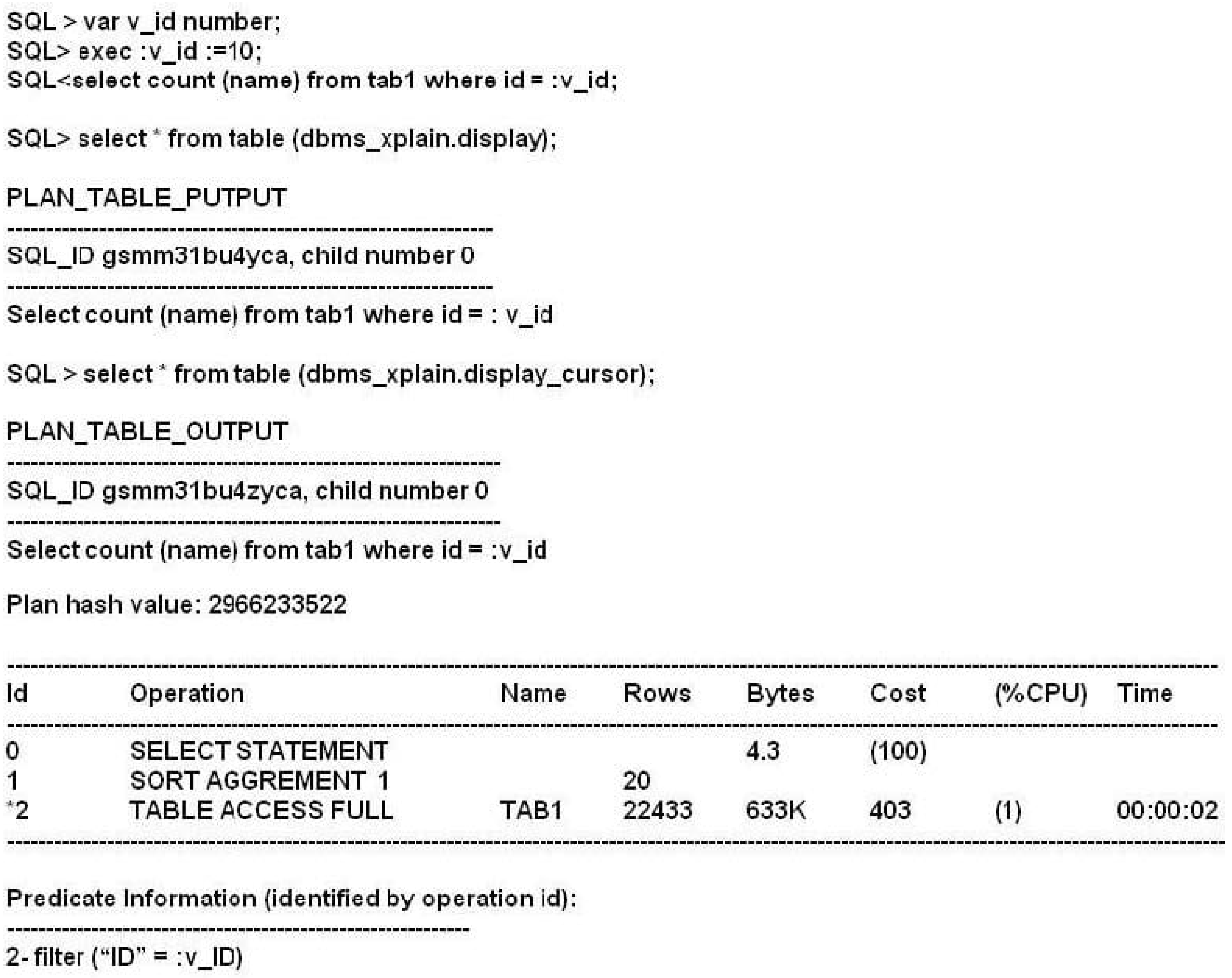
1Z0-117^Q&As

Oracle Database 11g Release 2: SQL Tuning Exam

QUESTION 1

Exhibit



```
SQL > var v_id number;
SQL> exec :v_id :=10;
SQL<select count (name) from tab1 where id = :v_id;

SQL> select * from table (dbms_xplain.display);

PLAN_TABLE_PUTPUT
------------------------------------------------------------
SQL_ID gsmm31bu4yca, child number 0
------------------------------------------------------------
Select count (name) from tab1 where id = : v_id

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

PLAN_TABLE_OUTPUT
------------------------------------------------------------
SQL_ID gsmm31bu4zyca, child number 0
------------------------------------------------------------
Select count (name) from tab1 where id = :v_id

Plan hash value: 2966233522

------------------------------------------------------------------------------------------------------------
Id        Operation                   Name     Rows     Bytes    Cost     (%CPU)   Time
------------------------------------------------------------------------------------------------------------
0         SELECT STATEMENT                              4.3      (100)
1         SORT AGGREGENT  1                    20
*2        TABLE ACCESS FULL           TAB1     22433    633K     403      (1)      00:00:02
------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
------------------------------------------------------------
2- filter ("ID" = :v_ID)
```

A table has three distinct values in its ID column. In the ID column, values 10 and 20 have more than 20000 rows each and value 30 has only five rows. The

statistics for the schema have been updated recently.

The CURSOR_SHARING parameter is set to EXACT.

The query was executed recently and the cursor for the query is bind aware. Examine the exhibits to view the commands executed.

You plan to execute the same query with a value of 30 for the bind variable V_ID.

Which statement is true in this scenario?

A. The same execution plan will always be used irrespective of the value in the bind variable.

B. A new execution plan will be generated depending on the access pattern and the bind value.

C. Adaptive cursor sharing will ensure that a new cursor is generated for each distinct value in the bind variable.

D. Adaptive cursor sharing will happen only if you use the literal values instead of bind variables in the query.

Correct Answer: C

Note:

*

CURSOR_SHARING determines what kind of SQL statements can share the same cursors.

*

Setting CURSOR_SHARING to EXACT allows SQL statements to share the SQL area only when their texts match exactly. This is the default behavior. Using

this setting, similar statements cannot shared; only textually exact statements can be shared.

*

Values:

*

FORCE

Forces statements that may differ in some literals, but are otherwise identical, to share a cursor, unless the literals affect the meaning of the statement.

*

SIMILAR

Causes statements that may differ in some literals, but are otherwise identical, to share a cursor, unless the literals affect either the meaning of the statement or

the degree to which the plan is optimized.

*

EXACT

Only allows statements with identical text to share the same cursor.

QUESTION 2

Which three are benefits of In-Memory Parallel Execution?

A. Reduction in the duplication of block images across multiple buffer caches

B. Reduction in CPU utilization

C. Reduction in the number of blocks accessed

D. Reduction in physical I/O for parallel queries

E. Ability to exploit parallel execution servers on remote instance

Correct Answer: ACD

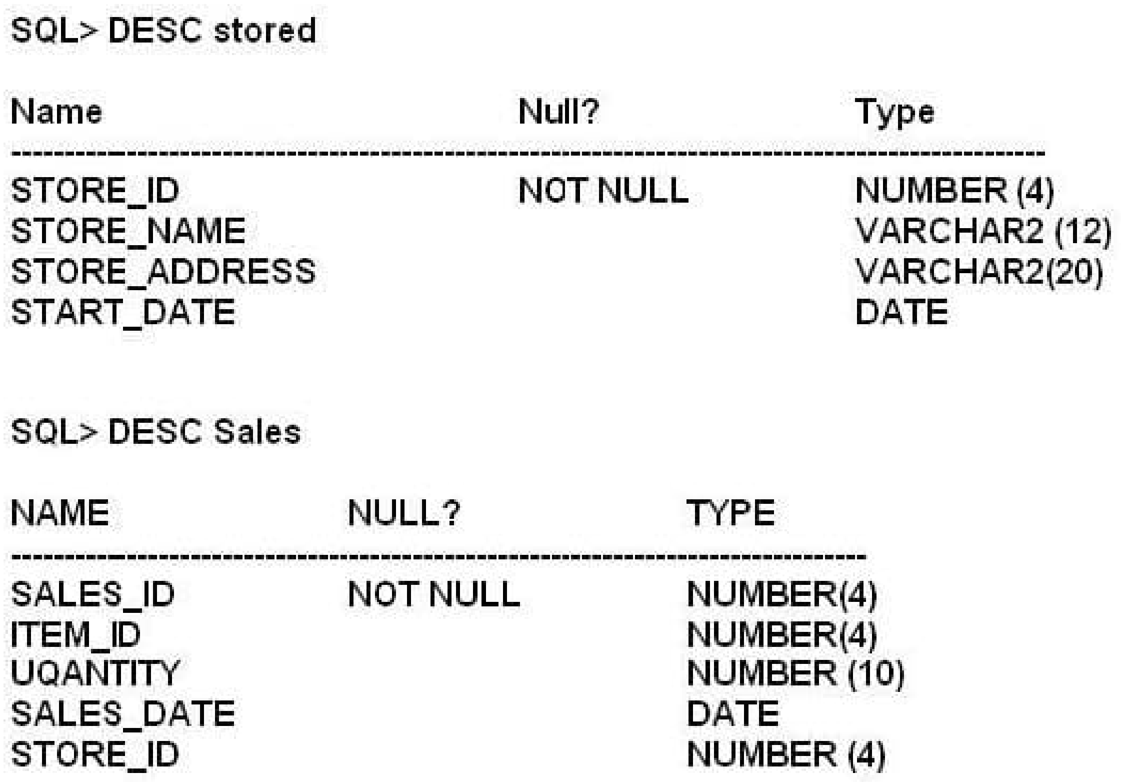
Note: In-Memory Parallel Execution

When the parameter PARALLEL_DEGREE_POLICY is set to AUTO, Oracle Database decides if an object that is accessed using parallel execution would benefit from being cached in the SGA (also called the buffer cache). The decision to cache an object is based on a well-defined set of heuristics including the size of the object and frequency on which it is accessed. In an Oracle RAC environment, Oracle Database maps pieces of the object into each of the buffer caches on the active instances. By creating this mapping, Oracle Database automatically knows which buffer cache to access to find different parts or pieces of the object. Using this information, Oracle Database prevents multiple instances from reading the same information from disk over and over again, thus maximizing the amount of memory that can cache objects. If the size of the object is larger than the size of the buffer cache (single instance) or the size of the buffer cache multiplied by the number of active instances in an Oracle RAC cluster, then the object is read using direct-path reads.

Reference: Oracle Database VLDB and Partitioning Guide 11g, How Parallel Execution Works

QUESTION 3

Refer to the Exhibit.

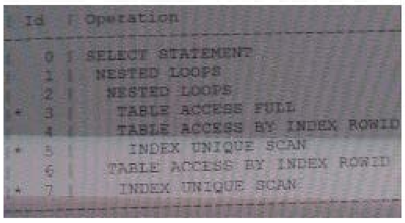


```
SQL> DESC stored

Name                    Null?           Type
-------------------------------------------------------------
STORE_ID                NOT NULL        NUMBER (4)
STORE_NAME                              VARCHAR2 (12)
STORE_ADDRESS                           VARCHAR2(20)
START_DATE                              DATE

SQL> DESC Sales

NAME            NULL?           TYPE
-----------------------------------------------
SALES_ID        NOT NULL        NUMBER(4)
ITEM_ID                         NUMBER(4)
UQANTITY                        NUMBER (10)
SALES_DATE                      DATE
STORE_ID                        NUMBER (4)
```

Execution plan: What must be the correct order of steps that the optimizer executes based on the ID column the execution plan?

```
Plan hash value: 29632623819
-------------------------------------------------------------------------------------------------------------
Id    Operation                        Name           Rows   Bytes   Cost   (%CPU)
-------------------------------------------------------------------------------------------------------------
0     SELECT STATEMENT                                3      189     10     (10)
1     NESTED LOOPS                                    3      189     10     (10)
2     NESTED LOOPS                                    3      141     7      (15)
*3    TABLE ACCESS FULL                EMPLOYEES      3      60      4      (25)
4     TABLE ACCESS BY INDEX ROWID      JOBS           19     513     2      (50)
*5    INDEX UNIQUE SCAN                JOB_ID_PK      1
6     TABLE ACCESS BY INDEX ROWID      DEPARTMENTS    27     432     2      (50)
7     INDEX UNIQUE SCAN                DEPT_ID_PK     1
-------------------------------------------------------------------------------------------------------------
```

```
PREDICATE Information (identified by operation id):
-----------------------------------------------------------------
3 – filter (“E”. “EMPLOYEE_ID”<103)
5 – access (“E”. “JOB_ID” = “J_ID”)
7 – access (“E”. “DEPARTMENT_ID” = “D”. “DEPARTMENT_ID”)
```



```
| Id  | Operation
---------------------------------------------
|   0 | SELECT STATEMENT
|   1 |  NESTED LOOPS
|   2 |   NESTED LOOPS
| * 3 |    TABLE ACCESS FULL
|   4 |    TABLE ACCESS BY INDEX ROWID
| * 5 |     INDEX UNIQUE SCAN
|   6 |   TABLE ACCESS BY INDEX ROWID
| * 7 |    INDEX UNIQUE SCAN
---------------------------------------------
```

A. 3, 5, 4, 6, 7

B. 3, 5, 4, 7, 6

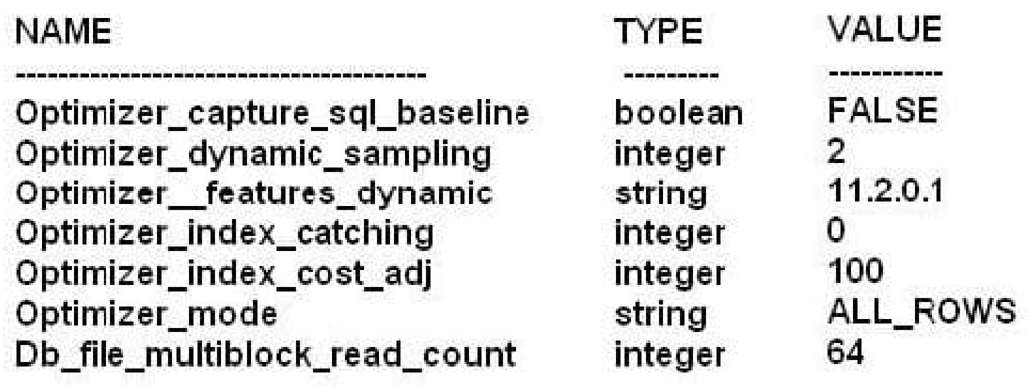
C. 3, 4, 5, 7, 6

D. 4, 5, 3, 7, 6

Correct Answer: D

QUESTION 4

Examine the utilization parameters for an instance:



NAME	TYPE	VALUE
Optimizer_capture_sql_baseline	boolean	FALSE
Optimizer_dynamic_sampling	integer	2
Optimizer__features_dynamic	string	11.2.0.1
Optimizer_index_catching	integer	0
Optimizer_index_cost_adj	integer	100
Optimizer_mode	string	ALL_ROWS
Db_file_multiblock_read_count	integer	64

You notice that despite having an index on the column used in the where clause, queries use full table scans with highly selective filters.

What are two possible reasons for the optimizer to use full table scans instead of index unique scans and index range scans?

A. The OPTIMIZER_MODE parameter is set to ALL_ROWS.

B. The clustering factor for the indexes is high.

C. The number of leaf blocks for the indexes is high.

D. The OPTIMIZER_INDEX_COST_ADJ initialization parameter is set to 100.

E. The blocks fetched by the query are greater than the value specified by the DB_FILE_MULTIBLOCK_READ_COUNT parameter.

Correct Answer: DE

D: OPTIMIZER_INDEX_COST_ADJ lets you tune optimizer behavior for access path selection to be more or less index friendly--that is, to make the optimizer more or less prone to selecting an index access path over a full table scan.

The default for this parameter is 100 percent, at which the optimizer evaluates index access paths at the regular cost. Any other value makes the optimizer evaluate the access path at that percentage of the regular cost. For example, a setting of 50 makes the index access path look half as expensive as normal.

E: DB_FILE_MULTIBLOCK_READ_COUNT is one of the parameters you can use to minimize I/O during table scans. It specifies the maximum number of blocks read in one I/O operation during a sequential scan. The total number of I/Os needed to perform a full table scan depends on such factors as the size of the table, the multiblock read count, and whether parallel execution is being utilized for the operation. g release 2, the default value of this parameter is a value that As of Oracle Database 10 corresponds to the maximum I/O size that can be performed efficiently. This value is platform-dependent and is 1MB for most platforms.Because the parameter is expressed in blocks, it will be set to a value that is equal to the maximum I/O size that can be performed efficiently divided by the standard block size. Note that if the number of sessions is extremely large the multiblock read count value is decreased to avoid the buffer cache getting flooded with too many table scan buffers.

Even though the default value may be a large value, the optimizer will not favor large plans if you do not set this parameter. It would do so only if you explicitly set this parameter to a large value.

Online transaction processing (OLTP) and batch environments typically have values in the range of 4 to 16 for this parameter. DSS and data warehouse environments tend to benefit most from maximizing the value of this parameter. The optimizer is more likely to choose a full table scan over an index if the value of this parameter is high.

Note:

* OPTIMIZER_MODE establishes the default behavior for choosing an optimization approach for the instance.

Values:

first_rows_n

The optimizer uses a cost-based approach and optimizes with a goal of best response time to return the first n rows (where n = 1, 10, 100, 1000).

first_rows

The optimizer uses a mix of costs and heuristics to find a best plan for fast delivery of the first few rows.

all_rows

The optimizer uses a cost-based approach for all SQL statements in the session and optimizes with a goal of best throughput (minimum resource use to complete

the entire statement).

QUESTION 5

You need to upgrade you Oracle Database 10g to 11g. You want to ensure that the same SQL plans that are currently in use in the 10g database are used in the upgraded database initially, but new, better plans are allowed subsequently.

Steps to accomplish the task:

1.

Set the OPTIMIZER_USE_SQL_BASELINE and OPTIMIZER_CAPTURE_SQL_PLAN_BASELINE to TRUE.

2.

Bulk load the SQL Management Base as part of an upgrade using an STS containing the plans captured in Oracle Database 10g.

3.

Evolve the plan baseline using the DBMS_SPM.EVOLVE_PLAN_BASELINE procedure.

4.

Fix the plan baseline using the DBMS_SPM.ALTER_SQL_PLANBASELINE procedure.

5.

Accept new, better plans using the DBMS_SPM.ALTER_SQL_PLAN_BASELINE procedure and manually load them to the existing baseline.

6.

Set OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES to FALSE.

Identify the required steps.

A. 1, 3, 4, 5

B. 1, 6, 3, 4, 5

C. 1, 2, 3, 5

D. 1, 2, 3, 4

E. 1, 6, 3

F. 1 and 2

Correct Answer: F

*

(1) OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES In Oracle Database 11g a new feature called SQL Plan Management (SPM) has been introduced to guarantees any plan changes that do occur lead to better performance. When OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is set to TRUE (default FALSE) Oracle will automatically capture a SQL plan baseline for every repeatable SQL statement on the system. The execution plan found at parse time will be added to the SQL plan baseline as an accepted plan.

*

(2) Once you have completed the software upgrade, but before you restart the applications and allow users back on the system, you should populate SQL Plan Management (SPM) with the 10g execution plans you captured before the upgrade. Seeding SPM with the 10g execution plans ensures that the application will continue to use the same execution plans you had before the upgrade. Any new execution plans found in Oracle Database 11g will be recorded in the plan history for that statement but they will not be used. When you are ready you can evolve or verify the new plans and only implement those that perform better than the 10g plan.

Incorrect:

Not (3): DBMS_SPM.EVOLVE_PLAN_BASELINE is not used to evolve new plans.

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE should be used:

It is possible to evolve a SQL statement\\'s execution plan using Oracle Enterprise Manager or by running the command-line function

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE. U

Note:

* SQL plan management (SPM) ensures that runtime performance will never degrade due to the change of an execution plan. To guarantee this, only accepted

(trusted) execution plans will be used; any plan will be tracked and evaluated at a later point in time and only accepted as verified if the new plan performs better

than an accepted plan. SQL Plan Management has three main components:

1.

SQL plan baseline capture:

Create SQL plan baselines that represents accepted execution plans for all relevant SQL statements. The SQL plan baselines are stored in a plan history inside

the SQL Management Base in the SYSAUX tablespace.

2.

SQL plan baseline selection

Ensure that only accepted execution plans are used for statements with a SQL plan baseline and track all new execution plans in the history for a statement as

unaccepted plan. The plan history consists of accepted and unaccepted plans. An unaccepted plan can be unverified (newly found but not verified) or rejected

(verified but not found to performant).

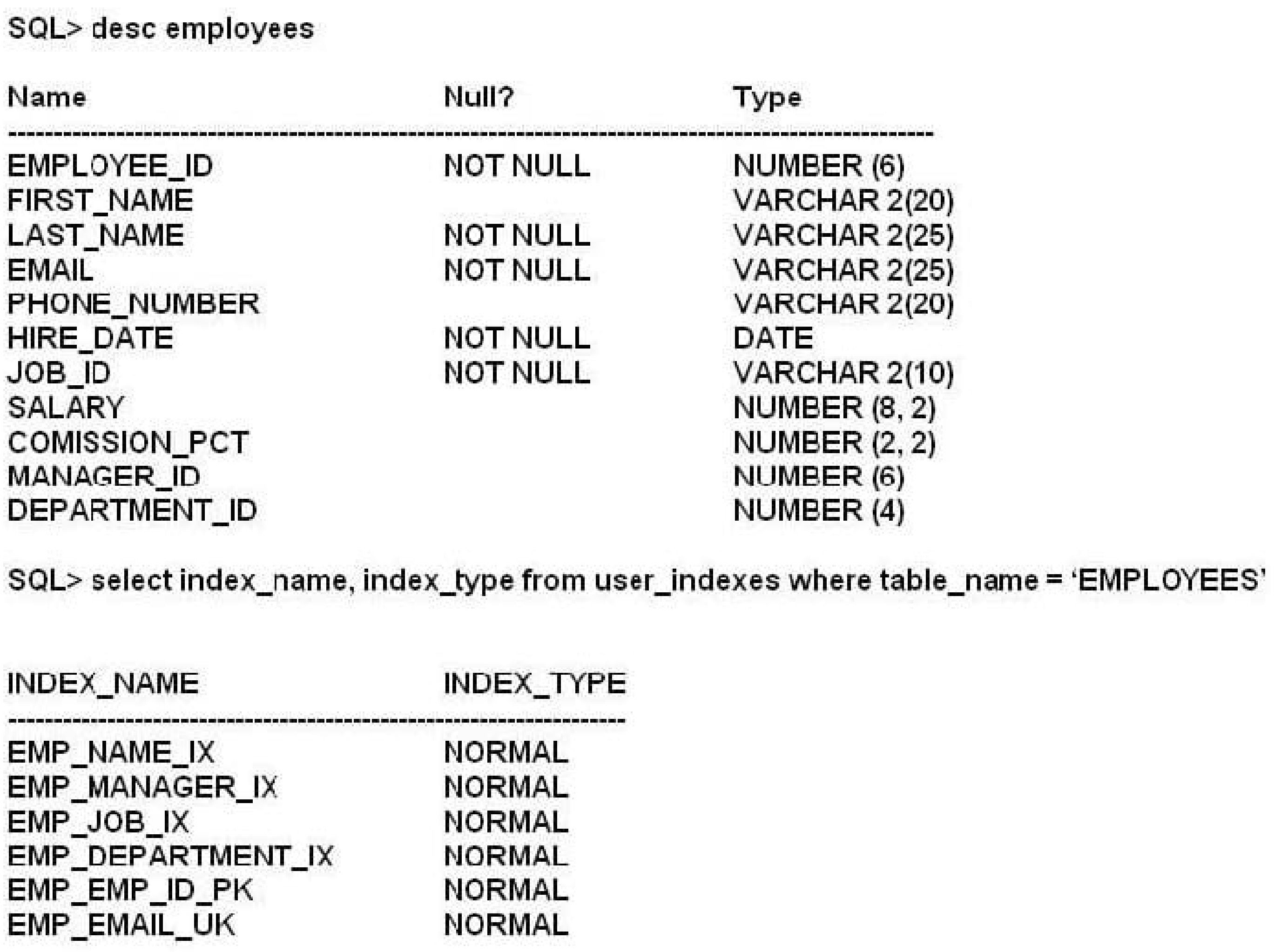
3.

SQL plan baseline evolution

Evaluate all unverified execution plans for a given statement in the plan history to become either accepted or rejected

QUESTION 6

Examine the Exhibit1 to view the structure of an indexes for the EMPLOYEES table.



```
SQL> desc employees

Name                    Null?          Type
---------------------------------------------------------------
EMPLOYEE_ID             NOT NULL       NUMBER (6)
FIRST_NAME                             VARCHAR 2(20)
LAST_NAME               NOT NULL       VARCHAR 2(25)
EMAIL                   NOT NULL       VARCHAR 2(25)
PHONE_NUMBER                           VARCHAR 2(20)
HIRE_DATE               NOT NULL       DATE
JOB_ID                  NOT NULL       VARCHAR 2(10)
SALARY                                 NUMBER (8, 2)
COMISSION_PCT                          NUMBER (2, 2)
MANAGER_ID                             NUMBER (6)
DEPARTMENT_ID                          NUMBER (4)

SQL> select index_name, index_type from user_indexes where table_name = 'EMPLOYEES'

INDEX_NAME              INDEX_TYPE
------------------------------------------
EMP_NAME_IX             NORMAL
EMP_MANAGER_IX          NORMAL
EMP_JOB_IX              NORMAL
EMP_DEPARTMENT_IX       NORMAL
EMP_EMP_ID_PK           NORMAL
EMP_EMAIL_UK            NORMAL
```

Examine the query:

SQL> SELECT * FROM employees WHERE employees_id IN (7876, 7900, 7902);

EMPLOYEE_ID is a primary key in the EMPLOYEES table that has 50000 rows.

Which statement is true regarding the execution of the query?

A. The query uses an index skip scan on the EMP_EMP_ID_PK index to fetch the rows.

B. The query uses the INLIST ITERATOR operator to iterate over the enumerated value list, and values are evaluated using an index range scan on the EMP_EMP_ID_PK index.

C. The query uses the INLIST ITERATOR operator to iterate over the enumerated value list, and values are evaluated using a fast full index scan on the EMP_EMP_ID_PK index.

D. The query uses the INLIST ITERATOR operator to iterate over the enumerated value list, and values are evaluated using an index unique scan on the EMP_EMP_ID_PK index.

E. The query uses a fast full index scan on the EMP_EMP_ID_PK index fetch the rows.

Correct Answer: B

How the CBO Evaluates IN-List Iterators

The IN-list iterator is used when a query contains an IN clause with values. The execution plan is identical to what would result for a statement with an equality clause instead of IN except for one additional step. That extra step occurs when the IN-list iterator feeds the equality clause with unique values from the IN-list.

Both of the statements in Example 2-1 and Example 2-1 are equivalent and produce the same plan.

Example 2-1 IN-List Iterators Initial Statement

SELECT header_id, line_id, revenue_amount FROM so_lines_all WHERE header_id IN (1011,1012,1013);

SELECT header_id, line_id, revenue_amount FROM so_lines_all WHERE header_id = 1011 OR header_id = 1012 OR header_id = 1013;

Plan

SELECT STATEMENT INLIST ITERATOR TABLE ACCESS BY INDEX ROWID SO_LINES_ALL INDEX RANGE SCAN SO_LINES_N1

Reference: Database Performance Tuning Guide and Reference

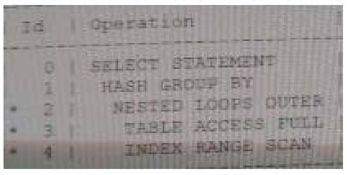
QUESTION 7

Examine the query and its execution plan: Which two statements are true regarding the execution plan?

```
SQL > SELECT cust_last_name, sum (nv12(o.customer_id, 0, 1)) “Count”
          FROM customers c, orders o
          WHERE c.credit_limit > 1000
          AND c.customer_id = o.customer_id(+)
          GROUP BY cust_last_name;
```

```
-----------------------------------------------------------------------------------------------------
Id      Operation                 Name                  Rows     Bytes    Cost    (% CPU)
-----------------------------------------------------------------------------------------------------
0       SELECT STATEMENT                                168      3192     6       (17)
1       HASH GROUP BY                                   168      3192     6       (17)
*2      NESTED LOOPS OUTER                              260      4940     5       (0)
*3      TABLE ACCESS FULL         CUSTOMERS             260      3900     5       (0)
*4      INDEX RANGE SCAN          ORD_CUSTOMERS_IX      105      420      0       (0)
-----------------------------------------------------------------------------------------------------
```

```
Predicate Information (Identified by operation id);
---------------------------------------------------------------------
3 – filter (“C”. “CREDIT_LIMIT”> 1000)
4 – access (“C”. “CUSTOMERS_ID”= “0”. “CUSTOMER_ID”(+))
   Filter (“O”. “CUSTOMER_ID”(+)>0)
```



```
 Id  | Operation
--------------------------
   0 | SELECT STATEMENT
   1 |  HASH GROUP BY
*  2 |   NESTED LOOPS OUTER
*  3 |    TABLE ACCESS FULL
*  4 |    INDEX RANGE SCAN
```

A. For every row of CUSTOMERS table, the row matching the join predicate from the ORDERS table are returned.

B. An outer join returns NULL for the ORDERS table columns along with the CUSTOMERS table rows when it does not find any corresponding rows in the ORDER table.

C. The data is aggregated from the ORDERS table before joining to CUSTOMERS.

D. The NESTED LOOP OUTER join is performed because the OPTIMZER_MODE parameter is set to ALL_ROWS.

Correct Answer: BD

B: An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

Note:

*

All_rows attempts to optimize the query to get the very last row as fast as possible. This makes sense in a stored procedure for example where the client does

not regain control until the stored procedure completes. You don\\'t care if you have to wait to get the first row if the last row gets back to you twice as fast. In a

client server/interactive application you may well care about that.

*

The optimizer uses nested loop joins to process an outer join in the following circumstances:

/ It is possible to drive from the outer table to inner table.

/ Data volume is low enough to make the nested loop method efficient.

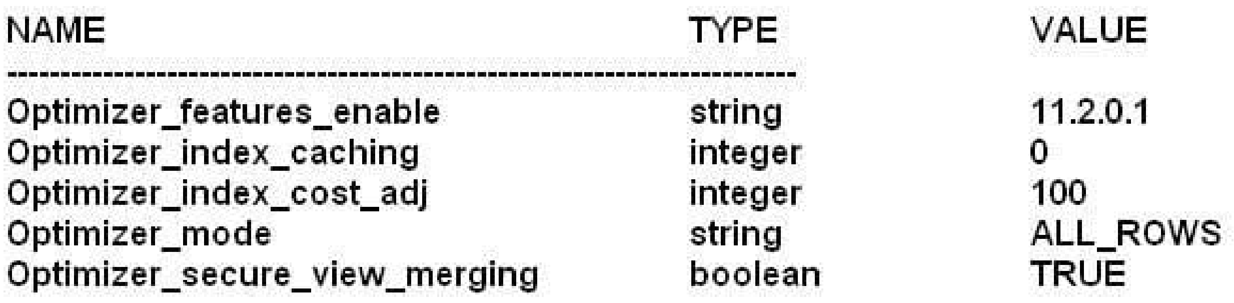
*

First_rows attempts to optimize the query to get the very first row back to the client as fast as possible. This is good for an interactive client server environment

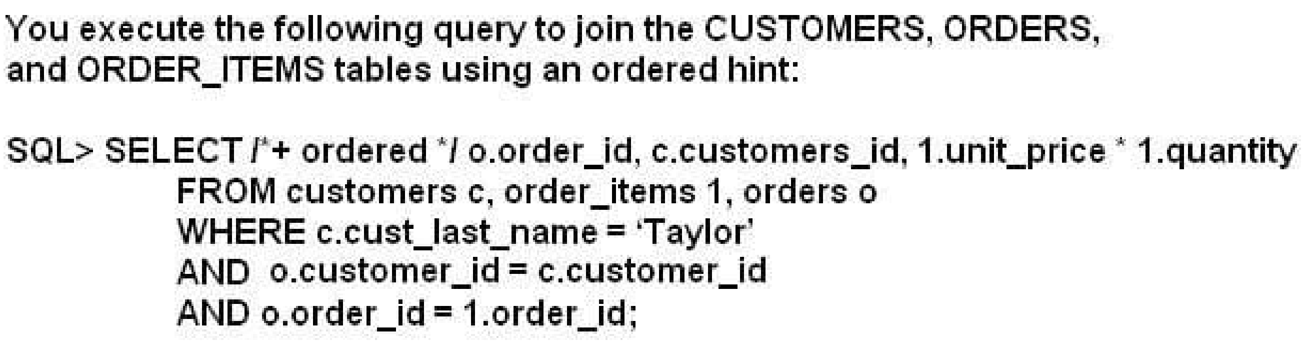
where the client runs a query and shows the user the first 10 rows or so and waits for them to page down to get more.

QUESTION 8

View the code sequence:



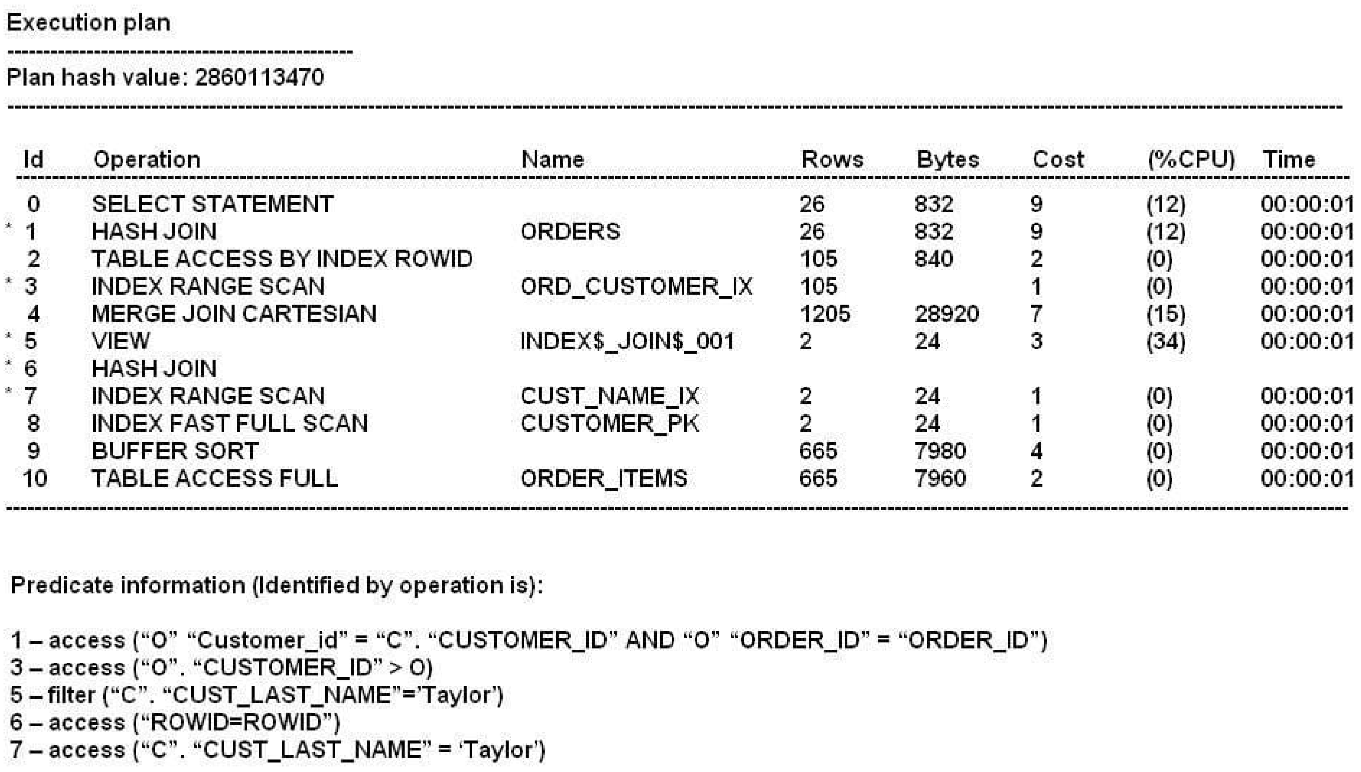
```
NAME                                    TYPE             VALUE
--------------------------------------------------------
Optimizer_features_enable               string           11.2.0.1
Optimizer_index_caching                 integer          0
Optimizer_index_cost_adj                integer          100
Optimizer_mode                          string           ALL_ROWS
Optimizer_secure_view_merging           boolean          TRUE
```



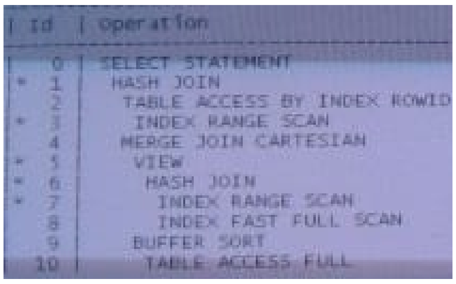
You execute the following query to join the CUSTOMERS, ORDERS, and ORDER_ITEMS tables using an ordered hint:

```
SQL> SELECT /*+ ordered */ o.order_id, c.customers_id, 1.unit_price * 1.quantity
          FROM customers c, order_items 1, orders o
          WHERE c.cust_last_name = 'Taylor'
          AND  o.customer_id = c.customer_id
          AND o.order_id = 1.order_id;
```

Examine the Exhibit to view the execution plan.



```
Execution plan
---------------------------------
Plan hash value: 2860113470
---------------------------------------------------------------------------------------------------------

 Id   Operation                      Name              Rows   Bytes   Cost   (%CPU)   Time
---------------------------------------------------------------------------------------------------------
  0   SELECT STATEMENT                                 26     832     9      (12)     00:00:01
* 1   HASH JOIN                      ORDERS            26     832     9      (12)     00:00:01
  2   TABLE ACCESS BY INDEX ROWID                      105    840     2      (0)      00:00:01
* 3   INDEX RANGE SCAN               ORD_CUSTOMER_IX   105            1      (0)      00:00:01
  4   MERGE JOIN CARTESIAN                             1205   28920   7      (15)     00:00:01
* 5   VIEW                           INDEX$_JOIN$_001  2      24      3      (34)     00:00:01
* 6   HASH JOIN
* 7   INDEX RANGE SCAN               CUST_NAME_IX      2      24      1      (0)      00:00:01
  8   INDEX FAST FULL SCAN           CUSTOMER_PK       2      24      1      (0)      00:00:01
  9   BUFFER SORT                                      665    7980    4      (0)      00:00:01
 10   TABLE ACCESS FULL              ORDER_ITEMS       665    7960    2      (0)      00:00:01
---------------------------------------------------------------------------------------------------------

Predicate information (Identified by operation is):

1 – access ("O" "Customer_id" = "C". "CUSTOMER_ID" AND "O" "ORDER_ID" = "ORDER_ID")
3 – access ("O". "CUSTOMER_ID" > O)
5 – filter ("C". "CUST_LAST_NAME"='Taylor')
6 – access ("ROWID=ROWID")
7 – access ("C". "CUST_LAST_NAME" = 'Taylor')
```



```
| Id  | Operation
-----------------------------------------
|   0 | SELECT STATEMENT
| * 1 |  HASH JOIN
|   2 |   TABLE ACCESS BY INDEX ROWID
| * 3 |    INDEX RANGE SCAN
|   4 |   MERGE JOIN CARTESIAN
| * 5 |    VIEW
| * 6 |     HASH JOIN
| * 7 |      INDEX RANGE SCAN
|   8 |      INDEX FAST FULL SCAN
|   9 |    BUFFER SORT
|  10 |     TABLE ACCESS FULL
```

Which two statements are true about the query execution?

A. The optimizer joins specified tables in the order as they appear in the FROM clause.

B. The CUSTOMERS and ORDERS tables are joined first and the resultant is then joined with rows returned by the ORDER_ITEMS table.

C. The CUSTOMERS and ORDER_ITEMS tables are joined first the resultant is then joined with rows by the ORDERS table.

D. The optimizer has chosen hash join as an access method as the OPTIMIZER_MODE parameter is set to FIRST_ROWS.

Correct Answer: CD

The first executed join is in line 6. The second executed join is in line 1.

Incorrect:

A: Line 7 and 8 are executed first.

QUESTION 9

You are working on a database that supports an OLTP workload. You see a large number of hard parses occurring and several almost identical SQL statements in the library cache that vary only in the literal values in the WHERE clause conditions.

Which two methods can you use to reduce hard parsing?

A. Replace literals with bind variables and evolve a baseline for the statement.

B. Use the RESULT_CACHE hint in the queries.

C. Create baselines for the almost identical SQL statement by manually loading them from the cursor cache.

D. Set the CURSOR_SHARING parameter to SIMILAR.

Correct Answer: AD

A: We can reduce this Hard parsing by using bindvariables

D: SIMILAR

Causes statements that may differ in some literals, but are otherwise identical, to share a cursor, unless the literals affect either the meaning of the statement or the degree to which the plan is optimized.

Note:

A hard parse is when your SQL must be re-loaded into the shared pool. A hard parse is worse than a soft parse because of the overhead involved in shared pool RAM allocation and memory management. Once loaded, the SQL must then be completely re-checked for syntax and semantics and an executable generated.

Excessive hard parsing can occur when your shared_pool_size is too small (and reentrant SQL is paged out), or when you have non-reusable SQL statements without host variables.

See the cursor_sharing parameter for a easy way to make SQL reentrant and remember that you should always use host variables in you SQL so that they can be reentrant.

Reference: Oracle Database Reference, CURSOR_SHARING

QUESTION 10

View the exhibit and examine the plans in the SQL baseline for a given statement. Which interpretation is correct?

A. A new plan cannot be evolved because SYS_SQL_bbedc41f554c408 is accepted.

B. Plan SYS_SQL_PLAN_bbdc741f554c408 will always be used by the optimizer for the query.

C. A new plan must be evolved using the DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE function before it can be used.

D. Plan SYS_SQL_bbedc741a57b5fc2 can be used by the optimizer if the cost of the query is less than plan SYS_SQL_PLAN_bbedc741f554c408.

E. Plan SYS_SQL_PLAN_bbedc741f554c408 will not be used until it is fixed by using the DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE function.

Correct Answer: C

Note:

*

Evolving a SQL plan baseline is the process by which the optimizer determines if non-accepted plans in the baseline should be accepted. As mentioned

previously, manually loaded plans are automatically marked as accepted, so manual loading forces the evolving process. When plans are loaded automatically,

the baselines are evolved using the

EVOLVE_SQL_PLAN_BASELINE function, which returns a CLOB reporting its results.

SET LONG 10000

SELECT DBMS_SPM.evolve_sql_plan_baseline(sql_handle =>

\\'SYS_SQL_7b76323ad90440b9\\')

FROM dual;

*

Manual plan loading can be used in conjunction with, or as an alternative to automatic plan capture. The load operations are performed using the DBMS_SPM

package, which allows SQL plan baselines to be loaded from SQL tuning sets or from specific SQL statements in the cursor cache. Manually loaded statements

are flagged as accepted by default. If a SQL plan baseline is present for a SQL statement, the plan is added to the baseline, otherwise a new baseline is created.

*

fixed (YES/NO) : If YES, the SQL plan baseline will not evolve over time. Fixed plans are used in preference to non-fixed plans.

QUESTION 11

Which are the two prerequisites for enabling star transformation on queries?

A. The STAR_TRANSFORMATION_ENABLED parameter should be set to TRUE or TEMP_DISABLE.

B. A B-tree index should be built on each of the foreign key columns of the fact table(s),

C. A bitmap index should be built on each of the primary key columns of the fact table(s).

D. A bitmap index should be built on each of the foreign key columns of the fact table(s).

E. A bitmap index must exist on all the columns that are used in the filter predicates of the query.

Correct Answer: AE

A: Enabling the transformation

E: Star transformation is essentially about adding subquery predicates corresponding to the constraint dimensions. These subquery predicates are referred to as bitmap semi-join predicates. The transformation is performed when there are indexes on the fact join columns (s.timeid, s.custid...). By driving bitmap AND and OR operations (bitmaps can be from bitmap indexes or generated from regular B-Tree indexes) of the key values supplied by the subqueries, only the relevant rows from the fact table need to be retrieved. If the filters on the dimension tables filter out a lot of data, this can be much more efficient than a full table scan on the fact table. After the relevant rows have been retrieved from the fact table, they may need to be joined back to the dimension tables, using the original predicates. In some cases, the join back can be eliminated.

Star transformation is controlled by the star_transformation_enabled parameter. The parameter takes 3 values.

TRUE - The Oracle optimizer performs transformation by identifying fact and constraint dimension tables automatically. This is done in a cost-based manner, i.e.

the transformation is performed only if the cost of the transformed plan is lower than the non-transformed plan. Also the optimizer will attempt temporary table

transformation automatically whenever materialization improves performance.

FALSE - The transformation is not tried.

TEMP_DISABLE - This value has similar behavior as TRUE except that temporary table transformation is not tried.

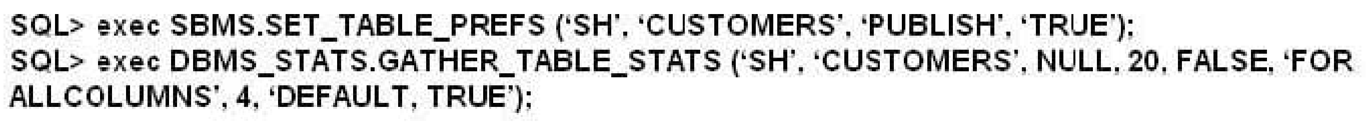
The default value of the parameter is FALSE. You have to change the parameter value and create indexes on the joining columns of the fact table to take

advantage of this transformation.

Reference: Optimizer Transformations: Star Transformation

QUESTION 12

You recently gathered statistics for a table by using the following commands:



```
SQL> exec SBMS.SET_TABLE_PREFS ('SH', 'CUSTOMERS', 'PUBLISH', 'TRUE');
SQL> exec DBMS_STATS.GATHER_TABLE_STATS ('SH', 'CUSTOMERS', NULL, 20, FALSE, 'FOR
ALLCOLUMNS', 4, 'DEFAULT, TRUE');
```

You noticed that the performance of queries has degraded after gathering statistics. You want to use the old statistics. The optimizer statistics retention period is

default.

What must you do to use the old statistics?

A. Use the flashback to bring back the statistics to the desired time.

B. Restore statistics from statistics history up to the desired time.

C. Delete all the statistics collected after the desired time.

D. Set OPTIMIZER_USE_PENDING_STATISTICS to TRUE.

Correct Answer: B

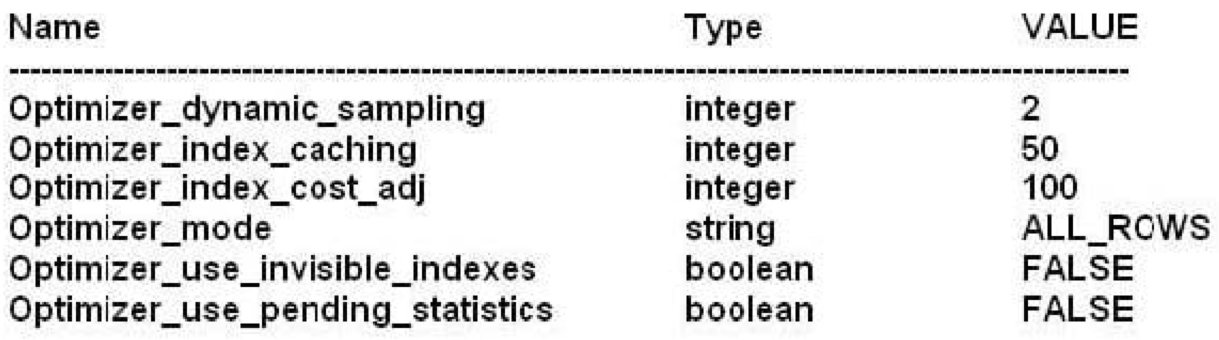
Whenever statistics in dictionary are modified, old versions of statistics are saved automatically for future restoration. Statistics can be restored using RESTORE procedures of DBMS_STATS package. These procedures use a time stamp as an argument and restore statistics as of that time stamp. This is useful in case newly collected statistics leads to some sub-optimal execution plans and the administrator wants to revert to the previous set of statistics.

Reference: Oracle Database Performance Tuning Guide, Restoring Previous Versions of Statistics

---

QUESTION 13

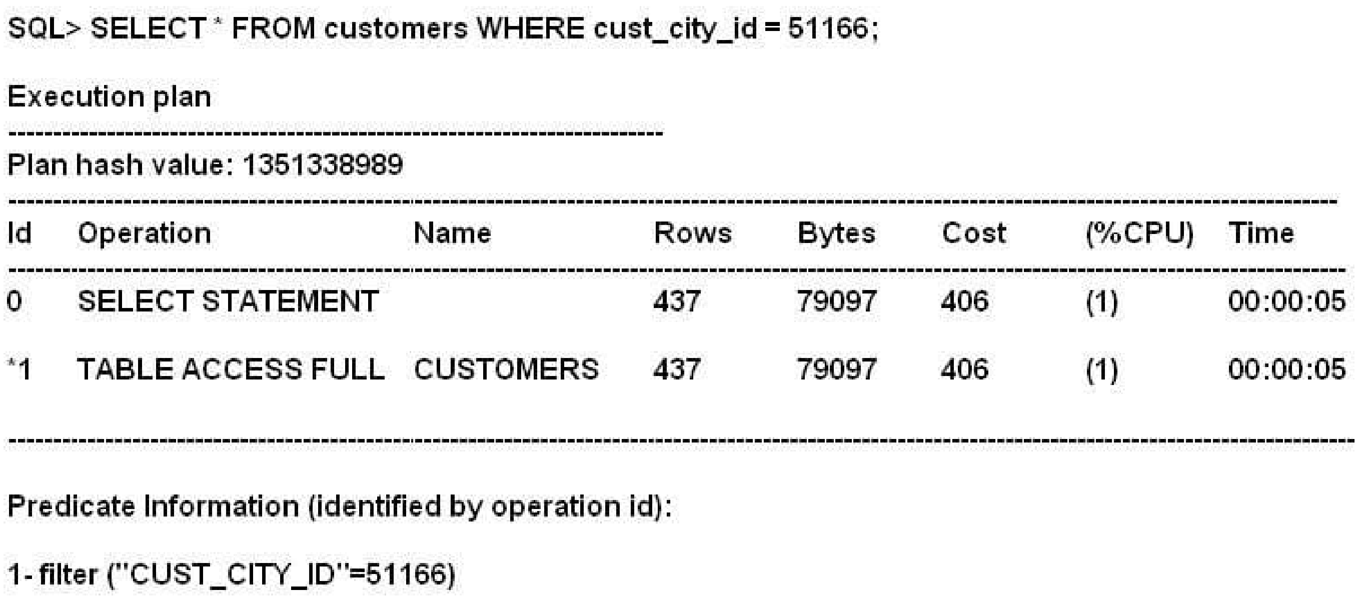
Examine the initializing parameters:



```
Name                                 Type            VALUE
------------------------------------------------------------------------
Optimizer_dynamic_sampling           integer         2
Optimizer_index_caching              integer         50
Optimizer_index_cost_adj             integer         100
Optimizer_mode                       string          ALL_ROWS
Optimizer_use_invisible_indexes      boolean         FALSE
Optimizer_use_pending_statistics     boolean         FALSE
```

An index exists on the column used in the WHERE of a query. You execute the query for the first time today and notice that the query is not using the index. The CUSTOMERS table has 55000 rows.

View the exhibit and examine the query and its execution plan.



```
SQL> SELECT * FROM customers WHERE cust_city_id = 51166;

Execution plan
------------------------------------------
Plan hash value: 1351338989
-----------------------------------------------------------------------------------------
Id   Operation            Name         Rows    Bytes    Cost    (%CPU)   Time
-----------------------------------------------------------------------------------------
0    SELECT STATEMENT                  437     79097    406     (1)      00:00:05

*1   TABLE ACCESS FULL    CUSTOMERS    437     79097    406     (1)      00:00:05

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

Predicate Information (identified by operation id):

1 - filter ("CUST_CITY_ID"=51166)
```

What can be the two reasons for full table scan?

A. The value of the OPTIMIZER_INDEX_COST_ADJ parameter is set to a low value.

B. The blocks fetched by the query are greater than the value specified by the DB_FILE_MULTIBLOCK_READ_COUNT parameter.

C. The statistics for the CUSTOMERS table and the indexes stale.

D. The OPTIMIZER_MODE parameter is set to ALL_ROWS.

E. Histogram statistics for CUST_CITY_ID are missing.

F. Average number of rows per block for the CUSTOMERS table is low.

Correct Answer: CE

C: Old statistics could cause this problem. "Histograms are feature in CBO and it helps to optimizer to determine how data are skewed(distributed) with in the column. Histogram is good to create for the column which are included in the WHERE clause where the column is highly skewed. Histogram helps to

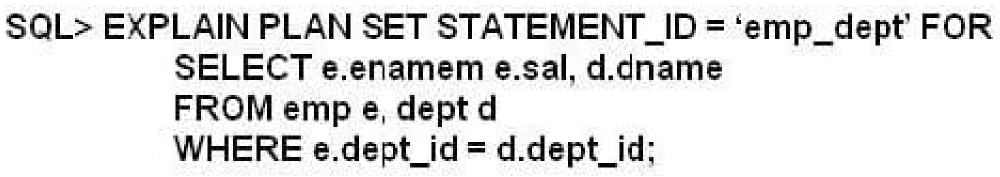
optimizer to decide whether to use an index or full-table scan or help the optimizer

"

determine the fastest table join order.

QUESTION 14

You executed the following statement:



```
SQL> EXPLAIN PLAN SET STATEMENT_ID = 'emp_dept' FOR
          SELECT e.enamem e.sal, d.dname
          FROM emp e, dept d
          WHERE e.dept_id = d.dept_id;
```

Which three statements are true about EXPLAIN PLAN?

A. The execution plan is saved in PLAN_TABLE without executing the query.

B. The execution plan for the query is generated and displayed immediately as the output.

C. The execution plan generated may not necessarily be the execution plan used during query execution.

D. The execution plan is saved in DBA_HIST_SQL_PLAN without executing the query.

E. The execution plan generated can be viewed using the DBMS_XPLAIN.DISPLAY function.

F. The execution plan generated can be fetched from the library cache by using the DBMS_XPLAIN.DISPLAY function.

Correct Answer: ACE

*

(A, not D): The explain plan process stores data in the PLAN_TABLE.

*

EXPLAIN PLAN

The EXPLAIN PLAN method doesn\\'t require the query to be run (A), greatly reducing the time it takes to get an execution plan for long-running queries compared to AUTOTRACE.

E: Use the DBMS_XPLAN.DISPLAY function to display the execution plan.

* The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several, predefined formats. You can also use the DBMS_XPLAN package to display the plan of a statement stored in the Automatic Workload Repository (AWR) or stored in a SQL tuning set. It further provides a way to display the SQL execution plan and SQL execution runtime statistics for cached SQL cursors based on the information stored in the V$SQL_PLAN and V$SQL_PLAN_STATISTICS_ALL fixed views.

Note:

*

First the query must be explained.

SQL> EXPLAIN PLAN FOR

2 SELECT *

3 FROM emp e, dept d

4 WHERE e.deptno = d.deptno

5 AND e.ename = \\'SMITH\\';

Explained.

SQL>

Then the execution plan displayed. (not B)

SQL> @$ORACLE_HOME/rdbms/admin/utlxpls.sql

Plan Table

-- | Operation | Name | Rows | Bytes| Cost | Pstart| Pstop |

-- | SELECT STATEMENT | | | | | | |

| NESTED LOOPS | | | | | | |

| TABLE ACCESS FULL |EMP | | | | | |

| TABLE ACCESS BY INDEX RO|DEPT | | | | | |

| INDEX UNIQUE SCAN |PK_DEPT | | | | | |

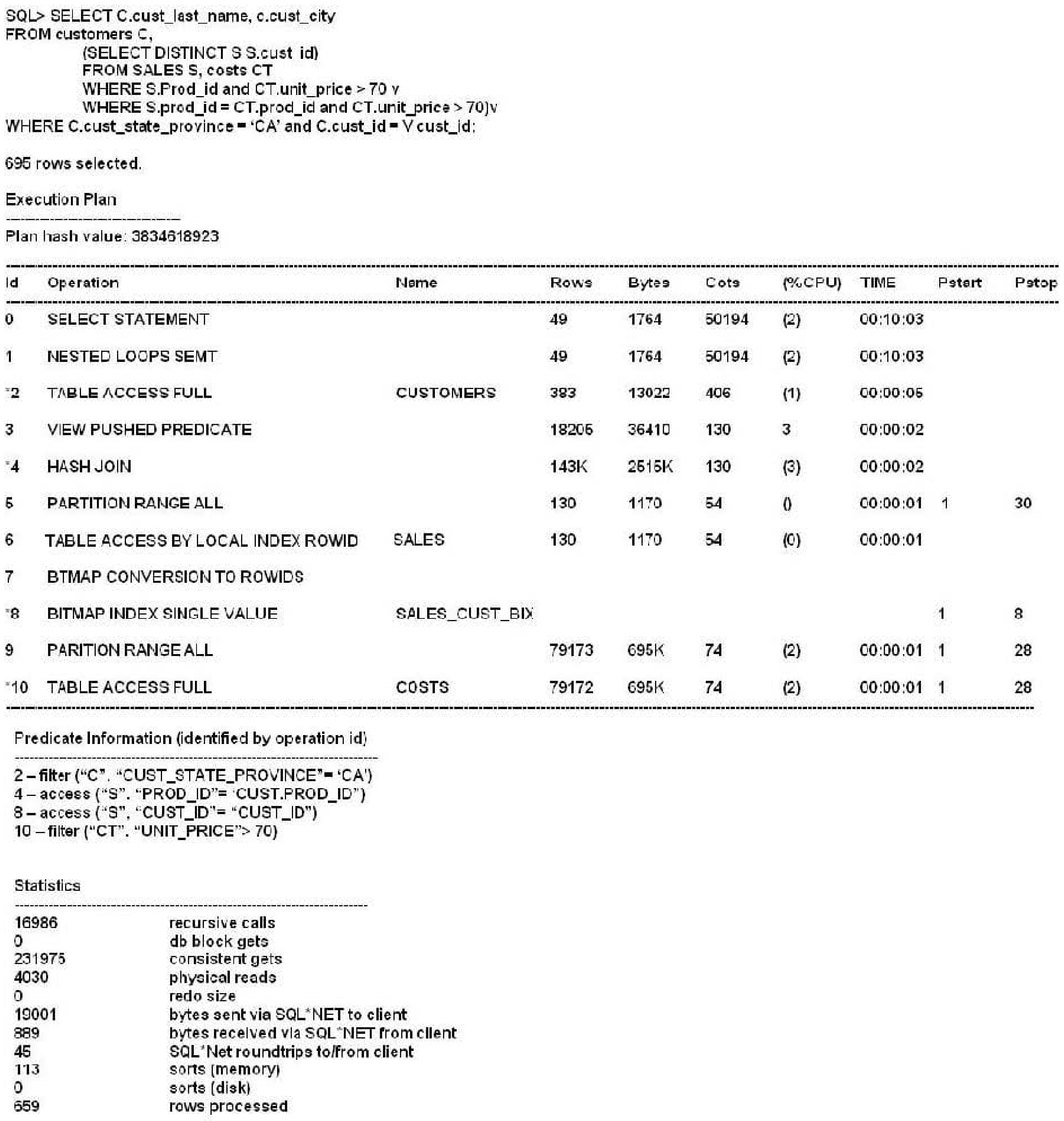
8 rows selected.

SQL>

For parallel queries use the "utlxplp.sql" script instead of "utlxpls.sql".

QUESTION 15

Examine the Following Query and execution plan:



```
SQL> SELECT C.cust_last_name, c.cust_city
FROM customers C,
            (SELECT DISTINCT S S.cust id)
            FROM SALES S, costs CT
            WHERE S.Prod_id and CT.unit_price > 70 v
            WHERE S.prod_id = CT.prod_id and CT.unit_price > 70)v
WHERE C.cust_state_province = 'CA' and C.cust_id = V cust_id;

695 rows selected.

Execution Plan
------------------------------
Plan hash value: 3834618923

------------------------------------------------------------------------------------------------------------------
Id   Operation                          Name              Rows    Bytes   Cots    (%CPU)  TIME       Pstart  Pstop
------------------------------------------------------------------------------------------------------------------
0    SELECT STATEMENT                                     49      1764    50194   (2)     00:10:03

1    NESTED LOOPS SEMT                                    49      1764    50194   (2)     00:10:03

*2   TABLE ACCESS FULL                  CUSTOMERS         383     13022   406     (1)     00:00:06

3    VIEW PUSHED PREDICATE                                18206   36410   130     3       00:00:02

*4   HASH JOIN                                            143K    2515K   130     (3)     00:00:02

5    PARTITION RANGE ALL                                  130     1170    54      ()      00:00:01   1       30

6    TABLE ACCESS BY LOCAL INDEX ROWID  SALES             130     1170    54      (0)     00:00:01

7    BTMAP CONVERSION TO ROWIDS

*8   BITMAP INDEX SINGLE VALUE          SALES_CUST_BIX                                               1       8

9    PARITION RANGE ALL                                   79173   695K    74      (2)     00:00:01   1       28

*10  TABLE ACCESS FULL                  COSTS             79172   695K    74      (2)     00:00:01   1       28
------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id)
-------------------------------------------------------------------
2 – filter ("C". "CUST_STATE_PROVINCE"= 'CA')
4 – access ("S". "PROD_ID"= 'CUST.PROD_ID")
8 – access ("S". "CUST_ID"= "CUST_ID")
10 – filter ("CT". "UNIT_PRICE"> 70)


Statistics
-------------------------------------------------------------------
16986         recursive calls
0             db block gets
231975        consistent gets
4030          physical reads
0             redo size
19001         bytes sent via SQL*NET to client
889           bytes received via SQL*NET from client
45            SQL*Net roundtrips to/from client
113           sorts (memory)
0             sorts (disk)
659           rows processed
```

Which query transformation technique is used by the optimizer?

A. Filter push down

B. Subquery factoring

C. Subquery unnesting

D. Predicate pushing

Correct Answer: D

Note:

* In the execution plan BX, note the keyword \\'VIEW PUSHED PREDICATE\\' indicates that the view has undergone the join predicate pushdown transformation.