Northumbria University NEWCASTLE

Department of Computer & Information Sciences

ASSESSMENT BRIEF

Module Title:	Advanced Databases
Module Code:	KL7011
Academic Year / Semester:	2021-22 / Semester 1
Module Tutor / Email (all queries):	Akhtar Ali akhtar.ali@northumbria.ac.uk
% Weighting (to overall module):	40%
Assessment Title:	Assignment 2: teamwork
Student IDs/Oracle Username (DWUs and DMU)	DMU6@cisbg DMU36@cisbg
Names of students in the Group	VENKATA SAI SUDHA VARSHINI ADUSUMALLI VENKATA SREEHARSHA ASAM
Group No	36

Assignment Questions

Part 1: Data Warehousing Tasks (50 Marks)

This part is based on the **Sales History** scenario as described in Appendix 1.

You must submit all the SQL queries and any other code that you wrote in answering any of the tasks / questions (e.g., the use of Explain Plan statements for the queries and their outputs using Spooling or other suitable means).

(A) Study the index definitions in sh_idx.sql. Discuss in detail (using cost-based analysis) why these indexes (choose *two* different ones) are useful for answering queries over the SH2 and DWU versions of the database. The queries should be complex and need to involve at least *two* different tables. *You should not run the sh_idx.sql script at all.*

(10 marks)

Answer Part 1 (A)

Provide the details of the 2 indexes you are going to compare their performance impact on SH2 (i.e., name the indexes and on which tables those indexes were created in SH2, these indexes must not exist in your DWU version) (1 Mark):

comparing 'products_prod_status_bix' created on Products table and 'customers_gender_bix' created on Customers table to compare their performance impact on SH2 and DWU.

Index 1:

```
CREATE BITMAP INDEX products_prod_status_bix
      ON products(prod_status)
        NOLOGGING COMPUTE STATISTICS;
```

Index 2:

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

Provide the 2 SQL queries you are going to run to compare the performance impact of the above 2 Indexes on SH2 and the version of the same queries on DWU (4 marks):

SH2 VERSION

INDEX 1:

```
SELECT p.prod_id, p.prod_status,
  ch.channel_desc, sum(s.quantity_sold) sum_sales
FROM sh2.sales s, sh2.products p, sh2.channels ch
WHERE p.prod_id= s.prod_id
AND ch.channel_id = s.channel_id
and p.prod_status = 'available, on stock'
and ch.channel_desc = 'Internet'
GROUP BY p.prod_status, p.prod_id, ch.channel_desc
order by p.prod_id;
```

INDEX 2:

```
SELECT   p.prod_id, p.prod_name, c.cust_gender,
              SUM(s.amount_sold) AS TotalSales
     FROM     sh2.sales s, sh2.products p, sh2.customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender ;
```

DWU VERSION

INDEX 1:

```
SELECT p.prod_id, p.prod_status,
  ch.channel_desc, sum(s.quantity_sold) sum_sales
FROM sales s, products p, channels ch
WHERE p.prod_id= s.prod_id
AND ch.channel_id = s.channel_id
and p.prod_status = 'available, on stock'
and ch.channel_desc = 'Internet'
GROUP BY p.prod_status, p.prod_id, ch.channel_desc
order by p.prod_id;
```

INDEX 2:

```
SELECT   p.prod_id, p.prod_name, c.cust_gender,
              SUM(s.amount_sold) AS TotalSales
     FROM     sales s, products p, customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender ;
```

Provide Explain Plan statements & outputs for the above 2 SQL queries you have run to compare the performance impact of those 2 Indexes on SH2 and their version of the same queries on DWU (2 marks):

SH2 VERSION

INDEX 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on
```

```
EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_status,
  ch.channel_desc, sum(s.quantity_sold) sum_sales
FROM sh2.sales s, sh2.products p, sh2.channels ch
WHERE p.prod_id= s.prod_id
AND ch.channel_id = s.channel_id
and p.prod_status = 'available, on stock'
and ch.channel_desc = 'Internet'
GROUP BY p.prod_status, p.prod_id, ch.channel_desc
order by p.prod_id;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 534141547

-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                           |  4255 |   191K|  2131   (1)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY                 |                           |  4255 |   191K|  2131   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                    |                           |   203K|  9130K|  2126   (1)| 00:00:01 |       |       |
|*  3 |    VIEW                        | index$_join$_002          |  8000 |   187K|    34   (0)| 00:00:01 |       |       |
|*  4 |     HASH JOIN                  |                           |       |       |            |          |       |       |
|   5 |      BITMAP CONVERSION TO ROWIDS|                          |  8000 |   187K|     1   (0)| 00:00:01 |       |       |
|*  6 |       BITMAP INDEX SINGLE VALUE | PRODUCTS_PROD_STATUS_BIX |       |       |            |          |       |       |
|   7 |      INDEX FAST FULL SCAN      | PRODUCTS_PK               |  8000 |   187K|    41   (0)| 00:00:01 |       |       |
|*  8 |    HASH JOIN                   |                           |   203K|  4366K|  2091   (1)| 00:00:01 |       |       |
|*  9 |     TABLE ACCESS FULL          | CHANNELS                  |     1 |    12 |     3   (0)| 00:00:01 |       |       |
|  10 |     PARTITION RANGE ALL        |                           |  1016K|  9924K|  2086   (1)| 00:00:01 |     1 |    17 |
|  11 |      TABLE ACCESS FULL         | SALES                     |  1016K|  9924K|  2086   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------------------------------------

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

   2 - access("P"."PROD_ID"="S"."PROD_ID")
   3 - filter("P"."PROD_STATUS"='available, on stock')
   4 - access(ROWID=ROWID)
   6 - access("P"."PROD_STATUS"='available, on stock')
   8 - access("CH"."CHANNEL_ID"="S"."CHANNEL_ID")
   9 - filter("CH"."CHANNEL_DESC"='Internet')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 3 Sql Plan Directives used for this statement

33 rows selected.

Elapsed: 00:00:00.04
```

INDEX 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT   p.prod_id, p.prod_name, c.cust_gender,
               SUM(s.amount_sold) AS TotalSales
     FROM      sh2.sales s, sh2.products p, sh2.customers c
     WHERE     p.prod_id = s.prod_id
       AND     c.cust_id = s.cust_id
       AND     p.prod_category = 'Men'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender ;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 2473104768

--------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
--------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |  2477 |   142K|  2467   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY                 |                     |  2477 |   142K|  2467   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                    |                     |   229K|    12M|  2461   (1)| 00:00:01 |       |       |
|   3 |    VIEW                        | index$_join$_003    | 50000 |   341K|   270   (1)| 00:00:01 |       |       |
|*  4 |     HASH JOIN                  |                     |       |       |            |          |       |       |
|   5 |      BITMAP CONVERSION TO ROWIDS|                    | 50000 |   341K|     4   (0)| 00:00:01 |       |       |
|   6 |       BITMAP INDEX FULL SCAN   | CUSTOMERS_GENDER_BIX|       |       |            |          |       |       |
|   7 |      INDEX FAST FULL SCAN      | CUSTOMERS_PK        | 50000 |   341K|   180   (0)| 00:00:01 |       |       |
|*  8 |    HASH JOIN                   |                     |   229K|    11M|  2190   (1)| 00:00:01 |       |       |
|*  9 |     TABLE ACCESS FULL          | PRODUCTS            |  2594 | 98572 |   102   (0)| 00:00:01 |       |       |
|  10 |     PARTITION RANGE ALL        |                     |  1016K|    13M|  2086   (1)| 00:00:01 |     1 |    17 |
|  11 |      TABLE ACCESS FULL         | SALES               |  1016K|    13M|  2086   (1)| 00:00:01 |     1 |    17 |
--------------------------------------------------------------------------------------------------------------------------------

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

   2 - access("C"."CUST_ID"="S"."CUST_ID")
   4 - access(ROWID=ROWID)
   8 - access("P"."PROD_ID"="S"."PROD_ID")
   9 - filter("P"."PROD_CATEGORY"='Men')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 4 Sql Plan Directives used for this statement

32 rows selected.

Elapsed: 00:00:00.10
```

DWU VERSION

INDEX 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_status,
  ch.channel_desc, sum(s.quantity_sold) sum_sales
FROM sales s, products p, channels ch
WHERE p.prod_id= s.prod_id
AND ch.channel_id = s.channel_id
and p.prod_status = 'available, on stock'
and ch.channel_desc = 'Internet'
GROUP BY p.prod_status, p.prod_id, ch.channel_desc
order by p.prod_id;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3152389671

-------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |  4255 |   191K|  3393   (1)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY         |           |  4255 |   191K|  3393   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN            |           |   203K|  9130K|  3388   (1)| 00:00:01 |       |       |
|   3 |    MERGE JOIN CARTESIAN|           |  8000 |   281K|   105   (0)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL  | CHANNELS  |     1 |    12 |     3   (0)| 00:00:01 |       |       |
|   5 |     BUFFER SORT        |           |  8000 |   187K|   102   (0)| 00:00:01 |       |       |
|*  6 |      TABLE ACCESS FULL | PRODUCTS  |  8000 |   187K|   102   (0)| 00:00:01 |       |       |
|   7 |    PARTITION RANGE ALL |           |  1016K|  9924K|  3280   (1)| 00:00:01 |     1 |    17 |
|   8 |     TABLE ACCESS FULL  | SALES     |  1016K|  9924K|  3280   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("P"."PROD_ID"="S"."PROD_ID" AND "CH"."CHANNEL_ID"="S"."CHANNEL_ID")
   4 - filter("CH"."CHANNEL_DESC"='Internet')
   6 - filter("P"."PROD_STATUS"='available, on stock')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement

27 rows selected.
```

INDEX 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT   p.prod_id, p.prod_name, c.cust_gender,
               SUM(s.amount_sold) AS TotalSales
     FROM      sales s, products p, customers c
     WHERE     p.prod_id = s.prod_id
       AND     c.cust_id = s.cust_id
       AND     p.prod_category = 'Men'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender ;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------
Plan hash value: 644657833

------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |  3669 |   207K|  3684   (2)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY           |           |  3669 |   207K|  3684   (2)| 00:00:01 |       |       |
|*  2 |   HASH JOIN              |           |  3669 |   207K|  3683   (2)| 00:00:01 |       |       |
|   3 |    JOIN FILTER CREATE    | :BF0000   |  2594 | 98572 |   102   (0)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL    | PRODUCTS  |  2594 | 98572 |   102   (0)| 00:00:01 |       |       |
|   5 |    VIEW                  | VW_GBC_9  |  7103 |   138K|  3580   (2)| 00:00:01 |       |       |
|   6 |     HASH GROUP BY        |           |  7103 |   145K|  3580   (2)| 00:00:01 |       |       |
|   7 |      JOIN FILTER USE     | :BF0000   |  1016K|    20M|  3556   (1)| 00:00:01 |       |       |
|*  8 |       HASH JOIN          |           |  1016K|    20M|  3556   (1)| 00:00:01 |       |       |
|   9 |        TABLE ACCESS FULL | CUSTOMERS | 50000 |   341K|   272   (0)| 00:00:01 |       |       |
|  10 |        PARTITION RANGE ALL|          |  1016K|    13M|  3280   (1)| 00:00:01 |     1 |    17 |
|  11 |         TABLE ACCESS FULL | SALES     |  1016K|    13M|  3280   (1)| 00:00:01 |     1 |    17 |
------------------------------------------------------------------------------------------------------------------

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

   2 - access("P"."PROD_ID"="ITEM_1")
   4 - filter("P"."PROD_CATEGORY"='Men')
   8 - access("C"."CUST_ID"="S"."CUST_ID")

25 rows selected.

Elapsed: 00:00:00.06
```

Provide discussion of the cost-based comparison of the above 2 sets of queries and their explain plan cost figures (3 marks):

Based on the above outputs, The Indexes 'products_prod_status_bix' and 'customers_gender_bix' are used by SH2 but DWU version didn't use them because those Indexes are present in DWU.

it is clear that costs of SH2 is less when compared with DWU for both query. And also, SH2 version used less space than DWU version because it uses Indexes.

(A) Identify *two* new indexes and justify why they could be useful. Write the SQL code for creating these indexes under your DWU account. Give example queries with cost-based analysis for both DWU account (which will have the new indexes) and SH2 shared schema (which will NOT have any of your new indexes). Alternatively, you may choose to run the same queries on your DWU account before and after creating your proposed two indexes. The queries should be complex and need to involve at least *two* different tables.

(10 marks)

Answer Part 1 (B)

Provide the SQL Code and output for the 2 new indexes you have created on your DWU database for comparing their performance impact on DWU (i.e., these indexes must not exist in SH2) (2 Marks):

creating country_name_bix created on countries table and customers_birth_bix created on Customers table on DWU, to compare their performance impact on SH2 and DWU.

Index 1:

```
CREATE BITMAP INDEX countries_name_bix
      ON countries (country_name)
        NOLOGGING COMPUTE STATISTICS;
```

Index 2:

```
CREATE BITMAP INDEX customers_birth_bix
      ON customers(cust_year_of_birth)
        NOLOGGING COMPUTE STATISTICS;
```

Provide 2 SQL queries you are going to run to compare the performance impact of your own 2 new indexes on DWU and the version of the same queries on SH2 (4 marks):

SH2 VERSION

INDEX 1:

```
SELECT co.country_id, co.country_name, c.cust_gender,
```

```
  sum(s.amount_sold) sum_sales
FROM sh2.sales s, sh2.customers c, sh2.countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;
```

INDEX 2:

```
select s.prod_id, c.* from sh2.sales s, sh2.customers c
     where s.cust_id = c.customer_id and
           cust_year_of_birth between 1910 and 1920;
```

DWU VERSION

INDEX 1:

```
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM sales s, customers c, countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;
```

INDEX 2:

```
select s.prod_id, c.* from sales s, customers c
     where s.cust_id = c.customer_id and
           cust_year_of_birth between 1910 and 1920;
```

Provide Explain Plan statements & outputs for the above 2 SQL queries you have run to compare the performance impact of your 2 indexes on DWU and their version of the same queries on SH2 (2 marks):

SH2 VERSION

INDEX 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on
```

```
EXPLAIN PLAN FOR
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM sh2.sales s, sh2.customers c, sh2.countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------
Plan hash value: 981620288

-------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |    39 |  2368   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY       |           |     1 |    39 |  2368   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN          |           |   155K|  5916K|  2364   (1)| 00:00:01 |       |       |
|*  3 |    HASH JOIN         |           |  7643 |   223K|   276   (1)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL| COUNTRIES |     1 |    14 |     3   (0)| 00:00:01 |       |       |
|   5 |     TABLE ACCESS FULL| CUSTOMERS | 50000 |   781K|   273   (1)| 00:00:01 |       |       |
|   6 |    PARTITION RANGE ALL|          |  1016K|  8932K|  2086   (1)| 00:00:01 |     1 |    17 |
|   7 |     TABLE ACCESS FULL| SALES     |  1016K|  8932K|  2086   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   4 - filter("CO"."COUNTRY_NAME"='United Kingdom')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 2 Sql Plan Directives used for this statement

27 rows selected.
```

INDEX 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
select s.prod_id, c.* from sh2.sales s, sh2.customers c
     where s.cust_id = c.cust_id and
           cust_year_of_birth between 1910 and 1920;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3216277529

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name              | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                   |  2710 |   407K|  2124   (1)| 00:00:01 |       |       |
|*  1 |  HASH JOIN                           |                   |  2710 |   407K|  2124   (1)| 00:00:01 |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMERS         |   157 | 22608 |    36   (0)| 00:00:01 |       |       |
|   3 |    BITMAP CONVERSION TO ROWIDS       |                   |       |       |            |          |       |       |
|*  4 |     BITMAP INDEX RANGE SCAN          | CUSTOMERS_YOB_BIX |       |       |            |          |       |       |
|   5 |   PARTITION RANGE ALL                |                   |  1016K|  9924K|  2086   (1)| 00:00:01 |     1 |    17 |
|   6 |    TABLE ACCESS FULL                 | SALES             |  1016K|  9924K|  2086   (1)| 00:00:01 |     1 |    17 |
-----------------------------------------------------------------------------------------------------------------------

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

   1 - access("S"."CUST_ID"="C"."CUST_ID")
   4 - access("CUST_YEAR_OF_BIRTH">=1910 AND "CUST_YEAR_OF_BIRTH"<=1920)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 1 Sql Plan Directive used for this statement

25 rows selected.
```

DWU VERSION

INDEX 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM sales s, customers c, countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 575917075

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name              | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                   |     2 |    78 |  3558   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY                       |                   |     2 |    78 |  3558   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                          |                   | 53488 |  2037K|  3556   (1)| 00:00:01 |       |       |
|*  3 |    HASH JOIN                         |                   |  2632 | 78960 |   275   (1)| 00:00:01 |       |       |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| COUNTRIES       |     1 |    14 |     2   (0)| 00:00:01 |       |       |
|   5 |      BITMAP CONVERSION TO ROWIDS     |                   |       |       |            |          |       |       |
|*  6 |       BITMAP INDEX SINGLE VALUE      | COUNTRIES_NAME_BIX|       |       |            |          |       |       |
|   7 |     TABLE ACCESS FULL                | CUSTOMERS         | 50000 |   781K|   273   (1)| 00:00:01 |       |       |
|   8 |    PARTITION RANGE ALL               |                   |  1016K|  8932K|  3279   (1)| 00:00:01 |     1 |    17 |
|   9 |     TABLE ACCESS FULL                | SALES             |  1016K|  8932K|  3279   (1)| 00:00:01 |     1 |    17 |
-----------------------------------------------------------------------------------------------------------------------

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

   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   6 - access("CO"."COUNTRY_NAME"='United Kingdom')

23 rows selected.

DWU36 >
```

INDEX 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
select s.prod_id, c.* from sales s, customers c
     where s.cust_id = c.cust_id and
           cust_year_of_birth between 1910 and 1920;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 2586604821

------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                     |  1016K|   143M|  3547   (1)| 00:00:01 |       |       |
|*  1 |  HASH JOIN                          |                     |  1016K|   143M|  3547   (1)| 00:00:01 |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| CUSTOMERS           |  2730 |   367K|   266   (0)| 00:00:01 |       |       |
|   3 |    BITMAP CONVERSION TO ROWIDS      |                     |       |       |            |          |       |       |
|*  4 |     BITMAP INDEX RANGE SCAN         | CUSTOMERS_BIRTH_BIX |       |       |            |          |       |       |
|   5 |   PARTITION RANGE ALL               |                     |  1016K|  9924K|  3279   (1)| 00:00:01 |     1 |    17 |
|   6 |    TABLE ACCESS FULL                | SALES               |  1016K|  9924K|  3279   (1)| 00:00:01 |     1 |    17 |
------------------------------------------------------------------------------------------------------------------------------

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

   1 - access("S"."CUST_ID"="C"."CUST_ID")
   4 - access("CUST_YEAR_OF_BIRTH">=1910 AND "CUST_YEAR_OF_BIRTH"<=1920)

19 rows selected.
```

Provide discussion of the cost-based comparison of the above 2 sets of queries and their explain plan cost figures (2 marks):

Based on the above outputs, The Indexes countries_name_bix and customers_birth_bix are used by DWU but SH2 version didn't use them because those Indexes are present in DWU.

it is clear that costs of DWU is less when compared with SH2 for both query. And also, DWU version used less space than SH2 version because it uses Indexes.

(A) Given the two materialized views (MVs) defined in sh_cremv.sql and already created under SH2 shared schema, discuss in detail why these MVs are useful for users of the SH database. You should provide detailed examples of cost based analysis, e.g., using Explain Plan for running sample queries on both SH2 and DWU to illustrate your answer. The queries should return subsets of the values contained in these MVs and you must not explicitly name these MVs in the FROM clause. *You should not run the sh_cremv.sql script at all.*

(8 marks)

Answer Part 1 (C)

Provide 2 SQL queries you are going to run to compare the performance impact of the 2 MVs in SH2 (i.e., cal_month_sales_mv and fweek_pscat_sales_mv) and the version of the same queries on DWU (4 marks):

REM SH2 VERSION

QUERY 1:

```
SELECT t.calendar_month_desc,
        sum(s.amount_sold) AS SOLD
FROM    sh2.sales s,
        sh2.times t
WHERE    s.time_id = t.time_id
AND     t.calendar_month_desc = '2001-01'
GROUP BY t.calendar_month_desc
order by t.calendar_month_desc;
```

QUERY 2:

```
SELECT   t.week_ending_day,
        p.prod_subcategory,
        sum(s.amount_sold) AS Money,
        s.channel_id,
        s.promo_id
FROM    sh2.sales s, sh2.times t, sh2.products p
```

```
WHERE    s.time_id = t.time_id
AND      s.prod_id = p.prod_id
AND      s.channel_id = 'D'
GROUP BY t.week_ending_day,
        p.prod_subcategory,
        s.channel_id,
        s.promo_id
order by t.week_ending_day;
```

REM DWU VERSION

QUERY 1:

```
SELECT  t.calendar_month_desc,
        sum(s.amount_sold) AS SOLD
FROM    sales s,
        times t
WHERE    s.time_id = t.time_id
AND     t.calendar_month_desc = '2001-01'
GROUP BY t.calendar_month_desc
order by t.calendar_month_desc;
```

QUERY 2:

```
SELECT   t.week_ending_day,
        p.prod_subcategory,
        sum(s.amount_sold) AS Money,
        s.channel_id,
        s.promo_id
FROM    sales s, times t, products p
WHERE    s.time_id = t.time_id
AND      s.prod_id = p.prod_id
AND      s.channel_id = 'D'
GROUP BY t.week_ending_day,
        p.prod_subcategory,
        s.channel_id,
        s.promo_id
order by t.week_ending_day;
```

Provide Explain Plan statements & outputs for the above 2 SQL queries you have run to compare the performance impact of those 2 MVs in SH2 and their version of the same queries on DWU (2 marks):

SH2 VERSION

QUERY 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT t.calendar_month_desc,
        sum(s.amount_sold) AS SOLD
FROM    sh2.sales s,
        sh2.times t
WHERE    s.time_id = t.time_id
AND     t.calendar_month_desc = '2001-01'
GROUP BY t.calendar_month_desc
```

```
order by t.calendar_month_desc;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
Plan hash value: 2368247697

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                   |     1 |    15 |     3   (0)| 00:00:01 |
|*  1 |  MAT_VIEW REWRITE ACCESS FULL| CAL_MONTH_SALES_MV |     1 |    15 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("CAL_MONTH_SALES_MV"."CALENDAR_MONTH_DESC"='2001-01')

13 rows selected.

Elapsed: 00:00:00.03
DWU36 >
```

QUERY 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT   t.week_ending_day,
        p.prod_subcategory,
        sum(s.amount_sold) AS Money,
        s.channel_id,
        s.promo_id
FROM    sh2.sales s, sh2.times t, sh2.products p
WHERE    s.time_id = t.time_id
AND      s.prod_id = p.prod_id
AND      s.channel_id = 'D'
GROUP BY t.week_ending_day,
        p.prod_subcategory,
        s.channel_id,
        s.promo_id
order by t.week_ending_day;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
Plan hash value: 1252709583

-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     | 10778 |   389K|   227   (2)| 00:00:01 |
|   1 |  SORT ORDER BY                |                     | 10778 |   389K|   227   (2)| 00:00:01 |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| FWEEK_PSCAT_SALES_MV | 10778 |   389K|   226   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

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

   2 - filter("FWEEK_PSCAT_SALES_MV"."CHANNEL_ID"='D')

14 rows selected.

Elapsed: 00:00:00.03
DWU36 >
```

DWU VERSION

QUERY 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT  t.calendar_month_desc,
        sum(s.amount_sold) AS SOLD
FROM    sales s,
        times t
WHERE    s.time_id = t.time_id
AND     t.calendar_month_desc = '2001-01'
GROUP BY t.calendar_month_desc
order by t.calendar_month_desc;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3097904601

-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     1 |    28 |  3296   (1)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY NOSORT         |          |     1 |    28 |  3296   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                   |          | 30587 |   836K|  3296   (1)| 00:00:01 |       |       |
|   3 |    PART JOIN FILTER CREATE    | :BF0000  |    31 |   496 |    13   (0)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL         | TIMES    |    31 |   496 |    13   (0)| 00:00:01 |       |       |
|   5 |    PARTITION RANGE JOIN-FILTER|          |  1016K|    11M|  3280   (1)| 00:00:01 |:BF0000|:BF0000|
|   6 |     TABLE ACCESS FULL         | SALES    |  1016K|    11M|  3280   (1)| 00:00:01 |:BF0000|:BF0000|
-------------------------------------------------------------------------------------------------------------------

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

   2 - access("S"."TIME_ID"="T"."TIME_ID")
   4 - filter("T"."CALENDAR_MONTH_DESC"='2001-01')

19 rows selected.

Elapsed: 00:00:00.05
```

QUERY 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT   t.week_ending_day,
        p.prod_subcategory,
        sum(s.amount_sold) AS Money,
        s.channel_id,
        s.promo_id
FROM    sales s, times t, products p
WHERE    s.time_id = t.time_id
AND      s.prod_id = p.prod_id
AND      s.channel_id = 'D'
GROUP BY t.week_ending_day,
        p.prod_subcategory,
        s.channel_id,
        s.promo_id
order by t.week_ending_day;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3567562560

-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |          | 60976 |  3691K|       |  4317   (1)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY                 |          | 60976 |  3691K|  4328K|  4317   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                    |          | 60976 |  3691K|       |  3402   (1)| 00:00:01 |       |       |
|   3 |    TABLE ACCESS FULL           | PRODUCTS | 10000 |   224K|       |   102   (0)| 00:00:01 |       |       |
|*  4 |    HASH JOIN                   |          | 60976 |  2322K|       |  3300   (1)| 00:00:01 |       |       |
|   5 |     PART JOIN FILTER CREATE    | :BF0000  |  1461 | 23376 |       |    13   (0)| 00:00:01 |       |       |
|   6 |      TABLE ACCESS FULL         | TIMES    |  1461 | 23376 |       |    13   (0)| 00:00:01 |       |       |
|   7 |     PARTITION RANGE JOIN-FILTER|          | 60976 |  1369K|       |  3286   (1)| 00:00:01 |:BF0000|:BF0000|
|*  8 |      TABLE ACCESS FULL         | SALES    | 60976 |  1369K|       |  3286   (1)| 00:00:01 |:BF0000|:BF0000|
-------------------------------------------------------------------------------------------------------------------------

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

   2 - access("S"."PROD_ID"="P"."PROD_ID")
   4 - access("S"."TIME_ID"="T"."TIME_ID")
   8 - filter("S"."CHANNEL_ID"='D')

22 rows selected.

Elapsed: 00:00:00.03
```

Provide Discussion of the cost-based comparison of the above 2 sets of queries and their explain plan cost figures (2marks):

Based on the above outputs, The MV's 'fweek_pscat_sales_mv' and 'cal_month_sales_mv' are used by SH2 but DWU version didn't use them because those MV's are present in SH2.

It is clear that costs of SH2 is less when compared with DWU for both query. And also SH2 version used less space than DWU version.

(A) Identify *three* new MVs based on the base tables in the SH schema under your DWU account and justify why they would be useful for the users of your data warehouse. Write the SQL code for creating these MVs. Moreover, run sample queries on both SH2 and DWU to ensure that queries running on DWU will be re-written by Oracle to use your proposed three MVs instead of the base tables used in the sample queries. Note that the queries should return subsets of the values contained in these MVs. Moreover, you must not query your MVs directly in the FROM clause; let the Oracle Query Optimizer re-write these queries and answer them using your proposed MVs.

(12 marks)

Answer Part 1 (D)

Provide SQL code and output you used to create the 3 new MVs in your own DWU database (i.e., these MVs *must not* exist in SH2) (3 marks):

Materialized View 1:

```
CREATE MATERIALIZED VIEW prod_cust_sales_mv
      PCTFREE 5
      STORAGE (INITIAL 8k NEXT 8k PCTINCREASE 0)
      BUILD IMMEDIATE
      REFRESH COMPLETE
      ENABLE QUERY REWRITE AS
SELECT   p.prod_id, p.prod_name, c.cust_gender,
       SUM(s.amount_sold) AS TotalSales
     FROM     sales s, products p, customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender ;
```

Materialized View 2:

```
CREATE MATERIALIZED VIEW country_cust_sales_mv
      PCTFREE 5
      STORAGE (INITIAL 8k NEXT 8k PCTINCREASE 0)
      BUILD IMMEDIATE
      REFRESH COMPLETE
      ENABLE QUERY REWRITE AS
      SELECT co.country_id, co.country_name, c.cust_gender,
        sum(s.amount_sold) sum_sales
    FROM sales s, customers c, countries co
      WHERE s.cust_id= c.cust_id
        AND c.country_id = co.country_id
      GROUP BY co.country_id, co.country_name,
       c.cust_gender, c.cust_marital_status;
```

Materialized View 3:

```
CREATE MATERIALIZED VIEW prod_sales_mv
      PCTFREE 5
      STORAGE (INITIAL 8k NEXT 8k PCTINCREASE 0)
      BUILD IMMEDIATE
      REFRESH COMPLETE
      ENABLE QUERY REWRITE AS
    SELECT p.prod_id, p.prod_name,
          sum(s.quantity_sold) sum_quantity,
          sum(s.amount_sold) sum_sales
      FROM sales s, products p
      WHERE s.prod_id= p.prod_id
        AND P.PROD_CATEGORY = 'Men'
      GROUP BY p.prod_id, p.prod_name
      order by p.prod_id;
```

Provide the 3 SQL queries you are going to run to compare the performance impact of your own 3 new MVs on DWU and the version of the same queries on SH2 (3 marks):

MV 1:

REM DWU VERSION

Materialized View 1:

```
SELECT p.prod_id, p.prod_name, c.cust_gender,
       SUM(s.amount_sold) AS TotalSales
     FROM     sales s, products p, customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
       AND    c.cust_gender = 'M'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender;
```

Materialized View 2:

```
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM sales s, customers c, countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;
```

Materialized View 3:

```
SELECT p.prod_id, p.prod_name,
    sum(s.quantity_sold) sum_quantity,
    sum(s.amount_sold) sum_sales
FROM sales s, products p
WHERE s.prod_id= p.prod_id
  AND P.PROD_CATEGORY = 'Men'
GROUP BY p.prod_id, p.prod_name;
```

REM SH2 VERSION

Materialized View 1:

```
SELECT p.prod_id, p.prod_name, c.cust_gender,
          SUM(s.amount_sold) AS TotalSales
     FROM     SH2.sales s, SH2.products p, SH2.customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
       AND    c.cust_gender = 'M'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender;
```

Materialized View 2:

```
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM SH2.sales s, SH2.customers c, SH2.countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
```

```
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;
```

Materialized View 3:

```
SELECT p.prod_id, p.prod_name,
    sum(s.quantity_sold) sum_quantity,
    sum(s.amount_sold) sum_sales
FROM SH2.sales s, SH2.products p
WHERE s.prod_id= p.prod_id
  AND P.PROD_CATEGORY = 'Men'
GROUP BY p.prod_id, p.prod_name;
```

Provide Explain Plan statements & outputs for the above 3 SQL queries you have run to compare the performance impact of your 3 MVs on DWU and their version of the same queries on SH2 (3 marks):

DWU VERSION

Materialized View 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_name, c.cust_gender,
       SUM(s.amount_sold) AS TotalSales
     FROM     sales s, products p, customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
       AND    c.cust_gender = 'M'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
Plan hash value: 2774803751

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                   |  1239 | 48321 |     7   (0)| 00:00:01 |
|*  1 |  MAT_VIEW REWRITE ACCESS FULL| PROD_CUST_SALES_MV |  1239 | 48321 |     7   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - filter("PROD_CUST_SALES_MV"."CUST_GENDER"='M')

13 rows selected.

Elapsed: 00:00:00.03
```

Materialized View 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on
```

```
EXPLAIN PLAN FOR
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM sales s, customers c, countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;


set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 2003742305

-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name             | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                  |     6 |   234 |  3562   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY                         |                  |     6 |   234 |  3562   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                            |                  |   148K|  5664K|  3558   (1)| 00:00:01 |       |       |
|*  3 |    HASH JOIN                           |                  |  7318 |   214K|   275   (1)| 00:00:01 |       |       |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| COUNTRIES        |     1 |    14 |     2   (0)| 00:00:01 |       |       |
|   5 |      BITMAP CONVERSION TO ROWIDS       |                  |       |       |            |          |       |       |
|*  6 |       BITMAP INDEX SINGLE VALUE        | COUNTRY_NAME_BIX |       |       |            |          |       |       |
|   7 |     TABLE ACCESS FULL                  | CUSTOMERS        | 50000 |   781K|   273   (1)| 00:00:01 |       |       |
|   8 |    PARTITION RANGE ALL                 |                  |  1016K|  8932K|  3280   (1)| 00:00:01 |     1 |    17 |
|   9 |     TABLE ACCESS FULL                  | SALES            |  1016K|  8932K|  3280   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------------------------------

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

   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   6 - access("CO"."COUNTRY_NAME"='United Kingdom')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 2 Sql Plan Directives used for this statement

28 rows selected.

Elapsed: 00:00:00.03
```

Materialized View 3:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_name,
    sum(s.quantity_sold) sum_quantity,
    sum(s.amount_sold) sum_sales
FROM sales s, products p
WHERE s.prod_id= p.prod_id
  AND P.PROD_CATEGORY = 'Men'
GROUP BY p.prod_id, p.prod_name;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------
Plan hash value: 95796306

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  1278 | 53676 |     5   (0)| 00:00:01 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| PROD_SALES_MV |  1278 | 53676 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

8 rows selected.

Elapsed: 00:00:00.03
```

SH2 VERSION

Materialized View 1:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_name, c.cust_gender,
          SUM(s.amount_sold) AS TotalSales
     FROM     SH2.sales s, SH2.products p, SH2.customers c
     WHERE    p.prod_id = s.prod_id
       AND    c.cust_id = s.cust_id
       AND    p.prod_category = 'Men'
       AND    c.cust_gender = 'M'
     GROUP BY p.prod_id, p.prod_name, c.cust_gender;
set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 1441809746

-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                 | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                      |  1259 | 74281 |  2345   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY                 |                      |  1259 | 74281 |  2345   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN                    |                      |   229K|    12M|  2339   (1)| 00:00:01 |       |       |
|*  3 |    VIEW                        | index$_join$_003     | 33685 |   230K|   148   (1)| 00:00:01 |       |       |
|*  4 |     HASH JOIN                  |                      |       |       |            |          |       |       |
|   5 |      BITMAP CONVERSION TO ROWIDS|                     | 33685 |   230K|     3   (0)| 00:00:01 |       |       |
|*  6 |       BITMAP INDEX SINGLE VALUE | CUSTOMERS_GENDER_BIX |       |       |            |          |       |       |
|   7 |      INDEX FAST FULL SCAN      | CUSTOMERS_PK         | 33685 |   230K|   180   (0)| 00:00:01 |       |       |
|*  8 |    HASH JOIN                   |                      |   229K|    11M|  2190   (1)| 00:00:01 |       |       |
|*  9 |     TABLE ACCESS FULL          | PRODUCTS             |  2594 | 98572 |   102   (0)| 00:00:01 |       |       |
|  10 |     PARTITION RANGE ALL        |                      |  1016K|    13M|  2086   (1)| 00:00:01 |     1 |    17 |
|  11 |      TABLE ACCESS FULL         | SALES                |  1016K|    13M|  2086   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------------------------

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

   2 - access("C"."CUST_ID"="S"."CUST_ID")
   3 - filter("C"."CUST_GENDER"='M')
   4 - access(ROWID=ROWID)
   6 - access("C"."CUST_GENDER"='M')
   8 - access("P"."PROD_ID"="S"."PROD_ID")
   9 - filter("P"."PROD_CATEGORY"='Men')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 5 Sql Plan Directives used for this statement

34 rows selected.

Elapsed: 00:00:00.07
```

Materialized View 2:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set echo on

EXPLAIN PLAN FOR
SELECT co.country_id, co.country_name, c.cust_gender,
  sum(s.amount_sold) sum_sales
FROM SH2.sales s, SH2.customers c, SH2.countries co
WHERE s.cust_id= c.cust_id
  AND c.country_id = co.country_id
  AND co.country_name = 'United Kingdom'
GROUP BY co.country_id, co.country_name,
 c.cust_gender, c.cust_marital_status;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
Plan hash value: 981620288

-------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |     1 |    39 |  2368   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY        |           |     1 |    39 |  2368   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN           |           |   155K|  5916K|  2364   (1)| 00:00:01 |       |       |
|*  3 |    HASH JOIN          |           |  7643 |   223K|   276   (1)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL | COUNTRIES |     1 |    14 |     3   (0)| 00:00:01 |       |       |
|   5 |     TABLE ACCESS FULL | CUSTOMERS | 50000 |   781K|   273   (1)| 00:00:01 |       |       |
|   6 |    PARTITION RANGE ALL|           |  1016K|  8932K|  2086   (1)| 00:00:01 |     1 |    17 |
|   7 |     TABLE ACCESS FULL | SALES     |  1016K|  8932K|  2086   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   4 - filter("CO"."COUNTRY_NAME"='United Kingdom')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 2 Sql Plan Directives used for this statement

27 rows selected.

Elapsed: 00:00:00.06
```

Materialized View 3:

```
alter session set query_rewrite_integrity = TRUSTED;
alter session set query_rewrite_enabled = TRUE;
set timing on
EXPLAIN PLAN FOR
SELECT p.prod_id, p.prod_name,
    sum(s.quantity_sold) sum_quantity,
    sum(s.amount_sold) sum_sales
FROM SH2.sales s, SH2.products p
WHERE s.prod_id= p.prod_id
  AND P.PROD_CATEGORY = 'Men'
GROUP BY p.prod_id, p.prod_name;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3535171836

-------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |  1277 | 63874 |  2196   (1)| 00:00:01 |       |       |
|   1 |  HASH GROUP BY        |          |  1277 | 63874 |  2196   (1)| 00:00:01 |       |       |
|*  2 |   HASH JOIN           |          |   229K|    10M|  2190   (1)| 00:00:01 |       |       |
|*  3 |    TABLE ACCESS FULL  | PRODUCTS |  2594 | 98572 |   102   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ALL|          |  1016K|    11M|  2086   (1)| 00:00:01 |     1 |    17 |
|   5 |     TABLE ACCESS FULL | SALES    |  1016K|    11M|  2086   (1)| 00:00:01 |     1 |    17 |
-------------------------------------------------------------------------------------------------------

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

   2 - access("S"."PROD_ID"="P"."PROD_ID")
   3 - filter("P"."PROD_CATEGORY"='Men')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
   - 3 Sql Plan Directives used for this statement

24 rows selected.

Elapsed: 00:00:00.05
```

Provide Discussion of the cost-based comparison of the above 3 sets of queries and their explain plan cost figures (3 marks):

Based on the above outputs, The MV's 'products_sales_mv', 'product_sales_channels_mv' and 'product_customer_sales_mv' are used by DWU36 but SH2 version didn't use them because those MV's are present in DWU36.

It is clear that costs of DWU36 is less when compared with SH2 for both query. And also, DWU36 version used less space than SH2 version because it is using Materialized View

(A) Prior to the introduction of the special aggregation function CUBE, there was no possibility to express an aggregation over different levels within a single SQL statement without using the set operation UNION ALL. Every different aggregation level needed its own SQL aggregation expression, operating on the exact same data set *n* times, once for each of the *n* different aggregation levels. With the introduction of CUBE in the recent database systems, Oracle provided a single SQL command for handling aggregations over different levels within a single SQL statement, not only improving the runtime of this operation but also reducing the number of internal operations necessary to run the query and results in reducing the workload on the system.

i. Using CUBE, write an SQL query over the SH schema under your DWU account involving **one** fact table (SALES or COSTS) and at **least two** dimension tables and at **least three** grouping attributes. Provide output of successful execution of your query. Provide reasons why your query may be useful for users of the SH data warehouse.

(3 marks)

Provide the CUBE query, its output / spool result and reasons why the query is useful for the users:

CODE:

```
column country format a25
column channel format a15
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month,
       co.country_name as Country,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
FROM sales s, customers c, times t, channels ch, countries co
WHERE s.time_id=t.time_id
    AND s.cust_id=c.cust_id
    AND s.channel_id= ch.channel_id
    AND c.country_id = co.country_id
    AND ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
GROUP BY ch.channel_desc, CUBE(t.calendar_month_desc, co.country_name)
Order By 1;
```

OUTPUT:

```
CHANNEL         CALENDER COUNTRY                   SALES
--------------- -------- ------------------------ -------------
Direct Sales    2003-09  United Kingdom               1,378,126
Direct Sales    2003-09  United States of America     2,835,557
Direct Sales    2003-09                               4,213,683
Direct Sales    2003-10  United Kingdom               1,388,051
Direct Sales    2003-10  United States of America     2,908,706
Direct Sales    2003-10                               4,296,757
Direct Sales             United Kingdom               2,766,177
Direct Sales             United States of America     5,744,263
Direct Sales                                          8,510,440
Internet        2003-09  United Kingdom                 911,739
Internet        2003-09  United States of America     1,732,240
Internet        2003-09                               2,643,979
Internet        2003-10  United Kingdom                 876,571
Internet        2003-10  United States of America     1,893,753
Internet        2003-10                               2,770,324
Internet                 United Kingdom               1,788,310
Internet                 United States of America     3,625,993
Internet                                              5,414,303

18 rows selected.

Elapsed: 00:00:00.31
```

ii. Using set operation UNION ALL (*and not CUBE*), write an SQL query that produces the same result as the query in (a) above. Provide output of successful execution of your query.

(4 marks)

Provide the UNION ALL query, its output / spool result:

CODE:

```
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month,
       co.country_name as Country,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  GROUP BY ch.channel_desc, t.calendar_month_desc, co.country_name
UNION ALL
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  GROUP BY ch.channel_desc, t.calendar_month_desc
UNION ALL
SELECT ch.channel_desc as Channel, NULL, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
```

```
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  GROUP BY ch.channel_desc
UNION ALL
SELECT NULL, NULL, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  ORDER BY 1, 2, 3;
```

OUTPUT:

```
CHANNEL         CALENDER COUNTRY                   SALES
--------------- -------- ------------------------- --------------
Direct Sales    2003-09  United Kingdom                1,378,126
Direct Sales    2003-09  United States of America      2,835,557
Direct Sales    2003-09                                4,213,683
Direct Sales    2003-10  United Kingdom                1,388,051
Direct Sales    2003-10  United States of America      2,908,706
Direct Sales    2003-10                                4,296,757
Direct Sales                                           8,510,440
Internet        2003-09  United Kingdom                  911,739
Internet        2003-09  United States of America      1,732,240
Internet        2003-09                                2,643,979
Internet        2003-10  United Kingdom                  876,571
Internet        2003-10  United States of America      1,893,753
Internet        2003-10                                2,770,324
Internet                                               5,414,303
                                                      13,924,743

15 rows selected.

Elapsed: 00:00:00.64
DWU36 >
```

iii. Using EXPLAIN PLAN, provide a detailed discussion analysing costs of evaluating the above queries (i.e., with and without CUBE).

(3 marks)

Provide Explain Plan statements & outputs for the above 2 SQL queries you have run to compare the performance of these 2 SQL queries and provide your discussion of their costs (3 marks):

CODE: (CUBE)

```
column country format a25
column channel format a20
EXPLAIN PLAN FOR
column country format a25
column channel format a15
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month,
       co.country_name as Country,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
FROM sales s, customers c, times t, channels ch, countries co
WHERE s.time_id=t.time_id
    AND s.cust_id=c.cust_id
    AND s.channel_id= ch.channel_id
    AND c.country_id = co.country_id
    AND ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
GROUP BY ch.channel_desc, CUBE(t.calendar_month_desc, co.country_name)
Order By 1;

set linesize 200
set pagesize 50

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

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3804022908

------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name              | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                   |    18 |  1242 |  1137   (1)| 00:00:01 |       |       |
|   1 |  SORT GROUP BY                         |                   |    18 |  1242 |  1137   (1)| 00:00:01 |       |       |
|   2 |   GENERATE CUBE                        |                   |    18 |  1242 |  1137   (1)| 00:00:01 |       |       |
|   3 |    SORT GROUP BY                       |                   |    18 |  1242 |  1137   (1)| 00:00:01 |       |       |
|*  4 |     HASH JOIN                          |                   | 24074 |  1622K|  1136   (1)| 00:00:01 |       |       |
|   5 |      INLIST ITERATOR                   |                   |       |       |            |          |       |       |
|   6 |       TABLE ACCESS BY INDEX ROWID      | COUNTRIES         |     2 |    28 |     2   (0)| 00:00:01 |       |       |
|*  7 |        INDEX UNIQUE SCAN               | COUNTRY_PK        |     2 |       |     1   (0)| 00:00:01 |       |       |
|*  8 |      HASH JOIN                         |                   | 24074 |  1293K|  1133   (1)| 00:00:01 |       |       |
|*  9 |       TABLE ACCESS FULL                | CUSTOMERS         | 22198 |   173K|   273   (1)| 00:00:01 |       |       |
|  10 |       NESTED LOOPS                     |                   | 24075 |  1105K|   860   (0)| 00:00:01 |       |       |
|  11 |        NESTED LOOPS                    |                   | 24075 |  1105K|   860   (0)| 00:00:01 |       |       |
|  12 |         MERGE JOIN CARTESIAN           |                   |   122 |  3416 |    28   (0)| 00:00:01 |       |       |
|* 13 |          TABLE ACCESS FULL             | CHANNELS          |     2 |    24 |     3   (0)| 00:00:01 |       |       |
|  14 |          BUFFER SORT                   |                   |    61 |   976 |    25   (0)| 00:00:01 |       |       |
|* 15 |           TABLE ACCESS FULL            | TIMES             |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|  16 |         PARTITION RANGE ITERATOR       |                   |       |       |            |          |   KEY |   KEY |
|  17 |          BITMAP CONVERSION TO ROWIDS   |                   |       |       |            |          |       |       |
|  18 |           BITMAP AND                   |                   |       |       |            |          |       |       |
|* 19 |            BITMAP INDEX SINGLE VALUE   | SALES_TIME_BIX    |       |       |            |          |   KEY |   KEY |
|* 20 |            BITMAP INDEX SINGLE VALUE   | SALES_CHANNEL_BIX |       |       |            |          |   KEY |   KEY |
|  21 |        TABLE ACCESS BY LOCAL INDEX ROWID| SALES             |   197 |  3743 |   860   (0)| 00:00:01 |     1 |     1 |
------------------------------------------------------------------------------------------------------------------------------------

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

   4 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   7 - access("CO"."COUNTRY_ID"='UK' OR "CO"."COUNTRY_ID"='US')
   8 - access("S"."CUST_ID"="C"."CUST_ID")
   9 - filter("C"."COUNTRY_ID"='UK' OR "C"."COUNTRY_ID"='US')
  13 - filter("CH"."CHANNEL_DESC"='Direct Sales' OR "CH"."CHANNEL_DESC"='Internet')
  15 - filter("T"."CALENDAR_MONTH_DESC"='2003-09' OR "T"."CALENDAR_MONTH_DESC"='2003-10')
  19 - access("S"."TIME_ID"="T"."TIME_ID")
  20 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 3 Sql Plan Directives used for this statement

45 rows selected.

Elapsed: 00:00:00.04
```

CODE: (UNION ALL)

```
column country format a25
column channel format a20
EXPLAIN PLAN FOR
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month,
       co.country_name as Country,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  GROUP BY ch.channel_desc, t.calendar_month_desc, co.country_name
UNION ALL
SELECT ch.channel_desc as Channel,
       t.calendar_month_desc as Calender_Month, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
```

```
  GROUP BY ch.channel_desc, t.calendar_month_desc
UNION ALL
SELECT ch.channel_desc as Channel, NULL, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
  FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  GROUP BY ch.channel_desc
UNION ALL
SELECT NULL, NULL, NULL,
       TO_CHAR(SUM(s.amount_sold), '9,999,999,999') as Sales
FROM sales s INNER JOIN times t ON s.time_id=t.time_id
    INNER JOIN channels ch ON s.channel_id= ch.channel_id
    INNER JOIN customers c ON s.cust_id=c.cust_id
    INNER JOIN countries co ON c.country_id = co.country_id
  WHERE ch.channel_desc IN ('Direct Sales', 'Internet')
    AND t.calendar_month_desc IN ('2003-09', '2003-10')
    AND c.country_id IN ('UK', 'US')
  ORDER BY 1, 2, 3;

set linesize 200
set pagesize 50
select * from table(dbms_xplan.display());
```

OUTPUT:

```
DWU36 > select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 2535861111

-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name              | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                   |    10 |   606 | 14297   (1)| 00:00:01 |       |       |
|   1 |  SORT ORDER BY                      |                   |    10 |   606 | 14296   (1)| 00:00:01 |       |       |
|   2 |   UNION-ALL                         |                   |       |       |            |          |       |       |
|   3 |    HASH GROUP BY                    |                   |     4 |   276 |  3575   (1)| 00:00:01 |       |       |
|*  4 |     HASH JOIN                       |                   | 24075 |  1622K|  3574   (1)| 00:00:01 |       |       |
|*  5 |      VIEW                           | index$_join$_008  |     2 |    28 |     1   (0)| 00:00:01 |       |       |
|*  6 |       HASH JOIN                     |                   |       |       |            |          |       |       |
|   7 |        INLIST ITERATOR              |                   |       |       |            |          |       |       |
|*  8 |         INDEX UNIQUE SCAN           | COUNTRY_PK        |     2 |    28 |     0   (0)| 00:00:01 |       |       |
|   9 |        BITMAP CONVERSION TO ROWIDS  |                   |     2 |    28 |     1   (0)| 00:00:01 |       |       |
|  10 |         BITMAP INDEX FULL SCAN      | COUNTRY_NAME_BIX  |       |       |            |          |       |       |
|* 11 |      HASH JOIN                      |                   | 24075 |  1293K|  3573   (1)| 00:00:01 |       |       |
|* 12 |       TABLE ACCESS FULL             | CUSTOMERS         | 21647 |   169K|   273   (1)| 00:00:01 |       |       |
|* 13 |       HASH JOIN                     |                   | 24075 |  1105K|  3299   (1)| 00:00:01 |       |       |
|* 14 |        TABLE ACCESS FULL            | CHANNELS          |     2 |    24 |     3   (0)| 00:00:01 |       |       |
|* 15 |        HASH JOIN                    |                   | 60187 |  2057K|  3296   (1)| 00:00:01 |       |       |
|  16 |         PART JOIN FILTER CREATE     | :BF0000           |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|* 17 |          TABLE ACCESS FULL          | TIMES             |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|  18 |         PARTITION RANGE JOIN-FILTER |                   |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0000|:BF0000|
|  19 |          TABLE ACCESS FULL          | SALES             |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0000|:BF0000|
|  20 |    HASH GROUP BY                    |                   |     3 |   165 |  3574   (1)| 00:00:01 |       |       |
|* 21 |     HASH JOIN                       |                   | 24075 |  1293K|  3573   (1)| 00:00:01 |       |       |
|* 22 |      TABLE ACCESS FULL              | CUSTOMERS         | 21647 |   169K|   273   (1)| 00:00:01 |       |       |
|* 23 |      HASH JOIN                      |                   | 24075 |  1105K|  3299   (1)| 00:00:01 |       |       |
|* 24 |       TABLE ACCESS FULL             | CHANNELS          |     2 |    24 |     3   (0)| 00:00:01 |       |       |
|* 25 |       HASH JOIN                     |                   | 60187 |  2057K|  3296   (1)| 00:00:01 |       |       |
|  26 |        PART JOIN FILTER CREATE      | :BF0001           |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|* 27 |         TABLE ACCESS FULL           | TIMES             |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|  28 |        PARTITION RANGE JOIN-FILTER  |                   |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0001|:BF0001|
|  29 |         TABLE ACCESS FULL           | SALES             |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0001|:BF0001|
|  30 |    HASH GROUP BY                    |                   |     2 |   110 |  3574   (1)| 00:00:01 |       |       |
|* 31 |     HASH JOIN                       |                   | 24075 |  1293K|  3573   (1)| 00:00:01 |       |       |
|* 32 |      TABLE ACCESS FULL              | CUSTOMERS         | 21647 |   169K|   273   (1)| 00:00:01 |       |       |
|* 33 |      HASH JOIN                      |                   | 24075 |  1105K|  3299   (1)| 00:00:01 |       |       |
|* 34 |       TABLE ACCESS FULL             | CHANNELS          |     2 |    24 |     3   (0)| 00:00:01 |       |       |
|* 35 |       HASH JOIN                     |                   | 60187 |  2057K|  3296   (1)| 00:00:01 |       |       |
|  36 |        PART JOIN FILTER CREATE      | :BF0002           |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|* 37 |         TABLE ACCESS FULL           | TIMES             |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|  38 |        PARTITION RANGE JOIN-FILTER  |                   |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0002|:BF0002|
|  39 |         TABLE ACCESS FULL           | SALES             |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0002|:BF0002|
|  40 |    SORT AGGREGATE                   |                   |     1 |    55 |            |          |       |       |
|* 41 |     HASH JOIN                       |                   | 24075 |  1293K|  3573   (1)| 00:00:01 |       |       |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------
|* 42 |      TABLE ACCESS FULL              | CUSTOMERS         | 21647 |   169K|   273   (1)| 00:00:01 |       |       |
|* 43 |      HASH JOIN                      |                   | 24075 |  1105K|  3299   (1)| 00:00:01 |       |       |
|* 44 |       TABLE ACCESS FULL             | CHANNELS          |     2 |    24 |     3   (0)| 00:00:01 |       |       |
|* 45 |       HASH JOIN                     |                   | 60187 |  2057K|  3296   (1)| 00:00:01 |       |       |
|  46 |        PART JOIN FILTER CREATE      | :BF0003           |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|* 47 |         TABLE ACCESS FULL           | TIMES             |    61 |   976 |    13   (0)| 00:00:01 |       |       |
|  48 |        PARTITION RANGE JOIN-FILTER  |                   |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0003|:BF0003|
```

```
|  49 |         TABLE ACCESS FULL          | SALES           |  1016K|    18M|  3280   (1)| 00:00:01 |:BF0003|:BF0003|
-----------------------------------------------------------------------------------------------------------------------

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

   4 - access("C"."COUNTRY_ID"="CO"."COUNTRY_ID")
   5 - filter("CO"."COUNTRY_ID"='UK' OR "CO"."COUNTRY_ID"='US')
   6 - access(ROWID=ROWID)
   8 - access("CO"."COUNTRY_ID"='UK' OR "CO"."COUNTRY_ID"='US')
  11 - access("S"."CUST_ID"="C"."CUST_ID")
  12 - filter("C"."COUNTRY_ID"='UK' OR "C"."COUNTRY_ID"='US')
  13 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
  14 - filter("CH"."CHANNEL_DESC"='Direct Sales' OR "CH"."CHANNEL_DESC"='Internet')
  15 - access("S"."TIME_ID"="T"."TIME_ID")
  17 - filter("T"."CALENDAR_MONTH_DESC"='2003-09' OR "T"."CALENDAR_MONTH_DESC"='2003-10')
  21 - access("S"."CUST_ID"="C"."CUST_ID")
  22 - filter("C"."COUNTRY_ID"='UK' OR "C"."COUNTRY_ID"='US')
  23 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
  24 - filter("CH"."CHANNEL_DESC"='Direct Sales' OR "CH"."CHANNEL_DESC"='Internet')
  25 - access("S"."TIME_ID"="T"."TIME_ID")
  27 - filter("T"."CALENDAR_MONTH_DESC"='2003-09' OR "T"."CALENDAR_MONTH_DESC"='2003-10')
  31 - access("S"."CUST_ID"="C"."CUST_ID")
  32 - filter("C"."COUNTRY_ID"='UK' OR "C"."COUNTRY_ID"='US')
  33 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
  34 - filter("CH"."CHANNEL_DESC"='Direct Sales' OR "CH"."CHANNEL_DESC"='Internet')
  35 - access("S"."TIME_ID"="T"."TIME_ID")
  37 - filter("T"."CALENDAR_MONTH_DESC"='2003-09' OR "T"."CALENDAR_MONTH_DESC"='2003-10')
  41 - access("S"."CUST_ID"="C"."CUST_ID")
  42 - filter("C"."COUNTRY_ID"='UK' OR "C"."COUNTRY_ID"='US')
  43 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
  44 - filter("CH"."CHANNEL_DESC"='Direct Sales' OR "CH"."CHANNEL_DESC"='Internet')
  45 - access("S"."TIME_ID"="T"."TIME_ID")
  47 - filter("T"."CALENDAR_MONTH_DESC"='2003-09' OR "T"."CALENDAR_MONTH_DESC"='2003-10')

88 rows selected.

Elapsed: 00:00:00.08
```

From the above explain plan, it is evident that cube uses less cost when compared to Union all.

Part 2: Data Mining Tasks (35 Marks)

This part is based on the Global Credit Finance credit card company's customers defaults scenario as described in Appendix 2. The main purpose of this part is correctly predicting if credit card customers will default on their due payments. You are required to perform the following tasks:

1. Explore the dataset and justify whether GlobalCRF's problem belongs to predictive or descriptive data mining models. Choose which data mining task (e.g., classification, association rules, clustering, regression, etc) will be used to produce data mining models for the GlobalCRF's scenario.

(5 marks)

Provide your answer here

GlobalCRF's problem belongs to predictive data mining models. We choose Classification data mining task for the process.

1. Prepare and setup your views and tables under your DMU account for accessing the shared `CreditCardsV2` dataset, which also includes splitting the dataset for building, testing and applying the data mining models.

(6 marks)

Provide whatever code and outputs you have used for this part or screenshots where relevant.

CODE:

```
REM from 1st to 30,000th  rows of the CreditCardsV2 table
REM by making a copy of it in your own schema

Create Table MyCreditCards AS
select * from CreditCardsV2;

REM Let us now divide it up to 3 subsets

REM from 1 to 10,000th of MyCreditCards order by edulevel
create or replace view mining_data_apply_v AS
select * FROM
(select c.*, row_number() over  (order by c.edulevel) as RNK
from  MyCreditCards c)
WHERE RNK <= 10000;

select count(*) from  mining_data_apply_v;

REM from 10,001th to 20,000th of MyCreditCards order by edulevel
create or replace view mining_data_build_v AS
select * FROM
(select c.*, row_number() over  (order by c.edulevel) as RNK
from  MyCreditCards c)
WHERE RNK > 10000 AND RNK <= 20000;

select count(*) from mining_data_build_v;

REM from 20,001th to 30,000th of MyCreditCards order by edulevel
create or replace view mining_data_test_v AS
select * FROM
(select c.*, row_number() over  (order by c.edulevel) as RNK
from  MyCreditCards c)
WHERE RNK > 20000 AND RNK <= 30000;

select count(*) from mining_data_test_v;
```

OUTPUT :

```
DWU36 > REM from 1st to 30,000th  rows of the CreditCardsV2 table
DWU36 > REM by making a copy of it in your own schema
DWU36 >
DWU36 > Create Table MyCreditCards AS
  2  select * from CreditCardsV2;

Table created.

DWU36 >
DWU36 > REM Let us now divide it up to 3 subsets
DWU36 >
DWU36 > REM from 1 to 10,000th of MyCreditCards order by edulevel
DWU36 > create or replace view mining_data_apply_v AS
  2  select * FROM
  3  (select c.*, row_number() over  (order by c.edulevel) as RNK
  4  from  MyCreditCards c)
  5  WHERE RNK <= 10000;

View created.

DWU36 >
DWU36 > select count(*) from  mining_data_apply_v;

  COUNT(*)
----------
     10000

DWU36 >
DWU36 > REM from 10,001th to 20,000th of MyCreditCards order by edulevel
DWU36 > create or replace view mining_data_build_v AS
  2  select * FROM
  3  (select c.*, row_number() over  (order by c.edulevel) as RNK
  4  from  MyCreditCards c)
  5  WHERE RNK > 10000 AND RNK <= 20000;

View created.

DWU36 >
DWU36 > select count(*) from mining_data_build_v;

  COUNT(*)
----------
     10000

DWU36 >
DWU36 > REM from 20,001th to 30,000th of MyCreditCards order by edulevel
DWU36 > create or replace view mining_data_test_v AS
  2  select * FROM
  3  (select c.*, row_number() over  (order by c.edulevel) as RNK
  4  from  MyCreditCards c)
  5  WHERE RNK > 20000 AND RNK <= 30000;

View created.

DWU36 >
DWU36 > select count(*) from mining_data_test_v;

  COUNT(*)
----------
     10000
```

2. Using the PL/SQL Data Mining API, develop at least TWO models using suitable algorithms for performing your chosen data mining task for the GlobalCRF's dataset.

(12 marks)

Provide here all the Oracle Data Mining PL/SQL API and SQL code you have used for this part including spool file contents / outputs; make sure that the output shows both the code and result / output when the code has been executed. Hint: Use **SET ECHO ON** and **SET SERVEROUTPUT ON**.

Model 1: NAIVE_BAYES

Code:

```
REM Create the settings table
CREATE TABLE naiveb_model_settings (
setting_name VARCHAR2(30),
setting_value VARCHAR2(30));
REM Populate the settings table
BEGIN
INSERT INTO naiveb_model_settings VALUES
(dbms_data_mining.algo_name,
dbms_data_mining.ALGO_NAIVE_BAYES);
INSERT INTO naiveb_model_settings VALUES
(dbms_data_mining.prep_auto,
dbms_data_mining.prep_auto_on);
COMMIT;
END;
/

REM Create the model using the specified settings
BEGIN
DBMS_DATA_MINING.CREATE_MODEL(
model_name => 'naiveb_model36',
mining_function => dbms_data_mining.classification,
data_table_name => 'mining_data_build_v',
case_id_column_name => 'card',
target_column_name => 'defaultpaynxtmnt',
settings_table_name => 'naiveb_model_settings');
END;
/

REM testing the model
SELECT defaultpaynxtmnt AS actual_target_value,
PREDICTION(naiveb_model36 USING *) AS predicted_target_value,
COUNT(*) AS total_value
FROM mining_data_test_v
GROUP BY defaultpaynxtmnt, PREDICTION(naiveb_model36 USING *)
ORDER BY 1, 2;

REM calculating the models accuracy
COLUMN ACCURACY FORMAT 99.99
SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
FROM (SELECT DECODE(defaultpaynxtmnt,
PREDICTION(naiveb_model36 USING *), 1, 0) AS correct
FROM mining_data_test_v);
```

OUTPUT:

ORACLE - SQLPLUS

```
DWU36 > CREATE TABLE naiveb_model_settings (
  2  setting_name VARCHAR2(30),
  3  setting_value VARCHAR2(30));

Table created.

DWU36 > REM Populate the settings table
DWU36 > BEGIN
  2  INSERT INTO naiveb_model_settings VALUES
  3  (dbms_data_mining.algo_name,
  4  dbms_data_mining.ALGO_NAIVE_BAYES);
  5  INSERT INTO naiveb_model_settings VALUES
  6  (dbms_data_mining.prep_auto,
  7  dbms_data_mining.prep_auto_on);
  8  COMMIT;
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 >
DWU36 > REM Create the model using the specified settings
DWU36 > BEGIN
  2  DBMS_DATA_MINING.CREATE_MODEL(
  3  model_name => 'naiveb_model36',
  4  mining_function => dbms_data_mining.classification,
  5  data_table_name => 'mining_data_build_v',
  6  case_id_column_name => 'card',
  7  target_column_name => 'defaultpaynxtmnt',
  8  settings_table_name => 'naiveb_model_settings');
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 >
DWU36 > REM testing the model
DWU36 > SELECT defaultpaynxtmnt AS actual_target_value,
  2  PREDICTION(naiveb_model36 USING *) AS predicted_target_value,
  3  COUNT(*) AS total_value
  4  FROM mining_data_test_v
  5  GROUP BY defaultpaynxtmnt, PREDICTION(naiveb_model36 USING *)
  6  ORDER BY 1, 2;

ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE TOTAL_VALUE
------------------- ---------------------- -----------
                  0                      0        6547
                  0                      1        1075
                  1                      0        1205
                  1                      1        1173

DWU36 >
DWU36 > REM calculating the models accuracy
DWU36 > COLUMN ACCURACY FORMAT 99.99
DWU36 > SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
  2  FROM (SELECT DECODE(defaultpaynxtmnt,
  3  PREDICTION(naiveb_model36 USING *), 1, 0) AS correct
  4  FROM mining_data_test_v);

ACCURACY
--------
   77.20
```

Model 2: Decision Tree

Code:

```
REM Create the settings table
CREATE TABLE dtree_model_settings (
setting_name VARCHAR2(30),
setting_value VARCHAR2(30));
REM Populate the settings table
BEGIN
INSERT INTO dtree_model_settings VALUES
(dbms_data_mining.algo_name,
dbms_data_mining.ALGO_DECISION_TREE);
INSERT INTO dtree_model_settings VALUES
(dbms_data_mining.prep_auto,
dbms_data_mining.prep_auto_on);
COMMIT;
END;
/

REM Create the model using the specified settings
BEGIN
DBMS_DATA_MINING.CREATE_MODEL(
model_name => 'decision_tree_model36',
mining_function => dbms_data_mining.classification,
data_table_name => 'mining_data_build_v',
case_id_column_name => 'card',
target_column_name => 'defaultpaynxtmnt',
settings_table_name => 'dtree_model_settings');
END;
/

REM testing the model
SELECT defaultpaynxtmnt AS actual_target_value,
PREDICTION(decision_tree_model36 USING *) AS predicted_target_value,
COUNT(*) AS total_value
FROM mining_data_test_v
GROUP BY defaultpaynxtmnt, PREDICTION(decision_tree_model36 USING *)
ORDER BY 1, 2;

REM calculating the models accuracy
COLUMN ACCURACY FORMAT 99.99
SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
FROM (SELECT DECODE(defaultpaynxtmnt,
PREDICTION(decision_tree_model36 USING *), 1, 0) AS correct
FROM mining_data_test_v);
```

OUTPUT :

```
DWU36 > REM Create the settings table
DWU36 > CREATE TABLE dtree_model_settings (
  2  setting_name VARCHAR2(30),
  3  setting_value VARCHAR2(30));

Table created.

DWU36 > REM Populate the settings table
DWU36 > BEGIN
  2  INSERT INTO dtree_model_settings VALUES
  3  (dbms_data_mining.algo_name,
  4  dbms_data_mining.ALGO_DECISION_TREE);
  5  INSERT INTO dtree_model_settings VALUES
  6  (dbms_data_mining.prep_auto,
  7  dbms_data_mining.prep_auto_on);
  8  COMMIT;
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 > REM Create the model using the specified settings
DWU36 > BEGIN
  2  DBMS_DATA_MINING.CREATE_MODEL(
  3  model_name => 'decision_tree_model36',
  4  mining_function => dbms_data_mining.classification,
  5  data_table_name => 'mining_data_build_v',
  6  case_id_column_name => 'card',
  7  target_column_name => 'defaultpaynxtmnt',
  8  settings_table_name => 'dtree_model_settings');
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 > REM testing the model
DWU36 > SELECT defaultpaynxtmnt AS actual_target_value,
  2  PREDICTION(decision_tree_model36 USING *) AS predicted_target_value,
  3  COUNT(*) AS total_value
  4  FROM mining_data_test_v
  5  GROUP BY defaultpaynxtmnt, PREDICTION(decision_tree_model36 USING *)
  6  ORDER BY 1, 2;

ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE TOTAL_VALUE
------------------- ---------------------- -----------
                  0                      0        7184
                  0                      1         438
                  1                      0        1548
                  1                      1         830

DWU36 >
DWU36 > REM calculating the models accuracy
DWU36 > COLUMN ACCURACY FORMAT 99.99
DWU36 > SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
  2  FROM (SELECT DECODE(defaultpaynxtmnt,
  3  PREDICTION(decision_tree_model36 USING *), 1, 0) AS correct
  4  FROM mining_data_test_v);

ACCURACY
--------
   80.14
```

Model 3: SVM model

Code:

```
REM Create the settings table
CREATE TABLE svm_model_settings (
setting_name VARCHAR2(30),
setting_value VARCHAR2(30));
REM Populate the settings table
BEGIN
INSERT INTO svm_model_settings VALUES
(dbms_data_mining.algo_name,
dbms_data_mining.algo_support_vector_machines);
INSERT INTO svm_model_settings VALUES
(dbms_data_mining.prep_auto,
dbms_data_mining.prep_auto_on);
COMMIT;
END;
/
REM Create the model using the specified settings
BEGIN
DBMS_DATA_MINING.CREATE_MODEL(
model_name => 'svm_model36',
mining_function => dbms_data_mining.classification,
data_table_name => 'mining_data_build_v',
case_id_column_name => 'card',
target_column_name => 'defaultpaynxtmnt',
settings_table_name => 'svm_model_settings');
END;
/
REM testing the model
SELECT defaultpaynxtmnt AS actual_target_value,
PREDICTION(svm_model36 USING *) AS predicted_target_value,
COUNT(*) AS total_value
FROM mining_data_test_v
GROUP BY defaultpaynxtmnt, PREDICTION(svm_model36 USING *)
ORDER BY 1, 2;

REM calculating the models accuracy
COLUMN ACCURACY FORMAT 99.99
SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
FROM (SELECT DECODE(defaultpaynxtmnt,
PREDICTION(svm_model36 USING *), 1, 0) AS correct
FROM mining_data_test_v);
```

OUTPUT:

```
DWU36 > REM Create the settings table
DWU36 > CREATE TABLE svm_model_settings (
  2  setting_name VARCHAR2(30),
  3  setting_value VARCHAR2(30));

Table created.

DWU36 > REM Populate the settings table
DWU36 > BEGIN
  2  INSERT INTO svm_model_settings VALUES
  3  (dbms_data_mining.algo_name,
  4  dbms_data_mining.algo_support_vector_machines);
  5  INSERT INTO svm_model_settings VALUES
  6  (dbms_data_mining.prep_auto,
  7  dbms_data_mining.prep_auto_on);
  8  COMMIT;
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 > REM Create the model using the specified settings
DWU36 > BEGIN
  2  DBMS_DATA_MINING.CREATE_MODEL(
  3  model_name => 'svm_model36',
  4  mining_function => dbms_data_mining.classification,
  5  data_table_name => 'mining_data_build_v',
  6  case_id_column_name => 'card',
  7  target_column_name => 'defaultpaynxtmnt',
  8  settings_table_name => 'svm_model_settings');
  9  END;
 10  /

PL/SQL procedure successfully completed.

DWU36 > REM testing the model
DWU36 > SELECT defaultpaynxtmnt AS actual_target_value,
  2  PREDICTION(svm_model36 USING *) AS predicted_target_value,
  3  COUNT(*) AS total_value
  4  FROM mining_data_test_v
  5  GROUP BY defaultpaynxtmnt, PREDICTION(svm_model36 USING *)
  6  ORDER BY 1, 2;

ACTUAL_TARGET_VALUE PREDICTED_TARGET_VALUE TOTAL_VALUE
------------------- ---------------------- -----------
                  0                      0        7311
                  0                      1         311
                  1                      0        1748
                  1                      1         630

DWU36 >
DWU36 > REM calculating the models accuracy
DWU36 > COLUMN ACCURACY FORMAT 99.99
DWU36 > SELECT (SUM(correct)/COUNT(*))*100 AS accuracy
  2  FROM (SELECT DECODE(defaultpaynxtmnt,
  3  PREDICTION(svm_model36 USING *), 1, 0) AS correct
  4  FROM mining_data_test_v);

ACCURACY
--------
   79.41
```

3. Evaluate capabilities of the models you have developed.

(6 marks)

Provide whatever code and outputs you have used for this part or screenshots where relevant.

Model 1: Naive Bayes

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	TOTAL_VALUE
0	0	6547
0	1	1075
1	0	1205
1	1	1173

ACCURACY
77.20

Model 2: Decision Tree

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	TOTAL_VALUE
0	0	7184
0	1	438
1	0	1548
1	1	830

DWU36 >

ACCURACY
80.14

Model 3: SVM model

ACTUAL_TARGET_VALUE	PREDICTED_TARGET_VALUE	TOTAL_VALUE
0	0	7311
0	1	311
1	0	1748
1	1	630

DWU36 >

ACCURACY
79.41

4. Present and discuss your findings and make recommendations for GlobalCRF

(6 marks)

Provide your answer here

From above results, Decision tree gives more accurate results than the Naïve bayes and SVM model. So Decision tree is recommended to GlobalCRF for the process.

Part 3 (15 marks)

Critically evaluate the SH data warehouse and the GlobalCRF's customer credit card defaults dataset in relation to the theory and best practices of data quality and standards.

The report should be concise and comprehensive and in the region of 900-1000 words. You should use Harvard style of citation and referencing by following the guidelines in Pears and Shields (2008).

Answer Part 3: 15 Marks [10 for the quality of your report addressing the above points, 3 for the quality of referencing and citation and adhering to the Harvard style, 2 for presentation of the report]

Data Quality:
Data quality is a metric that assesses the state of data based on parameters such as accuracy, completeness, consistency, reliability, and timeliness. Measuring data quality levels can aid organisations in identifying data issues and determining whether the data in the database is fit for the intended purpose.

In today's world of data-driven decision making, data must be seen as an organisational asset; data without quality serves no use. Data quality is an assessment of data's usefulness for a certain purpose. Data quality is a key factor influencing the reliability of data for decision making. If the data is untrustworthy, then so are the analytics and reports that rely on it. To put it another way, if your data is of high quality, it can provide the insight you want. In contrast, if you don't have data quality, there is an issue with your data that will prohibit you from using it to reach your goals.

Bad data may have serious commercial ramifications for businesses. Poor-quality data is frequently blamed for operational blunders, erroneous analytics, and ill-conceived corporate initiatives. Additional expenses when products are delivered to the wrong client addresses, lost sales opportunities due to the incorrect or incomplete client information, and consequences for improper financial or regulatory compliance reporting are just a few examples of the economic damage that data quality issues can cause (Donald P. Ballou, 1985).

Data verification typically assesses the analytical process's conformity with the project plan and other project requirements papers, as well as the statement of work (SOW), and details compliance and noncompliance in a data verification report. In contrast to the inspections and reviews performed by lab and field workers throughout implementation, data verification is a distinct operation. Documentation created during the implementation phase will be utilised to verify if suitable processes were followed and compliance with project plan papers, service agreement requirements, and

measurement quality targets. Any data connected with failure to comply will be flagged as an exception, requiring additional inquiry during data validation.
A data warehouse is a sort of data management system that enables and supports business intelligence (BI) operations, particularly analytics. Data warehouses are purely meant for query and analytical purposes, and they frequently hold vast volumes of historical data. A data warehouse's data is often generated from a variety of sources, including applications log files and transactional programmes (Leo L. Pipino, 2002).

The **EXPLAIN PLAN** command reveals the Oracle optimizer's execution plans for SELECT, UPDATE, INSERT, and DELETE queries. The execution plan of a statement is the series of activities performed by Oracle to execute the statement.
The execution plan revolves around the row source tree. It displays the following data:

- An ordering of the tables referenced by the statement
- An access method for each table stated in the statement
- A join method for tables impacted by join operations in the statement
- Data operations like filter, sort, or aggregation

The EXPLAIN PLAN results help you know whether the optimizer chooses a specific execution plan, such as nested loops join. It also allows you to understand optimizer decisions, like as why the optimizer picked a nested loops join over a hash join, and it allows you to evaluate query performance.
An **Oracle view** is a representation of data from one or more oracle tables or views. Views do not contain any data; they are simply a database query. All the information displayed is derived from the basic tables. A view may be thought of as a virtual table or mapping of data from one or more sources. Except for the definition of the view in the data dictionary, a view requires no storage space. A view could be used to display a subset of data, a superset of data (joining many tables to one view), to conceal difficult joins, to offer descriptive names for columns, and to reduce application and data source modifications.

```
CREATE VIEW view_name AS

SELECT columns

FROM tables

[WHERE conditions];
```

DML operations like insert, delete and update on the view just like the oracle table can be created with some restrictions. Advantage of view is the View is based on a single SQL query, and the View definition code will not change. As a result, when a view definition is invoked, less parsing is required (Lee, Y.W., 2006). In Oracle, a **materialised view** is a database object that includes the results of a query. They are either local copies of distant data or are used to generate tables depending on aggregations of a table's data. Snapshots are materialised views that store data based on distant tables.

Materialized views let you to keep copies of distant data on the local node for replication reasons. These are read-only copies. It should utilise the Advanced Replication functionality to update the local copies. A materialised view may be used to choose data in the same way that a table or view can (Cai L., 2015).
Materialized views for data warehousing are often developed as aggregate views, single-table aggregate views, and join views.
In replication environments, the materialized views commonly created are

- primary key materialized views

```
SQL>       CREATE MATERIALIZED VIEW mv_emp_pk
           REFRESH FAST START WITH SYSDATE
  NEXT  SYSDATE + 1/48
  WITH PRIMARY KEY
  AS SELECT * FROM emp@remote_db;

Materialized view created.
```

To create a materialized view using the FAST option there is a need to create a view log on the master tables as shown below:

```
SQL> CREATE MATERIALIZED VIEW LOG ON emp;
Materialized view log created.
```

- rowed materialized views

```
SQL>       CREATE MATERIALIZED VIEW mv_emp_rowid
  REFRESH WITH ROWID
           AS SELECT * FROM emp@remote_db;

Materialized view log created.
```

- subquery materialized views

```
SQL> CREATE MATERIALIZED VIEW  mv_empdept
AS SELECT * FROM emp@remote_db e
WHERE EXISTS
    (SELECT * FROM dept@remote_db d
    WHERE e.dept_no = d.dept_no)
```

References:

[1] Donald P. Ballou, Harold L. Pazer, (1985) Modeling Data and Process Quality in Multi-Input, Multi-Output Information Systems. Management Science 31(2):150-162. http://dx.doi.org/10.1287/mnsc.31.2.150

[2] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. 2002. Data quality assessment. Commun. ACM 45, 4 (April 2002), 211–218.
DOI: https://doi.org/10.1145/505248.506010

[3] Lee, Y.W., Pipino, L.L., Funk, J.D., and Wang, R.Y., 2006. *Journey to data quality*. The MIT Press.

[4] Cai, L. and Zhu, Y., 2015. The challenges of data quality and data quality assessment in the big data era. *Data science journal*, *14*.