Practical No.1

Aim: To implement Basic SQL commands and to access & modify Data using SQL. Create and populate database using Data Definition Language (DDL) and DML Commands

Theory:
Implementing basic SQL commands involves utilizing Data Definition Language (DDL) and Data Manipulation Language (DML) to create, access, and modify a database. DDL commands like CREATE DATABASE and CREATE TABLE establish the database's structure. For instance, "CREATE DATABASE Library;" generates a new database named "Library". Tables are designed using CREATE TABLE, defining columns like BookID, Title, and AuthorID. Foreign keys ensure data consistency and relationships.

With the structure in place, DML commands enable data interactions. INSERT statements add data; "INSERT INTO Books (BookID, Title, AuthorID) VALUES (1, 'SQL Basics', 1);" populates the "Books" table. SELECT queries retrieve data; "SELECT * FROM Books;" fetches all book records. UPDATE statements modify data; "UPDATE Books SET Title = 'SQL Fundamentals' WHERE BookID = 1;" changes the book's title.

INSERT introduces new records, as in "INSERT INTO Books (BookID, Title, AuthorID) VALUES (2, 'Database Design', 2);". DELETE commands remove data; "DELETE FROM Books WHERE BookID = 2;" deletes the book with ID 2. However, cautious usage is advised to prevent accidental data loss.

In conclusion, SQL proficiency is essential for managing data. DDL creates databases and tables, while DML provides the means to insert, retrieve, modify, and delete data. Balancing these actions while prioritizing data integrity and security ensures efficient and safe data management.

1)Change the price of „plate“ from 1500 to 2000.

```
mysql> /*202203103510124*/
mysql> UPDATE PRODUCT
    -> SET PRICE = 2000
    -> WHERE DESCRIPTION = 'PLATE';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from product;
+------------+----------------+----------+-------------+------------------+-----------------+
| PRODUCT_NO | DESCRIPTION    | PRICE    | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+----------------+----------+-------------+------------------+-----------------+
|        120 | REDUCER        |  1200.00 |        1005 |                5 |               6 |
|        121 | PLATE          |  2000.00 |        1004 |                3 |               1 |
|        122 | HANDLE         |   700.00 |        1003 |                2 |               4 |
|        124 | WIDGET REMOVER |   900.00 |        1005 |                4 |               2 |
|        136 | SIZE WIDGET    |  1000.00 |        1001 |                1 |               5 |
|        137 | SIZE WIDGET    | 15000.00 |        1002 |                2 |              16 |
+------------+----------------+----------+-------------+------------------+-----------------+
6 rows in set (0.00 sec)
```

2) Modify the credit limit to 8000 for those customers who live in „grange“.

```
mysql> /*202203103510124*/
mysql> UPDATE CUSTOMER
    -> SET CREDIT_LIMIT = 8000
    -> WHERE ADDRESS = 'GRANGE';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from customer;
+-------------+--------------+---------------+----------+--------------+
| CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
|          10 | GARRY SMITH  | BRIXTON       |        6 |      1000.00 |
|          20 | PATEL        | GRANGE        |        1 |      8000.00 |
|          30 | DRAKE        | BRIXTON       |        4 |      7000.00 |
|          40 | BOB SMITH    | LONDON        |        2 |     10000.00 |
|          50 | JAMES        | GRANGE        |        3 |      8000.00 |
|          60 | NORTON       | SAN FRANCISCO |        5 |     17000.00 |
|          70 | JOHN MICHAEL | EUROPE        |       16 |      8000.00 |
+-------------+--------------+---------------+----------+--------------+
7 rows in set (0.00 sec)
```

3) Change the size of the customer address to 30.

```
mysql> /*202203103510124*/
mysql> ALTER TABLE CUSTOMER
    -> MODIFY ADDRESS VARCHAR(30);
Query OK, 7 rows affected (0.02 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

4) Create a table cust1 with the attributes and formats

Customer_no number (10)

Name varchar2 (20)

Address varchar2 (20)

Rep_no number (10)

```
mysql> /*202203103510124*/
mysql> CREATE TABLE CUST1 (
    ->     CUSTOMER_NO int(10),
    ->     NAME VARCHAR(20),
    ->     ADDRESS VARCHAR(20),
    ->     REP_NO int(10)
    -> );
Query OK, 0 rows affected, 2 warnings (0.01 sec)
```

5) Add a new field email id in the cust1 table.

```
mysql> /*202203103510124*/
mysql> ALTER TABLE CUST1
    -> ADD EMAIL_ID VARCHAR(50);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

6) Display the structure of the cust1 table.

```
mysql> /*202203103510124*/
mysql> DESC CUST1;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| CUSTOMER_NO | int         | YES  |     | NULL    |       |
| NAME        | varchar(20) | YES  |     | NULL    |       |
| ADDRESS     | varchar(20) | YES  |     | NULL    |       |
| REP_NO      | int         | YES  |     | NULL    |       |
| EMAIL_ID    | varchar(50) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

7) Display the content of the cust1 table.

```
mysql> /*202203103510124*/
mysql> SELECT * FROM CUST1;
Empty set (0.00 sec)
```

8) Delete details of customer no 2 from cust1 table.

```
mysql> /*202203103510124*/
mysql> DELETE FROM CUST1
    -> WHERE CUSTOMER_NO = 2;
Query OK, 0 rows affected (0.00 sec)
```

9) Delete email id field from cust1 table.

```
mysql> /*202203103510124*/
mysql> ALTER TABLE CUST1
    -> DROP COLUMN EMAIL_ID;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10) Delete all the data rows from cust1 and look at the contents again.

```
mysql> /*202203103510124*/
mysql> DELETE FROM CUST1;
Query OK, 0 rows affected (0.00 sec)
```

11) Delete the table cust1 and then try to look at its contents again.

```
mysql> /*202203103510124*/
mysql> DROP TABLE CUST1;
Query OK, 0 rows affected (0.01 sec)
```

12) List the customer numbers (customer_no) and names (name) of all customers.

```
mysql> /*202203103510124*/
mysql> SELECT CUSTOMER_NO, NAME
    -> FROM CUSTOMER;
+-------------+--------------+
| CUSTOMER_NO | NAME         |
+-------------+--------------+
|          10 | GARRY SMITH  |
|          20 | PATEL        |
|          30 | DRAKE        |
|          40 | BOB SMITH    |
|          50 | JAMES        |
|          60 | NORTON       |
|          70 | JOHN MICHAEL |
+-------------+--------------+
7 rows in set (0.00 sec)
```

13) List all details of the product with a product number (product_no) of 121 and 136.(use Or).

```
mysql> /*202203103510124*/
mysql> SELECT *
    -> FROM PRODUCT
    -> WHERE PRODUCT_NO IN (121, 136);
+------------+-------------+---------+-------------+------------------+-----------------+
| PRODUCT_NO | DESCRIPTION | PRICE   | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+-------------+---------+-------------+------------------+-----------------+
|        121 | PLATE       | 2000.00 |        1004 |                3 |               1 |
|        136 | SIZE WIDGET | 1000.00 |        1001 |                1 |               5 |
+------------+-------------+---------+-------------+------------------+-----------------+
2 rows in set (0.00 sec)
```

14) List all details of depots with rep 5 as their rep(rep_no).

```
mysql> /*202203103510124*/
mysql> SELECT *
    -> FROM DEPOT
    -> WHERE REP_NO = 5;
+----------+----------+---------+--------+
| DEPOT_NO | LOCATION | ADDRESS | REP_NO |
+----------+----------+---------+--------+
|        5 | WALES UK | 5       |      5 |
+----------+----------+---------+--------+
1 row in set (0.00 sec)
```

15) List the product number (product_no) and description only of all products from supplier number 1005 (supplier_no).

```
mysql> /*202203103510124*/
mysql> SELECT PRODUCT_NO, DESCRIPTION
    -> FROM PRODUCT
    -> WHERE SUPPLIER_NO = 1005;
+------------+----------------+
| PRODUCT_NO | DESCRIPTION    |
+------------+----------------+
|        120 | REDUCER        |
|        124 | WIDGET REMOVER |
+------------+----------------+
2 rows in set (0.00 sec)
```

16)List the sales rep number (rep_no), depot number and address for depots located at NORTH and address is UK.

```
mysql> /*202203103510124*/
mysql> SELECT REP_NO, DEPOT_NO, ADDRESS
    -> FROM DEPOT
    -> WHERE LOCATION = 'NORTH' AND ADDRESS = 'UK';
Empty set (0.00 sec)
```

Conclusion:

Mastering basic SQL commands enables creating, accessing, and modifying databases. DDL crafts the structure, DML handles data. Balancing efficacy with data integrity and security ensures successful management.

Practical No.2

Aim: The aim of this practical exercise is to develop practical skills in querying a relational database. Through this practical we will gain hands-on experience in retrieving specific information from a database using SQL query.

Theory:

The theory behind this practical exercise is to understand and apply Structured Query Language (SQL) for data retrieval. Participants will learn how to write SQL queries to extract specific data from relational databases, focusing on SELECT statements, filtering conditions, and pattern matching using SQL.

1) List the customer numbers (customer_no) and names (name) of all customers.

```
ysql> /* 202203103510124*/
ysql> SELECT * FROM CUSTOMER;
+-------------+--------------+---------------+----------+--------------+
 CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
          10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
          20 | PATEL        | GRANGE        |        1 |         8000 |
          30 | DRAKE        | BRIXTON       |        4 |         7000 |
          40 | BOB SMITH    | LONDON        |        2 |        10000 |
          50 | JAMES        | GRANGE        |        3 |         8000 |
          60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
          70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
+-------------+--------------+---------------+----------+--------------+
 rows in set (0.00 sec)

ysql> SELECT CUSTOMER_NO , NAME FROM CUSTOMER;
+-------------+--------------+
 CUSTOMER_NO | NAME         |
+-------------+--------------+
          10 | GARRY SMITH  |
          20 | PATEL        |
          30 | DRAKE        |
          40 | BOB SMITH    |
          50 | JAMES        |
          60 | NORTON       |
          70 | JOHN MICHAEL |
+-------------+--------------+
 rows in set (0.00 sec)
```

2) List all details of the product with a product number (product_no) of 121 and 136.

```
mysql> /* 202203103510124 */
mysql> SELECT * FROM PRODUCT;
+------------+----------------+-------+-------------+------------------+-----------------+
| PRODUCT_NO | DESCRIPTION    | PRICE | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+----------------+-------+-------------+------------------+-----------------+
|        120 | REDUCER        |  1200 |        1005 |                5 |               6 |
|        121 | PLATE          |  2000 |        1004 |                3 |               1 |
|        122 | HANDEL         |   700 |        1003 |                2 |               4 |
|        124 | WIDGET REMOVER |   900 |        1005 |                4 |               2 |
|        136 | SIZE WIDGET    |  1000 |        1001 |                1 |               5 |
|        137 | SIZE WIDGET    | 15000 |        1002 |                2 |              16 |
+------------+----------------+-------+-------------+------------------+-----------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM PRODUCT WHERE PRODUCT_NO > 120 AND PRODUCT_NO < 137;
+------------+----------------+-------+-------------+------------------+-----------------+
| PRODUCT_NO | DESCRIPTION    | PRICE | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+----------------+-------+-------------+------------------+-----------------+
|        121 | PLATE          |  2000 |        1004 |                3 |               1 |
|        122 | HANDEL         |   700 |        1003 |                2 |               4 |
|        124 | WIDGET REMOVER |   900 |        1005 |                4 |               2 |
|        136 | SIZE WIDGET    |  1000 |        1001 |                1 |               5 |
+------------+----------------+-------+-------------+------------------+-----------------+
4 rows in set (0.02 sec)
```

3) List all details of depots with rep 5 as their rep(rep_no).

```
mysql> /*202203103510124 */
mysql> SELECT * FROM DEPOT;
+----------+-------------+---------+--------+
| DEPOT_NO | LOCATION    | ADDRESS | REP_NO |
+----------+-------------+---------+--------+
|        1 | NORTH       | UK      |      1 |
|        2 | SOUTH       | USA     |      2 |
|        3 | LONDON WEST | USA     |      3 |
|        4 | EAST        | NZ      |      4 |
|        5 | WALES       | UK      |      5 |
|        6 | NORTH       | KENYA   |      6 |
|       16 | SOUTH       | UK      |      2 |
+----------+-------------+---------+--------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM DEPOT WHERE REP_NO=5 ;
+----------+----------+---------+--------+
| DEPOT_NO | LOCATION | ADDRESS | REP_NO |
+----------+----------+---------+--------+
|        5 | WALES    | UK      |      5 |
+----------+----------+---------+--------+
1 row in set (0.00 sec)
```

4) List the product number (product_no) and description only of all products from supplier number 1005 (supplier_no).

```
mysql> /*202203103510124 */
mysql> SELECT * FROM PRODUCT;
+------------+----------------+-------+-------------+------------------+-----------------+
| PRODUCT_NO | DESCRIPTION    | PRICE | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+----------------+-------+-------------+------------------+-----------------+
|        120 | REDUCER        |  1200 |        1005 |                5 |               6 |
|        121 | PLATE          |  2000 |        1004 |                3 |               1 |
|        122 | HANDEL         |   700 |        1003 |                2 |               4 |
|        124 | WIDGET REMOVER |   900 |        1005 |                4 |               2 |
|        136 | SIZE WIDGET    |  1000 |        1001 |                1 |               5 |
|        137 | SIZE WIDGET    | 15000 |        1002 |                2 |              16 |
+------------+----------------+-------+-------------+------------------+-----------------+
6 rows in set (0.00 sec)

mysql> SELECT PRODUCT_NO, DESCRIPTION FROM PRODUCT WHERE SUPPLIER_NO = 1005;
+------------+----------------+
| PRODUCT_NO | DESCRIPTION    |
+------------+----------------+
|        120 | REDUCER        |
|        124 | WIDGET REMOVER |
+------------+----------------+
2 rows in set (0.00 sec)
```

5) List all details for all customers with names (name) starting from ga followed by 2 character followed by y followed by anything.

```
> /* 202203103510124 */
> SELECT * FROM CUSTOMER;
---------+--------------+---------------+----------+--------------+
TOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
---------+--------------+---------------+----------+--------------+
      10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
      20 | PATEL        | GRANGE        |        1 |         8000 |
      30 | DRAKE        | BRIXTON       |        4 |         7000 |
      40 | BOB SMITH    | LONDON        |        2 |        10000 |
      50 | JAMES        | GRANGE        |        3 |         8000 |
      60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
      70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
---------+--------------+---------------+----------+--------------+
s in set (0.00 sec)

> SELECT * FROM CUSTOMER WHERE NAME LIKE 'GA__Y%';
---------+-------------+---------+----------+--------------+
TOMER_NO | NAME        | ADDRESS | DEPOT_NO | CREDIT_LIMIT |
---------+-------------+---------+----------+--------------+
      10 | GARRY SMITH | BRIXTON |        6 |         1000 |
---------+-------------+---------+----------+--------------+
 in set (0.00 sec)
```

6) List all details for all orders with date_placed from 01-jan-1993 to 31-mar-1996.

```
mysql> /* 202203103510124 */
mysql> SELECT * FROM CORDER;
+-----------+-------------+-------------+----------------+
| CORDER_NO | CUSTOMER_NO | DATE_PLACED | DATE_DELIVERED |
+-----------+-------------+-------------+----------------+
|       200 |          20 | 01-JAN-1993 | 04-JAN-1993    |
|       201 |          40 | 17-JAN-1993 | 20-JAN-1993    |
|       202 |          20 | 01-JAN-1993 | 04-JAN-1993    |
|       203 |          30 | 02-FEB-1995 | 05-FEB-1995    |
|       204 |          10 | 13-MAR-1996 | 16-MAR-1996    |
|       205 |          70 | 31-JAN-1993 | 03-FEB-1993    |
|       206 |          40 | 01-JAN-1994 | 04-JAN-1994    |
|       207 |          20 | 02-AUG-1994 | 05-AUG-1994    |
+-----------+-------------+-------------+----------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM CORDER WHERE DATE_PLACED BETWEEN '01-JAN-1993' AND '13-MAR-1996';
+-----------+-------------+-------------+----------------+
| CORDER_NO | CUSTOMER_NO | DATE_PLACED | DATE_DELIVERED |
+-----------+-------------+-------------+----------------+
|       200 |          20 | 01-JAN-1993 | 04-JAN-1993    |
|       202 |          20 | 01-JAN-1993 | 04-JAN-1993    |
|       203 |          30 | 02-FEB-1995 | 05-FEB-1995    |
|       204 |          10 | 13-MAR-1996 | 16-MAR-1996    |
|       206 |          40 | 01-JAN-1994 | 04-JAN-1994    |
|       207 |          20 | 02-AUG-1994 | 05-AUG-1994    |
+-----------+-------------+-------------+----------------+
6 rows in set (0.00 sec)
```

7) List the sales rep number (rep_no), depot number and address for depots located at NORTH and address is UK.

```
mysql> /* 202203103510124 */
mysql> SELECT * FROM DEPOT;
+----------+-------------+---------+--------+
| DEPOT_NO | LOCATION    | ADDRESS | REP_NO |
+----------+-------------+---------+--------+
|        1 | NORTH       | UK      |      1 |
|        2 | SOUTH       | USA     |      2 |
|        3 | LONDON WEST | USA     |      3 |
|        4 | EAST        | NZ      |      4 |
|        5 | WALES       | UK      |      5 |
|        6 | NORTH       | KENYA   |      6 |
|       16 | SOUTH       | UK      |      2 |
+----------+-------------+---------+--------+
7 rows in set (0.00 sec)

mysql> SELECT REP_NO , DEPOT_NO ,ADDRESS FROM DEPOT WHERE LOCATION='NORTH' AND ADDRESS = 'UK';
+--------+----------+---------+
| REP_NO | DEPOT_NO | ADDRESS |
+--------+----------+---------+
|      1 |        1 | UK      |
+--------+----------+---------+
1 row in set (0.00 sec)
```

8) Give the total number of items (quantity) in stock in all depots.

```
mysql> /* 202203103510124  */
mysql> SELECT * FROM STOCK;
+----------+------------+----------+------+--------+
| DEPOT_NO | PRODUCT_NO | QUANTITY | RACK | BIN_NO |
+----------+------------+----------+------+--------+
|        1 |        120 |       50 |    1 |      1 |
|        2 |        137 |      100 |   10 |      2 |
|        3 |        136 |       40 |    2 |      3 |
|        4 |        120 |       60 |    7 |      1 |
|        5 |        121 |       90 |    5 |      4 |
|        6 |        124 |      120 |    4 |      7 |
|       16 |        122 |       80 |   10 |      8 |
+----------+------------+----------+------+--------+
7 rows in set (0.00 sec)

mysql> SELECT SUM(QUANTITY) FROM STOCK;
+---------------+
| SUM(QUANTITY) |
+---------------+
|           540 |
+---------------+
1 row in set (0.03 sec)
```

9) Give the total number of items (order line quantity) which have been ordered with corder_no 200.

```
mysql> /* 202203103510124  */
mysql> SELECT * FROM OLINE;
+-----------+------------+----------+
| CORDER_NO | PRODUCT_NO | QUANTITY |
+-----------+------------+----------+
|       200 |        120 |        5 |
|       201 |        121 |       10 |
|       202 |        120 |        5 |
|       203 |        122 |       20 |
|       204 |        136 |       30 |
|       205 |        124 |       15 |
|       206 |        136 |       30 |
+-----------+------------+----------+
7 rows in set (0.00 sec)

mysql> SELECT SUM(QUANTITY) AS TOTAL_ITEMS_ORDERED FROM OLINE WHERE CORDER_NO = 200;
+---------------------+
| TOTAL_ITEMS_ORDERED |
+---------------------+
|                   5 |
+---------------------+
1 row in set (0.00 sec)
```

10) List product descriptions in reverse alphabetical order.

```
mysql> /* 202203103510124  */
mysql> SELECT * FROM PRODUCT;
+------------+----------------+-------+-------------+------------------+------------------+
| PRODUCT_NO | DESCRIPTION    | PRICE | SUPPLIER_NO | MARKETING_REP_NO | SUPPLY_DEPOT_NO |
+------------+----------------+-------+-------------+------------------+------------------+
|        120 | REDUCER        |  1200 |        1005 |                5 |                6 |
|        121 | PLATE          |  2000 |        1004 |                3 |                1 |
|        122 | HANDEL         |   700 |        1003 |                2 |                4 |
|        124 | WIDGET REMOVER |   900 |        1005 |                4 |                2 |
|        136 | SIZE WIDGET    |  1000 |        1001 |                1 |                5 |
|        137 | SIZE WIDGET    | 15000 |        1002 |                2 |               16 |
+------------+----------------+-------+-------------+------------------+------------------+
6 rows in set (0.00 sec)

mysql> SELECT DESCRIPTION FROM PRODUCT ORDER BY DESCRIPTION DESC;
+----------------+
| DESCRIPTION    |
+----------------+
| WIDGET REMOVER |
| SIZE WIDGET    |
| SIZE WIDGET    |
| REDUCER        |
| PLATE          |
| HANDEL         |
+----------------+
6 rows in set (0.00 sec)
```

11) List the customer details with the name ending with N.

```
mysql> /* 202203103510124 */
mysql> SELECT * FROM CUSTOMER;
+-------------+--------------+---------------+----------+--------------+
| CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
|          10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
|          20 | PATEL        | GRANGE        |        1 |         8000 |
|          30 | DRAKE        | BRIXTON       |        4 |         7000 |
|          40 | BOB SMITH    | LONDON        |        2 |        10000 |
|          50 | JAMES        | GRANGE        |        3 |         8000 |
|          60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
|          70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
+-------------+--------------+---------------+----------+--------------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM CUSTOMER WHERE NAME LIKE '%N';
+-------------+--------+---------------+----------+--------------+
| CUSTOMER_NO | NAME   | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------+---------------+----------+--------------+
|          60 | NORTON | SAN FRANSISCO |        5 |        17000 |
+-------------+--------+---------------+----------+--------------+
1 row in set (0.00 sec)
```

12) List the customers details with a CustomerName that have “r” in the second position:

```
mysql> /*202203103510124 */
mysql> SELECT * FROM CUSTOMER;
+-------------+--------------+---------------+----------+--------------+
| CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
|          10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
|          20 | PATEL        | GRANGE        |        1 |         8000 |
|          30 | DRAKE        | BRIXTON       |        4 |         7000 |
|          40 | BOB SMITH    | LONDON        |        2 |        10000 |
|          50 | JAMES        | GRANGE        |        3 |         8000 |
|          60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
|          70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
+-------------+--------------+---------------+----------+--------------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM CUSTOMER WHERE SUBSTRING(NAME,2,1)='R';
+-------------+-------+---------+----------+--------------+
| CUSTOMER_NO | NAME  | ADDRESS | DEPOT_NO | CREDIT_LIMIT |
+-------------+-------+---------+----------+--------------+
|          30 | DRAKE | BRIXTON |        4 |         7000 |
+-------------+-------+---------+----------+--------------+
1 row in set (0.00 sec)
```

13) List the customers with a CustomerName that starts with “N” and is at least 4 characters in length.

```
mysql> /* 202203103510124  */
mysql> SELECT * FROM CUSTOMER;
+-------------+--------------+---------------+----------+--------------+
| CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
|          10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
|          20 | PATEL        | GRANGE        |        1 |         8000 |
|          30 | DRAKE        | BRIXTON       |        4 |         7000 |
|          40 | BOB SMITH    | LONDON        |        2 |        10000 |
|          50 | JAMES        | GRANGE        |        3 |         8000 |
|          60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
|          70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
+-------------+--------------+---------------+----------+--------------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM CUSTOMER WHERE NAME LIKE '%N' AND LENGTH(NAME) >= 4;
+-------------+--------+---------------+----------+--------------+
| CUSTOMER_NO | NAME   | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------+---------------+----------+--------------+
|          60 | NORTON | SAN FRANSISCO |        5 |        17000 |
+-------------+--------+---------------+----------+--------------+
1 row in set (0.00 sec)
```

14) Find all suppliers with a City containing the pattern “ny”.

```
mysql> /* 202203103510124  */
mysql> SELECT * FROM SUPPLIER;
+-------------+-----------+-----------+
| SUPPLIER_NO | NAME      | ADDRESS   |
+-------------+-----------+-----------+
|        1001 | MICHAEL   | BASILDON  |
|        1002 | RINGWORLD | GERMANY   |
|        1003 | BABYLON   | LONDON    |
|        1004 | JOHN      | BASILDON  |
|        1005 | SMITH     | GERMANY   |
+-------------+-----------+-----------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM SUPPLIER WHERE UPPER(ADDRESS) LIKE '%NY';
+-------------+-----------+---------+
| SUPPLIER_NO | NAME      | ADDRESS |
+-------------+-----------+---------+
|        1002 | RINGWORLD | GERMANY |
|        1005 | SMITH     | GERMANY |
+-------------+-----------+---------+
2 rows in set (0.00 sec)
```

15) selects all customers with a City starting with “L”, followed by any character, followed by “n”, followed by 2 character, followed by “n”:

```
mysql> /* 202203103510124 */
mysql> SELECT * FROM CUSTOMER;
+-------------+--------------+---------------+----------+--------------+
| CUSTOMER_NO | NAME         | ADDRESS       | DEPOT_NO | CREDIT_LIMIT |
+-------------+--------------+---------------+----------+--------------+
|          10 | GARRY SMITH  | BRIXTON       |        6 |         1000 |
|          20 | PATEL        | GRANGE        |        1 |         8000 |
|          30 | DRAKE        | BRIXTON       |        4 |         7000 |
|          40 | BOB SMITH    | LONDON        |        2 |        10000 |
|          50 | JAMES        | GRANGE        |        3 |         8000 |
|          60 | NORTON       | SAN FRANSISCO |        5 |        17000 |
|          70 | JOHN MICHAEL | EUROPE        |       16 |         8000 |
+-------------+--------------+---------------+----------+--------------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM CUSTOMER WHERE ADDRESS LIKE 'L_N__N';
+-------------+-----------+---------+----------+--------------+
| CUSTOMER_NO | NAME      | ADDRESS | DEPOT_NO | CREDIT_LIMIT |
+-------------+-----------+---------+----------+--------------+
|          40 | BOB SMITH | LONDON  |        2 |        10000 |
+-------------+-----------+---------+----------+--------------+
1 row in set (0.00 sec)
```

Conclusion:

In conclusion, this practical exercise provides valuable experience in using SQL for data retrieval. Through this practical we have successfully practiced writing SQL queries to retrieve data from the given database, demonstrating their ability to select and filter data based on specific criteria.

Practical No. 3

Aim: To implement aggregate functions.

Theory:

Aggregate functions in MySQL are powerful tools for summarizing and analysing data in database tables. These functions allow you to perform calculations on sets of values and return a single result. Here's a brief overview of common aggregate functions:

- **COUNT()**: Counts the number of rows in a result set. Useful for determining the size of a dataset.
- **SUM()**: Calculates the sum of values in a numeric column, such as total sales or quantities.
- **AVG()**: Computes the average of values in a numeric column, providing insight into the typical value.
- **MIN() and MAX()**: Identify the minimum and maximum values in a column, helping to find extremes in the data.
- **GROUP BY**: Groups rows based on a specified column, allowing you to perform aggregate functions on subsets of data. Useful for creating summaries or reports.

Queries:

```
mysql> /*202203103510124*/
mysql> SELECT count(*) AS total;
+-------+
| total |
+-------+
|     1 |
+-------+
1 row in set (0.00 sec)

mysql> /*202203103510124*/
mysql> select sum(10+20+30) as sum;
+------+
| sum  |
+------+
|   60 |
+------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> select avg(price) as avg from product;
+-------------+
| avg         |
+-------------+
| 3383.333333 |
+-------------+
1 row in set (0.00 sec)
```

```
mysql> select min(price) as min, max(price) as max from product;
+--------+----------+
| min    | max      |
+--------+----------+
| 700.00 | 15000.00 |
+--------+----------+
1 row in set (0.00 sec)
```

Conclusion:

In conclusion, the practical exploration of implementing aggregate functions in MySQL has provided valuable insights into the capabilities of these functions. Aggregate functions, such as COUNT, SUM, AVG, MIN, and MAX, empower us to summarize and analyse data effectively. They enable us to derive meaningful information from large datasets, such as calculating totals, averages, and identifying extreme values. Additionally, using the GROUP BY clause in combination with aggregate functions allows us to perform calculations on specific subsets of data, facilitating the creation of informative reports and summaries. Mastering these aggregate functions is essential for anyone working with databases, as they are fundamental tools for data analysis and decision-making.

Practical No. 4

Aim: To implement Integrity Constraints. Queries (along with sub Queries)

Theory:

The practical exercises you've been working on involve SQL queries and operations on a set of database tables. Below, I'll provide a comprehensive theory section that covers the key concepts and SQL statements used in these exercises:

SQL (Structured Query Language): SQL is a domain-specific language used for managing and manipulating relational databases. It's a standard for communicating with and querying databases.

Relational Database: A relational database is a structured collection of data organized into tables with rows and columns, where each row represents a record, and each column represents an attribute.

SELECT Statement: The SELECT statement is used to retrieve data from one or more database tables. It allows you to specify which columns to retrieve and can include conditions for filtering the data.

FROM Clause: The FROM clause in a SELECT statement specifies the tables from which to retrieve data.

WHERE Clause: The WHERE clause is used to filter rows based on specified conditions. It acts as a condition or filter applied to the rows in the result set.

INNER JOIN: An INNER JOIN is used to combine rows from two or more tables based on a related column between them. It returns only the rows where there is a match in both tables.

LEFT JOIN (or LEFT OUTER JOIN): A LEFT JOIN returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for the right table's columns.

RIGHT JOIN (or RIGHT OUTER JOIN): A RIGHT JOIN is similar to a LEFT JOIN but returns all rows from the right table and the matched rows from the left table.

FULL JOIN (or FULL OUTER JOIN): A FULL JOIN returns all rows when there is a match in either the left or right table. If there's no match, NULL values are returned.

GROUP BY Clause: The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows.

HAVING Clause: The HAVING clause is used in combination with GROUP BY to filter grouped rows based on specified conditions.

ORDER BY Clause: The ORDER BY clause is used to sort the result set based on one or more columns, either in ascending (ASC) or descending (DESC) order.

Aggregate Functions: Aggregate functions perform a calculation on a set of values and return a single value. Common aggregate functions include COUNT, SUM, AVG, MIN, and MAX.

SUBQUERY: A subquery, also known as a nested query, is a query embedded within another query. It can be used to retrieve data that will be used as a condition in the outer query.

EXISTS: The EXISTS condition is used to test whether a subquery returns any rows. If the subquery returns at least one row, the condition is true.

UNION Operator: The UNION operator is used to combine the result sets of two or more SELECT statements into a single result set, removing duplicate rows.

EXCEPT Operator: The EXCEPT operator is used to return the distinct rows from the left SELECT statement that are not present in the right SELECT statement.

Queries:

(1) List the description of product which are supplied by supplier SMITH using

IN.

```
mysql> /*202203103510124*/
mysql> SELECT P.DESCRIPTION
    -> FROM PRODUCT P
    -> WHERE P.SUPPLIER_NO IN (SELECT SUPPLIER_NO FROM SUPPLIER WHERE NAME = 'SMITH');
+----------------+
| DESCRIPTION    |
+----------------+
| REDUCER        |
| WIDGET REMOVER |
+----------------+
2 rows in set (0.01 sec)
```

(2) List all product no which are not ordered by the customer having same CORDER_NO

As the CUSTOMER_NO 20.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT P.PRODUCT_NO
    -> FROM PRODUCT P
    -> WHERE P.PRODUCT_NO NOT IN (SELECT OL.PRODUCT_NO
    ->                            FROM OLINE OL
    ->                            JOIN CORDER O ON OL.CORDER_NO = O.CORDER_NO
    ->                            WHERE O.CUSTOMER_NO = 20);
+------------+
| PRODUCT_NO |
+------------+
|        121 |
|        122 |
|        124 |
|        136 |
|        137 |
+------------+
5 rows in set (0.01 sec)
```

(3) List the locations and addresses of all depots which stock any product which is supplied to the depot whose location is wales.

```
mysql> /*202203103510124*/
mysql> SELECT D.LOCATION, D.ADDRESS
    -> FROM DEPOT D
    -> WHERE EXISTS (SELECT 1
    ->               FROM STOCK S
    ->               JOIN PRODUCT P ON S.PRODUCT_NO = P.PRODUCT_NO
    ->               WHERE S.DEPOT_NO = D.DEPOT_NO
    ->                 AND P.SUPPLIER_NO IN (SELECT SUPPLIER_NO
    ->                                       FROM SUPPLIER
    ->                                       WHERE LOCATION = 'WALES'));
Empty set (0.00 sec)
```

(4) List the customer_no, date_placed and date_delivered for all orders which contain order lines for the product with product_no 137 using existential quantification (ie the where exists condition).

```
mysql> /*202203103510124*/
mysql> SELECT O.CUSTOMER_NO, O.DATE_PLACED, O.DATE_DELIVERED
    -> FROM CORDER O
    -> WHERE EXISTS (SELECT 1
    ->               FROM OLINE OL
    ->               WHERE OL.CORDER_NO = O.CORDER_NO
    ->                 AND OL.PRODUCT_NO = 137);
Empty set (0.00 sec)
```

(5) List the depots which do not stock any product supplied by the supplier whose name is ringworld.

```
mysql> /*202203103510124*/
mysql> SELECT D.LOCATION, D.ADDRESS
    -> FROM DEPOT D
    -> WHERE D.DEPOT_NO NOT IN (SELECT DISTINCT S2.DEPOT_NO
    ->                          FROM STOCK S2
    ->                          JOIN PRODUCT P2 ON S2.PRODUCT_NO = P2.PRODUCT_NO
    ->                          WHERE P2.SUPPLIER_NO IN (SELECT SUPPLIER_NO
    ->                                                   FROM SUPPLIER
    ->                                                   WHERE NAME = 'RINGWORLD'));
+-----------------+---------+
| LOCATION        | ADDRESS |
+-----------------+---------+
| NORTH UK        | 1       |
| LONDON WEST USA | 3       |
| EAST NZ         | 4       |
| WALES UK        | 5       |
| NORTH KENYA     | 6       |
| SOUTH UK        | 2       |
+-----------------+---------+
6 rows in set (0.00 sec)
```

(6) List the locations and addresses of all depots which stock all products supplied by the supplier babylon 5.

```
mysql> /*202203103510124*/
mysql> SELECT D.LOCATION, D.ADDRESS
    -> FROM DEPOT D
    -> WHERE NOT EXISTS (SELECT P3.PRODUCT_NO
    ->                   FROM PRODUCT P3
    ->                   WHERE P3.SUPPLIER_NO = (SELECT SUPPLIER_NO
    ->                                           FROM SUPPLIER
    ->                                           WHERE NAME = 'BABYLON 5')
    ->                     AND P3.PRODUCT_NO NOT IN (SELECT S3.PRODUCT_NO
    ->                                               FROM STOCK S3
    ->                                               WHERE S3.DEPOT_NO = D.DEPOT_NO));
+-----------------+---------+
| LOCATION        | ADDRESS |
+-----------------+---------+
| NORTH UK        | 1       |
| SOUTH USA       | 2       |
| LONDON WEST USA | 3       |
| EAST NZ         | 4       |
| WALES UK        | 5       |
| NORTH KENYA     | 6       |
| SOUTH UK        | 2       |
+-----------------+---------+
7 rows in set (0.00 sec)
```

(7) List the number of different products supplied by each supplier_no.

```
mysql> /*202203103510124*/
mysql> SELECT S.SUPPLIER_NO, COUNT(DISTINCT P4.PRODUCT_NO) AS NUM_PRODUCTS_SUPPLIED
    -> FROM SUPPLIER S
    -> LEFT JOIN PRODUCT P4 ON S.SUPPLIER_NO = P4.SUPPLIER_NO
    -> GROUP BY S.SUPPLIER_NO;
+-------------+-----------------------+
| SUPPLIER_NO | NUM_PRODUCTS_SUPPLIED |
+-------------+-----------------------+
|        1001 |                     1 |
|        1002 |                     1 |
|        1003 |                     1 |
|        1004 |                     1 |
|        1005 |                     2 |
+-------------+-----------------------+
5 rows in set (0.00 sec)
```

(8) List the name of each supplier with the location of each depot and the number of products supplied by that supplier and stocked at that depot.

```
mysql> /*202203103510124*/
mysql> SELECT S.NAME AS SUPPLIER_NAME, D.LOCATION AS DEPOT_LOCATION, COUNT(DISTINCT P5.PRODUCT_NO) AS NUM_PRODUCTS_STOCKED
    -> FROM SUPPLIER S
    -> JOIN PRODUCT P5 ON S.SUPPLIER_NO = P5.SUPPLIER_NO
    -> JOIN STOCK S4 ON P5.PRODUCT_NO = S4.PRODUCT_NO
    -> JOIN DEPOT D ON S4.DEPOT_NO = D.DEPOT_NO
    -> GROUP BY S.NAME, D.LOCATION;
+---------------+----------------+----------------------+
| SUPPLIER_NAME | DEPOT_LOCATION | NUM_PRODUCTS_STOCKED |
+---------------+----------------+----------------------+
| BABYLON       | SOUTH UK       |                    1 |
| JOHN          | WALES UK       |                    1 |
| MICHAEL       | LONDON WEST USA |                   1 |
| RINGWORLD     | SOUTH USA      |                    1 |
| SMITH         | EAST NZ        |                    1 |
| SMITH         | NORTH KENYA    |                    1 |
| SMITH         | NORTH UK       |                    1 |
+---------------+----------------+----------------------+
7 rows in set (0.00 sec)
```

(9) List all product descriptions with the product's supplie name, sorted by product description within supplier name(i.e. all products for a supplier listed together in alphabetic order).

```
mysql> /*202203103510124*/
mysql> SELECT P6.DESCRIPTION AS PRODUCT_DESCRIPTION, S3.NAME AS SUPPLIER_NAME
    -> FROM PRODUCT P6
    -> JOIN SUPPLIER S3 ON P6.SUPPLIER_NO = S3.SUPPLIER_NO
    -> ORDER BY S3.NAME, P6.DESCRIPTION;
+---------------------+---------------+
| PRODUCT_DESCRIPTION | SUPPLIER_NAME |
+---------------------+---------------+
| HANDLE              | BABYLON       |
| PLATE               | JOHN          |
| SIZE WIDGET         | MICHAEL       |
| SIZE WIDGET         | RINGWORLD     |
| REDUCER             | SMITH         |
| WIDGET REMOVER      | SMITH         |
+---------------------+---------------+
6 rows in set (0.00 sec)
```

(10) Display customer name who has ordered on same date.

```
mysql> /* EN.NO - 202203103510223 */
mysql> SELECT
    ->     C1.NAME AS CustomerName1,
    ->     C2.NAME AS CustomerName2,
    ->     O1.DATE_PLACED AS OrderDate
    -> FROM
    ->     corder AS O1
    -> INNER JOIN
    ->     corder AS O2 ON O1.DATE_PLACED = O2.DATE_PLACED
    -> INNER JOIN
    ->     customer AS C1 ON O1.CUSTOMER_NO = C1.CUSTOMER_NO
    -> INNER JOIN
    ->     customer AS C2 ON O2.CUSTOMER_NO = C2.CUSTOMER_NO
    -> WHERE
    ->     O1.CORDER_NO <> O2.CORDER_NO
    -> ORDER BY
    ->     O1.DATE_PLACED;
+---------------+---------------+-------------+
| CustomerName1 | CustomerName2 | OrderDate   |
+---------------+---------------+-------------+
| PATEL         | PATEL         | 01-JAN-1993 |
| PATEL         | PATEL         | 01-JAN-1993 |
+---------------+---------------+-------------+
2 rows in set (0.01 sec)
```

Conclusion:

These SQL exercises cover a range of essential SQL concepts and operations, including data retrieval, filtering, joining tables, aggregating data, and using subqueries. Understanding these concepts and practicing SQL queries is crucial for effectively working with relational databases and extracting meaningful insights from data.

Practical No. 5

Aim: Queries using

1.string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr, ELT(), Char_Length(), Format(), Find_In_Set, Oct(), Reverse(), Repeat(), Ascii())
ii.date functions (Sysdate, next_day, add_months, last_day, months_between, least, greatest, trunc, round, to_char, to_date, PERIOD_DIFF)
iii. numeric functions: Abs (), power (), sqrt (), greatest (), least (), round (), mod ().
iv. time functions (Localtime,Minute(datetime), Microsecond)

Theory:

1. **String Functions**:
   - String functions are used for manipulating text data.
   - Concatenation combines two or more strings.
   - Lpad and Rpad add characters to the left or right of a string.
   - Ltrim and Rtrim remove spaces from the left or right of a string.
   - Lower converts text to lowercase, while Upper converts it to uppercase.
   - Initcap capitalizes the first letter of each word.
   - Length returns the length of a string.
   - Substr extracts a portion of a string.
   - Instr finds the position of a substring in a string.
   - ELT() returns the Nth element from a comma-separated list.
   - Char_Length() returns the character length of a string.
   - Format(), Find_In_Set, Oct(), Reverse(), Repeat(), and Ascii() perform various string operations.
2. **Date Functions**:
   - Date functions are used for manipulating date and time data.
   - Sysdate retrieves the current date and time.
   - Next_day returns the next specified day of the week.
   - Add_months adds or subtracts months from a date.
   - Last_day returns the last day of the month.
   - Months_between calculates the difference in months between two dates.
   - Least returns the smallest date from a list.
   - Greatest returns the largest date from a list.
   - Trunc and Round modify the precision of a date or timestamp.
   - To_char and To_date convert between date formats.
   - PERIOD_DIFF calculates the difference between two periods.
3. **Numeric Functions**:
   - Numeric functions are used for mathematical operations.
   - Abs() returns the absolute value of a number.
   - Power() raises a number to a specified power.
   - Sqrt() calculates the square root of a number.
   - Greatest() returns the largest number from a list.
   - Least() returns the smallest number from a list.

- Round() rounds a number to a specified number of decimal places.
- Mod() calculates the remainder of a division operation.

4. **Time Functions**:
   - Time functions are used for working with time data.
   - Localtime retrieves the current local time.
   - Minute(datetime) returns the minute component of a datetime value.
   - Microsecond returns the microsecond component of a datetime value.

Queries:

1.string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr, ELT(), Char_Length(), Format(), Find_In_Set, Oct(), Reverse(), Repeat(), Ascii())

```
mysql> /*202203103510124*/
mysql> -- Concatenation
mysql> SELECT CONCAT('Hello', ' ', 'World') AS Result;
+-------------+
| Result      |
+-------------+
| Hello World |
+-------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LPAD
mysql> SELECT LPAD('123', 5, '0') AS Result;
+--------+
| Result |
+--------+
| 00123  |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- RPAD
mysql> SELECT RPAD('123', 5, '0') AS Result;
+--------+
| Result |
+--------+
| 12300  |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LTRIM
mysql> SELECT LTRIM('   Hello') AS Result;
+--------+
| Result |
+--------+
| Hello  |
+--------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- RTRIM
mysql> SELECT RTRIM('Hello   ') AS Result;
+--------+
| Result |
+--------+
| Hello  |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LOWER
mysql> SELECT LOWER('Hello World') AS Result;
+-------------+
| Result      |
+-------------+
| hello world |
+-------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- UPPER
mysql> SELECT UPPER('Hello World') AS Result;
+-------------+
| Result      |
+-------------+
| HELLO WORLD |
+-------------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- INITCAP
mysql> SELECT INITCAP('hello world') AS Result;
ERROR 1305 (42000): FUNCTION dbms_tables.INITCAP does not exist
mysql>
mysql> /*202203103510124*/
mysql> -- LENGTH
mysql> SELECT LENGTH('Hello World') AS Result;
+--------+
| Result |
+--------+
|     11 |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- SUBSTR (substring)
mysql> SELECT SUBSTR('Hello World', 7, 5) AS Result;
+--------+
| Result |
+--------+
| World  |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- INSTR (find position)
mysql> SELECT INSTR('Hello World', 'World') AS Result;
+--------+
| Result |
+--------+
|      7 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- ELT (element at a specified position)
mysql> SELECT ELT(2, 'Apple', 'Banana', 'Cherry') AS Result;
+--------+
| Result |
+--------+
| Banana |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- CHAR_LENGTH (character length)
mysql> SELECT CHAR_LENGTH('Hello World') AS Result;
+--------+
| Result |
+--------+
|     11 |
+--------+
```

```
mysql> /*202203103510124*/
mysql> -- FORMAT (number formatting)
mysql> SELECT FORMAT(1234567.89, 2) AS Result;
+--------------+
| Result       |
+--------------+
| 1,234,567.89 |
+--------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- FIND_IN_SET (find value in a comma-separated list)
mysql> SELECT FIND_IN_SET('Banana', 'Apple,Banana,Cherry') AS Result;
+--------+
| Result |
+--------+
|      2 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- OCT (convert to octal)
mysql> SELECT OCT(10) AS Result;
+--------+
| Result |
+--------+
| 12     |
+--------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- REVERSE (reverse a string)
mysql> SELECT REVERSE('Hello World') AS Result;
+-------------+
| Result      |
+-------------+
| dlroW olleH |
+-------------+
```

```
mysql> /*202203103510124*/
mysql> -- REPEAT (repeat a string multiple times)
mysql> SELECT REPEAT('ABC', 3) AS Result;
+-----------+
| Result    |
+-----------+
| ABCABCABC |
+-----------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- ASCII (get ASCII value of a character)
mysql> SELECT ASCII('A') AS Result;
+--------+
| Result |
+--------+
|     65 |
+--------+
1 row in set (0.00 sec)
```

ii.Date Functions (Sysdate, next_day, add_months, last_day, months_between, least, greatest, trunc, round, to_char, to_date, PERIOD_DIFF)

```
mysql> SELECT NOW() AS CurrentDate;
+---------------------+
| CurrentDate         |
+---------------------+
| 2023-10-03 19:34:38 |
+---------------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- NEXT_DAY (next occurrence of a specific day)
mysql> SELECT DATE_ADD('2023-09-21', INTERVAL (7 - DAYOFWEEK('2023-09-21')) % 7 + 1 DAY) AS NextSaturday;
+--------------+
| NextSaturday |
+--------------+
| 2023-09-24   |
+--------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- ADD_MONTHS (add months to a date)
mysql> SELECT DATE_ADD('2023-09-21', INTERVAL 3 MONTH) AS ThreeMonthsLater;
+------------------+
| ThreeMonthsLater |
+------------------+
| 2023-12-21       |
+------------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LAST_DAY (last day of the month)
mysql> SELECT LAST_DAY('2023-09-21') AS LastDayOfMonth;
+----------------+
| LastDayOfMonth |
+----------------+
| 2023-09-30     |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- ROUND (round date to a specific precision)
mysql> SELECT DATE_FORMAT('2023-09-21 14:30:45', '%Y-%m-%d %H:00:00') AS RoundedDate;
+---------------------+
| RoundedDate         |
+---------------------+
| 2023-09-21 14:00:00 |
+---------------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- TO_CHAR (convert date to string)
mysql> SELECT DATE_FORMAT('2023-09-21', '%M %d, %Y') AS FormattedDate;
+--------------------+
| FormattedDate      |
+--------------------+
| September 21, 2023 |
+--------------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- TO_DATE (convert string to date)
mysql> SELECT STR_TO_DATE('2023-09-21', '%Y-%m-%d') AS ConvertedDate;
+---------------+
| ConvertedDate |
+---------------+
| 2023-09-21    |
+---------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- PERIOD_DIFF (difference in periods between two dates)
mysql> SELECT PERIOD_DIFF('202301', '202201') AS PeriodDiff;
+------------+
| PeriodDiff |
+------------+
|         12 |
+------------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- MONTHS_BETWEEN (difference in months between two dates)
mysql> SELECT PERIOD_DIFF(EXTRACT(YEAR_MONTH FROM '202312'), EXTRACT(YEAR_MONTH FROM '202301')) AS MonthsDiff;
+------------+
| MonthsDiff |
+------------+
|       NULL |
+------------+
1 row in set, 1 warning (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LEAST (returns the smallest date among multiple dates)
mysql> SELECT LEAST('2023-09-21', '2023-10-01', '2023-11-15') AS SmallestDate;
+--------------+
| SmallestDate |
+--------------+
| 2023-09-21   |
+--------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- GREATEST (returns the largest date among multiple dates)
mysql> SELECT GREATEST('2023-09-21', '2023-10-01', '2023-11-15') AS LargestDate;
+-------------+
| LargestDate |
+-------------+
| 2023-11-15  |
+-------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- TRUNC (truncate date to a specific precision)
mysql> SELECT DATE_FORMAT('2023-09-21 14:30:45', '%Y-%m-01') AS TruncatedDate;
+---------------+
| TruncatedDate |
+---------------+
| 2023-09-01    |
+---------------+
1 row in set (0.00 sec)
```

iii. Numeric Functions: Abs (), power (), sqrt (), greatest (), least (),

round (), mod ().

```
mysql> /*202203103510124*/
mysql> -- MOD (modulus, returns the remainder of a division)
mysql> SELECT MOD(10, 3) AS ModulusValue;
+---------------+
| ModulusValue |
+---------------+
|             1 |
+---------------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- ABS (absolute value)
mysql> SELECT ABS(-5) AS AbsoluteValue;
+--------------+
| AbsoluteValue |
+--------------+
|            5 |
+--------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- POWER (exponentiation)
mysql> SELECT POWER(2, 3) AS Exponentiation;
+----------------+
| Exponentiation |
+----------------+
|              8 |
+----------------+
```

```
mysql> /*202203103510124*/
mysql> -- SQRT (square root)
mysql> SELECT SQRT(25) AS SquareRoot;
+------------+
| SquareRoot |
+------------+
|          5 |
+------------+
1 row in set (0.00 sec)
```

```
mysql> /*202203103510124*/
mysql> -- GREATEST (returns the largest value among multiple values)
mysql> SELECT GREATEST(10, 20, 30) AS LargestValue;
+--------------+
| LargestValue |
+--------------+
|           30 |
+--------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- LEAST (returns the smallest value among multiple values)
mysql> SELECT LEAST(10, 20, 30) AS SmallestValue;
+---------------+
| SmallestValue |
+---------------+
|            10 |
+---------------+
1 row in set (0.00 sec)

mysql>
mysql> /*202203103510124*/
mysql> -- ROUND (rounds a number to a specified decimal place)
mysql> SELECT ROUND(3.14159, 2) AS RoundedValue;
+--------------+
| RoundedValue |
+--------------+
|         3.14 |
+--------------+
1 row in set (0.00 sec)
```

iv. Time Functions (Localtime,Minute(datetime), Microsecond)

```
mysql> /*202203103510124*/
mysql> select localtime() as time;
+---------------------+
| time                |
+---------------------+
| 2023-10-03 19:42:28 |
+---------------------+
1 row in set (0.00 sec)

mysql> select minut(now()) as minuts;
ERROR 1305 (42000): FUNCTION dbms_tables.minut does not exist
mysql> select minute(now()) as minuts;
+--------+
| minuts |
+--------+
|     43 |
+--------+
1 row in set (0.00 sec)

mysql> select microsecond(now()) as ms;
+------+
| ms   |
+------+
|    0 |
+------+
1 row in set (0.00 sec)
```

Conclusion:

In this practical exploration of SQL functions, we delved into four categories: string, date, numeric, and time functions, within the MySQL database system. String functions empowered us to manipulate text data, date functions facilitated date and time-related operations, numeric functions enabled mathematical calculations, and time functions offered tools for working with temporal data. These functions are indispensable in database management, allowing for data transformation, analysis, and presentation. Mastering these functions is essential for proficient database querying and data manipulation, enhancing the capabilities of database professionals in handling diverse types of data.

Practical No. 6

Aim: To implement Joins.

Theory:

1. **Database Tables:** We worked with several tables, including customer, corder, depot, oline, product, salesrep, stock, and suppliers. Each table represents a different aspect of a business scenario, such as customer data, orders, products, and sales representatives.
2. **SQL Queries:** We used SQL (Structured Query Language) to query the database and retrieve meaningful information. SQL allows us to interact with the database by performing operations like selecting, filtering, joining, and aggregating data.
3. **SELECT Statement:** We frequently used the SELECT statement to specify which columns we wanted to retrieve from a table. This statement is fundamental in SQL and forms the basis of most queries.
4. **JOIN Operations:** We used various types of joins, including INNER JOIN and LEFT JOIN, to combine data from multiple tables based on matching keys. Joins are essential for retrieving related data from different tables.
5. **Filtering Data:** The WHERE clause allowed us to filter rows based on specific conditions. We used it to narrow down our results and retrieve only the data that met certain criteria.
6. **Aggregation:** We employed aggregate functions like SUM, COUNT, and COALESCE to perform calculations on groups of data. Aggregation functions are useful for obtaining summary information, such as total quantities and averages.
7. **Subqueries:** Subqueries were used to create nested queries within our main queries. They helped us retrieve data from one table based on information obtained from another table.
8. **Aliases:** We used aliases to assign temporary names to columns or tables. This made our query results more readable and provided clarity when dealing with complex queries.

we delved into the realm of relational databases and SQL (Structured Query Language). Databases are the backbone of modern information management, and SQL is the language used to interact with them. We explored the fundamental aspects of database querying, including the use of SQL statements like SELECT, JOIN, and WHERE to retrieve, combine, and filter data from multiple tables. We learned how to perform calculations, aggregate data, and use subqueries to tackle complex questions. Additionally, we employed aliases to enhance query readability. These practical exercises provided valuable hands-on experience and insight into the world of relational databases and SQL, which are vital skills for data professionals and anyone seeking to harness the power of data for decision-making and analysis.

Queries:

(1) Give a list of depot locations paired with the name of the sales rep who covers that depot.

```
mysql> /*202203103510124*/
mysql> SELECT D.LOCATION, SR.NAME AS SALES_REP_NAME
    -> FROM DEPOT D
    -> JOIN SALESREP SR ON D.REP_NO = SR.REP_NO;
+-----------------+----------------+
| LOCATION        | SALES_REP_NAME |
+-----------------+----------------+
| NORTH UK        | MIKE           |
| SOUTH USA       | FRED           |
| LONDON WEST USA | ALI            |
| EAST NZ         | SAM            |
| WALES UK        | BILL ADAMS     |
| NORTH KENYA     | SAM            |
| SOUTH UK        | FRED           |
+-----------------+----------------+
```

(2) List the customer name and the depot location for the depot delivering to that customer for all customers who receive deliveries from depots looked after by sales rep number (rep_no) 3.

```
mysql> /*202203103510124*/
mysql> SELECT C.NAME AS CUSTOMER_NAME, D.LOCATION AS DEPOT_LOCATION
    -> FROM CUSTOMER C
    -> JOIN DEPOT D ON C.DEPOT_NO = D.DEPOT_NO
    -> WHERE D.REP_NO = 3;
+---------------+-----------------+
| CUSTOMER_NAME | DEPOT_LOCATION  |
+---------------+-----------------+
| JAMES         | LONDON WEST USA |
+---------------+-----------------+
1 row in set (0.00 sec)
```

(3) List the sales rep number (rep_no) and depot location and address for depots looked after by the sales rep whose name is mike.

```
mysql> /*202203103510124*/
mysql> SELECT SR.REP_NO, D.LOCATION, D.ADDRESS
    -> FROM DEPOT D
    -> JOIN SALESREP SR ON D.REP_NO = SR.REP_NO
    -> WHERE SR.NAME = 'MIKE';
+--------+----------+---------+
| REP_NO | LOCATION | ADDRESS |
+--------+----------+---------+
|      1 | NORTH UK | 1       |
+--------+----------+---------+
1 row in set (0.00 sec)
```

(4) For all order lines (oline) for all orders (corder) for customers whose name is patel, list the customer address, the date_placed, the product_no and the quantity.

```
mysql> /*202203103510124*/
mysql> SELECT C.NAME AS CUSTOMER_NAME, C.ADDRESS AS CUSTOMER_ADDRESS, O.DATE_PLACED, OL.PRODUCT_NO, OL.QUANTITY
    -> FROM CUSTOMER C
    -> JOIN CORDER O ON C.CUSTOMER_NO = O.CUSTOMER_NO
    -> JOIN OLINE OL ON O.CORDER_NO = OL.CORDER_NO
    -> WHERE C.NAME = 'PATEL';
+---------------+------------------+-------------+------------+----------+
| CUSTOMER_NAME | CUSTOMER_ADDRESS | DATE_PLACED | PRODUCT_NO | QUANTITY |
+---------------+------------------+-------------+------------+----------+
| PATEL         | GRANGE           | 1993-01-01  |        120 |        5 |
| PATEL         | GRANGE           | 1993-01-01  |        120 |        5 |
+---------------+------------------+-------------+------------+----------+
```

(5) Give the total number of items (quantity) in stock in all depots.

```
mysql> /*202203103510124*/
mysql> SELECT SUM(QUANTITY) AS TOTAL_STOCK
    -> FROM STOCK;
+-------------+
| TOTAL_STOCK |
+-------------+
|         540 |
+-------------+
```

(6) Give the total number of items (order line quantity) which have been ordered on the order with corder_no 200.

```
mysql> /*202203103510124*/
mysql> SELECT SUM(OL.QUANTITY) AS ORDER_TOTAL
    -> FROM OLINE OL
    -> WHERE OL.CORDER_NO = 200;
+-------------+
| ORDER_TOTAL |
+-------------+
|           5 |
+-------------+
```

(7) List the names of all customers who receive deliveries from depots which are looked after by the sales rep whose name is fred.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT C.NAME AS CUSTOMER_NAME
    -> FROM CUSTOMER C
    -> JOIN DEPOT D ON C.DEPOT_NO = D.DEPOT_NO
    -> JOIN SALESREP SR ON D.REP_NO = SR.REP_NO
    -> WHERE SR.NAME = 'FRED';
+---------------+
| CUSTOMER_NAME |
+---------------+
| BOB SMITH     |
| JOHN MICHAEL  |
+---------------+
```

(8) List the customer name, order date_placed, order line quantity and product description for each order line (with its linked, order, customer and product rows) for customers who receive deliveries from depot number 2.

```
mysql> /*202203103510124*/
mysql> SELECT C.NAME AS CUSTOMER_NAME, O.DATE_PLACED, OL.QUANTITY, P.DESCRIPTION AS PRODUCT_DESCRIPTION
    -> FROM CUSTOMER C
    -> JOIN CORDER O ON C.CUSTOMER_NO = O.CUSTOMER_NO
    -> JOIN OLINE OL ON O.CORDER_NO = OL.CORDER_NO
    -> JOIN PRODUCT P ON OL.PRODUCT_NO = P.PRODUCT_NO
    -> WHERE C.DEPOT_NO = 2;
+---------------+-------------+----------+--------------------+
| CUSTOMER_NAME | DATE_PLACED | QUANTITY | PRODUCT_DESCRIPTION |
+---------------+-------------+----------+--------------------+
| BOB SMITH     | 1993-01-17  |       10 | PLATE              |
| BOB SMITH     | 1994-01-01  |       30 | SIZE WIDGET        |
+---------------+-------------+----------+--------------------+
```

(9) List supplier names paired with the names of the sales reps who market products supplied by that supplier.

```
mysql> /*202203103510124*/
mysql> SELECT S.NAME AS SUPPLIER_NAME, SR.NAME AS SALES_REP_NAME
    -> FROM SUPPLIER S
    -> JOIN PRODUCT P ON S.SUPPLIER_NO = P.SUPPLIER_NO
    -> JOIN SALESREP SR ON P.MARKETING_REP_NO = SR.REP_NO;
+---------------+----------------+
| SUPPLIER_NAME | SALES_REP_NAME |
+---------------+----------------+
| SMITH         | BILL ADAMS     |
| JOHN          | ALI            |
| BABYLON       | FRED           |
| SMITH         | SAM            |
| MICHAEL       | MIKE           |
| RINGWORLD     | FRED           |
+---------------+----------------+
```

(10) List supplier names paired with the names of the sales reps who look after the depots where products from that supplier are delivered.

```
mysql> /*202203103510124*/
mysql> SELECT S.NAME AS SUPPLIER_NAME, SR.NAME AS SALES_REP_NAME
    -> FROM SUPPLIER S
    -> JOIN PRODUCT P ON S.SUPPLIER_NO = P.SUPPLIER_NO
    -> JOIN STOCK ST ON P.PRODUCT_NO = ST.PRODUCT_NO
    -> JOIN DEPOT D ON ST.DEPOT_NO = D.DEPOT_NO
    -> JOIN SALESREP SR ON D.REP_NO = SR.REP_NO;
+---------------+----------------+
| SUPPLIER_NAME | SALES_REP_NAME |
+---------------+----------------+
| SMITH         | MIKE           |
| RINGWORLD     | FRED           |
| MICHAEL       | ALI            |
| SMITH         | SAM            |
| JOHN          | BILL ADAMS     |
| SMITH         | SAM            |
| BABYLON       | FRED           |
+---------------+----------------+
```

(11) List the names of all customers who have ordered products which are marketed by the sales rep whose name is ali.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT C.NAME AS CUSTOMER_NAME
    -> FROM CUSTOMER C
    -> JOIN CORDER O ON C.CUSTOMER_NO = O.CUSTOMER_NO
    -> JOIN PRODUCT P ON C.CUSTOMER_NO = P.SUPPLIER_NO
    -> JOIN SALESREP SR ON P.MARKETING_REP_NO = SR.REP_NO
    -> WHERE SR.NAME = 'ALI';
Empty set (0.00 sec)
```

(12) List the names of all customers who are delivered to by the depot which delivers to the customer whose name is drake.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT C1.NAME AS CUSTOMER_NAME
    -> FROM CUSTOMER C1
    -> JOIN DEPOT D1 ON C1.DEPOT_NO = D1.DEPOT_NO
    -> JOIN DEPOT D2 ON D1.LOCATION = D2.LOCATION
    -> JOIN CUSTOMER C2 ON D2.DEPOT_NO = C2.DEPOT_NO
    -> WHERE C2.NAME = 'DRAKE';
+---------------+
| CUSTOMER_NAME |
+---------------+
| DRAKE         |
+---------------+
```

(13) List each product description and its price increased by 10%.

```
mysql> /*202203103510124*/
mysql> SELECT P.DESCRIPTION, P.PRICE * 1.1 AS INCREASED_PRICE
    -> FROM PRODUCT P;
+----------------+-----------------+
| DESCRIPTION    | INCREASED_PRICE |
+----------------+-----------------+
| REDUCER        |        1320.000 |
| PLATE          |        1650.000 |
| HANDLE         |         770.000 |
| WIDGET REMOVER |         990.000 |
| SIZE WIDGET    |        1100.000 |
| SIZE WIDGET    |       16500.000 |
+----------------+-----------------+
```

(14) List all order lines for the customer with customer_no 20 giving the product description, the order line quantity and the value of the order line. (i.e. the order line quantity * the price from the linked product row)

```
mysql> /*202203103510124*/
mysql> SELECT P.DESCRIPTION, OL.QUANTITY, P.PRICE * OL.QUANTITY AS ORDER_LINE_VALUE
    -> FROM CUSTOMER C
    -> JOIN CORDER O ON C.CUSTOMER_NO = O.CUSTOMER_NO
    -> JOIN OLINE OL ON O.CORDER_NO = OL.CORDER_NO
    -> JOIN PRODUCT P ON OL.PRODUCT_NO = P.PRODUCT_NO
    -> WHERE C.CUSTOMER_NO = 20;
+-------------+----------+------------------+
| DESCRIPTION | QUANTITY | ORDER_LINE_VALUE |
+-------------+----------+------------------+
| REDUCER     |        5 |          6000.00 |
| REDUCER     |        5 |          6000.00 |
+-------------+----------+------------------+
```

(15) List the locations and addresses of all depots which do not stock product number 122. (ie where there is no stock row for that product for the depot)

```
mysql> /*202203103510124*/
mysql> SELECT D.LOCATION, D.ADDRESS
    -> FROM DEPOT D
    -> LEFT JOIN STOCK S ON D.DEPOT_NO = S.DEPOT_NO AND S.PRODUCT_NO = 122
    -> WHERE S.PRODUCT_NO IS NULL;
+-----------------+---------+
| LOCATION        | ADDRESS |
+-----------------+---------+
| NORTH UK        | 1       |
| SOUTH USA       | 2       |
| LONDON WEST USA | 3       |
| EAST NZ         | 4       |
| WALES UK        | 5       |
| NORTH KENYA     | 6       |
+-----------------+---------+
```

(16) Set up a query which lists the names of all customers who have placed an order with the order number (corder_no) of the order merged with the names of all customers who have never placed an order (shown once, with the order number attribute null) i.e. an outer join.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT C1.NAME AS CUSTOMER_NAME, C2.NAME AS MERGED_CUSTOMER_NAME
    -> FROM CUSTOMER C1
    -> LEFT JOIN CORDER O ON C1.CUSTOMER_NO = O.CUSTOMER_NO
    -> LEFT JOIN CUSTOMER C2 ON O.CUSTOMER_NO = C2.CUSTOMER_NO;
+---------------+----------------------+
| CUSTOMER_NAME | MERGED_CUSTOMER_NAME |
+---------------+----------------------+
| GARRY SMITH   | GARRY SMITH          |
| PATEL         | PATEL                |
| DRAKE         | DRAKE                |
| BOB SMITH     | BOB SMITH            |
| JAMES         | NULL                 |
| NORTON        | NULL                 |
| JOHN MICHAEL  | JOHN MICHAEL         |
+---------------+----------------------+
```

Conclusion:

In this series of practical exercises, we gained hands-on experience with SQL, a powerful language for managing and querying relational databases. We learned how to retrieve data from multiple tables, join data together, filter results, and perform calculations. These skills

are fundamental for anyone working with databases, whether in a professional context or for personal projects.

We also explored various real-world scenarios, such as customer orders, product management, and sales representatives, which helped us apply SQL concepts in practical situations. By practicing these exercises, we've developed a strong foundation for working with databases and have learned how to extract valuable insights from data.

Overall, these practical exercises provide a solid introduction to SQL and relational databases, which are essential tools in today's data-driven world.

Practical No. 7

Aim: To Perform Simple queries, string manipulation operations implement groups by having.

Theory:

1. **List the number of different products supplied by each supplier_no:** ○ This query involves the use of the `COUNT` function and the `GROUP BY` clause. ○ `COUNT(DISTINCT PRODUCT_NO)` is used to count the unique product numbers supplied by each supplier.
   ○ `GROUP BY SUPPLIER_NO` groups the results by supplier number, allowing you to count products for each supplier separately.
2. **List the name of each supplier with the location of each depot and the number of products supplied by that supplier and stocked at that depot:**
   ○ This query combines data from multiple tables using `JOIN` operations. ○ It utilizes `COUNT(DISTINCT PRODUCT_NO)` again to count the unique products supplied by each supplier.
   ○ The `GROUP BY` clause is used to group results by supplier name, depot location, and depot address.
3. **List the depot_no's of all depots where the average credit_limit for all the customers receiving deliveries from the depot is > 20,000:**
   ○ This query calculates the average (`AVG`) credit limit for each depot.
   ○ It uses the `HAVING` clause to filter depots based on the average credit limit condition.
4. **List the total quantity and product number ordered by each customer:** ○ This query uses the `SUM` function to calculate the total quantity of products ordered by each customer.
   ○ It groups the results by customer number.
5. **Give the product number that has the maximum quantity stocked at any depot:** ○ This query uses the `SUM` function to calculate the total quantity of each product stocked at different depots.
   ○ The `ORDER BY` clause sorts the results in descending order of quantity.
6. **Give the customer address with the minimum credit limit:**
   ○ This query uses a subquery with `MIN(CREDIT_LIMIT)` to find the minimum credit limit.
   ○ It then selects the customer address corresponding to that minimum credit limit.
7. **List supplier numbers who have supplied products whose total price is < 1000:** ○ This query involves subqueries to filter suppliers based on the price condition. ○ It uses `DISTINCT` to ensure that each supplier number appears only once in the result.
8. **Display the total number of customers who have ordered products on the same date:**
   ○ This query involves a subquery to count the number of customers who ordered on the same date.
9. **List the sum of quantities stocked at each rack:**
   ○ This query uses the `SUM` function to calculate the total quantity of products stocked at each rack.
   ○ It groups the results by the rack number.
10. **Display the total number of customers who have received products from the**

same location:

- This query combines data from the `depot` and `customer` tables.
- It counts the number of customers for each location that received products.
- The `HAVING` clause filters locations with more than one customer.

Queries:

(1) List the number of different products supplied by each supplier_no.

```
mysql> /*202203103510124*/
mysql> SELECT SUPPLIER_NO, COUNT(DISTINCT PRODUCT_NO) AS NUM_PRODUCTS_SUPPLIED
    -> FROM PRODUCT
    -> GROUP BY SUPPLIER_NO;
+-------------+------------------------+
| SUPPLIER_NO | NUM_PRODUCTS_SUPPLIED |
+-------------+------------------------+
|        1001 |                      1 |
|        1002 |                      1 |
|        1003 |                      1 |
|        1004 |                      1 |
|        1005 |                      2 |
+-------------+------------------------+
5 rows in set (0.00 sec)
```

(2) List the name of each supplier with the location of each depot and the number of products supplied by that supplier and stocked at that depot.

```
mysql> /*202203103510124*/
mysql> SELECT S.NAME AS SUPPLIER_NAME, D.LOCATION AS DEPOT_LOCATION, COUNT(DISTINCT P.PRODUCT_NO) AS NUM_PRODUCTS_SUPPLIED
    -> FROM SUPPLIER S
    -> JOIN PRODUCT P ON S.SUPPLIER_NO = P.SUPPLIER_NO
    -> JOIN STOCK ST ON P.PRODUCT_NO = ST.PRODUCT_NO
    -> JOIN DEPOTE D ON ST.DEPOT_NO = D.DEPOT_NO
    -> GROUP BY S.NAME, D.LOCATION;
+--------------+---------------+----------------------+
| SUPPLIER_NAME | DEPOT_LOCATION | NUM_PRODUCTS_SUPPLIED |
+--------------+---------------+----------------------+
| BABYLON       | SOUTH         |                    1 |
| JOHN          | WALES         |                    1 |
| MICHAEL       | LONDON        |                    1 |
| RINGWORLD     | SOUTH         |                    1 |
| SMITH         | EAST          |                    1 |
| SMITH         | NORTH         |                    2 |
+--------------+---------------+----------------------+
6 rows in set (0.00 sec)
```

(3) List the depot_no's of all depots where the average credit_limit for all the customers receiving deliveries from the depot is > 20,000.

```
mysql> /*202203103510124*/
mysql> SELECT D.DEPOT_NO
    -> FROM DEPOTE D
    -> JOIN CUSTOMER C ON D.DEPOT_NO = C.DEPOT_NO
    -> GROUP BY D.DEPOT_NO
    -> HAVING AVG(C.CREDIT_LIMIT) > 20000;
Empty set (0.00 sec)
```

(4) List total no of quantity and product number ordered by customer.

```
mysql> /*202203103510124*/
mysql> SELECT C.CUSTOMER_NO, SUM(OL.QUANTITY) AS TOTAL_QUANTITY_ORDERED
    -> FROM CUSTOMER C
    -> JOIN CORDER O ON C.CUSTOMER_NO = O.CUSTOMER_NO
    -> JOIN OLINE OL ON O.CORDER_NO = OL.CORDER_NO
    -> GROUP BY C.CUSTOMER_NO;
+-------------+------------------------+
| CUSTOMER_NO | TOTAL_QUANTITY_ORDERED |
+-------------+------------------------+
|          10 |                     30 |
|          20 |                     10 |
|          30 |                     20 |
|          40 |                     40 |
|          70 |                     15 |
+-------------+------------------------+
5 rows in set (0.00 sec)
```

(5) Give product number which has maximum quantity stock at any depot.

```
mysql> /*202203103510124*/
mysql> SELECT PRODUCT_NO
    -> FROM STOCK
    -> GROUP BY PRODUCT_NO
    -> HAVING SUM(QUANTITY) = (SELECT MAX(TOTAL_QUANTITY) FROM (SELECT PRODUCT_NO, SUM(QUANTITY) AS TOTAL_QUANTITY FROM STOCK GROUP BY PRODUCT_NO) AS
TOTALS);
+------------+
| PRODUCT_NO |
+------------+
|        124 |
+------------+
1 row in set (0.00 sec)
```

(6) Give customer address which has minimum credit limit.

```
mysql> /*202203103510124*/
mysql> SELECT CUSTOMER_NO, ADDRESS
    -> FROM CUSTOMER
    -> ORDER BY CREDIT_LIMIT ASC
    -> LIMIT 1;
+-------------+---------+
| CUSTOMER_NO | ADDRESS |
+-------------+---------+
|          10 | BRIXTON |
+-------------+---------+
1 row in set (0.00 sec)
```

(7) List supplier no who has supplied products whose total price is < 1000.

```
mysql> /*202203103510124*/
mysql> SELECT S.SUPPLIER_NO
    -> FROM SUPPLIER S
    -> JOIN PRODUCT P ON S.SUPPLIER_NO = P.SUPPLIER_NO
    -> GROUP BY S.SUPPLIER_NO
    -> HAVING SUM(P.PRICE) < 1000;
+-------------+
| SUPPLIER_NO |
+-------------+
|        1003 |
+-------------+
1 row in set (0.00 sec)
```

(8) Give total number of customers who has ordered the product on same date.

```
mysql> /*202203103510124*/
mysql> SELECT DATE_PLACED, COUNT(*) AS CUSTOMER_COUNT
    -> FROM CORDER
    -> GROUP BY DATE_PLACED
    -> HAVING COUNT(*) > 1;
+-------------+----------------+
| DATE_PLACED | CUSTOMER_COUNT |
+-------------+----------------+
| 1993-JAN-01 |              2 |
+-------------+----------------+
1 row in set (0.00 sec)
```

(9) List sum of quantity stocked at each rack.

```
mysql> /*202203103510124*/
mysql> SELECT RACK, SUM(QUANTITY) AS TOTAL_QUANTITY_STOCKED
    -> FROM STOCK
    -> GROUP BY RACK;
+------+-------------------------+
| RACK | TOTAL_QUANTITY_STOCKED |
+------+-------------------------+
| 1    |                      50 |
| 10   |                     180 |
| 2    |                      40 |
| 4    |                     120 |
| 5    |                      90 |
| 7    |                      60 |
+------+-------------------------+
6 rows in set (0.00 sec)
```

(10) Display total no of customers who have received product from same location.

```
mysql> /*202203103510124*/
mysql> select d.location, count(distinct c.customer_no) as cust_count from depot d inner join customer c on d.depot_no =
 c.depot_no group by d.location;
+----------------+------------+
| location       | cust_count |
+----------------+------------+
| EAST NZ        |          1 |
| LONDON WEST USA |          1 |
| NORTH KENYA    |          1 |
| NORTH UK       |          1 |
| SOUTH UK       |          1 |
| SOUTH USA      |          1 |
| WALES UK       |          1 |
+----------------+------------+
```

Conclusion:

We learned how to retrieve data, count unique values, and perform aggregations using functions like `COUNT`, `SUM`, and `AVG`. The `GROUP BY` clause allowed us to organize data into meaningful groups, making it invaluable for summarizing information across various categories.

Joining tables using the `JOIN` operation enabled us to link data from different parts of the database, creating a comprehensive view of relationships between entities. Subqueries proved handy for dynamic filtering and calculations, enhancing the flexibility of our queries.

Practical No. 8

Aim : Identify the case study and detailed statement of the problem. Design an Entity Relationship (ER) / Extended Entity-Relationship (EER) Model.

Project Description :-

- This web app is created for Sentosa Enclave residential society.
- Aims to optimize society management and communication.
- Enables convenient online maintenance payments, event updates, clubhouse bookings, and more.
- Provides a user-friendly platform for easy access to essential society information.
- Empowers residents with personalized features such as digital notices, intercom, and complaint resolution.

Expected Outcomes:-

- Efficient Management: Streamlined management of society activities, notices, and payments.
- Improved Communication: Enhanced resident communication through notices and events.
- Cost and Time Savings: Reduced administrative workload and costs.
- Better User Experience: User-friendly design for easy access and interaction.
- Community Building: Fostering a sense of community and engagement
- Data-Driven Decisions: Data insights for informed decision-making.
- Scalability: Potential to expand to more societies.

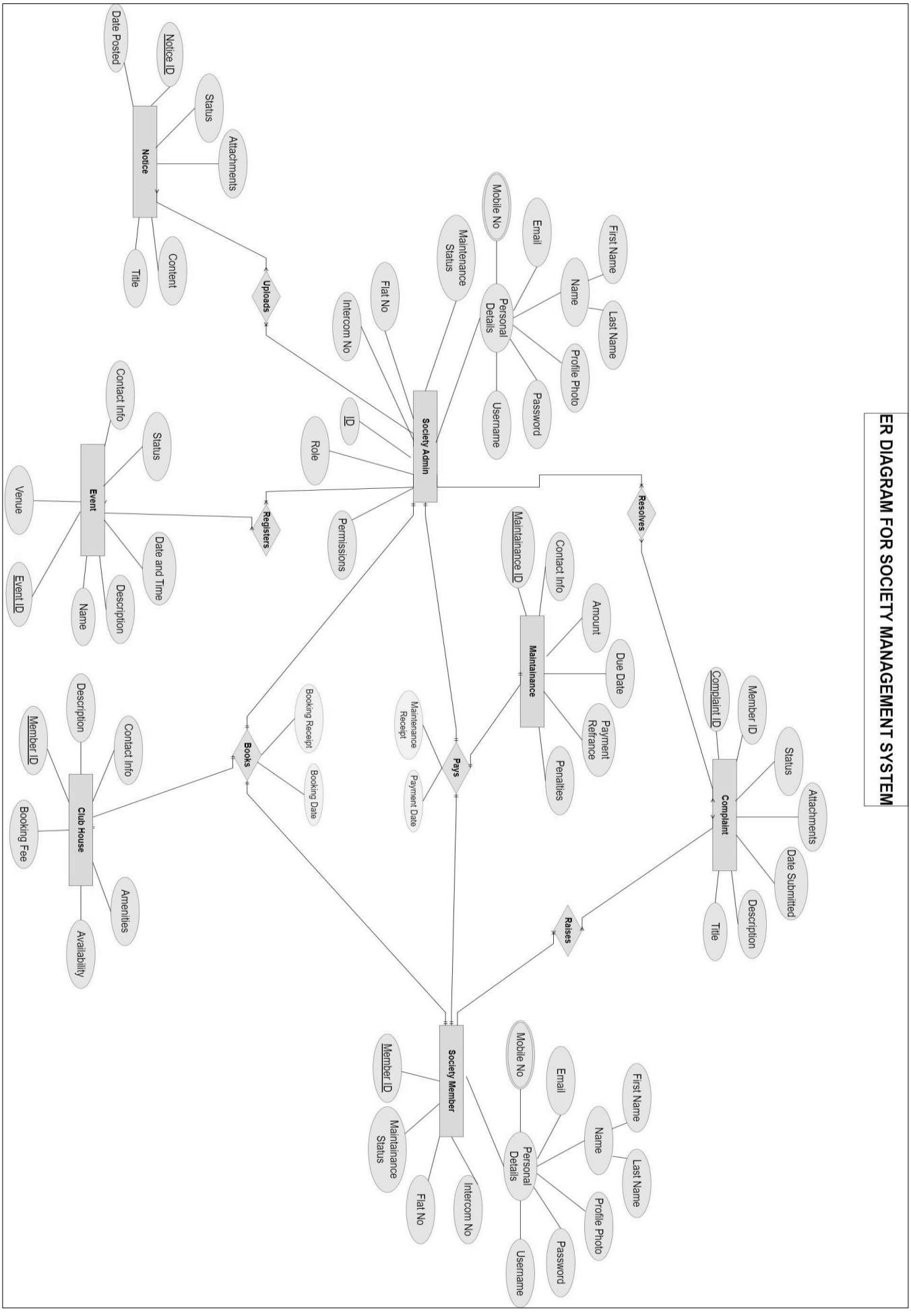
Statement Of Problem:-

1. Communication Challenges
2. Lack of Financial Transparency
3. Administrative Complexity

Conclusion :-

In conclusion, our project, tailored for Sentosa Enclave residential society, offers an efficient, user-friendly web-based solution for society management and communication. By enabling online maintenance payments, event updates, and seamless access to vital information, it enhances resident living. Anticipated outcomes encompass improved communication, reduced administrative workload, and data-driven decision-making.

ER DIAGRAM FOR SOCIETY MANAGEMENT SYSTEM



Practical No. 9

Aim: Implement queries related to order by and having clause.(To perform TCL and DCL commands.).

Theory:

- ## TCL Commands:

1. **Transaction:** A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. It is important for maintaining data consistency.

2. **TCL Commands**: TCL commands are used to manage transactions. The two primary TCL commands are:

3. **COMMIT:** This command is used to save all the changes made during the current transaction. It marks the successful end of a transaction.

4. **ROLLBACK:** This command is used to undo all the changes made during the current transaction and return the database to its previous state. It is used when an error or issue occurs during a transaction.

5. **SAVEPOINT:** TCL allows the use of savepoints within a transaction. A savepoint is a point in a transaction to which you can later roll back. You can use SAVEPOINT to create a savepoint within a transaction and ROLLBACK TO to revert to a specific savepoint.

- **DCL Commands:**

Data Control Language (DCL): DCL is a subset of SQL (Structured Query Language) commands used for managing access and permissions to database objects. DCL commands are focused on controlling who can access, modify, or manipulate data and database objects.

Key DCL Commands:
1. GRANT: The GRANT command is used to give specific privileges or permissions to users or roles on database objects, such as tables, views, procedures, and more. It allows you to specify what actions a user or role is allowed to perform on the specified objects.

2. REVOKE: The REVOKE command is used to take away previously granted privileges or permissions from users or roles. It is used to restrict or revoke access to database objects.

Queries:

- **TCL COMMAND:**

(1) COMMIT:

```
mysql> /*202203103510124*/
mysql> CREATE TABLE Students3 (
    ->     StudentID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> CREATE TABLE Courses3 (
    ->     CourseID INT PRIMARY KEY,
    ->     CourseName VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> INSERT INTO Students3 (StudentID, FirstName, LastName)
    -> VALUES
    ->     (1, 'John', 'Doe'),
    ->     (2, 'Jane', 'Smith'),
    ->     (3, 'Tom', 'Wilson');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO Courses3 (CourseID, CourseName)
    -> VALUES
    ->     (101, 'Mathematics'),
    ->     (102, 'History'),
    ->     (103, 'Science');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> /*202203103510124*/
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO Students3 (StudentID, FirstName, LastName)
    -> VALUES (4, 'Sarah', 'Johnson');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> UPDATE Courses3
    -> SET CourseName = 'English'
    -> WHERE CourseID = 102;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SELECT * FROM Students3;
+-----------+-----------+----------+
| StudentID | FirstName | LastName |
+-----------+-----------+----------+
|         1 | John      | Doe      |
|         2 | Jane      | Smith    |
|         3 | Tom       | Wilson   |
|         4 | Sarah     | Johnson  |
+-----------+-----------+----------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM Courses3;
+----------+-------------+
| CourseID | CourseName  |
+----------+-------------+
|      101 | Mathematics |
|      102 | English     |
|      103 | Science     |
+----------+-------------+
3 rows in set (0.00 sec)
```

(2) ROLLBACK:

```
mysql> /*202203103510124*/
mysql> CREATE TABLE Employees2 (
    ->     EmployeeID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     Salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> INSERT INTO Employees2 (EmployeeID, FirstName, LastName, Salary)
    -> VALUES
    ->     (1, 'John', 'Doe', 50000.00),
    ->     (2, 'Jane', 'Smith', 60000.00),
    ->     (3, 'Tom', 'Wilson', 55000.00);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> /*202203103510124*/
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> UPDATE Employees2
    -> SET Salary = 65000.00
    -> WHERE EmployeeID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> DELETE FROM Employees2
    -> WHERE EmployeeID = 3;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> SELECT * FROM Employees2;
+------------+-----------+----------+----------+
| EmployeeID | FirstName | LastName | Salary   |
+------------+-----------+----------+----------+
|          1 | John      | Doe      | 65000.00 |
|          2 | Jane      | Smith    | 60000.00 |
+------------+-----------+----------+----------+
2 rows in set (0.00 sec)

mysql>
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SELECT * FROM Employees2;
+------------+-----------+----------+----------+
| EmployeeID | FirstName | LastName | Salary   |
+------------+-----------+----------+----------+
|          1 | John      | Doe      | 50000.00 |
|          2 | Jane      | Smith    | 60000.00 |
|          3 | Tom       | Wilson   | 55000.00 |
+------------+-----------+----------+----------+
3 rows in set (0.00 sec)
```

(3) SAVEPOINT:

```
mysql> /*202203103510124*/
mysql> CREATE TABLE Students4 (
    ->     StudentID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     GPA DECIMAL(3, 2)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> INSERT INTO Students4 (StudentID, FirstName, LastName, GPA)
    -> VALUES
    ->     (1, 'John', 'Doe', 3.65),
    ->     (2, 'Jane', 'Smith', 3.80),
    ->     (3, 'Tom', 'Wilson', 3.55);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> /*202203103510124*/
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> UPDATE Students
    -> SET GPA = 3.95
    -> WHERE StudentID = 1;
ERROR 1054 (42S22): Unknown column 'GPA' in 'field list'
mysql>
mysql> SAVEPOINT before_delete;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELETE FROM Students
    -> WHERE StudentID = 3;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> SELECT * FROM Students;
+-----------+-----------+----------+------+--------+
| StudentID | FirstName | LastName | Age  | Gender |
+-----------+-----------+----------+------+--------+
|         1 | John      | Smith    |   22 | Male   |
|         2 | Emily     | Johnson  |   20 | Female |
|         4 | Sophia    | Davis    |   23 | Female |
+-----------+-----------+----------+------+--------+
3 rows in set (0.00 sec)

mysql>
mysql> ROLLBACK TO before_delete;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SELECT * FROM Students;
+-----------+-----------+----------+------+--------+
| StudentID | FirstName | LastName | Age  | Gender |
+-----------+-----------+----------+------+--------+
|         1 | John      | Smith    |   22 | Male   |
|         2 | Emily     | Johnson  |   20 | Female |
|         3 | Michael   | Brown    |   21 | Male   |
|         4 | Sophia    | Davis    |   23 | Female |
+-----------+-----------+----------+------+--------+
4 rows in set (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

(4) ROLLBACK TO:

```
mysql> /*202203103510124*/
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> UPDATE Students4
    -> SET GPA = 3.95
    -> WHERE StudentID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> SAVEPOINT before_delete;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELETE FROM Students4
    -> WHERE StudentID = 3;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> SELECT * FROM Students4;
+-----------+-----------+----------+------+
| StudentID | FirstName | LastName | GPA  |
+-----------+-----------+----------+------+
|         1 | John      | Doe      | 3.95 |
|         2 | Jane      | Smith    | 3.80 |
+-----------+-----------+----------+------+
2 rows in set (0.00 sec)

mysql>
mysql> ROLLBACK TO before_delete;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> SELECT * FROM Students4;
+-----------+-----------+----------+------+
| StudentID | FirstName | LastName | GPA  |
+-----------+-----------+----------+------+
|         1 | John      | Doe      | 3.95 |
|         2 | Jane      | Smith    | 3.80 |
|         3 | Tom       | Wilson   | 3.55 |
+-----------+-----------+----------+------+
3 rows in set (0.00 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

- **DCL Commands:**

(1) GRANT: GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

(2) REVOKE: REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

Conclusion: TCL is a scripting language used for controlling applications and systems, whereas DCL is a subset of SQL specifically designed for managing access to data within a database. Both TCL and DCL serve different purposes within the realm of computing, with TCL focusing on scripting and automation, while DCL manages data access control within databases.

Practical No. 11

Aim: Implement set operations, case statement and view queries.

Theory:

1. **Set Operations**:
   - Set operations are fundamental database operations used to manipulate data.
   - Common set operations include union, intersection, difference, and Cartesian product.
   - These operations help combine or compare data from one or more database tables.
2. **Case Statement**:
   - A case statement is a conditional statement used in SQL to perform conditional logic.
   - It allows you to perform different actions based on specified conditions.
   - Commonly used for data transformation and customization in query results.
3. **View Queries**:
   - A view is a virtual table created by a query that can be used like a regular table.
   - View queries allow you to encapsulate complex queries into reusable objects.
   - They enhance security by limiting direct access to underlying tables and simplify query composition.

Queries:

(1) Implement “IF” Condition in Query

a) Put if condition on “price” attribute (IF Else)

```
mysql> SELECT IF((SELECT PRICE FROM product WHERE PRODUCT_NO=120) = (SELECT PRICE FROM product WHERE PRODUCT_NO=122),'YES','NO')AS IF_STATEMENT;
+--------------+
| IF_STATEMENT |
+--------------+
| NO           |
+--------------+
1 row in set (0.01 sec)

mysql> /*202203103510203*/
```

b) Try nested IF on the “price” attribute.

c) Display Price and quantity and there rating with “High” , “Medium” and “Low”

```
mysql> /*202203103510124*/
mysql> SELECT
    ->     IF(
    ->         (SELECT MAX(PRICE) FROM PRODUCT WHERE PRODUCT_NO = 120) = (SELECT MAX(PRICE) FROM PRODUCT WHERE PRODUCT_NO = 122),
    ->         'YES',
    ->         IF(
    ->             (SELECT MAX(PRICE) FROM PRODUCT WHERE PRODUCT_NO = 120) < (SELECT MAX(PRICE) FROM PRODUCT WHERE PRODUCT_NO = 122),
    ->             'LOW',
    ->             'HIGH'
    ->         )
    ->     ) AS NESTED_IF_STATEMENT;
+---------------------+
| NESTED_IF_STATEMENT |
+---------------------+
| HIGH                |
+---------------------+
```

(2) Create a view VProduct of the product‟s id, description and price.

```
mysql> /*202203103510124*/
mysql> CREATE VIEW VPRODUCT AS SELECT PRODUCT_NO , DESCRIPTION, PRICE FROM PRODUCT;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT * FROM VPRODUCT;
+------------+-----------------+----------+
| PRODUCT_NO | DESCRIPTION     | PRICE    |
+------------+-----------------+----------+
|        120 | REDUCER         |  1200.00 |
|        121 | PLATE           |  2000.00 |
|        122 | HANDLE          |   700.00 |
|        124 | WIDGET REMOVER  |   900.00 |
|        136 | SIZE WIDGET     |  1000.00 |
|        137 | SIZE WIDGET     | 15000.00 |
+------------+-----------------+----------+
```

(3) Create a view of Vorder to get orders (order_id, product_id, description, customer_name,quantity) placed by customer who belongs to “BRIXTON”.

```
mysql> CREATE VIEW VORDER AS SELECT O.CORDER_NO AS ORDER_ID, OL.PRODUCT_NO, P.DESCRIPTION, C.NAME A
NE OL ON O.CORDER_NO = OL.CORDER_NO JOIN PRODUCT P ON OL.PRODUCT_NO = P.PRODUCT_NO JOIN CUSTOMER C
RIXTON';
Query OK, 0 rows affected (0.04 sec)

mysql> /*202203103510124*/
mysql> CREATE VIEW VORDER AS SELECT O.CORDER_NO AS ORDER_ID, OL.PRODUCT_NO, P.DESCRIPTION, C.NAME A
NE OL ON O.CORDER_NO = OL.CORDER_NO JOIN PRODUCT P ON OL.PRODUCT_NO = P.PRODUCT_NO JOIN CUSTOMER C
RIXTON';
ERROR 1050 (42S01): Table 'VORDER' already exists
mysql> SELECT * FROM VORDER;
+----------+------------+-------------+---------------+----------+
| ORDER_ID | PRODUCT_NO | DESCRIPTION | CUSTOMER_NAME | QUANTITY |
+----------+------------+-------------+---------------+----------+
|      203 |        122 | HANDLE      | DRAKE         |       20 |
|      204 |        136 | SIZE WIDGET | GARRY SMITH   |       30 |
+----------+------------+-------------+---------------+----------+
```

(4) Insert a new row in VProduct 135, „Sofa“ and 35000.

```
mysql> /*202203103510124*/
mysql> INSERT INTO VProduct (PRODUCT_NO, DESCRIPTION, PRICE)
    -> VALUES (135, 'Sofa', 35000);
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM VProduct;
+------------+----------------+----------+
| PRODUCT_NO | DESCRIPTION    | PRICE    |
+------------+----------------+----------+
|        120 | REDUCER        |  1200.00 |
|        121 | PLATE          |  2000.00 |
|        122 | HANDLE         |   700.00 |
|        124 | WIDGET REMOVER |   900.00 |
|        135 | Sofa           | 35000.00 |
|        136 | SIZE WIDGET    |  1000.00 |
|        137 | SIZE WIDGET    | 15000.00 |
+------------+----------------+----------+
```

(5) Update product‟s quantity to 25 which is brought by customer „DRAKE‟ in Vorder.

```
mysql> UPDATE VORDER SET QUANTITY = 25 WHERE CUSTOMER_NAME = 'DRAKE';
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM VORDER;
+----------+------------+-------------+---------------+----------+
| ORDER_ID | PRODUCT_NO | DESCRIPTION | CUSTOMER_NAME | QUANTITY |
+----------+------------+-------------+---------------+----------+
|      203 |        122 | HANDLE      | DRAKE         |       25 |
|      204 |        136 | SIZE WIDGET | GARRY SMITH   |       30 |
+----------+------------+-------------+---------------+----------+
2 rows in set (0.00 sec)

mysql> /*202203103510124*/
```

(6) Delete details of product id 121 from VProduct.

```
mysql> DELETE FROM VProduct WHERE PRODUCT_NO = 121;
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM VProduct;
+------------+----------------+----------+
| PRODUCT_NO | DESCRIPTION    | PRICE    |
+------------+----------------+----------+
|        120 | REDUCER        |  1200.00 |
|        122 | HANDLE         |   700.00 |
|        124 | WIDGET REMOVER |   900.00 |
|        135 | Sofa           | 35000.00 |
|        136 | SIZE WIDGET    |  1000.00 |
|        137 | SIZE WIDGET    | 15000.00 |
+------------+----------------+----------+
6 rows in set (0.00 sec)

mysql> /*202203103510124*/
```

(7) Delete view VProduct.

```
mysql> /*202203103510124*/
mysql> DROP VIEW VProduct;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM VProduct;
ERROR 1146 (42S02): Table 'Tutorial1_Ankit.VProduct' doesn't exist
```

(8) Display the name of all customers and all suppliers with their id by using union operator.

```
mysql> /*202203103510124*/
mysql> SELECT NAME, CUSTOMER_NO
    -> FROM CUSTOMER
    -> UNION
    -> SELECT NAME, SUPPLIER_NO
    -> FROM SUPPLIER;
+--------------+-------------+
| NAME         | CUSTOMER_NO |
+--------------+-------------+
| GARRY SMITH  |          10 |
| PATEL        |          20 |
| DRAKE        |          30 |
| BOB SMITH    |          40 |
| JAMES        |          50 |
| NORTON       |          60 |
| JOHN MICHAEL |          70 |
| MICHAEL      |        1001 |
| RINGWORLD    |        1002 |
| BABYLON      |        1003 |
| JOHN         |        1004 |
| SMITH        |        1005 |
+--------------+-------------+
```

(9) List products which are not bought by any customer using the minus operator.

```
mysql> /*202203103510124*/
mysql> SELECT PRODUCT_NO, DESCRIPTION
    -> FROM PRODUCT
    -> MINUS
    -> SELECT PRODUCT_NO, DESCRIPTION
    -> FROM Vorder;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use n
ear 'SELECT PRODUCT_NO, DESCRIPTION
FROM Vorder' at line 4
```

(10) Give the name of suppliers who are also customers.

```
mysql> /*202203103510124*/
mysql> SELECT DISTINCT S.NAME AS SUPPLIER_NAME
    -> FROM SUPPLIER S
    -> JOIN CUSTOMER C ON S.NAME = C.NAME;
Empty set (0.00 sec)
```

Conclusion:

Incorporated set operations, case statements, and view queries to enhance data manipulation and retrieval capabilities. These features have significantly improved the flexibility and efficiency of database queries, allowing for more complex and customized data processing.