LAB CYCLE 1
Experiment No: 1
Date:21/02/24
Familiarization of DDL Commands

Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

Some common DDL commands:

- Create Database
  Syntax: create database database_name;
- Drop Database
  Syntax: drop database [if exists] database_name;
- Create Table
  Syntax: create table *table_name* (*column1 datatype*,....);
- Drop Table
  Syntax: drop table table_name;
- Alter Table Add Column
  Syntax: alter table *table_name* add *column_name datatype*; o
  Drop Column
  Syntax: alter table *table_name* drop column *column_name*; o
  Rename Column

Syntax: alter table *table_name* rename column *old_name* to *new_name*; o
Modify Column properties
Syntax: alter table *table_name* modify *column_name datatype*;

A. Consider the database for a college. Write SQL commands to implement the following:

1. Create a database

   SQL : CREATE DATABASE CollegeDB;

   Output:

```
Query OK, 1 row affected (0.02 sec)
```

2. Select the current database

   SQL : USE CollegeDB;

   Output:

```
Database changed
```

3. Create the following tables:

a) Student (roll_no integer, name varchar, dob date, address text, phone_no varchar, blood_grp varchar)

SQL : CREATE TABLE Student(roll_no INT PRIMARY KEY,name VARCHAR(255),dob DATE,address TEXT, phone_no VARCHAR(15),blood_grp VARCHAR(5));

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

b) Course (Course_id integer, Course_name varchar, course_duration integer)

SQL : CREATE TABLE Course(Course_id INT,Course_name VARCHAR(255),Course_duration INT);

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

4. List all tables in the current database.

SQL : SHOW TABLES;

Output:

```
+----------------------+
| Tables_in_CollegeDB |
+----------------------+
| Course               |
| Student              |
+----------------------+
2 rows in set (0.00 sec)
```

5. Display the structure of the Student table.

SQL : DESCRIBE Student;

Output:

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| roll_no  | int          | NO   | PRI | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| dob      | date         | YES  |     | NULL    |       |
| address  | text         | YES  |     | NULL    |       |
| phone_no | varchar(15)  | YES  |     | NULL    |       |
| blood_grp| varchar(5)   | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

6. Drop the column blood_grp from Student table.

SQL : ALTER TABLE Student DROP COLUMN blood_grp;

Output:

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| roll_no  | int          | NO   | PRI | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| dob      | date         | YES  |     | NULL    |       |
| address  | text         | YES  |     | NULL    |       |
| phone_no | varchar(15)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

7. Add a new column Adar_no with domain number to the table Student.

SQL : ALTER TABLE Student ADD Adar_no INT;

Output:

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| roll_no  | int          | NO   | PRI | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| dob      | date         | YES  |     | NULL    |       |
| address  | text         | YES  |     | NULL    |       |
| phone_no | varchar(15)  | YES  |     | NULL    |       |
| adhaar_no| int          | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

8. Change the datatype of phone_no from varchar to int

SQL : ALTER TABLE Student MODIFY COLUMN phone_no INT;

Output:

```
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| roll_no  | int          | NO   | PRI | NULL    |       |
| name     | varchar(255) | YES  |     | NULL    |       |
| dob      | date         | YES  |     | NULL    |       |
| address  | text         | YES  |     | NULL    |       |
| phone_no | int          | YES  |     | NULL    |       |
| adhaar_no| int          | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

9. Drop the tables.

SQL : DROP TABLE Student,Course;

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

10. Delete the database.

SQL : DROP DATABASE CollegeDB;

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

Date:21/02/24

B. Consider the database for an organization. Write SQL commands to implement the following:

1. Create a database

SQL : CREATE DATABASE OrganizationDB;

Output:

```
Query OK, 1 row affected (0.01 sec)
```

2. Select the current database

SQL : USE OrganizationDB;

Output:

```
Database changed
```

3. Create the following tables:

a) Employee (emp_no varchar, emp_name varchar, dob date, address text, mobile_no integer, dept_no varchar, salary integer)

SQL : CREATE TABLE Employee(emp_no VARCHAR(10),emp_name VARCHAR(255),dob DATE,address TEXT,mobile_no INT,dept_no VARCHAR(10),salary INT);

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

b) Department (dept_no varchar, dept_name varchar, location varchar)

SQL : CREATE TABLE Department(dept_no VARCHAR(10),dept_name VARCHAR(255),location VARCHAR(255));

Output:

```
Query OK, 0 rows affected (0.03 sec)
```

4. List all tables in the current database.

SQL : SHOW TABLES;

Output:

```
+--------------------------+
| Tables_in_OrganizationDB |
+--------------------------+
| Department               |
| Employee                 |
+--------------------------+
2 rows in set (0.00 sec)
```

5. Display the structure of the Employee table and Department table.

SQL : DESCRIBE Employee;

DESCRIBE Department;

Output:

Field	Type	Null	Key	Default	Extra
emp_no	varchar(10)	YES		NULL	
emp_name	varchar(255)	YES		NULL	
dob	date	YES		NULL	
address	text	YES		NULL	
mobile_no	int	YES		NULL	
dept_no	varchar(10)	YES		NULL	
salary	int	YES		NULL	

7 rows in set (0.00 sec)

Field	Type	Null	Key	Default	Extra
dept_no	varchar(10)	YES		NULL	
dept_name	varchar(255)	YES		NULL	
location	varchar(255)	YES		NULL	

3 rows in set (0.00 sec)

6. Add a new column 'Designation' to the table Employee.

SQL : ALTER TABLE Employee ADD Designation VARCHAR(255);

Output:

Field	Type	Null	Key	Default	Extra
emp_no	varchar(10)	YES		NULL	
emp_name	varchar(255)	YES		NULL	
dob	date	YES		NULL	
address	text	YES		NULL	
mobile_no	int	YES		NULL	
dept_no	varchar(10)	YES		NULL	
salary	int	YES		NULL	
Designation	varchar(255)	YES		NULL	

7. Drop the column ‘location’ from Department table.

SQL : ALTER TABLE Department DROP COLUMN location;

Output:

```
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| dept_no   | varchar(10)  | YES  |     | NULL    |       |
| dept_name | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Result:

Query is executed successfully and has been executed

LAB CYCLE 1

Experiment No: 2

Date:06/03/24

Familiarization of SQL Constraints.

Constraints are used to specify rules for data in a table. Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

- With Create Table

  Syntax: create table table_name (column1 datatype constraint,…);

- With Alter Table

  Syntax: alter table persons add constraint <name> column name Different constraints

- NOT NULL - Ensures that a column cannot have a NULL value Syntax: create table table_name (column_name datatype NOT NULL,..);
- UNIQUE - Ensures that all values in a column are different

  Syntax: create table table_name (column_name datatype UNIQUE,..);
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

  Syntax: create table table_name (column_name datatype PRIMARY KEY,..);
- FOREIGN KEY - a foreign key is a field or a column that is used to establish a link between two tables.

  Syntax: create table table_name (column_list,.., foreign key (column_list) references parent_table(column_list));
- CHECK - Ensures that all values in a column satisfies a specific condition
  Syntax: create table table_name (column_name datatype check (expression),..);
- DEFAULT - Sets a default value for a column when no value is specified.

  Syntax: create table table_name (column_name datatype default value,..);

1. Create new table Persons with attributes PersonID (integer, PRIMARY KEY), Name (varchar , NOT NULL), Aadhar (Number, NOT NULL, UNIQUE), Age (integer , CHECK>18).

SQL: create table Persons(PersonID int, Name varchar(30), Aadhar numeric NOT NULL UNIQUE, Age int CHECK(Age>18));

Output:

```
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| PersonID | int           | NO   | PRI | NULL    |       |
| Name     | varchar(30)   | NO   |     | NULL    |       |
| Aadhar   | decimal(10,0) | YES  | UNI | NULL    |       |
| Age      | int           | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

2.Create table Orders with attributes OrderID (PRIMARY KEY), OrderNumber(NOT NULL) and PersonID(set FOREIGN KEY on attribute PersonID referencing the column PersonId of Person table)

SQL: create table Orders(OrderID int PRIMARY KEY, OrderNumber int NOT NULL, PersonID int, FOREIGN KEY (PersonID) REFERENCES Persons (PersonID));

Output:

```
+-------------+------+------+-----+---------+-------+
| Field       | Type | Null | Key | Default | Extra |
+-------------+------+------+-----+---------+-------+
| OrderID     | int  | NO   | PRI | NULL    |       |
| OrderNumber | int  | NO   |     | NULL    |       |
| PersonID    | int  | YES  | MUL | NULL    |       |
+-------------+------+------+-----+---------+-------+
```

3. Display the structure of Persons tables.

SQL: desc Persons;

Output:

```
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| PersonID | int           | NO   | PRI | NULL    |       |
| Name     | varchar(30)   | NO   |     | NULL    |       |
| Aadhar   | decimal(10,0) | YES  | UNI | NULL    |       |
| Age      | int           | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

4. Display the structure of Orders tables.

SQL: desc Orders;

Output:

```
+-------------+------+------+-----+---------+-------+
| Field       | Type | Null | Key | Default | Extra |
+-------------+------+------+-----+---------+-------+
| OrderID     | int  | NO   | PRI | NULL    |       |
| OrderNumber | int  | NO   |     | NULL    |       |
| PersonID    | int  | YES  | MUL | NULL    |       |
+-------------+------+------+-----+---------+-------+
```

5. Add emp_no as the primary key of the table Employee.

SQL:alter table Employee add PRIMARY KEY (emp_no);

Output:

```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| emp_no      | varchar(5)  | NO   | PRI | NULL    |       |
| emp_name    | varchar(20) | YES  |     | NULL    |       |
| dob         | date        | YES  |     | NULL    |       |
| address     | text        | YES  |     | NULL    |       |
| mobile_no   | int         | YES  |     | NULL    |       |
| dept_no     | varchar(5)  | YES  |     | NULL    |       |
| salary      | int         | YES  |     | NULL    |       |
| Designation | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

6.Add dept_no as the primary key of the table Department.
SQL: alter table Department add PRIMARY KEY (dept_no);
Output:

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| dept_no   | varchar(5)  | NO   | PRI | NULL    |       |
| dept_name | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

7. Add dept_no in Employee table as the foreign key reference to the table Department with on delete cascade.

SQL: ALTER table EMPLOYEE add FOREIGN KEY(dept_no) REFERENCES DEPARTMENT(dept_no) on delete cascade;
Output:

```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| emp_no      | varchar(5)  | NO   | PRI | NULL    |       |
| emp_name    | varchar(20) | YES  |     | NULL    |       |
| dob         | date        | YES  |     | NULL    |       |
| address     | text        | YES  |     | NULL    |       |
| mobile_no   | int         | YES  |     | NULL    |       |
| dept_no     | varchar(5)  | YES  | MUL | NULL    |       |
| salary      | int         | YES  |     | NULL    |       |
| Designation | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

8. Drop the primary key of the table Orders.

SQL: alter table Orders drop PRIMARY KEY;

Output:

```
+-------------+------+------+-----+---------+-------+
| Field       | Type | Null | Key | Default | Extra |
+-------------+------+------+-----+---------+-------+
| OrderID     | int  | NO   |     | NULL    |       |
| OrderNumber | int  | NO   |     | NULL    |       |
| PersonID    | int  | YES  | MUL | NULL    |       |
+-------------+------+------+-----+---------+-------+
```

Output

Query is executed successfully and has been executed

LAB CYCLE 1
Experiment No: 3
Date:13/03/24
Familiarization of DML Commands.

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

- Insert command
  Syntax: insert into tablename (columnname1, columnname2, ...) values (column1_value, column2_value, ...);
- Update command
  Syntax: UPDATE tablename SET column1 = new_value1, column2 = new_value2... WHERE search condition;
- Select command
  Syntax: select [distinct] <select-list> from <from-list> [where <qualification>]
- Delete command
  Syntax: delete from table_name where some_condition;

1. Add at least 10 rows into the table Employee and Department.

   SQL:

   -->>INSERT INTO Department (dept_no, dept_name, location) VALUES
   ('D01', 'Finance', 'New York'),
   ('D02', 'Marketing', 'Los Angeles'),
   ('D03', 'Human Resources', 'Chicago'),
   ('D04', 'Operations', 'Houston'),
   ('D05', 'IT', 'San Francisco');

   -->>INSERT INTO Employee (emp_no, emp_name, dob, address, mobile_no, dept_no, salary, designation) VALUES
   ('emp1', 'John Doe', '1990-05-15', '123 Main St, City', '1234567890', 'D01', 30000, 'Manager'),
   ('emp2', 'Jane Smith', '1992-08-20', '456 Elm St, Town', '9876543210', 'D02', 19000, 'Marketing Specialist'),
   ('emp3', 'Michael Johnson', '1985-03-10', '789 Oak St, Village', '5554443333', 'D03', 40000, 'HR Coordinator'),
   ('emp4', 'Emily Brown', '1988-11-25', '567 Pine St, Hamlet', '2223334444', 'D01', 32000, 'Computer Assistant'),
   ('emp5', 'Alice Wilson', '1993-02-28', '890 Cedar St, Suburb', '3332221111', 'D02', 38000, 'Marketing Associate'),

('emp6', 'Jessica Lee', '1991-07-12', '234 Maple St, Rural', '7778889999', 'D03', 175000, 'HR Specialist'),
('emp7', 'David Garcia', '1987-09-05', '678 Birch St, Countryside', '9998887777', 'D04', 200000, 'Operations Manager'),
('emp8', 'Sophia Martinez', '1990-12-30', '345 Walnut St, Urban', '1112223333', 'D02', 37000, 'Marketing Analyst'),
('emp9', 'Matthew Anderson', '1989-04-18', '901 Cherry St, Coastal', '6665554444', 'D05', 7000, 'Intern'),
('emp10', 'Olivia Taylor', '1994-06-22', '432 Spruce St, Lakeside', '8889990000', 'D01', 19500, 'Finance Assistant');

2. Display all the records from the above tables.

SQL:

-->> SELECT * FROM Employee;

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	1234567890	D01	30000	Manager
emp2	Jane Smith	1992-08-20	456 Elm St, Town	9876543210	D02	19000	Marketing Specialist
emp3	Michael Johnson	1985-03-10	789 Oak St, Village	5554443333	D03	40000	HR Coordinator
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant
emp5	Alice Wilson	1993-02-28	890 Cedar St, Suburb	3332221111	D02	38000	Marketing Associate
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp8	Sophia Martinez	1990-12-30	345 Walnut St, Urban	1112223333	D02	37000	Marketing Analyst
emp9	Matthew Anderson	1989-04-18	901 Cherry St, Coastal	6665554444	D05	7000	Intern
emp10	Olivia Taylor	1994-06-22	432 Spruce St, Lakeside	8889990000	D01	19500	Finance Assistant

SQL:

-->> SELECT * FROM Department;

Output:

dept_no	dept_name	location
D01	Finance	New York
D02	Marketing	Los Angeles
D03	Human Resources	Chicago
D04	Operations	Houston
D05	IT	San Francisco

3. Display the emp_no and name of employees from department no ‘D02’.

SQL:

-->> SELECT emp_no, emp_name FROM Employee WHERE dept_no = 'D02';

Output:

```
emp_no |  emp_name
-------|-----------------
emp2   | Jane Smith
emp5   | Alice Wilson
emp8   | Sophia Martinez
```

4. Display emp_no, emp_name , designation, deptno and salary of employees in the descending order of salary.

   SQL:
   --> SELECT emp_no, emp_name, dept_no, salary, designation FROM Employee ORDER BY salary DESC;

   Output:

```
emp_no |  emp_name       | dept_no | salary | designation
-------|-----------------|---------|--------|-------------------
emp7   | David Garcia    | D04     | 200000 | Operations Manager
emp6   | Jessica Lee     | D03     | 175000 | HR Specialist
emp5   | Alice Wilson    | D02     | 38000  | Marketing Associate
emp8   | Sophia Martinez | D02     | 37000  | Marketing Analyst
emp3   | Michael Johnson | D03     | 40000  | HR Coordinator
emp4   | Emily Brown     | D01     | 32000  | Computer Assistant
emp1   | John Doe        | D01     | 30000  | Manager
emp10  | Olivia Taylor   | D01     | 19500  | Finance Assistant
emp2   | Jane Smith      | D02     | 19000  | Marketing Specialist
emp9   | Matthew Anderson| D05     | 7000   | Intern
```

5. Display the emp_no , name of employees whose salary is between 2000 and 5000

   SQL:
   --> SELECT emp_no, emp_name FROM Employee WHERE salary BETWEEN 2000 AND 5000;

   Output:

```
emp_no |  emp_name       |
-------|-----------------|
emp9   | Matthew Anderson|
```

6. Display the designations without duplicate values
   SQL:
   ----> SELECT DISTINCT design FROM EMPLOYEE;

Output:

designation
Manager
Marketing Specialist
HR Coordinator
Computer Assistant
Marketing Associate
HR Specialist
Operations Manager
Marketing Analyst
Intern
Finance Assistant

7. Change the salary of employee to 200000 whose designation is manager
   SQL:
   ---> UPDATE Employee SET salary = 200000 WHERE designation = 'Manager';

   Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	1234567890	D01	200000	Manager
emp2	Jane Smith	1992-08-20	456 Elm St, Town	9876543210	D02	19000	Marketing Specialist
emp3	Michael Johnson	1985-03-10	789 Oak St, Village	5554443333	D03	40000	HR Coordinator
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant
emp5	Alice Wilson	1993-02-28	890 Cedar St, Suburb	3332221111	D02	38000	Marketing Associate
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp8	Sophia Martinez	1990-12-30	345 Walnut St, Urban	1112223333	D02	37000	Marketing Analyst
emp9	Matthew Anderson	1989-04-18	901 Cherry St, Coastal	6665554444	D05	7000	Intern
emp10	Olivia Taylor	1994-06-22	432 Spruce St, Lakeside	8889990000	D01	19500	Finance Assistant

8. Change the mobile number of employees named John

   SQL:
   UPDATE Employee SET mobile_no = '7994924584' WHERE emp_name LIKE 'John%';
   Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	7994924584	D01	200000	Manager
emp2	Jane Smith	1992-08-20	456 Elm St, Town	9876543210	D02	19000	Marketing Specialist
emp3	Michael Johnson	1985-03-10	789 Oak St, Village	5554443333	D03	40000	HR Coordinator
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant
emp5	Alice Wilson	1993-02-28	890 Cedar St, Suburb	3332221111	D02	38000	Marketing Associate
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp8	Sophia Martinez	1990-12-30	345 Walnut St, Urban	1112223333	D02	37000	Marketing Analyst
emp9	Matthew Anderson	1989-04-18	901 Cherry St, Coastal	6665554444	D05	7000	Intern
emp10	Olivia Taylor	1994-06-22	432 Spruce St, Lakeside	8889990000	D01	19500	Finance Assistant

9. Delete all employees whose salary is equal to Rs.7000

SQL:
--->> DELETE FROM Employee WHERE salary = 7000;

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	7994924584	D01	200000	Manager
emp2	Jane Smith	1992-08-20	456 Elm St, Town	9876543210	D02	19000	Marketing Specialist
emp3	Michael Johnson	1985-03-10	789 Oak St, Village	5554443333	D03	40000	HR Coordinator
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant
emp5	Alice Wilson	1993-02-28	890 Cedar St, Suburb	3332221111	D02	38000	Marketing Associate
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp8	Sophia Martinez	1990-12-30	345 Walnut St, Urban	1112223333	D02	37000	Marketing Analyst
emp10	Olivia Taylor	1994-06-22	432 Spruce St, Lakeside	8889990000	D01	19500	Finance Assistant

10. Retrieve the name, mobile number of all employees whose name start with “A”.
SQL:
-->>SELECT emp_name, mobile_no FROM Employee WHERE emp_name LIKE 'A%';

Output:

emp_name	mobile_no
Alice Wilson	3332221111

11. Display the details of the employee whose name has at least three characters and salary greater than 20000.

SQL:
--->> SELECT * FROM Employee WHERE LENGTH(emp_name) >= 3 AND salary > 20000;

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	7994924584	D01	200000	Manager
emp3	Michael Johnson	1985-03-10	789 Oak St, Village	5554443333	D03	40000	HR Coordinator
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant
emp5	Alice Wilson	1993-02-28	890 Cedar St, Suburb	3332221111	D02	38000	Marketing Associate
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp8	Sophia Martinez	1990-12-30	345 Walnut St, Urban	1112223333	D02	37000	Marketing Analyst

12. Display the details of employees with empid ‘emp1’, ‘emp2’ and ‘emp6’.

SQL:
--->>SELECT * FROM Employee WHERE emp_no IN ('emp1','emp2','emp3');

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	7994924584	D01	200000	Manager
emp2	Jane Smith	1992-08-20	456 Elm St, Town	9876543210	D02	19000	Marketing Specialist
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist

13. Display employee name and employee id of those who have salary between 120000 and 300000.

SQL:
SELECT emp_name, emp_no FROM Employee WHERE salary BETWEEN 120000 AND 300000;

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp7	David Garcia	1987-09-05	678 Birch St, Countryside	9998887777	D04	200000	Operations Manager
emp6	Jessica Lee	1991-07-12	234 Maple St, Rural	7778889999	D03	175000	HR Specialist

14. Display the details of employees whose designation is ‘Manager’ or ‘Computer Assistant’.

SQL:
-->>SELECT * FROM Employee WHERE designation IN ('Manager', 'Computer Assistant');

Output:

emp_no	emp_name	dob	address	mobile_no	dept_no	salary	designation
emp1	John Doe	1990-05-15	123 Main St, City	7994924584	D01	200000	Manager
emp4	Emily Brown	1988-11-25	567 Pine St, Hamlet	2223334444	D01	32000	Computer Assistant

15. Displays how many employees work for each department.
SQL:
-->>SELECT dept_no, COUNT(*) AS employee_count FROM Employee GROUP BY dept_no;

Output:

dept_no	employee_count
D01	3
D02	2
D03	2
D04	1
D05	1

16. Displays average salary of employees in each department.

SQL:

-->>SELECT dept_no, AVG(salary) AS average_salary FROM Employee
GROUP BY dept_no;

Output:

dept_no	average_salary
D01	87916.6667
D02	31333.3333
D03	105000.0000
D04	200000.0000

17. Displays total salary of employees in each department.

SQL:

SELECT dept_no, SUM(salary) AS total_salary FROM Employee
GROUP BY dept_no;

Output:

dept_no	total_salary
D01	278500
D02	110000
D03	217000
D04	200000

18. Displays top and lower salary of employees in each department.

SQL:

SELECT dept_no, MAX(salary) AS top_salary, MIN(salary) AS lowest_salary
FROM Employee
GROUP BY dept_no;

Output:

dept_no	top_salary	lowest_salary
D01	200000	19500
D02	38000	19000
D03	175000	40000
D04	200000	200000

19. Displays average salary of employees in all departments except department with department number ‘D05’.

SQL:

-->>SELECT AVG(salary) AS average_salary FROM Employee
WHERE dept_no <> 'D05';

Output:

```
average_salary
---------------
100695.4545
```

20. Displays average salary of employees in all departments except department with department number ‘D01’ and average salary greater than 20000 in the ascending order of average salary.

SQL:
-->>SELECT dept_no, AVG(salary) AS average_salary
FROM Employee
WHERE dept_no <> 'D01'
GROUP BY dept_no
HAVING AVG(salary) > 20000
ORDER BY AVG(salary) ASC;

Output:

```
dept_no | average_salary
--------|---------------
D03     | 105000.0000
```

Result:

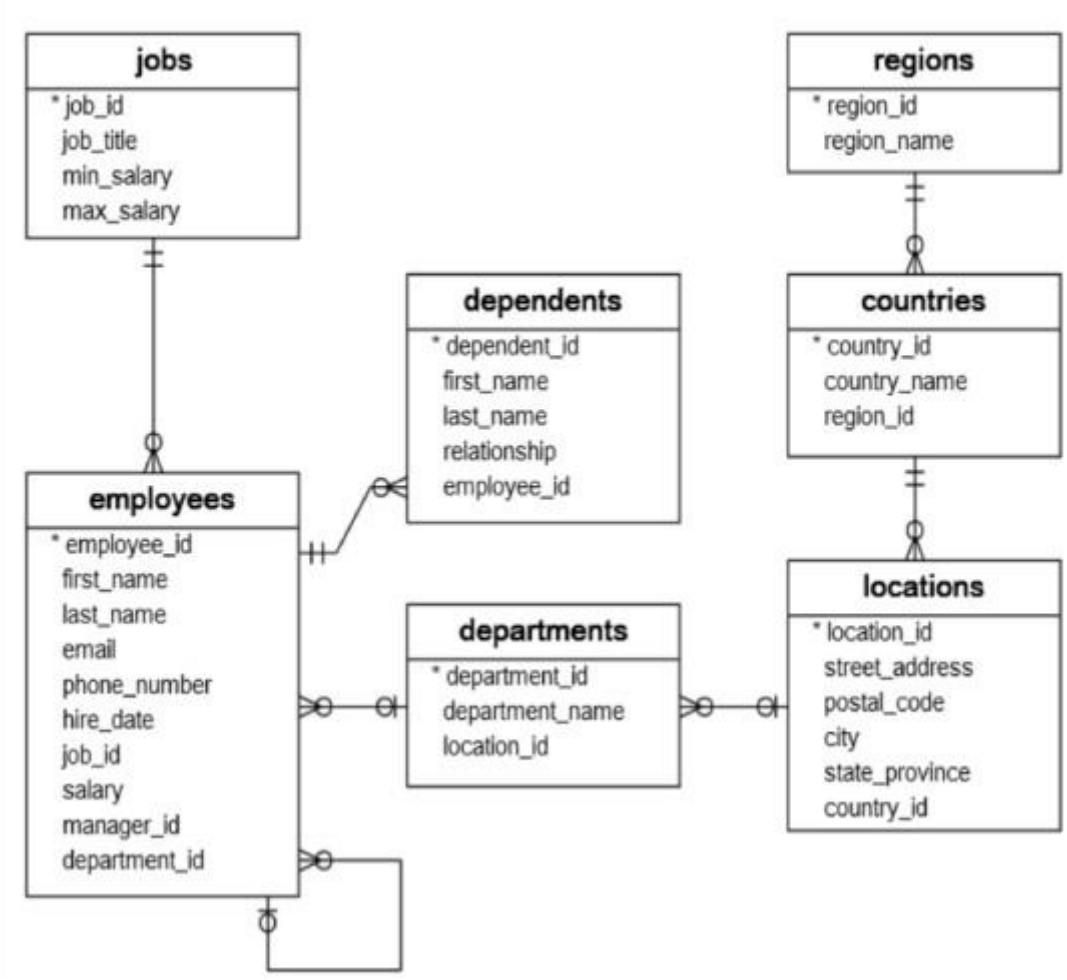
Query is executed successfully and has been executed

LAB CYCLE 2
Experiment No: 4
Date:20/03/24
Familiarization of Subquery, Joins, Views and Set Operations.

Subqueries are queries nested within other queries, allowing for more complex conditions or calculations. Joins in SQL combine data from two or more tables based on a related column between them. Common types include INNER JOIN, LEFT JOIN, and RIGHT JOIN. Views are virtual tables generated from the result set of a SELECT query. They don't store data themselves but provide a way to simplify complex queries. Set operations—Union, Intersect, and Difference—are used to combine or compare the results of two or more SELECT queries. UNION merges the results of multiple queries into a single result set, while INTERSECT returns only the rows common to all queries. Difference returns rows from the first query that are not found in the second query.

- Subquery
  Syntax: SELECT column1, column2, ...FROM table_name WHERE column_name OPERATOR (SELECT column_name FROM table_name WHERE condition);
- Join operation
  Syntax: SELECT table1.column1, table1.column2, table2.column1, ... FROM table1 JOIN table2 ON table1.common_column = table2.common_column;
- View Creation
  Syntax: CREATE VIEW view_name AS SELECT column1, column2, ...FROM table_name WHERE condition;
- Set Operations - Union
  Syntax: SELECT column1, column2, ... FROM table1 UNION SELECT column1, column2, ... FROM table2;
- Set Operations – Intersection
  Syntax: SELECT column1, column2, ... FROM table1 INTERSECT SELECT column1, column2, ... FROM table2;
- Set Operations – Set Difference
  Syntax: SELECT column1, column2, ... FROM table1 MINUS SELECT column1, column2, ... FROM table2;

Consider the following Database Schema



1. Create table regions.

SQL : CREATE TABLE regions (region_id INT PRIMARY KEY,region_name VARCHAR(255));
-->> DESC regions;

Output:

```
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| region_id   | int          | NO   | PRI | NULL    |       |
| region_name | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

2. Create table countries **.**

SQL : CREATE TABLE countries (country_id VARCHAR(2) PRIMARY KEY,country_name VARCHAR(255),region_id INT,FOREIGN KEY (region_id) REFERENCES regions(region_id));
-->> DESC countries;

Output :

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| country_id   | varchar(2)   | NO   | PRI | NULL    |       |
| country_name | varchar(255) | YES  |     | NULL    |       |
| region_id    | int          | YES  | MUL | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

3. Create table locations **.**

SQL : CREATE TABLE locations (location_id INT PRIMARY KEY,street_address VARCHAR(255),postal_code VARCHAR(20),city VARCHAR(255),state_province VARCHAR(255),country_id VARCHAR(2),FOREIGN KEY (country_id) REFERENCES countries(country_id));
-->> DESC locations;

Output :

```
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| location_id    | int          | NO   | PRI | NULL    |       |
| street_address | varchar(255) | YES  |     | NULL    |       |
| postal_code    | varchar(20)  | YES  |     | NULL    |       |
| city           | varchar(255) | YES  |     | NULL    |       |
| state_province | varchar(255) | YES  |     | NULL    |       |
| country_id     | varchar(2)   | YES  | MUL | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
```

4. Create table departments **.**

SQL : CREATE TABLE locations (location_id INT PRIMARY KEY,street_address VARCHAR(255),postal_code VARCHAR(20),city VARCHAR(255),state_province VARCHAR(255),country_id VARCHAR(2),FOREIGN KEY (country_id) REFERENCES countries(country_id));
-->> DESC departments;

Output :

```
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| department_id   | int          | NO   | PRI | NULL    |       |
| department_name | varchar(255) | YES  |     | NULL    |       |
| location_id     | int          | YES  | MUL | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
```

5. Create table employees.

SQL: CREATE TABLE employees (employee_id INT PRIMARY KEY,first_name VARCHAR(255),last_name VARCHAR(255),email VARCHAR(255),phone_number VARCHAR(20),hire_date DATE,job_id INT,salary DECIMAL(10, 2),manager_id INT,department_id INT,FOREIGN KEY (job_id) REFERENCES jobs(job_id),FOREIGN KEY (manager_id) REFERENCES employees(employee_id),FOREIGN KEY (department_id) REFERENCES departments(department_id));
-->> DESC employees;

Output:

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| employee_id   | int           | NO   | PRI | NULL    |       |
| first_name    | varchar(255)  | YES  |     | NULL    |       |
| last_name     | varchar(255)  | YES  |     | NULL    |       |
| email         | varchar(255)  | YES  |     | NULL    |       |
| phone_number  | varchar(20)   | YES  |     | NULL    |       |
| hire_date     | date          | YES  |     | NULL    |       |
| job_id        | int           | YES  | MUL | NULL    |       |
| salary        | decimal(10,2) | YES  |     | NULL    |       |
| manager_id    | int           | YES  | MUL | NULL    |       |
| department_id | int           | YES  | MUL | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

6. Create table dependents.

SQL: CREATE TABLE dependents (dependent_id INT PRIMARY KEY,first_name VARCHAR(255),last_name VARCHAR(255),relationship VARCHAR(255), employee_id INT,FOREIGN KEY (employee_id) REFERENCES employees(employee_id));
-->> DESC dependents;

Output:

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| dependent_id | int          | NO   | PRI | NULL    |       |
| first_name   | varchar(255) | YES  |     | NULL    |       |
| last_name    | varchar(255) | YES  |     | NULL    |       |
| relationship | varchar(255) | YES  |     | NULL    |       |
| employee_id  | int          | YES  | MUL | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

7. Create table jobs.

SQL: CREATE TABLE jobs (job_id INT PRIMARY KEY,job_title VARCHAR(255),min_salary DECIMAL(10, 2),max_salary DECIMAL(10, 2));
-->> DESC jobs;

Output :

```
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| job_id     | int           | NO   | PRI | NULL    |       |
| job_title  | varchar(255)  | YES  |     | NULL    |       |
| min_salary | decimal(10,2) | YES  |     | NULL    |       |
| max_salary | decimal(10,2) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

8. Insert records into table regions

   SQL:

   -->> INSERT INTO regions (region_id, region_name) VALUES (1, 'Asia'),(2, 'Americas'),(3, 'Europe');

9. Insert into table countries.

   SQL:

   -->> INSERT INTO countries (country_id, country_name, region_id) VALUES ('IN', 'India', 3),('US','United States', 2),('UK', 'United Kingdom', 2);

10. Insert into table locations.

    SQL:

    -->> INSERT INTO locations (location_id, street_address, postal_code, city, state_province, country_id) VALUES (1700, '123 Main St', '12345', 'Delhi', 'DL', 'IN'),(1701, '456 Elm St', '54321', 'Mumbai', 'MH', 'IN'),(1702, '789 Oak St', '67890', 'Bangalore', 'KA', 'IN');

11. Insert into table departments.

    SQL:

    -->> INSERT INTO departments (department_id, department_name, location_id) VALUES (1, 'Finance', 1700),(2, 'Engineering', 1701),(3, 'Sales', 1702);

12. Insert into table employees.

SQL:

-->> INSERT INTO employees (employee_id, first_name, last_name, email, phone_number, hire_date, job_id, salary, manager_id, department_id) VALUES (1, 'John', 'Doe', 'john.doe@example.com', '123456789', '2023-01-01', 1, 50000, NULL, 1),(2, 'Jane', 'Smith', 'jane.smith@example.com', '987654321', '2023-01-15', 2, 60000, 1, 1),(3, 'Michael', 'Taylor', 'michael.taylor@example.com', '555555555', '2023-02-01', 3, 70000, 2, 2),(4, 'Emily', 'Brown', 'emily.brown@example.com', '333333333', '2023-02-15', 3, 80000, 1, 2),(5, 'David', 'Johnson', 'david.johnson@example.com', '111111111', '2023-03-01', 4, 90000, 2, 3);

13. Insert into table dependents.

SQL:

-->> INSERT INTO dependents (dependent_id, first_name, last_name, relationship, employee_id) VALUES (1, 'Alice', 'Doe', 'Child', 1),(2, 'Bob', 'Doe', 'Child', 1),(3, 'Emma', 'Smith', 'Spouse', 2),(4, 'Olivia', 'Taylor', 'Child', 3),(5, 'Noah', 'Brown', 'Child', 4);

14. Insert into jobs.

SQL:

-->> INSERT INTO jobs (job_id, job_title, min_salary, max_salary) VALUES (1, 'Manager', 50000, 100000),(2, 'Software Engineer', 40000, 80000),(3, 'Sales Representative', 30000, 60000),(4, 'HR Specialist', 35000, 70000);

15. Display all the records from the above tables.

SQL:
-->> select * from regions;
Output :

```
+-----------+-------------+
| region_id | region_name |
+-----------+-------------+
|         1 | Asia        |
|         2 | Americas    |
|         3 | Europe      |
+-----------+-------------+
```

SQL:
-->> select * from countries;
Output:

```
+------------+----------------+-----------+
| country_id | country_name   | region_id |
+------------+----------------+-----------+
| IN         | India          |         3 |
| UK         | United Kingdom |         2 |
| US         | United States  |         2 |
+------------+----------------+-----------+
```

SQL:

-->> select * from locations;

Output:

location_id	street_address	postal_code	city	state_province	country_id
1700	123 Main St	12345	Delhi	DL	IN
1701	456 Elm St	54321	Mumbai	MH	IN
1702	789 Oak St	67890	Bangalore	KA	IN

SQL:

-->> select * from departments;

Output:

department_id	department_name	location_id
1	Finance	1700
2	Engineering	1701
3	Sales	1702

SQL:

-->> select * from employees;

Output:

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id	department_id
1	John	Doe	john.doe@example.com	123456789	2023-01-01	1	50000.00	NULL	1
2	Jane	Smith	jane.smith@example.com	987654321	2023-01-15	2	60000.00	1	1
3	Michael	Taylor	michael.taylor@example.com	555555555	2023-02-01	3	70000.00	2	2
4	Emily	Brown	emily.brown@example.com	333333333	2023-02-15	3	80000.00	1	2
5	David	Johnson	david.johnson@example.com	111111111	2023-03-01	4	90000.00	2	3

SQL:

-->> select * from dependents;

Output:

dependent_id	first_name	last_name	relationship	employee_id
1	Alice	Doe	Child	1
2	Bob	Doe	Child	1
3	Emma	Smith	Spouse	2
4	Olivia	Taylor	Child	3
5	Noah	Brown	Child	4

SQL:

-->> select * from jobs;

Output:

```
+--------+----------------------+------------+------------+
| job_id | job_title            | min_salary | max_salary |
+--------+----------------------+------------+------------+
|      1 | Manager              |   50000.00 |  100000.00 |
|      2 | Software Engineer    |   40000.00 |   80000.00 |
|      3 | Sales Representative |   30000.00 |   60000.00 |
|      4 | HR Specialist        |   35000.00 |   70000.00 |
+--------+----------------------+------------+------------+
```

16. Find all employees who locate in the location with the id 1700.

SQL:

-->> SELECT e.employee_id, e.first_name, e.last_name FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN locations l ON d.location_id = l.location_id WHERE l.location_id = 1700;

Output:

```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|           1 | John       | Doe       |
|           2 | Jane       | Smith     |
+-------------+------------+-----------+
```

17. Find all employees who do not locate at the location 1700.

SQL:

-->> SELECT e.employee_id, e.first_name, e.last_name FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN locations l ON d.location_id = l.location_id WHERE l.location_id != 1700;

Output:

```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|           3 | Michael    | Taylor    |
|           4 | Emily      | Brown     |
|           5 | David      | Johnson   |
+-------------+------------+-----------+
```

18. Finds the employees who have the highest salary.

SQL:

-->> select * from employees where salary =(select max(salary) from employees);

Output:

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id	department_id
5	David	Johnson	david.johnson@example.com	111111111	2023-03-01	4	90000.00	2	3

19. Finds all employees who salaries are greater than the average salary of all employees.

SQL:

-->> SELECT employee_id, first_name, last_name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);

Output:

employee_id	first_name	last_name
4	Emily	Brown
5	David	Johnson

20. Finds all departments (Department Id, Name) which have at least one employee with the salary is greater than 10,000.

SQL:

-->> SELECT d.department_id, d.department_name FROM departments d WHERE EXISTS (SELECT * FROM employees e WHERE e.department_id = d.department_id AND e.salary > 10000);

Output:

department_id	department_name
1	Finance
2	Engineering
3	Sales

21. Finds all departments (Department Id, Name) that do not have any employee with the salary greater than 10,000.

SQL:

-->> SELECT d.department_id, d.department_name FROM departments d WHERE NOT EXISTS (SELECT * FROM employees e WHERE e.department_id = d.department_id AND e.salary > 10000);

Output:

```
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             3 | Sales           |
+---------------+-----------------+
```

22. Finds all employees whose salaries are greater than the lowest salary of every department.

SQL:

-->>**S** ELECT employee_id, first_name, last_name FROM employees e WHERE salary > (SELECT MIN(salary) FROM employees WHERE department_id = e.department_id);

Output:

```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|           2 | Jane       | Smith     |
|           4 | Emily      | Brown     |
+-------------+------------+-----------+
```

23. Finds all employees whose salaries are greater than or equal to the highest salary of every department.

SQL:

-->> SELECT employee_id, first_name, last_name FROM employees e WHERE salary >= (SELECT MAX(salary) FROM employees WHERE department_id = e.department_id);

Output:

```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|           2 | Jane       | Smith     |
|           4 | Emily      | Brown     |
|           5 | David      | Johnson   |
+-------------+------------+-----------+
```

24. Calculate the average of average salary of departments. (Hint: SQL subquery in the FROM clause)

SQL:

-->> select avg(dept_salary) as overall_avg_salary from(select department_id,avg(salary) as dept_salary from employees group by department_id)as totalsalary;

Output:

```
+--------------------+
| overall_avg_salary |
+--------------------+
|   73333.3333333333 |
+--------------------+
```

25. Finds the salaries of all employees, their average salary, and the difference between the salary of each employee and the average salary. (Hint: SQL Subquery in the SELECT clause)

SQL:

-->> SELECT employee_id, first_name, last_name, salary, (salary - (SELECT AVG(salary) FROM employees)) AS salary_difference FROM employees;

Output:

```
+-------------+------------+-----------+----------+-------------------+
| employee_id | first_name | last_name | salary   | salary_difference |
+-------------+------------+-----------+----------+-------------------+
|           1 | John       | Doe       | 50000.00 |     -20000.000000 |
|           2 | Jane       | Smith     | 60000.00 |     -10000.000000 |
|           3 | Michael    | Taylor    | 70000.00 |          0.000000 |
|           4 | Emily      | Brown     | 80000.00 |      10000.000000 |
|           5 | David      | Johnson   | 90000.00 |      20000.000000 |
+-------------+------------+-----------+----------+-------------------+
```

26. Finds all employees whose salary is higher than the average salary of the employees in their departments. (Hint : Use Correlated Subquery).

SQL:

-->> SELECT employee_id, first_name, last_name, salary FROM employees e WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.department_id GROUP BY department_id);

Output:

```
+-------------+------------+-----------+----------+
| employee_id | first_name | last_name | salary   |
+-------------+------------+-----------+----------+
|           2 | Jane       | Smith     | 60000.00 |
|           4 | Emily      | Brown     | 80000.00 |
+-------------+------------+-----------+----------+
```

27. Returns all employees who have no dependents.

SQL:

-->> SELECT e.employee_id, e.first_name, e.last_name FROM employees e LEFT JOIN dependents d ON e.employee_id = d.employee_id WHERE d.employee_id IS NULL;

Output:

```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|           5 | David      | Johnson   |
+-------------+------------+-----------+
```

28. Display first name, last name, department name of employees of the Department with id 1, 2 and 3.

SQL:

-->> SELECT e.first_name, e.last_name, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id WHERE d.department_id IN (1, 2, 3);

Output:

```
 +------------+-----------+-----------------+
| first_name | last_name | department_name |
+------------+-----------+-----------------+
| John       | Doe       | Finance         |
| Jane       | Smith     | Finance         |
| Michael    | Taylor    | Engineering     |
| Emily      | Brown     | Engineering     |
| David      | Johnson   | Sales           |
+------------+-----------+-----------------+
```

29. Display the first name, last name, job title, and department name of employees who work in department with id 1, 2, and 3 and salary greater than 10000.

SQL:

-->> SELECT e.first_name, e.last_name, j.job_title, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN jobs j ON e.job_id = j.job_id WHERE e.department_id IN (1, 2, 3) AND e.salary > 10000;

Output:

```
+------------+-----------+----------------------+-----------------+
| first_name | last_name | job_title            | department_name |
+------------+-----------+----------------------+-----------------+
| John       | Doe       | Manager              | Finance         |
| Jane       | Smith     | Software Engineer    | Finance         |
| Michael    | Taylor    | Sales Representative | Engineering     |
| Emily      | Brown     | Sales Representative | Engineering     |
| David      | Johnson   | HR Specialist        | Sales           |
+------------+-----------+----------------------+-----------------+
```

30. Display Department name, street address, postal code, country name and region name of all departments.

SQL:

-->> SELECT d.department_name, l.street_address, l.postal_code, c.country_name,r.region_name FROM departments d JOIN locations l ON d.location_id = l.location_id JOIN countries c ON l.country_id = c.country_id JOIN regions r ON c.region_id = r.region_id;

Output:

```
+-----------------+----------------+-------------+--------------+-------------+
| department_name | street_address | postal_code | country_name | region_name |
+-----------------+----------------+-------------+--------------+-------------+
| Sales           | 789 Oak St     | 67890       | India        | Europe      |
| Engineering     | 456 Elm St     | 54321       | India        | Europe      |
| Finance         | 123 Main St    | 12345       | India        | Europe      |
+-----------------+----------------+-------------+--------------+-------------+
```

31. Write a SQL query to find out which employees have or do not have a department. Return first name, last name, department ID, department name.

SQL:

-->> SELECT e.first_name, e.last_name, e.department_id, COALESCE(d.department_name, 'No Department') AS department_name FROM employees e LEFT JOIN departments d ON e.department_id = d.department_id;

Output:

```
+------------+-----------+---------------+-----------------+
| first_name | last_name | department_id | department_name |
+------------+-----------+---------------+-----------------+
| John       | Doe       |             1 | Finance         |
| Jane       | Smith     |             1 | Finance         |
| Michael    | Taylor    |             2 | Engineering     |
| Emily      | Brown     |             2 | Engineering     |
| David      | Johnson   |             3 | Sales           |
+------------+-----------+---------------+-----------------+
```

32. Write a SQL query to find those employees whose first name contains the letter ‘Z’. Return first name, last name, department, city, and state province.

SQL:

-->> SELECT e.first_name, e.last_name, d.department_name AS department, l.city, l.state_province FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN locations l ON d.location_id = l.location_id WHERE e.first_name LIKE '%z%';

Output:

```
+------------+-----------+------------+-------+----------------+
| first_name | last_name | department | city  | state_province |
+------------+-----------+------------+-------+----------------+
| zen        | Doe       | Finance    | Delhi | DL             |
+------------+-----------+------------+-------+----------------+
```

33. Write a SQL query to find all departments, including those without employees Return first name, last name, department ID, department name

SQL:

-->> SELECT e.first_name, e.last_name, d.department_id, d.department_name FROM departments d LEFT JOIN employees e ON d.department_id = e.department_id;

Output:

```
+------------+-----------+---------------+-----------------+
| first_name | last_name | department_id | department_name |
+------------+-----------+---------------+-----------------+
| john       | Doe       |             1 | Finance         |
| Jane       | Smith     |             1 | Finance         |
| Michael    | Taylor    |             2 | Engineering     |
| Emily      | Brown     |             2 | Engineering     |
| David      | Johnson   |             3 | Sales           |
+------------+-----------+---------------+-----------------+
```

34. Write a SQL query to find the employees and their managers. . Those managers do not work under any manager also appear in the list. Return the first name of the employee and manager.

SQL:

-->> SELECT e.first_name AS employee_first_name, COALESCE(m.first_name, 'No Manager') AS manager_first_name FROM employees e LEFT JOIN employees m ON e.manager_id = m.employee_id;

Output:

```
+---------------------+--------------------+
| employee_first_name | manager_first_name |
+---------------------+--------------------+
| john                | No Manager         |
| Jane                | john               |
| Michael             | Jane               |
| Emily               | john               |
| David               | Jane               |
+---------------------+--------------------+
```

35. Write a SQL query to find the employees who work in the same department as the employee with the last name Taylor. Return first name, last name and department ID.

SQL:

-->> SELECT e.first_name, e.last_name, e.department_id FROM employees e WHERE e.department_id = (SELECT department_id FROM employees WHERE last_name = 'Taylor');

Output:

```
+------------+-----------+---------------+
| first_name | last_name | department_id |
+------------+-----------+---------------+
| Michael    | Taylor    |             2 |
| Emily      | Brown     |             2 |
+------------+-----------+---------------+
```

36. Write a SQL query to calculate the difference between the maximum salary of the job and the employee's salary. Return job title, employee name, and salary difference.

SQL:

-->> SELECT j.job_title, CONCAT(e.first_name, ' ', e.last_name) AS employee_name, (j.max_salary - e.salary) AS salary_difference FROM employees e JOIN jobs j ON e.job_id = j.job_id;

Output:

```
+----------------------+---------------+-------------------+
| job_title            | employee_name | salary_difference |
+----------------------+---------------+-------------------+
| Manager              | john Doe      |          50000.00 |
| Software Engineer    | Jane Smith    |          20000.00 |
| Sales Representative | Michael Taylor|         -10000.00 |
| Sales Representative | Emily Brown   |         -20000.00 |
| HR Specialist        | David Johnson |         -20000.00 |
+----------------------+---------------+-------------------+
```

37. Write a SQL query to calculate the average salary, the number of employees receiving commissions in that department. Return department name, average salary and number of employees of all departments.

SQL:

-->> SELECT d.department_name, AVG(e.salary) AS average_salary, COUNT(e.employee_id) AS number_of_employee FROM departments d LEFT JOIN employees e ON d.department_id = e.department_id GROUP BY d.department_id, d.department_name;

Output:

```
+-----------------+----------------+--------------------+
| department_name | average_salary | number_of_employee |
+-----------------+----------------+--------------------+
| Finance         |   55000.000000 |                  2 |
| Engineering     |   75000.000000 |                  2 |
| Sales           |   90000.000000 |                  1 |
+-----------------+----------------+--------------------+
```

38. Create a view which contains employee name, employee id, phone number, job title, department name, manager name of employees belongs to department whose location is in ‘Delhi’ and display the details,

SQL:

-->> CREATE VIEW EmployeeDetailsInDelhi AS SELECT e.employee_id, CONCAT(e.first_name, ' ', e.last_name) AS employee_name, e.phone_number, j.job_title, d.department_name, CONCAT(m.first_name, ' ', m.last_name) AS manager_name FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN jobs j ON e.job_id = j.job_id LEFT JOIN employees m ON e.manager_id = m.employee_id JOIN locations l ON d.location_id = l.location_id WHERE l.city = 'Delhi';

-->> select*from EmployeeDetailsInDelhi;

Output:

```
+-------------+---------------+--------------+-------------------+-----------------+--------------+
| employee_id | employee_name | phone_number | job_title         | department_name | manager_name |
+-------------+---------------+--------------+-------------------+-----------------+--------------+
|           1 | john Doe      | 123456789    | Manager           | Finance         | NULL         |
|           2 | Jane Smith    | 987654321    | Software Engineer | Finance         | john Doe     |
+-------------+---------------+--------------+-------------------+-----------------+--------------+
```

39. Use the above created view to obtain the names of employees whose job title is ‘Manager’ and department is ‘Finance’.

SQL:

-->> SELECT employee_name FROM EmployeeDetailsInDelhi WHERE job_title = 'Manager' AND department_name = 'Finance';

Output:

```
+----------------+
| employee_name |
+----------------+
| john Doe       |
+----------------+
```

40. Check whether it is possible to update the phone number of employee whose first name is ‘Smith’ by using the above created view.

SQL:

-->> UPDATE EmployeeDetailsInDelhi SET phone_number = 'new_phone_number' WHERE employee_name LIKE 'Smith%';

Output:

ERROR 1288 (HY000): The target table EmployeeDetailsInDelhi of the UPDATE is not updatable

41. Display the details of employee who have no dependents.

SQL:

-->> select e.employee_id,e.first_name,e.last_name,e.phone_number from employees e left join dependents d on e.employee_id=d.employee_id where d.employee_id is NULL;

Output:

employee_id	first_name	last_name	phone_number
5	David	Johnson	111111111

42. Display the details of employee who manager id is 101 or 201. (Use Union Clause).

SQL:

-->> SELECT * FROM employees WHERE manager_id = 101 UNION SELECT * FROM employees WHERE manager_id = 201;

Output:

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id	department_id
102	Anu	j	anu@gmail.com	687533218	2007-02-01	4	25000.00	101	1

43. . Display the details of employees who have at least one dependent.

SQL:

-->> select * from employees where employee_id in(select distinct(employee_id) from dependents);

Output:

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id	department_id
1	john	Doe	john.doe@example.com	123456789	2023-01-01	1	50000.00	NULL	1
2	Jane	Smith	jane.smith@example.com	987654321	2023-01-15	2	60000.00	1	1
3	Michael	Taylor	michael.taylor@example.com	555555555	2023-02-01	3	70000.00	2	2
4	Emily	Brown	emily.brown@example.com	333333333	2023-02-15	3	80000.00	1	2

Result:

Query is executed successfully and has been executed

LAB CYCLE 3
Experiment No: 5
Date: 03/04/24
Familiarization of Stored Procedure, Function, Cursor and Triggers

- **Stored Procedure**

  A stored procedure is a set of SQL commands that can be executed repeatedly with a single call.It is stored in the database and can be executed by calling its name.
  Syntax: CREATE PROCEDURE procedure_name (parameters) AS BEGIN SQL commands END;
  Example: CREATE PROCEDURE get_employee_details (emp_id INT) AS BEGIN SELECT * FROM employees WHERE emp_id = ?; END;

- **Function**

  A function is a self-contained block of code that performs a specific task.It returns a value and can be used in a SELECT statement.
  Syntax: CREATE FUNCTION function_name (parameters) RETURNS data_type AS BEGIN SQL commands RETURN result; END;
  Example: CREATE FUNCTION get_employee_name (emp_id INT) RETURNS VARCHAR(255) AS BEGIN SELECT name FROM employees WHERE emp_id = ?; END;

- **Cursor**

  A cursor is a temporary result set that allows you to process data row by row.It is used to retrieve and manipulate data in a loop.
  Syntax: DECLARE cursor_name CURSOR FOR SELECT statement;
  Example: DECLARE emp_cursor CURSOR FOR SELECT * FROM employees WHERE department = 'Sales';

- **Trigger**

  A trigger is a set of SQL commands that are executed automatically when specific event occurs (e.g., insert, update, delete).It is used to maintain data integrity and perform actions automatically.
  Syntax: CREATE TRIGGER trigger_name BEFORE/AFTER INSERT/UPDATE/DELETE ON table_name FOR EACH ROW BEGIN SQL commands END;
  Example: CREATE TRIGGER update_employee_salary BEFORE UPDATE ON employees FOR EACH ROW BEGIN IF NEW.salary > 100000 THEN SET NEW.salary = 100000; END;

1. Create a procedure which will receive account_id and amount to withdraw. If the account does not exist, it will display a message. Otherwise, if the account exists, it will allow the withdrawal only if the new balance after the withdrawal is at least 1000.

SQL:

```
delimiter //
CREATE PROCEDURE WithdrawAmount(IN account_id INT, IN
withdraw_amount DECIMAL(10,2))
BEGIN
DECLARE current_balance DECIMAL(10,2);
DECLARE account_exists INT;
SELECT COUNT(*) INTO account_exists FROM Accounts
WHERE id = account_id;
IF account_exists = 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Account does not exist.';
ELSE
SELECT balance INTO current_balance FROM Accounts
WHERE id = account_id;
IF current_balance - withdraw_amount < 1000 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Insufficient balance. The balance after withdrawal
must be at least 1000.';
ELSE
UPDATE Accounts SET balance = balance - withdraw_amount
WHERE id = account_id;
SIGNAL SQLSTATE '02000'
SET MESSAGE_TEXT = 'Withdrawal successful.';
END IF;
END IF;
END //
```

Output:

```
Query OK, 0 rows affected (0.07 sec)
```

SQL:

```
CALL WithdrawAmount(123, 500.00) ;
```

Output:

```
ERROR 1643 (02000): Withdrawal successful.
```

SQL:

```
CALL WithdrawAmount(123, 1500.00);
```

Output:

```
ERROR 1644 (45000): Insufficient balance. The balance after withdrawal must
be at least 1000.
```

2. Create a 'Customer' table with attributes customer id, name, city and credits. Write a stored procedure to display the details of a particular customer from the customer table, where name is passed as a parameter.

SQL:

```
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    city VARCHAR(100),
    credits DECIMAL(10,2)
);

CREATE PROCEDURE GetCustomerDetailsByName(IN customer_name
VARCHAR(100))
  BEGIN
    SELECT * FROM Customer
    WHERE name = customer_name;
  END;
CALL GetCustomerDetailsByName('John');
```

Output:

```
+-------------+------+----------+---------+
| customer_id | name | city     | credits |
+-------------+------+----------+---------+
|           1 | John | New York | 1500.00 |
+-------------+------+----------+---------+
```

3. Create a stored procedure to determine membership of a particular customer based on the following credits:
Above 5000 = Membership Platinum
1000 to 5000 = Gold
< 1000 = silver
[Use IN and OUT Parameters]

SQL:

```
CREATE PROCEDURE DetermineMembership(IN customer_credits DECIMAL(10,2),
OUT membership_level VARCHAR(50))
  BEGIN
  IF customer_credits >= 5000 THEN
      SET membership_level = 'Membership Platinum';
      ELSEIF customer_credits >= 1000 THEN
      SET membership_level = 'Gold';
    ELSE
      SET membership_level = 'Silver';
    END IF;
```

```
CALL DetermineMembership(6000.00, @membership1);
CALL DetermineMembership(3500.00, @membership2);
CALL DetermineMembership(800.00, @membership3);
CALL DetermineMembership(10000.00, @membership4);

SELECT @membership1 AS John_Membership, @membership2 AS
Alice_Membership, @membership3 AS Bob_Membership, @membership4 AS Emma;
```

Output:

```
+--------------------+-----------------+---------------+--------------------+
| John_Membership    | Alice_Membership | Bob_Membershi |Emma    |
+--------------------+-----------------+---------------+--------------------+
| Membership Platinum | Gold            | Silver        | Membership Platinum |
+--------------------+-----------------+---------------+--------------------+
```

4. Write a function that takes employee name as parameter and returns the number of employees with this name. Use the function to update details of employees with unique names. For other cases, the program (not the function) should display error messages - "No Employee" or "Multiple employees".

SQL:

```
CREATE FUNCTION CountEmployeesWithName(employee_name VARCHAR(100))
RETURNS INT
    DETERMINISTIC
    BEGIN
     DECLARE employee_count INT;

     SELECT COUNT(*) INTO employee_count FROM Employees
      WHERE name = employee_name;

     RETURN employee_count;
   END ;

SELECT CountEmployeesWithName('John');
```

Output:

```
+---------------------------------+
| CountEmployeesWithName('John') |
+---------------------------------+
|                               2 |
+---------------------------------+
```

5. Write a stored procedure using cursor to calculate salary of each employee. Consider an Emp_salary table have the following attributes emp_id, emp_name, no_of_working_days, designation and salary.

SQL:

```
CREATE PROCEDURE CalculateEmployeeSalary()
   -> BEGIN
   ->     DECLARE done INT DEFAULT FALSE;
   ->     DECLARE emp_id_val INT;
   ->     DECLARE emp_name_val VARCHAR(100);
   ->     DECLARE working_days_val INT;
   ->     DECLARE designation_val VARCHAR(100);
   ->     DECLARE daily_wage DECIMAL(10, 2);
   ->     DECLARE emp_salary DECIMAL(10, 2);
   ->
   ->     DECLARE emp_cursor CURSOR FOR
   ->         SELECT emp_id, emp_name, no_of_working_days, designation
   ->         FROM Emp_salary;
   ->
   ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   ->
   ->     OPEN emp_cursor;
   ->
   ->     emp_loop: LOOP
   ->         FETCH emp_cursor INTO emp_id_val, emp_name_val, working_days_val,
designation_val;
   ->
   ->         IF done THEN
   ->             LEAVE emp_loop;
   ->         END IF;
   ->
   ->         CASE designation_val
   ->             WHEN 'Assistant Professor' THEN SET daily_wage := 1750.00;
   ->             WHEN 'Clerk' THEN SET daily_wage := 750.00;
   ->             WHEN 'Programmer' THEN SET daily_wage := 1250.00;
   ->             ELSE SET daily_wage := 0.00;
   ->         END CASE;
   ->
   ->         SET emp_salary := daily_wage * working_days_val;
   ->
   ->         SELECT CONCAT(emp_name_val, "'s salary is $", emp_salary) AS Result;
   ->
   ->     END LOOP;
   ->
   ->     CLOSE emp_cursor;
   -> END ;

CALL CalculateEmployeeSalary();
```

Output:

```
+----------------------------+
| Result                     |
+----------------------------+
| John's salary is $35000.00 |
+----------------------------+

+----------------------------+
| Result                     |
+----------------------------+
| Alice's salary is $18750.00 |
+----------------------------+
+----------------------------+
| Result                     |
+----------------------------+
| Bob's salary is $27500.00 |
+----------------------------+
```

6. Write a procedure to calculate the electricity bill of all customers. Electricity board charges the following rates to domestic uses to find the consumption of energy.

a) For first 100 units Rs:2 per unit.
b) 101 to 200 units Rs:2.5 per unit.
c) 201 to 300 units Rs: 3 per unit.
d) Above 300 units Rs: 4 per unit

Consider the table 'Bill' with fields customer_id, name, pre_reading, cur_reading , unit, and amount.

SQL:

```
CREATE PROCEDURE CalculateElectricityBill()
   -> BEGIN
   ->    DECLARE customer_id_val INT;
   ->    DECLARE name_val VARCHAR(100);
   ->    DECLARE pre_reading_val INT;
   ->    DECLARE cur_reading_val INT;
   ->    DECLARE unit_val INT;
   ->    DECLARE amount_val DECIMAL(10, 2);
   ->
   ->    DECLARE done INT DEFAULT FALSE;
   ->
   ->    DECLARE bill_cursor CURSOR FOR
   ->       SELECT customer_id, name, pre_reading, cur_reading
   ->       FROM Bill;
   ->
   ->    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   ->
   ->    OPEN bill_cursor;
   ->
```

```
    ->     bill_loop: LOOP
    ->         FETCH bill_cursor INTO customer_id_val, name_val, pre_reading_val,
cur_reading_val;
    ->
    ->         IF done THEN
    ->             LEAVE bill_loop;
    ->         END IF;
    ->
    ->         SET unit_val := cur_reading_val - pre_reading_val;
    ->
    ->         IF unit_val <= 100 THEN
    ->             SET amount_val := unit_val * 2.00;
    ->         ELSEIF unit_val <= 200 THEN
    ->             SET amount_val := 100 * 2.00 + (unit_val - 100) * 2.50;
    ->         ELSEIF unit_val <= 300 THEN
    ->             SET amount_val := 100 * 2.00 + 100 * 2.50 + (unit_val - 200) * 3.00;
    ->         ELSE
    ->             SET amount_val := 100 * 2.00 + 100 * 2.50 + 100 * 3.00 + (unit_val - 300)
* 4.00;
    ->         END IF;
    ->
    ->         -- Update the amount in the Bill table
    ->         UPDATE Bill
    ->         SET amount = amount_val
    ->         WHERE customer_id = customer_id_val;
    ->     END LOOP;
    ->
    ->     CLOSE bill_cursor;
    -> END ;
CALL CalculateElectricityBill();
```

Output:

```
+-------------+--------+-------------+-------------+------+--------+
| customer_id | name   | pre_reading | cur_reading | unit | amount |
+-------------+--------+-------------+-------------+------+--------+
|           1 | John   |         100 |         200 | NULL | 200.00 |
|           2 | Alice  |         150 |         300 | NULL | 325.00 |
|           3 | Bob    |         200 |         400 | NULL | 450.00 |
|           4 | Emma   |         250 |         500 | NULL | 600.00 |
|           5 | Sophia |         300 |         450 | NULL | 325.00 |
+-------------+--------+-------------+-------------+------+--------+
```

7. Create a trigger on employee table such that whenever a row is deleted, it is moved to history table named ‘Emp_history’ with the same structure as employee table. ‘Emp_history’ will contain an additional column “Date_of_deletion” to store the date on which the row is removed. [After Delete Trigger]

SQL:

```
CREATE TRIGGER MoveDeletedEmployeeToHistory
   -> AFTER DELETE ON Employee
   -> FOR EACH ROW
   -> BEGIN
   ->     INSERT INTO Emp_history (employee_id, name, department, salary)
   ->     VALUES (OLD.employee_id, OLD.name, OLD.department, OLD.salary);
   -> END ;
```

```
select * from Emp_history
```

Output:

```
+-------------+-------+------------+----------+---------------------+
| employee_id | name  | department | salary   | date_of_deletion    |
+-------------+-------+------------+----------+---------------------+
|           2 | Alice | HR         | 45000.00 | 2024-06-03 01:11:43 |
+-------------+-------+------------+----------+---------------------
```

8. Before insert a new record in emp_details table, create a trigger that check the column value of FIRST_NAME, LAST_NAME, JOB_ID and if there are any space(s) before or after the FIRST_NAME, LAST_NAME, TRIM () function will remove those. The value of the JOB_ID will be converted to upper cases by UPPER () function. [Before Insert Trigger]

SQL:

```
CREATE TRIGGER BeforeInsertEmpDetails
   -> BEFORE INSERT ON emp_details
   -> FOR EACH ROW
   -> BEGIN
   ->     -- Trim leading and trailing spaces from FIRST_NAME and LAST_NAME
   ->     SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
   ->     SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
   ->
   ->     -- Convert JOB_ID to upper case
   ->     SET NEW.JOB_ID = UPPER(NEW.JOB_ID);
   -> END ;
```

```
INSERT INTO emp_details (FIRST_NAME, LAST_NAME, JOB_ID) VALUES ('
Michael   ', '   Smith   ', 'analyst');
```

```
SELECT * FROM emp_details;
```

Output:

```
+--------+------------+-------------+----------+
| emp_id | FIRST_NAME | LAST_NAME   | JOB_ID   |
+--------+------------+-------------+----------+
|      1 |    John    |    Doe      | engineer |
|      2 | Alice      | Smith       | clerk    |
|      3 |   Bob      |   Johnson   | manager  |
|      4 | Michael    | Smith       | ANALYST  |
+--------+------------+-------------+----------+
```

9. Consider the following table with sample data. Create a trigger to calculate total marks, percentage and grade of the students, when marks of the subjects are updated. [After Update Trigger]

For this sample calculation, the following conditions are assumed: Total Marks (will be stored in TOTAL column) : TOTAL = SUB1 + SUB2 + SUB3 + SUB4 + SUB5. Percentage of Marks (will be stored in PER_MARKS column): PER_MARKS = (TOTAL)/5 Grade (will be stored in GRADE column):
- If PER_MARKS>=90 -> 'EXCELLENT'
- If PER_MARKS>=75 AND PER_MARKS 'VERY GOOD'
- If PER_MARKS>=60 AND PER_MARKS 'GOOD'
- If PER_MARKS>=40 AND PER_MARKS 'AVERAGE'
- If PER_MARKS 'NOT PROMOTED

STUDENT_ID	NAME	SUB1	SUB2	SUB3	SUB4	SUB5	TOTAL	PER_MARKS	GRADE
1	Steven King	0	0	0	0	0	0	0.00	
2	Neena Kochhar	0	0	0	0	0	0	0.00	
3	Lex De Haan	0	0	0	0	0	0	0.00	
4	Alexander Hunold	0	0	0	0	0	0	0.00	

SQL:

```
CREATE TRIGGER CalculateStudentDetails
 -> BEFORE UPDATE ON student_marks
 -> FOR EACH ROW
 -> BEGIN
 ->     -- Calculate total marks
 ->     SET NEW.total = NEW.sub1 + NEW.sub2 + NEW.sub3 + NEW.sub4 +
NEW.sub5;
 ->
 ->     -- Calculate percentage of marks
 ->     SET NEW.per_marks = NEW.total / 5;
 ->
 ->     -- Determine grade based on percentage
 ->     IF NEW.per_marks >= 90 THEN
 ->         SET NEW.grade = 'EXCELLENT';
 ->     ELSEIF NEW.per_marks >= 75 THEN
 ->         SET NEW.grade = 'VERY GOOD';
 ->     ELSEIF NEW.per_marks >= 60 THEN
 ->         SET NEW.grade = 'GOOD';
 ->     ELSEIF NEW.per_marks >= 40 THEN
 ->         SET NEW.grade = 'AVERAGE';
```

```
    ->     ELSE
    ->         SET NEW.grade = 'NOT PROMOTED';
    ->     END IF;
    -> END ;
UPDATE student_marks
    -> SET sub1 = 80, sub2 = 85, sub3 = 90, sub4 = 75, sub5 = 85
    -> WHERE student_id = 1;
select * from student_marks;
```

Output:

student_id	name	sub1	sub2	sub3	sub4	sub5	total	per_marks	grade
1	John	80	85	90	75	85	415	83.00	VERY GOOD
2	Alice	70	80	65	75	85	NULL	NULL	NULL
3	Bob	60	65	70	55	80	NULL	NULL	NULL
4	Emma	90	85	95	88	92	NULL	NULL	NULL

Result:

Query is executed successfully and has been executed

LAB CYCLE 4

DATE: 24/04/2024

Experiment No: 6

1. **Build sample collections/documents to perform query operation using MongoDB**

- **Insert**
  The insert() method in MongoDB is used to insert documents into a collection. There are multiple variations, including insertOne(), insertMany(), and the deprecated insert()
  Syntax: db.collection.insertOne(document)
- **Find**
  db.collection.find().pretty() fetches and displays all documents in a collection in a readable format.
- **Sort**
  sorts the query results in ascending order, while `-1` sorts in descending order.
  Syntax db.collection.find().sort({ field: 1 })
- **Limit**
  db.collection.find().limit(number) limits the number of documents returned in the query result.
- **Skip**
  db.collection.find().skip(number) skips the specified number of documents in the query result
- **Update**
  db.collection.updateOne({ condition }, { $set: { field: value } }) updates a single document that matches the condition
- **Delete**
  db.collection.deleteOne({ condition }) deletes a single document that matches the condition
- **Drop**
  db.collection.drop() removes the entire collection from the database.

1.Create a database (Eg : MyCev)

Query:

use MyCev

Output:

Switched to db MyCev

2.Create a collection (Eg: db_mca)

Query:

ii. db.createCollection("db_mca")

Output:

{ ok: 1 }

3 .Create a collection (Eg: db_cs)

Query:

iii. db.createCollection("db_cs")

Output:

{ ok: 1 }

4. Insert 10 data to the collection

Query:

```
db.db_mca.insertMany([
...     { "name": "Alice", "age": 25, "course": "MCA" },
...     { "name": "Bob", "age": 27, "course": "MCA" },
...     { "name": "Charlie", "age": 22, "course": "MCA" },
...     { "name": "David", "age": 24, "course": "MCA" },
...     { "name": "Eve", "age": 23, "course": "MCA" },
...     { "name": "Frank", "age": 26, "course": "MCA" },
...     { "name": "Grace", "age": 28, "course": "MCA" },
...     { "name": "Hank", "age": 21, "course": "MCA" },
...     { "name": "Ivy", "age": 29, "course": "MCA" },
...     { "name": "Jack", "age": 30, "course": "MCA" }
... ])
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66636c19c04587550246b799'),
    '1': ObjectId('66636c19c04587550246b79a'),
    '2': ObjectId('66636c19c04587550246b79b'),
    '3': ObjectId('66636c19c04587550246b79c'),
    '4': ObjectId('66636c19c04587550246b79d'),
    '5': ObjectId('66636c19c04587550246b79e'),
    '6': ObjectId('66636c19c04587550246b79f'),
    '7': ObjectId('66636c19c04587550246b7a0'),
    '8': ObjectId('66636c19c04587550246b7a1'),
    '9': ObjectId('66636c19c04587550246b7a2')
  }
}
```

5. List the first 5 data from the collection (limit)

Query:

v. db.db_mca.find().limit(5).pretty()

Output:

```
[
  {
    _id: ObjectId('66636c19c04587550246b799'),
    name: 'Alice',
    age: 25,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79a'),
    name: 'Bob',
    age: 27,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79b'),
    name: 'Charlie',
    age: 22,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79c'),
    name: 'David',
    age: 24,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79d'),
    name: 'Eve',
```

```
  age: 23,
  course: 'MCA'
 }
]
```

6. List the entire data except first 2 data (skip)

Query:

vi. db.db_mca.find().skip(2).pretty()

Output:

```
[
 {
  _id: ObjectId('66636c19c04587550246b79b'),
  name: 'Charlie',
  age: 22,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79c'),
  name: 'David',
  age: 24,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79d'),
  name: 'Eve',
  age: 23,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79e'),
  name: 'Frank',
  age: 26,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79f'),
  name: 'Grace',
  age: 28,
```

```
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b7a0'),
  name: 'Hank',
  age: 21,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b7a1'),
  name: 'Ivy',
  age: 29,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b7a2'),
  name: 'Jack',
  age: 30,
  course: 'MCA'
 }
]
```

7. Sort the data by choosing any field in the collection

Query:

vii. db.db_mca.find().sort({ age: 1 }).pretty()

Output:

```
[
 {
  _id: ObjectId('66636c19c04587550246b7a0'),
  name: 'Hank',
  age: 21,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79b'),
  name: 'Charlie',
  age: 22,
  course: 'MCA'
 },
 {
  _id: ObjectId('66636c19c04587550246b79d'),
  name: 'Eve',
  age: 23,
  course: 'MCA'
 },
```

```
  {
    _id: ObjectId('66636c19c04587550246b79c'),
    name: 'David',
    age: 24,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b799'),
    name: 'Alice',
    age: 25,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79e'),
    name: 'Frank',
    age: 26,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79a'),
    name: 'Bob',
    age: 27,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b79f'),
    name: 'Grace',
    age: 28,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b7a1'),
    name: 'Ivy',
    age: 29,
    course: 'MCA'
  },
  {
    _id: ObjectId('66636c19c04587550246b7a2'),
    name: 'Jack',
    age: 30,
    course: 'MCA'
  }
]
```

8. Delete data from collection

Query:

viii. db.db_mca.deleteOne({ name: "Alice" })

Output:

{ acknowledged: true, deletedCount: 1 }

9. Drop the collection(db_cs)

Query:

ix. db.db_cs.drop()

Output:

True

10. Drop database

Query:

x. Db.dropDatabase()

Output:

{ ok: 1, dropped: 'MyCev' }

Result:

Query has been executed successfully and output is obtained

LAB CYCLE 4

Experiment No: 7

Date: 24/05/2024

1. **Design Databases using MongoDB and perform CRUD operations**

- **Insert**
  The insert() method in MongoDB is used to insert documents into a collection. There are multiple variations, including insertOne(), insertMany(), and the deprecated insert()
  Syntax: db.collection.insertOne(document)
- **Find**
  db.collection.find().pretty() fetches and displays all documents in a collection in a readable format.
- **Sort**
  sorts the query results in ascending order, while `-1` sorts in descending order.
  Syntax db.collection.find().sort({ field: 1 })
- **Limit**
  db.collection.find().limit(number) limits the number of documents returned in the query result.
- **Skip**
  db.collection.find().skip(number) skips the specified number of documents in the query result
- **Update**
  db.collection.updateOne({ condition }, { $set: { field: value } }) updates a single document that matches the condition
- **Delete**
  db.collection.deleteOne({ condition }) deletes a single document that matches the condition
- **Drop**
  db.collection.drop() removes the entire collection from the database.

1. i.Create a database Myclass.
   Query:
   i use Myclass
   Output:
   switched to db Myclass

2. ii.Create a collection named “db_students” Should contain this fields : { student_name, student_rollno, mark[subject, mark] }
   Nb: Mark should be stored as array
   Query:
   ii. db.createCollection("db_students")
   Output:
   { ok: 1 }

3. .Insert details of 10 students in a class

Query:

iii. db.db_students.insertMany([

... { student_name: "Alice", student_rollno: 1, mark: [{subject: "Math", mark: 85}, {subject: "English", mark: 78}] },

... { student_name: "Bob", student_rollno: 2, mark: [{subject: "Math", mark: 90}, {subject: "English", mark: 88}] },

... { student_name: "Charlie", student_rollno: 3, mark: [{subject: "Math", mark: 75}, {subject: "English", mark: 67}] },

... { student_name: "David", student_rollno: 4, mark: [{subject: "Math", mark: 80}, {subject: "English", mark: 70}] },

... { student_name: "Eve", student_rollno: 5, mark: [{subject: "Math

", mark: 92}, {subject: "English", mark: 95}] },

... { student_name: "Frank", student_rollno: 6, mark: [{subject: "Math", mark: 65}, {subject: "English", mark: 72}] },

... { student_name: "Grace", student_rollno: 7, mark: [{subject: "Math", mark: 88}, {subject: "English", mark: 85}] },

... { student_name: "Hank", student_rollno: 8, mark: [{subject: "Math", mark: 77}, {subject: "English", mark: 82}] },

... { student_name: "Ivy", student_rollno: 9, mark: [{subject: "Math", mark: 80}, {subject: "English", mark: 89}] },

... { student_name: "Jack", student_rollno: 10, mark: [{subject: "Math", mark: 83}, {subject: "English", mark: 87}] }

...])

Output:

{

acknowledged: true,

insertedIds: {

'0': ObjectId('66637148c04587550246b7a3'),

'1': ObjectId('66637148c04587550246b7a4'),

'2': ObjectId('66637148c04587550246b7a5'),

'3': ObjectId('66637148c04587550246b7a6'),

'4': ObjectId('66637148c04587550246b7a7'),

```
    '5': ObjectId('66637148c04587550246b7a8'),
    '6': ObjectId('66637148c04587550246b7a9'),
    '7': ObjectId('66637148c04587550246b7aa'),
    '8': ObjectId('66637148c04587550246b7ab'),
    '9': ObjectId('66637148c04587550246b7ac')
  }
}
```

4. List the entire students in the class

Query:

iv .db.db_students.find().pretty()

Output:

```
[
  {
    _id: ObjectId('66637148c04587550246b7a3'),
    student_name: 'Alice',
    student_rollno: 1,
    mark: [ { subject: 'Math', mark: 85 }, { subject: 'English', mark: 78 } ]
  },
  {
    _id: ObjectId('66637148c04587550246b7a4'),
    student_name: 'Bob',
    student_rollno: 2,
    mark: [ { subject: 'Math', mark: 90 }, { subject: 'English', mark: 88 } ]
  },
  {
    _id: ObjectId('66637148c04587550246b7a5'),
    student_name: 'Charlie',
    student_rollno: 3,
    mark: [ { subject: 'Math', mark: 75 }, { subject: 'English', mark: 67 } ]
  },
```

```
{
  _id: ObjectId('66637148c04587550246b7a6'),
  student_name: 'David',
  student_rollno: 4,
  mark: [ { subject: 'Math', mark: 80 }, { subject: 'English', mark: 70 } ]
},
{
  _id: ObjectId('66637148c04587550246b7a7'),
  student_name: 'Eve',
  student_rollno: 5,
  mark: [ { subject: 'Math', mark: 92 }, { subject: 'English', mark: 95 } ]
},
{
  _id: ObjectId('66637148c04587550246b7a8'),
  student_name: 'Frank',
  student_rollno: 6,
  mark: [ { subject: 'Math', mark: 65 }, { subject: 'English', mark: 72 } ]
},
{
  _id: ObjectId('66637148c04587550246b7a9'),
  student_name: 'Grace',
  student_rollno: 7,
  mark: [ { subject: 'Math', mark: 88 }, { subject: 'English', mark: 85 } ]
},
{
  _id: ObjectId('66637148c04587550246b7aa'),
  student_name: 'Hank',
  student_rollno: 8,
  mark: [ { subject: 'Math', mark: 77 }, { subject: 'English', mark: 82 } ]
},
```

```
  {
    _id: ObjectId('66637148c04587550246b7ab'),
    student_name: 'Ivy',
    student_rollno: 9,
    mark: [ { subject: 'Math', mark: 80 }, { subject: 'English', mark: 89 } ]
  },
  {
    _id: ObjectId('66637148c04587550246b7ac'),
    student_name: 'Jack',
    student_rollno: 10,
    mark: [ { subject: 'Math', mark: 83 }, { subject: 'English', mark: 87 } ]
  }
]
```

5. Update mark of any students in the collection “db_students”

Query:

```
v. db.db_students.updateOne(
...     { student_name: "Bob" },
...     { $set: { "mark.$[elem].mark": 90 } },
...     { arrayFilters: [ { "elem.subject": "English" } ] }
... )
```

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

6. .Delete the data of first student in the collection

Query:

vi. db.db_students.deleteOne({ student_rollno: 1 })

Output:

{ acknowledged: true, deletedCount: 1 }

Result:

Query has been executed successfully and output is obtained.

LAB CYCLE 4

Experiment No: 8

Date: 08/05/2024

1. **Design Databases using MongoDB and use aggregate function**

- **Insert**
  The insert() method in MongoDB is used to insert documents into a collection. There are multiple variations, including insertOne(), insertMany(), and the deprecated insert()
  Syntax: db.collection.insertOne(document)
- **Find**
  db.collection.find().pretty() fetches and displays all documents in a collection in a readable format.
- **Sort**
  sorts the query results in ascending order, while -1 sorts in descending order.
  Syntax db.collection.find().sort({ field: 1 })
- **Limit**
  db.collection.find().limit(number) limits the number of documents returned in the query result.
- **Skip**
  db.collection.find().skip(number) skips the specified number of documents in the query result
- **Update**
  db.collection.updateOne({ condition }, { $set: { field: value } }) updates a single document that matches the condition
- **Delete**
  db.collection.deleteOne({ condition }) deletes a single document that matches the condition
- **Drop**
  db.collection.drop() removes the entire collection from the database.

1. {emp_name : “Sharath”, designation: “sales”, salary: 15000}
   {emp_name : “Shyam”, designation: “manager”, salary: 50000}
   {emp_name : “Abraham”, designation: “superwiser”, salary: 35000}
   {emp_name : “Muhammed”, designation: “sales”, salary: 15000}
   {emp_name : “Rohith”, designation: “sales”, salary: 20000}
   {emp_name : “Nirmal”, designation: “driver”, salary: 20000}
   {emp_name : “Samuel”, designation: “superwiser”, salary: 35000}
   {emp_name : “Johns”, designation: “sales”, salary: 15000}
   1. Create a database Employee

      Query:

      i use Employee

Output:

switched to db Employee

2. Create a collection “db_employee”

Query:

ii. db.createCollection("db_employee")

Output:

{ ok: 1 }

3. Insert the above employee details to the collection called“db_employee”

Query:

iii. db.db_employee.insertMany([

{ emp_name: "Sharath", designation: "sales", salary: 15000 },

{ emp_name: "Shyam", designation: "manager", salary: 50000 },

{ emp_name: "Abraham", designation: "supervisor", salary: 35000 },

{ emp_name: "Muhammed", designation: "sales", salary: 15000 },

{ emp_name: "Rohith", designation: "sales", salary: 20000 },

{ emp_name: "Nirmal", designation: "driver", salary: 20000 },

{ emp_name: "Samuel", designation: "supervisor", salary: 35000 },

{ emp_name: "Johns", designation: "sales", salary: 15000 }

])

Output:

{

acknowledged: true,

insertedIds: {

'0': ObjectId('66637690c04587550246b7ad'),

'1': ObjectId('66637690c04587550246b7ae'),

```
    '2': ObjectId('66637690c04587550246b7af'),
    '3': ObjectId('66637690c04587550246b7b0'),
    '4': ObjectId('66637690c04587550246b7b1'),
    '5': ObjectId('66637690c04587550246b7b2'),
    '6': ObjectId('66637690c04587550246b7b3'),
    '7': ObjectId('66637690c04587550246b7b4')
  }
}
```

4. List the details of employee having ‘salary > 15000’ AND designation = “supervisor”

Query:

```
iv..db.db_employee.find({ salary: { $gt: 15000 }, designation: "supervisor" }).pretty()
```

Output:

```
[
  {
    _id: ObjectId('66637690c04587550246b7af'),
    emp_name: 'Abraham',
    designation: 'supervisor',
    salary: 35000
  },
  {
    _id: ObjectId('66637690c04587550246b7b3'),
    emp_name: 'Samuel',
    designation: 'supervisor',
    salary: 35000
  }
]
```

5. List the details of employee who working in ‘sales’ department

Query:

v. db.db_employee.find({ designation: "sales" }).pretty()

Output:

```
[
 {
  _id: ObjectId('66637690c04587550246b7ad'),
  emp_name: 'Sharath',
  designation: 'sales',
  salary: 15000
 },
 {
  _id: ObjectId('66637690c04587550246b7b0'),
  emp_name: 'Muhammed',
  designation: 'sales',
  salary: 15000
 },
 {
  _id: ObjectId('66637690c04587550246b7b1'),
  emp_name: 'Rohith',
  designation: 'sales',
  salary: 20000
 },
 {
  _id: ObjectId('66637690c04587550246b7b4'),
  emp_name: 'Johns',
  designation: 'sales',
  salary: 15000
 }
```

]

6. Update the emp_name :’’Sharath” to Abhijith

Query:

vi. db.db_employee.updateOne({ emp_name: "Sharath" }, { $set: { emp_name: "Abhijith" } })

Output:

{

acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0

}

7. Find the total sum of salary of employees under the sales department

Query:

```
db.db_employee.aggregate([
        { $match: { designation: "sales" } },
        { $group: { _id: null, totalSalary: { $sum: "$salary" } } }
    ])
```

Output

[{ _id: null, totalSalary: 65000 }]

Result:

Query has been executed successfully and output is obtained.