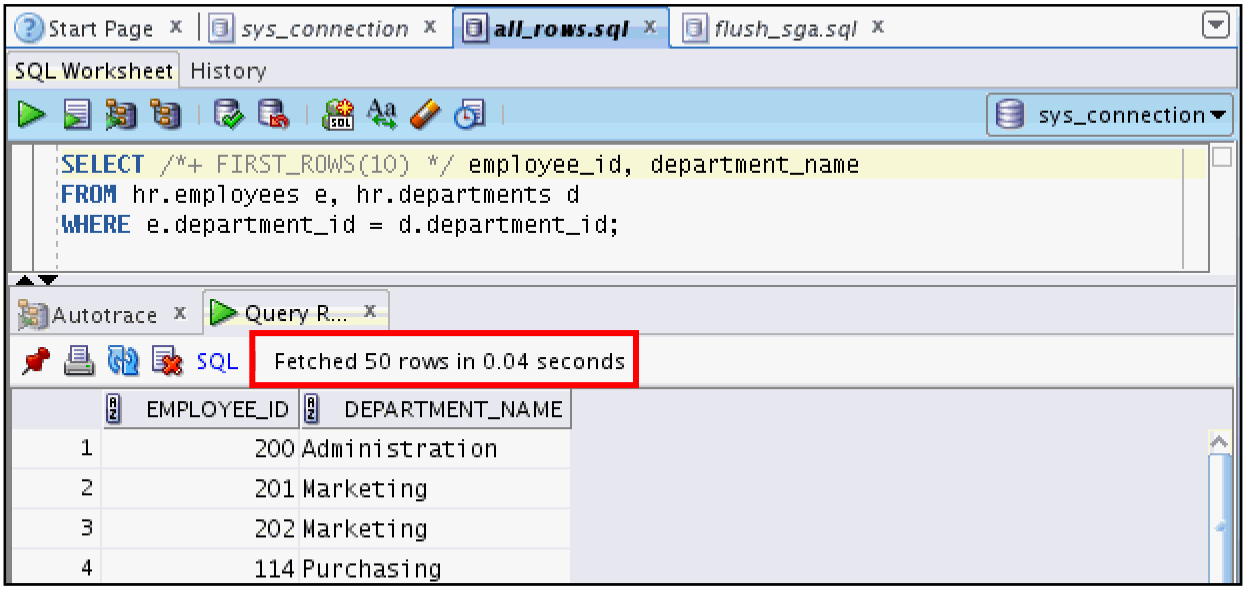
a. It takes some time before the result appears on the screen. This is because a merge join is used. It needs to sort all data before producing rows.

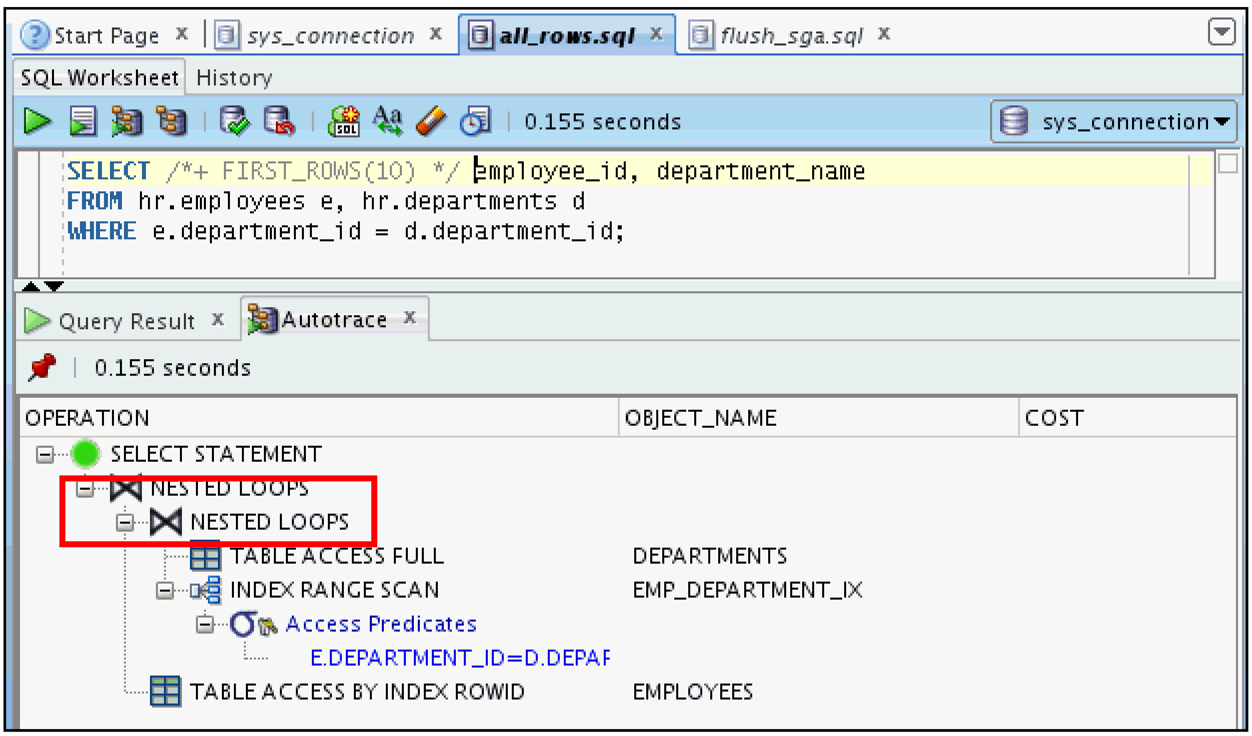




27. Using a hint, how can you ensure that the previous query starts fetching rows faster? Test your solution. The `first_rows.sql` script has the query with the hint.

    a. Using the `FIRST_ROWS(10)` hint, a nested loop is chosen by the optimizer. It is faster to retrieve the first rows.





28. In a SQL Developer session, close all connections by selecting File > Close All. Then exit SQL Developer by selecting File > Exit.
29. Exit from any SQL*Plus sessions and clean up your environment by executing the `sh_index_cleanup.sh` script. Look for the user dropped message, if this is not successful, there is most likely a session connected as the `SH` user. In this case, exit the session and execute the script again.

```
$ ./sh_index_cleanup.sh

…

SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> SQL> Connected.
```

```
SQL> SQL>
User altered.

SQL> SQL> SQL> Rem
SQL> Rem $Header: sh_main.sql 06-mar-2008.15:00:45 cbauwens Exp $
SQL> Rem
SQL> Rem sh_main.sql
SQL> Rem
...
SQL> Rem
SQL>
SQL> SET ECHO OFF

specify password for SH as parameter 1:

specify default tablespace for SH as parameter 2:

specify temporary tablespace for SH as parameter 3:

specify password for SYS as parameter 4:

specify directory path for the data files as parameter 5:

writeable directory path for the log files as parameter 6:

specify version as parameter 7:


Session altered.


User dropped.

old   1: CREATE USER sh IDENTIFIED BY &pass
new   1: CREATE USER sh IDENTIFIED BY sh

User created.

old   1: ALTER USER sh DEFAULT TABLESPACE &tbs
new   1: ALTER USER sh DEFAULT TABLESPACE example
old   2:  QUOTA UNLIMITED ON &tbs
new   2:  QUOTA UNLIMITED ON example

User altered.

old   1: ALTER USER sh TEMPORARY TABLESPACE &ttbs
```

```
new   1: ALTER USER sh TEMPORARY TABLESPACE temp

User altered.


Grant succeeded.



...



Grant succeeded.


PL/SQL procedure successfully completed.

Connected.

Grant succeeded.

old   1: CREATE OR REPLACE DIRECTORY data_file_dir AS '&data_dir'
new   1: CREATE OR REPLACE DIRECTORY data_file_dir AS
'/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/'

Directory created.

old   1: CREATE OR REPLACE DIRECTORY log_file_dir AS '&log_dir'
new   1: CREATE OR REPLACE DIRECTORY log_file_dir AS '/home/oracle/'

Directory created.


Grant succeeded.


Grant succeeded.


Grant succeeded.

Connected.

Session altered.


Session altered.
```

```
Table created.


...


Comment created.


Creating OLAP metadata ...
<<<<< CREATE CWMLite Metadata for the Sales History Schema >>>>>
…
<<<<< 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.

SQL> SQL> Disconnected …
$

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

#!/bin/bash

cd /home/oracle/solutions/SQL_Access_Advisor/sh

cp * $ORACLE_HOME/demo/schema/sales_history

sqlplus / as sysdba <<FIN!

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
```

```
SET LINESIZE 8000
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET LONG 1000

CONNECT / AS SYSDBA

alter user sh identified by sh account unlock;

@sh_main sh example temp oracle
/u01/app/oracle/product/11.2.0/dbhome_1/demo/schema/sales_history/
/home/oracle/ v3

exit;

FIN!
```