Assignment 1 (10% of total marks)

Due date: Saturday, 26 April 2020

Scope:

The tasks of this assignment cover **functional dependency and indexing**. The assignment covers the topics discussed in lecture 1, 2, and 3.

Assessment criteria:

Marks will be awarded for:

- Correct,
- Comprehensive, and
- Appropriate

application of the materials covered in this subject.

Assignment Specification:

Task 1 (5.0 marks)
Normalization

Consider the specifications of the sample database domains and the respective relational tables given below. For each one of the database domains listed below discover the respective sets of valid functional dependencies in the tables, and **identify its highest normal form**. Next **normalize the relational table to the required normal form specified in the question**. **Provide justification for each answer. A solution with no comprehensive justification scores no marks**.

a. ABC is an international company that own many buildings in a Science Park. In the current COVID-19 situation, company ABC has increase its cleaning activities. A group of cleaners are engaged to clean the company's buildings several times a day. Groups of 5 cleaners will be assigned to each building. Cleaners are not attached to any specific rooms within a building, that is, cleaners are able to clean any rooms. Once a cleaner has finished cleaning, the cleaner can move on to clean the next room schedule to be cleaned in a cleaning roster for a building. The room number is unique within a building, but not across the company.

   The company stores the information about the cleaning schedule for the building

   CLEANING-SCHEDULE(bldgNum, {roomNum}, sittingCapacity, {cleanerNum})

   into an un-normalized relational table as follow:

   CLEANING-SCHEDULE(bldgNum, roomNum, sittingCapacity, cleanerNum)

Decompose the relational table CLEANING-SCHEDULE into a minimal number of relational tables in 4NF.

(5.0 marks)

Sample solution include but not limited to the following:

- Identify all possible functional dependencies and the minimal superkeys of the relation:

  CLEANING-SCHEDULE(bldgNum, roomNum, sittingCapacity, cleanerNum)

  - "ABC is an international company that own many buildings in a Science Park."

    This statement indicates there are multiple building involve.
  - "A group of cleaners are engaged to clean the company's buildings several times a day."

    This statement indicates there are multiple cleaners involve.
  - "Groups of 5 cleaners will be assigned to each building."

    bldgNum ->> cleanerNum
  - "Cleaners are not attached to any specific rooms within a building"

    This statement indicates cleanerNum -/-> room (the functionality cleanerNum -> room is NOT valid.)
  - "The room number is unique within a building, but not across the company."

    bldgNum ->> roomNum

    We have established the following multi-value functional dependencies:
    bldgNum ->> cleanerNum
    bldgNum ->> roomNum

    Of the four attributes in the relational table CLEANING-SCHEDULE, the functional dependency of the attribute sittingCapacity is not established yet. The sittingCapacity is specific to each room. Hence, through common sense, knowing which room in a building, we will know the sitting capacity. The following functional dependency can be established.
    (bldgNum, roomNum) -> sittingCapacity

    (Note: Attributes in a relational table must be atomic. Hence, a normalized form of the relational would means the determinant and the multi-value attributes need to be composited to uniquely identify the multi-value functional dependency.)

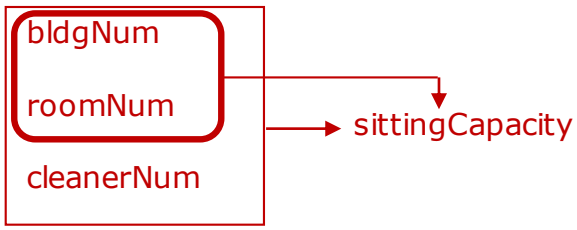
    The above two multi-value functional dependencies:
    (bldgNum, roomNum)
    (bldgNum, cleanerNum)

and one functional dependency:

(bldgNum, roomNum) -> sittingCapacity

can be combined to form the functional dependency through Augmentation axiom as follow:

(bldgNum, roomNum, cleanerNum) -> sittingCapacity



The minimal superkey of the relation consists of a composite attribute (bdlgNum, roomNum, cleanerNum) identifying the non-key attribute sittingCapacity.

From the functional dependency diagram, it is noted that partial-functional dependency exist, hence, the relational table CLEANNING-SCHEDULE has violated the requirement to be in 2NF. The highest normal form of the relational table is 1NF.

To normalize (decompose) the relational to 4NF, we need to normalize the table to 2NF. From 2NF table, then we normalize it to 3NF, BCNF and finally to 4NF as follow:

i. Normalize the table to 2NF.

Remove the partial-functional dependency from the CLEANING-SCHEDULE table and form a new relational table named ROOM-CAPACITY.

- ROOM-CAPACITY(bldgNum, roomNum, sittingCapacity)

  PK: (bldgNum, roomNum)

  The original CLEANING-SCHEDULE table will become:

  CLEANING-SCHEDULE(bldgNum, roomNum, cleanerNum)
  PK: (bldgNum, roomNum, cleanerNum)

FK: (bldgNum, roomNum) reference ROOM-CAPACITY(bldgNum, roomNum)

Checking the two relational tables revealed that there is no more partial-functional dependency, hence the two relational tables are now in 2NF.

ii. Normalize the tables to 3NF.

Testing the two relational table revealed that there is no transitive-functional dependency exists in the two relational table, and since the two relational tables are already in 2NF, hence, the two relational tables are now in 3NF.

iii. Normalize the tables to BCNF.

Testing the two relational table revealed that there is no non-trivial functional dependency exists. All the determinants of all the functional dependency established for each table are candidate keys. Hence the two relational table are in BCNF.

iv. Normalize the table to 4NF.

Testing the two relational table revealed that in CLEANING-SCHEDULE table, there exists multiple-multi-value functional dependency. The first multi-value functional dependency is bldgNum ->> roomNum and the second multi-value functional dependency is bldgNum ->> cleanerNum. There is one multi-value functional dependency in the ROOM-CAPACITY table, but no multiple multi-value, hence the relational table ROOM-CAPACITY has no violation of 4NF requirement, and hence the relational table ROOM-CAPACITY is now in 4NF.

To normalize the CLEANING-SCHEDULE to 4NF, we need to remove the multiple multi-value functional dependencies. Since there are two multi-value functional dependencies found in the table, the table need to be split into two separate tables each contains only one multi-value functional dependency as follow:

BUILDING-ROOM(bldgNum, roomNum)
PK: (bldgNum, roomNum)
FK: (bldgNum, roomNum) reference ROOM-CAPACITY(bldgNum, roomNum)

And

BUILDING-CLEANER(bldgNum, cleanerNum)
PK: (bldgNum, cleanerNum)

The normalization process is completed and the normalized 4NF tables are:

ROOM-CAPACITY(bldgNum, roomNum, sittingCapacity)
PK: (bldgNum, roomNum)

BUILDING-ROOM(bldgNum, roomNum)
PK: (bldgNum, roomNum)
FK: (bldgNum, roomNum) references ROOM-CAPACITY(bldgNum, roomNum)

BUILDING-CLEANER(bldgNum, cleanerNum)
PK: (bldgNum, cleanerNum)

Deliverables

Submit a pdf file consisting of the normalization process and explanation of the processes of the above relational tables. Name your pdf file as **solution1.pdf**.

Task 2 (5.0 marks)

Indexing

a. Consider the TPCHR benchmark database created through processing of CREATE TABLE statements include in the file TPCHR.

An index partIdx(p_name, p_type, p_retailprice) has been created over the relational table PART.

Construct SELECT statements that will use the index partIdx in the following ways:

i. Execution of the first SELECT statement must traverse the index vertically and it **MUST NOT** access a relational table PART.

(0.5 mark)

ii. Execution of the second SELECT statement must traverse the index vertically and then horizontally at the leaf level of the index and it **MUST NOT** access the relational table PART.

(0.5 mark)

iii. Execution of the third SELECT statement must traverse the leaf level of the index horizontally and it **MUST NOT** access the relational table PART.

(0.5 mark)

iv. Execution of the fourth SELECT statement must traverse the index

vertically and it **MUST** access the relational table PART.

(0.5 mark)

v. Execution of the fifth SELECT statement must traverse the index vertically and then horizontally and it **MUST** access a relational table PART.

(0.5 mark)

Use 'Explain plan for …' to generate the execution plan for each of the SELECT statements that you have proposed and created. Next use the 'SELECT * FROM TABLE(dbms_xplan.display) to display the generated execution plan to show that your proposed SELECT statements indeed perform according to the specification.

```
SQL> set feedback on
SQL> drop index partIdx;

Index dropped.

SQL>
SQL> create index partIdx on part(p_name, p_type, p_retailprice);

Index created.

SQL>
SQL> -- Question Task 1(i)
SQL> -- Generate the execution plan for the query
SQL> explain plan for
  2  select count(*)
  3  from part
  4  where p_name = 'Any Name';

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
-------------------------------------
Plan hash value: 1358544420

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

```
|   0 | SELECT STATEMENT  |         |     1 |    35 |     3   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |         |     1 |    35 |            |          |
|*  2 |   INDEX RANGE SCAN| PARTIDX |     1 |    35 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):

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

   2 - access("P_NAME"='Any Name')

14 rows selected.

SQL>
SQL> -- Question Task 1(ii)
SQL> -- Generate the execution plan for the query
SQL> explain plan for
  2  select count(*)
  3  from part
  4  where p_name > 'Some Name';

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------
Plan hash value: 1568896578

---------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |         |     1 |    35 |   165   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE      |         |     1 |    35 |            |          |
|*  2 |   INDEX FAST FULL SCAN| PARTIDX | 60000 |  2050K|   165   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------
```

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

   2 - filter("P_NAME">'Some Name')

14 rows selected.

SQL>
SQL> -- Question Task 1(iii)
SQL> -- Generate the executionn plan for the query
SQL> explain plan for
  2  select p_name
  3  from part;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
-----------------------------------
Plan hash value: 756900672

-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        | 60000 |  2050K|   165   (0)| 00:00:01 |
|   1 |  INDEX FAST FULL SCAN| PARTIDX | 60000 |  2050K|   165   (0)| 00:00:01 |
-------------------------------------------------------------------------------

8 rows selected.

SQL>
SQL> -- Question Task 1(iv)
SQL> -- Generate the execution plan for the query
SQL> explain plan for
  2  select *
  3  from part
  4  where p_name = 'Some Name'
  5  and p_type = 'A Type'
  6  and p_retailprice = 100;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------
Plan hash value: 371993851

--------------------------------------------------------------------------------
| Id  | Operation                          | Name    | Rows  | Bytes | Cost (%CPU)| Time
|
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |         |     1 |   121 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PART    |     1 |   121 |     4
(0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                 | PARTIDX |     1 |       |     3   (0)|
00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

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

   2 - access("P_NAME"='Some Name' AND "P_TYPE"='A Type' AND
"P_RETAILPRICE"=100)

14 rows selected.

SQL>
SQL> -- Question Task 1(v)
SQL> -- Generate the execution plan for the query
SQL> explain plan for
  2  select *
  3  from part
  4  where p_name = 'Some Part'
  5  and p_retailprice > 100;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------
Plan hash value: 371993851
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                          | Name    | Rows  | Bytes | Cost (%CPU)| Time
|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |         |     1 |   121 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PART    |     1 |   121 |     4
(0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                 | PARTIDX |     1 |       |     3   (0)|
00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):

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

   2 - access("P_NAME"='Some Part' AND "P_RETAILPRICE">100)
       filter("P_RETAILPRICE">100)

15 rows selected.

SQL>
SQL> spool off
```

b. Consider the TPCHR benchmark database created through processing of CREATE TABLE statements include in the file TPCHR.

Consider the following SELECT statements:

i.
```
SELECT   distinct o_total, o_orderDate
FROM     ORDERS
ORDER BY O_ORDERDATE;
```
(0.5 mark)

ii.
```
SELECT   *
FROM     PART
WHERE P_BRAND = 'GOLDEN BOLTS'
AND P_SIZE = 25;
```
(0.5 mark)

iii.
```
SELECT   c_custKey, c_name, c_address
FROM     CUSTOMER;
```
(0.5 mark)

iv.
```
SELECT   l_partKey, count(*)
FROM     LINEITEM
GROUP BY L_PARTKEY
HAVING COUNT(l_tax) > 2;
```

(0.5 mark)

v.
```
SELECT   *
FROM     LINEITEM
WHERE    l_quantity = 100
OR       l_shipMode = 'FAST';
```

(0.5 mark)

Find index that speeds up processing of the statements in the best possible way listed above, and create the index. You are only allowed to create **ONE** index per query. The best possible way means the database system will execute a query and uses the index proposed by you.

Use 'Explain plan for …' to generate the execution plan for each of the SELECT statements that you have proposed and created. Next use the 'SELECT * FROM TABLE(dbms_xplan.display) to display the generated execution plan to show that your proposed SELECT statements indeed perform according to the specification. If you think that no index can help to speed up the query processing, justify and give explanation.

Sample solution included but not limited to the following:

For Question A1T2bv, there is no index can be used to improve/speed up the processing of the query because the compound condition specified in the 'WHERE' clause include an 'OR' operation/operator. The 'OR' operator implies that the condition may NOT necessarily apply to a record being indexed; that is, a record that has l_quantity = 100 may not mean there is NO other record in the database with l_shipMode = 'FAST". Similarly, a record that has l_shipMode = 'FAST' does not mean there is NO other record in the database with l_quantity = 100. The records that meet the two conditions can be another in the database and no index can uniquely identify these record and hence, no index can be used. The query optimizer will perform a full-table scan because the records/rows meeting the condition can be anywhere in the table (No index can help speed up the query process).

```
SQL> set feedback on
SQL> set line 132
SQL>
SQL> -- Soltuion to Question A1T2(bi)
SQL> -- Generate the execution plan for the SELECT statement
SQL> explain plan for
  2  SELECT  distinct o_totalPrice, o_orderDate
  3  FROM    ORDERS
```

```
  4  ORDER BY o_orderDate;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------
Plan hash value: 4122230513

-------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |   450K|  6152K|       |  6395   (1)| 00:00:01 |
|   1 |  SORT UNIQUE       |         |   450K|  6152K|    10M|  4172   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| ORDERS |   450K|  6152K|       |  1949   (1)| 00:00:01
|
-------------------------------------------------------------------------------------

9 rows selected.

SQL>
SQL> -- Create the required index to speed up processing
SQL> create index A1T2biIdx on ORDERS(o_orderDate, o_totalPrice);

Index created.

SQL>
SQL> -- Generate the execution plan for the SELECT statement after the creation of
index
SQL> explain plan for
  2  SELECT  distinct o_totalPrice, o_orderDate
  3  FROM    ORDERS
  4  ORDER BY o_orderDate;

Explained.

SQL>
SQL> -- Display the generated execution plan after the creation of index
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------
Plan hash value: 2602696672
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time
|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |   449K|  6152K|       |  4874   (1)| 00:00:01
|
|   1 |  SORT UNIQUE        |           |   449K|  6152K|   10M|  2651   (1)| 00:00:01 |
|   2 |   INDEX FAST FULL SCAN| A1T2BIIDX |   450K|  6152K|       |  428   (1)|
00:00:01 |
-------------------------------------------------------------------------------------

9 rows selected.

SQL>
SQL> -- Soltuion to Question A1T2(bii)
SQL> -- Generate the execution plan for the SELECT statement
SQL> explain plan for
  2  SELECT  *
  3  FROM    PART
  4  WHERE   P_BRAND = 'GOLDEN BOLTS'
  5  AND     P_SIZE = 25;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------
Plan hash value: 673417232

----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |   121 |   291   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PART |     1 |   121 |   291   (1)| 00:00:01 |
----------------------------------------------------------------------------

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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
--------------------------------
```

```
   1 - filter("P_SIZE"=25 AND "P_BRAND"='GOLDEN BOLTS')

13 rows selected.

SQL>
SQL> -- Create the required index to speed up processing
SQL> create index A1T2biiIdx on part(p_brand, p_size);

Index created.

SQL>
SQL> -- Generate the execution plan for the SELECT statement after the creation of
index
SQL> explain plan for
  2  SELECT  *
  3  FROM    PART
  4  WHERE   P_BRAND = 'GOLDEN BOLTS'
  5  AND     P_SIZE = 25;

Explained.

SQL>
SQL> -- Display the generated execution plan after the creation of index
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------
-----------------------------------
Plan hash value: 3198386858

--------------------------------------------------------------------------------------------
--
| Id  | Operation                           | Name       | Rows  | Bytes | Cost (%CPU)| Time
|
--------------------------------------------------------------------------------------------
--
|   0 | SELECT STATEMENT                    |            |     1 |   121 |     2   (0)| 00:00:01
|
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PART       |     1 |   121 |     2
(0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | A1T2BIIIDX |     1 |       |     1   (0)|
00:00:01 |
--------------------------------------------------------------------------------------------
--

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
```

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

   2 - access("P_BRAND"='GOLDEN BOLTS' AND "P_SIZE"=25)

14 rows selected.

SQL>
SQL>
SQL> -- Soltuion to Question A1T2(biii)
SQL> -- Generate the execution plan for the SELECT statement
SQL> explain plan for
  2  SELECT  c_custkey, c_name, c_address
  3  FROM    CUSTOMER;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
---------------------------------
Plan hash value: 2844954298

-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 45000 |  2241K|   282   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| CUSTOMER | 45000 |  2241K|   282   (0)| 00:00:01 |
-------------------------------------------------------------------------------

8 rows selected.

SQL>
SQL> -- Create the required index to speed up processing
SQL> create index A1T2biiiIdx on CUSTOMER(c_custKey, c_name, c_address);

Index created.

SQL>
SQL> -- Generate the execution plan for the SELECT statement after creation of index
SQL> explain plan for
  2  SELECT  c_custkey, c_name, c_address
  3  FROM    CUSTOMER;
```

```
Explained.

SQL>
SQL> -- Display the generated execution plan after creation of index
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
----------------------------------
Plan hash value: 160150316

-------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 45000 |  2241K|   105   (0)| 00:00:01 |
|   1 |  INDEX FAST FULL SCAN| A1T2BIIIIDX | 45000 |  2241K|   105   (0)| 00:00:01
|
-------------------------------------------------------------------------------

8 rows selected.

SQL>
SQL> -- Soltuion to Question A1T2(biv)
SQL> -- Generate the execution plan for the SELECT statement
SQL> explain plan for
  2  SELECT  l_partKey, count(*)
  3  FROM    LINEITEM
  4  GROUP BY l_partKey
  5  HAVING COUNT(l_partKey) > 2;

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
----------------------------------
Plan hash value: 2487493660

-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |        |  3023 | 15115 |  8821   (1)| 00:00:01 |
|*  1 |  FILTER             |        |       |       |            |          |
|   2 |   HASH GROUP BY     |        |  3023 | 15115 |  8821   (1)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| LINEITEM |  1800K|  8789K|  8775   (1)| 00:00:01 |
```

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


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

   1 - filter(COUNT(*)>2)

15 rows selected.

SQL>
SQL> -- Create the required index to speed up processing
SQL> create index A1T2bivIdx on LINEITEM(l_partKey);

Index created.

SQL>
SQL> -- Generate the execution plan for the SELECT statement after the creation of
index
SQL> explain plan for
  2  SELECT  l_partKey, count(*)
  3  FROM    LINEITEM
  4  GROUP BY l_partKey
  5  HAVING COUNT(l_partKey) > 2;

Explained.

SQL>
SQL> -- Display the generated execution plan after the creation of index
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
-----------------------------------
Plan hash value: 2619016170

-------------------------------------------------------------------------------------
| Id  | Operation             | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |            |  3023 | 15115 |  1130   (5)| 00:00:01 |
|*  1 |  FILTER               |            |       |       |            |          |
|   2 |   HASH GROUP BY       |            |  3023 | 15115 |  1130   (5)| 00:00:01 |
|   3 |    INDEX FAST FULL SCAN| A1T2BIVIDX |  1800K|  8789K|  1084   (1)|
00:00:01 |
-------------------------------------------------------------------------------------
```

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

   1 - filter(COUNT(*)>2)

15 rows selected.

SQL>
SQL>
SQL> -- Soltuion to Question A1T2(bv)
SQL> -- Generate the execution plan for the SELECT statement
SQL> explain plan for
  2  SELECT  *
  3  FROM    LINEITEM
  4  WHERE   l_quantity = 100
  5  OR      l_shipMode = 'FAST';

Explained.

SQL>
SQL> -- Display the generated execution plan
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------
Plan hash value: 98068815

-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |   128K|    15M|  8802   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| LINEITEM |   128K|    15M|  8802   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

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

   1 - filter("L_SHIPMODE"='FAST' OR "L_QUANTITY"=100)
```

```
13 rows selected.

SQL>
SQL> -- Create the required index to speed up processing
SQL> create index A1T2bvIdx on LINEITEM(l_quantity, l_shipMode);

Index created.

SQL>
SQL> -- Generate the execution plan for the SELECT statement after the creation of
the index
SQL> explain plan for
  2  SELECT  *
  3  FROM    LINEITEM
  4  WHERE   l_quantity = 100
  5  OR      l_shipMode = 'FAST';

Explained.

SQL>
SQL> -- Display the generated execution plan after the creation of the index
SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------
----------------------------------
Plan hash value: 98068815

-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |   128K|    15M|  8802   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| LINEITEM |   128K|    15M|  8802   (1)| 00:00:01 |
-------------------------------------------------------------------------------

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

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

   1 - filter("L_SHIPMODE"='FAST' OR "L_QUANTITY"=100)

13 rows selected.

SQL>
```

```
SQL> -- Drop all created indexes after completion of assignment
SQL> drop index A1T2biIdx;

Index dropped.

SQL> drop index A1T2biiIdx;

Index dropped.

SQL> drop index A1T2biiiIdx;

Index dropped.

SQL> drop index A1T2bivIdx;

Index dropped.

SQL> drop index A1T2bvIdx;

Index dropped.

SQL>
SQL> -- Done
SQL> set echo off
```

Deliverables

A file pdf file with the SELECT statements created for Part 2(a) above, and the index proposed for each statement in Part 2(b). You may put your solution for Part 2(a) and Part 2(b) together in **one** pdf file named as **solution2.pdf**.

Submissions

This assignment is due by 9:00 pm (21:00 hours) Saturday, 26 April 2020, **Singapore time**.

Submit the files **solution1.pdf** and **solution2.pdf** through Moodle in the following way:

1) Zip all the files (Solution1.pdf and solution2.pdf into one zipped folder.)
2) Access Moodle at **http://moodle.uowplatform.edu.au/**
3) To login use a Login link located in the right upper corner the Web page or in the middle of the bottom of the Web page
4) When successfully logged in, select a site CSCI235 (SP220) Database Systems
5) Scroll down to a section Submissions of Assignments

6) Click at Submit your Assignment 1 here link.
7) Click at a button Add Submission
8) Move the zipped file created in Step 1 above into an area provided in Moodle. You can drag and drop files here to add them. You can also use a link *Add…*
9) Click at a button Save changes,
10) Click at check box to confirm authorship of a submission,
11) When you are satisfied, remember to click at a button Submit assignment.

A policy regarding late submissions is included in the subject

outline. Only one submission per student is accepted.

Assignment 1 is an individual assignment and it is expected that all its tasks will be solved individually without any cooperation with the other students. Plagiarism is treated seriously. Students involved will likely receive zero. If you have any doubts, questions, etc. please consult your lecturer or tutor during lab classes or over e-mail.

End of specification