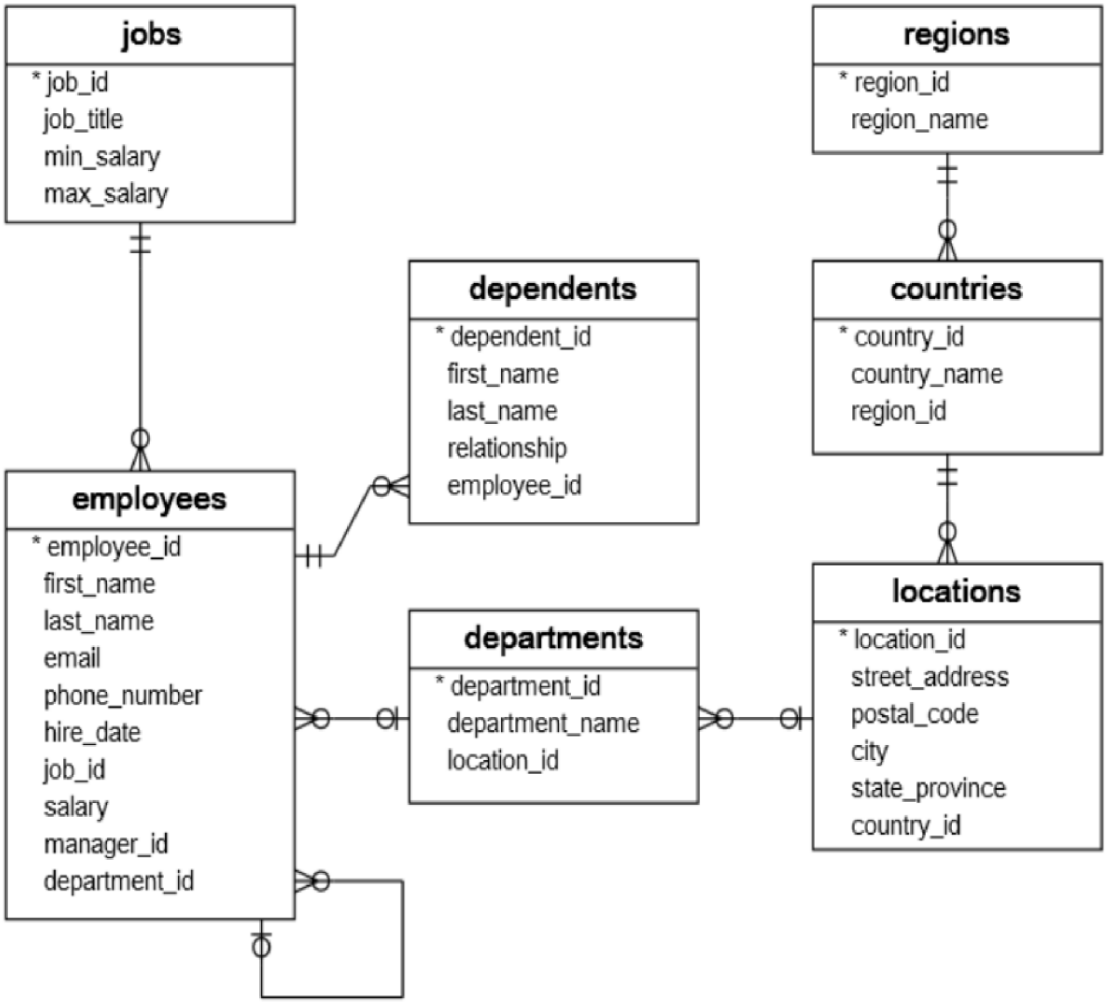
Lab Cycle 2 **Date:18/04/23**

Experiment No :3

AIM:

Familiarisation of subquery, join, views and set operation.

Consider the following Database Schema.



Query

regions TABLE

```
SQL> CREATE TABLE regions (
   region_id INT PRIMARY KEY,
   region_name VARCHAR(20)
);
```

countries TABLE

```
SQL> CREATE TABLE countries (
   country_id INT PRIMARY KEY,
   country_name VARCHAR(20),
   region_id INT,
   FOREIGN KEY (region_id) REFERENCES regions (region_id)
);
```

locations TABLE

```
SQL> CREATE TABLE locations (
   location_id INT PRIMARY KEY,
   street_address VARCHAR(30),
   postal_code BIGINT(6),
   city VARCHAR(30),
   state_province VARCHAR(35),
   country_id INT,
   FOREIGN KEY (country_id) REFERENCES countries (country_id)
);
```

departments TABLE

```
SQL> CREATE TABLE departments (
   department_id INT PRIMARY KEY,
```

```
    department_name VARCHAR(30),
    location_id INT,
    FOREIGN KEY (location_id) REFERENCES locations (location_id)
);
```

jobs TABLE

```
SQL> CREATE TABLE jobs (
    job_id INT PRIMARY KEY,
    job_title VARCHAR(50),
    min_salary FLOAT,
    max_salary FLOAT
);
```

employees TABLE

```
SQL>CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(10),
    last_name VARCHAR(10),
    email VARCHAR(10),
    phone_number NUMERIC(10),
    hire_date DATE,
    job_id INT,
    salary INT,
    manager_id INT,
    department_id INT,
```

```
    FOREIGN KEY (job_id) REFERENCES jobs (job_id),
    FOREIGN KEY (department_id) REFERENCES departments
(department_id)
);
```

dependents TABLE

```
SQL> CREATE TABLE dependents (
    dependent_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    relationship VARCHAR(80) DEFAULT NULL,
    employee_id INT,
    FOREIGN KEY (employee_id) REFERENCES employees (employee_id)
);
```

Values:

```
SQL> INSERT INTO `countries` VALUES
(1,'India',1),(2,'Japan',1),(3,'China',1),(4,'United
States',2),(5,'Canada',2),(6,'Mexico',2);

SQL> INSERT INTO `department` VALUES
(1,'Finance',1500),(2,'Marketing',1700),(3,'Human Resources',1500),(4,'Tech
Support',1700);
SQL> INSERT INTO `dependents` VALUES
(1,'Devika','Kumar','Sister',103),(2,'Jessica','Pearson','Wife',100);

SQL> INSERT INTO `employees` VALUES
(100,'Dave','Patel','dave@gmail.com',100007,'2023-06-
13',4,16000,NULL,3),(101,'Taylor','Smith','taysmith@gmail.com',10101010,
```

'2023-06-13', 4,15000, NULL,1), (103,'Devi','Kumar','devi@gmail.com', 100008,'2023-06-13',4,19000,NULL,3), (104,'Neil','Patrick','neil@gmail.com', 100009,'2023-06-13',1,9000,201,2), (105,'Michael','Jacob', 'michael@gmail.com', 1000100,'2023-06-13',2,6000,103,2), (201,'Smith','Anderson','smithy@gmail.com',1010220,'2023-06-13',4,6000,NULL,2),(204,'Zack','Damon','zacku@gmail.com',138138138,'2023-07-04',3,11000, 101,NULL),(205,'Stephan','Taylor','steph@gmail.com', 123123123,'2023-07-05',1,12000,NULL,1);

SQL> INSERT INTO `jobs` VALUES (1,'Software Engineer',10000,15000),(2,'Data Entry Clerk',5000,7000),(3,'Executive assistant',7000,8000),(4,'Manager',10000,20000);

SQL> INSERT INTO `locations` VALUES (1500,'1st Floor, Armenian Street',600001,'Kambam','Tamil Nadu',1),(1700,'Arya Samaj Road',110059,'Kanjhawala','Delhi',1);

SQL> INSERT INTO `regions` VALUES (1,'Asia'),(2,'North America'),(3,'Carribbean'),(4,'Europe');

1.
```
+-------------+-----------+-----------+
| employee_id | first_name | last_name |
+-------------+-----------+-----------+
|         104 | Neil       | Patrick   |
|         105 | Michael    | Jacob     |
|         201 | Smith      | Anderson  |
+-------------+-----------+-----------+
3 rows in set (0.0006 sec)
```

2.
```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|         100 | Dave       | Patel     |
|         101 | Taylor     | Smith     |
|         103 | Devi       | Kumar     |
+-------------+------------+-----------+
3 rows in set (0.0011 sec)
```

3.
```
+-------------+------------+-----------+
| employee_id | first_name | last_name |
+-------------+------------+-----------+
|         100 | Dave       | Patel     |
|         201 | Smith      | Anderson  |
+-------------+------------+-----------+
2 rows in set (0.0074 sec)
```

4.
```
+-------------+------------+-----------+--------+
| employee_id | first_name | last_name | salary |
+-------------+------------+-----------+--------+
|         100 | Dave       | Patel     |  16000 |
|         101 | Taylor     | Smith     |  15000 |
|         201 | Smith      | Anderson  |  16000 |
+-------------+------------+-----------+--------+
3 rows in set (0.0010 sec)
```

1. Find all employees who locate in the location with the id 1700.
SQL> elect e.employee_id,e.first_name,e.last_name from employees e join department d on e.department_id=d.department_id where location_id=1700 ;

2. 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 left join department d on e.department_id=d.department_id where location_id!=1700;

3. Finds the employees who have the highest salary.
SQL> select e.employee_id,e.first_name,e.last_name from employees e where e.salary=(select max(salary) from employees);

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

SQL > select e.employee_id,e.first_name,e.last_name,e.salary from employees e where e.salary >(select avg(salary) as avgSal from employees);

5.

```
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             3 | Human Resources |
|             1 | Finance         |
|             2 | Marketing       |
+---------------+-----------------+
3 rows in set (0.0007 sec)
```

6.

```
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             2 | Marketing       |
+---------------+-----------------+
1 row in set (0.0013 sec)
```

7.

```
+-------------+------------+-----------+--------+
| employee_id | first_name | last_name | salary |
+-------------+------------+-----------+--------+
|         103 | Devi       | Kumar     |  19000 |
+-------------+------------+-----------+--------+
1 row in set (0.0013 sec)
```

8.

```
+-------------+------------+-----------+--------+
| employee_id | first_name | last_name | salary |
+-------------+------------+-----------+--------+
|         103 | Devi       | Kumar     |  19000 |
+-------------+------------+-----------+--------+
1 row in set (0.0013 sec)
```

5. 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 department d JOIN employees e ON d.department_id = e.department_id WHERE e.salary > 10000 GROUP BY d.department_id;

6. 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 department d LEFT JOIN employees e ON d.department_id = e.department_id AND e.salary < 10000 where e.employee_id is not null group by d.department_id,d.department_name;

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

SQL > select e.employee_id,e.first_name,e.last_name,e.salary from employees e where e.salary> ALL (select min(salary) from employees group by department_id);

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

SQL > select e.employee_id,e.first_name,e.last_name,e.salary from employees e where e.salary>= All (select max(salary) from employees group by department_id);

9.

```
+----------------+
| Average salary |
+----------------+
| 13166.66666667 |
+----------------+
1 row in set (0.0012 sec)
```

10.

```
+-------------+------------+--------+--------------------+
| employee_id | first_name | salary | Salary difference  |
+-------------+------------+--------+--------------------+
|         100 | Dave       |  16000 |         -1500.0000 |
|         101 | Taylor     |  15000 |             0.0000 |
|         103 | Devi       |  19000 |          1500.0000 |
|         104 | Neil       |   9000 |          2000.0000 |
|         105 | Michael    |   6000 |         -1000.0000 |
|         201 | Smith      |   6000 |         -1000.0000 |
+-------------+------------+--------+--------------------+
6 rows in set (0.0013 sec)
```

11

```
+-------------+------------+--------+---------------+
| employee_id | first_name | salary | AverageSalary |
+-------------+------------+--------+---------------+
|         103 | Devi       |  19000 |    17500.0000 |
|         104 | Neil       |   9000 |     7000.0000 |
+-------------+------------+--------+---------------+
2 rows in set (0.0013 sec)
```

12. .

```
+-------------+------------+
| employee_id | first_name |
+-------------+------------+
|         100 | Dave       |
|         103 | Devi       |
+-------------+------------+
2 rows in set (0.0009 sec)
```

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

SQL > select avg(sub.avgSalary) as 'Average salary' from (select avg(e.salary) as avgSalary from employees e group by e.department_id) sub ;

10. 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 e.employee_id,e.first_name,e.salary,salary-(select avg(salary) from employees where department_id=e.department_id) as 'Salary difference' from employees e;

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

SQL > select e.employee_id,e.first_name,e.salary,(select avg(salary) from employees where department_id=e.department_id) as AverageSalary from employees e where e.salary>(select avg(salary) from employees where department_id=e.department_id);

12. Returns all employees who have no dependents.

SQL > select e.employee_id,e.first_name from employees e left join dependents d on e.employee_id=d.employee_id where d.dependent_id is not null;

13.

first_name	last_name	department_name
Taylor	Smith	Finance
Neil	Patrick	Marketing
Michael	Jacob	Marketing
Smith	Anderson	Marketing
Dave	Patel	Human Resources
Devi	Kumar	Human Resources

6 rows in set (0.0012 sec)

14.

first_name	last_name	Job Title	department_name
Taylor	Smith	Manager	Finance
Dave	Patel	Manager	Human Resources
Devi	Kumar	Manager	Human Resources

3 rows in set (0.0016 sec)

15.

department_name	street_address	postal_code	country_name
Human Resources	1st Floor, Armenian Street	600001	India
Finance	1st Floor, Armenian Street	600001	India
Marketing	Arya Samaj Road	110059	India

3 rows in set (0.0011 sec)

16.

first_name	last_name	department_id	department_name	Is in a department
Dave	Patel	3	Human Resources	True
Taylor	Smith	1	Finance	True
Devi	Kumar	3	Human Resources	True
Neil	Patrick	2	Marketing	True
Michael	Jacob	2	Marketing	True
Smith	Anderson	2	Marketing	True

6 rows in set (0.0013 sec)

13. 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 department d ON e.department_id = d.department_id WHERE d.department_id IN (1, 2, 3);

14. 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,(select job_title from jobs where job_id=e.job_id) as 'Job Title', d.department_name FROM employees e JOIN department d ON e.department_id = d.department_id WHERE d.department_id IN (1, 2, 3) AND e.salary>10000;

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

SQL> select dl.department_name, dl.street_address, dl.postal_code, c.country_name from(select d.department_name, l.street_address, l.postal_code, l.country_id from department d join locations l on d.location_id=l.location_id) dl join countries c on dl.country_id = c.country_id;

16. 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, d.department_name, CASE WHEN e.department_id IS NOT NULL THEN 'True' ELSE 'False' END AS 'Is in a department' FROM employees e LEFT JOIN department d ON e.department_id = d.department_id;

17.

first_name	last_name	department_id
Taylor	Smith	1
Stephan	Taylor	1

2 rows in set (0.0009 sec)

18.

first_name	last_name	department_id	department_name
Dave	Patel	3	Human Resources
Taylor	Smith	1	Finance
Devi	Kumar	3	Human Resources
Neil	Patrick	2	Marketing
Michael	Jacob	2	Marketing
Smith	Anderson	2	Marketing
Zack	Taylor	NULL	NULL
NULL	NULL	4	Tech Support

8 rows in set (0.0008 sec)

19.

first_name	manager_name
Dave	NULL
Taylor	NULL
Devi	NULL
Neil	NULL
Michael	Devi
Smith	NULL
Zack	Taylor

20.

first_name	last_name	department_id
Taylor	Smith	1
Stephan	Taylor	1

17. 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 ed.first_name, ed.last_name,ed.department_name, l.city,l.state_province from (select e.first_name, e.last_name, d.department_name, d.location_id from employees e join department d on e.department_id=d.department_id) ed join locations l on ed.location_id=l.location_id where ed.first_name like 'Z%';

18. 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 employees e left join department d on e.department_id=d.department_id union select e.first_name,e.last_name,d.department_id,d.department_name from employees e right join department d on e.department_id=d.department_id ;

19. 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 empData.first_name,empData.manager_name from (select e.first_name,(select first_name from employees where employee_id=e.manager_id)as manager_name from employees e)empData;

20. 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,d.department_id from employees e join department d on e.department_id=d.department_id where d.department_id=(select department_id from employees where last_name like 'Taylor') ;

21.

job_title	full_name	salary_difference
Software Engineer	Neil Patrick	6000
Software Engineer	Stephan Taylor	3000
Data Entry Clerk	Michael Jacob	1000
Executive assistant	Zack Damon	-3000
Manager	Dave Patel	4000
Manager	Taylor Smith	5000
Manager	Devi Kumar	1000
Manager	Smith Anderson	14000

8 rows in set (0.0012 sec)

22.

name	employee_id	phone_number	job_title	department_name	manager_name
Stephan Taylor	205	123123123	Software Engineer	Finance	NULL
Taylor Smith	101	10101010	Manager	Finance	NULL
Neil Patrick	104	100009	Software Engineer	Marketing	NULL
Michael Jacob	105	1000100	Data Entry Clerk	Marketing	Devi Kumar
Smith Anderson	201	1010220	Manager	Marketing	NULL
Dave Patel	100	100007	Manager	Human Resources	NULL
Devi Kumar	103	100008	Manager	Human Resources	NULL

23.

name
Taylor Smith

1 row in set (0.0009 sec)

24.

```
ERROR: 1288: The target table delhi_employees of the UPDATE is not updatable
```

21. 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 job_title,full_name,salary_difference from (select j.job_title,concat(e.fname,' ',lname) as full_name,j.max_sal-e.salary as salary_difference from employees e join jobs j on j.jobid=e.jobid) ej;

22. 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 delhi_employees as select ej.name,ej.employee_id,ej.phone_number,ej.job_title,d.department_name,ej.manager_name from (select concat(e.first_name,' ',e.last_name) as name,e.employee_id,e.department_id,e.phone_number,j.job_title,(select concat(first_name, ' ' ,last_name) as fullname from employees where employee_id=e.manager_id) as manager_name from employees e join jobs j on j.job_id=e.job_id) ej join department d on d.department_id=ej.department_id ;

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

SQL > select name from delhi_employees where job_title like 'manager' and department_name like 'finance';

24. 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 delhi_employees set phone_number=121121121 where name like 'Smith%';

25.

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id	department_id	dependent_id
101	Taylor	Smith	taysmith@gmail.com	10101010	2023-06-13	4	15000	NULL	1	NULL
104	Neil	Patrick	neil@gmail.com	100009	2023-06-13	1	9000	NULL	2	NULL
105	Michael	Jacob	michael@gmail.com	1000100	2023-06-13	2	6000	103	2	NULL
201	Smith	Anderson	smithy@gmail.com	1010220	2023-06-13	4	6000	NULL	2	NULL
204	Zack	Damon	zacku@gmail.com	138138138	2023-07-04	3	11000	101	NULL	NULL
205	Stephan	Taylor	steph@gmail.com	123123123	2023-07-05	1	12000	NULL	1	NULL

6 rows in set (0.0012 sec)

26.

employee_id	first_name	last_name	email	phone_number	manager_id
204	Zack	Damon	zacku@gmail.com	138138138	101
104	Neil	Patrick	neil@gmail.com	100009	201

27.

employee_id	first_name	last_name	No_of_Dependents
100	Dave	Patel	1
103	Devi	Kumar	1

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

SQL > select e.employee_id, e.first_name, e.last_name,e.email,e.phone_number,e.hire_date, e.job_id,e.salary, e.manager_id,e.department_id,d.dependent_id from employees e left join dependents d on d.employee_id=e.employee_id where dependent_id is null;

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

SQL> select employee_id, first_name,last_name,email,phone_number, manager_id from employees where manager_id=101 union select employee_id,first_name,last_name,email,phone_number,manager_id from employees where manager_id=201;

27. Display the details of employees who have at least one dependent.
SQL> select ed.employee_id, ed.first_name, ed.last_name, count(ed.employee_id) as No_of_Dependents from (select e.employee_id, e.first_name,e.last_name,d.dependent_id from employees e join dependents d on e.employee_id=d.employee_id) ed group by e.employee_id having No_of_Dependents>=1;

Result:

SQL query executed successfully and output is verified.

Lab Cycle 3 **Date: 06/06/23**

Experiment No :4

AIM:

Familiarization of Stored Procedure, Function, Cursor and Triggers.

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> CREATE TABLE ACCOUNT (
   account_id INT PRIMARY KEY,
   balance DECIMAL(10, 2)
);

SQL>INSERT INTO ACCOUNT (account_id, balance)
VALUES(1, 1500.00), (2, 2200.50), (3, 1800.75), (4, 1300.25), (5, 2500.00);

SQL> DELIMITER //
CREATE PROCEDURE WithdrawMoney (
   IN acID INT,
   IN withdraw_amount DECIMAL(10, 2)
)
BEGIN
   DECLARE current_balance DECIMAL(10, 2);
   DECLARE updatedBalance DECIMAL(10, 2);
```

i.

```
+-------------------------+
| status                  |
+-------------------------+
| Withdrawal successful. |
+-------------------------+
1 row in set (0.0086 sec)
```

ii.

```
+-----------------------------------------------------------------------------+
| status                                                                      |
+-----------------------------------------------------------------------------+
| Insufficient balance after withdrawal. Minimum balance requirement: 1000 |
+-----------------------------------------------------------------------------+
1 row in set (0.0009 sec)
```

```
 IF EXISTS (SELECT 1 FROM account WHERE account_id = acID) THEN
      SELECT balance INTO current_balance FROM account WHERE
account_id = acID;
      IF current_balance - withdraw_amount >= 1000 THEN
         SET updatedBalance = current_balance - withdraw_amount;
         UPDATE account SET balance = updatedBalance WHERE account_id
= acID;
         SELECT 'Withdrawal successful.' as status;
      ELSE
         SELECT 'Insufficient balance after withdrawal. Minimum balance
requirement: 1000' as status;
      END IF;
   ELSE
      SELECT 'Account does not exist.';
   END IF;
END;
//
DELIMITER ;
```

i. SQL> call WithdrawMoney(5,500);

ii. SQL> call WithdrawMoney(1,600);

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> SQL> CREATE TABLE Customer (
   customer_id INT PRIMARY KEY,
   name VARCHAR(50),
   city VARCHAR(50),
   credits DECIMAL(10, 2)
);
SQL> INSERT INTO Customer (customer_id, name, city, credits)
VALUES
   (1, 'John Doe', 'New York', 1500.00),
   (2, 'Jane Smith', 'Los Angeles', 2000.50),
   (3, 'Michael Johnson', 'Chicago', 1800.75),
   (4, 'Emily Brown', 'Houston', 1200.25),
   (5, 'William Wilson', 'Miami', 2500.00);

SQL> DELIMITER //

CREATE PROCEDURE GetCustomerDetailsByName ( IN customer_name
VARCHAR(50))
BEGIN
   IF EXISTS (SELECT 1 FROM CUSTOMER WHERE name =
customer_name) THEN
   SELECT * FROM Customer WHERE name = customer_name;
   ELSE
      SELECT 'Customer not found.';
   END IF;
```

i.

```
+-------------+----------+----------+---------+
| customer_id | name     | city     | credits |
+-------------+----------+----------+---------+
|           1 | John Doe | New York | 1500.00 |
+-------------+----------+----------+---------+
1 row in set (0.0009 sec)

Query OK, 0 rows affected (0.0009 sec)
```

ii.

```
+---------------------+
| Customer not found. |
+---------------------+
| Customer not found. |
+---------------------+
1 row in set (0.0008 sec)

Query OK, 0 rows affected (0.0008 sec)
```

```
END;
//
DELIMITER ;
```

i. CALL GetCustomerDetailsByName('John Doe');

ii. CALL GetCustomerDetailsByName('Johny');

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>DELIMITER //
CREATE PROCEDURE DetermineMembership (
   IN customer_credits DECIMAL(10, 2),
   OUT membership_level VARCHAR(50)
)
BEGIN
   DECLARE local_membership_level VARCHAR(50);

   IF customer_credits > 5000 THEN
      SET local_membership_level = 'Membership Platinum';
   ELSEIF customer_credits >= 1000 AND customer_credits <= 5000 THEN
      SET local_membership_level = 'Gold';
   ELSE
      SET local_membership_level = 'Silver';
   END IF;
      SET membership_level = local_membership_level;
END;
//
DELIMITER ;
```

```
+---------------------+
| @membership_level   |
+---------------------+
| Membership Platinum |
+---------------------+
1 row in set (0.0014 sec)
```

```
SQL> SET @membership_level = '';
SQL> CALL DetermineMembership(7000, @membership_level);
SQL> SELECT @membership_level;
```

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

```
SQL> CREATE TABLE employees (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   salary DECIMAL(10, 2)
);
SQL> INSERT INTO employees (id, name, salary) VALUES
   (1, 'John', 10000),
   (2, 'James', 10000),
   (3, 'James', 12000);
```

Function

```
DELIMITER //
CREATE FUNCTION UpdateEmployeeSalaryByName (employee_name
VARCHAR(100), new_salary DECIMAL(10, 2)) RETURNS INT
BEGIN
   DECLARE employee_count INT;
   SELECT COUNT(*) INTO employee_count
   FROM employees WHERE name = employee_name;
   IF employee_count = 1 THEN
      UPDATE employees SET salary = new_salary WHERE name =
employee_name;
   END IF;
   RETURN employee_count;
```

i.

```
CALL UpdateEmployeeDetails('John', 15000);
+----------------------+
| Status               |
+----------------------+
| Updated successfully |
+----------------------+
1 row in set (0.0078 sec)
```

ii.

```
CALL UpdateEmployeeDetails('James', 15000);
+--------------------+
| Status             |
+--------------------+
| Multiple employees |
+--------------------+
1 row in set (0.0005 sec)
```

iii.

```
CALL UpdateEmployeeDetails('Haris', 15000);
+-------------+
| Status      |
+-------------+
| No Employee |
+-------------+
1 row in set (0.0005 sec)
```

```
END;
//
DELIMITER ;
```

Procedure

```
DELIMITER //
CREATE PROCEDURE UpdateEmployeeDetails (
    IN employee_name VARCHAR(100),
    IN new_salary DECIMAL(10, 2)
)
BEGIN
    DECLARE employee_count INT;
    SET employee_count =
UpdateEmployeeSalaryByName(employee_name,new_salary);
    IF employee_count = 0 THEN
        SELECT 'No Employee' AS Status;
    ELSEIF employee_count = 1 THEN
        SELECT 'Updated successfully' AS Status;
    ELSE
        SELECT 'Multiple employees' AS Status;
    END IF;
END;
//
DELIMITER ;
```

i. CALL UpdateEmployeeDetails('John', 15000);

ii. CALL UpdateEmployeeDetails('James', 15000);

iii. CALL UpdateEmployeeDetails('Haris', 15000);

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.

Designation	Daily Wage Amount
Assistance Professor	1750/day
Clerk	750/day
Programmer	1250/day

```
SQL> CREATE TABLE Emp_salary (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100),
    no_of_working_days INT,
    designation VARCHAR(50),
    salary DECIMAL(10, 2)
);
SQL> INSERT INTO Emp_salary (emp_id, emp_name, no_of_working_days, designation)
VALUES
    (1, 'Employee1', 20, 'Assistance Professor'),
    (2, 'Employee2', 25, 'Clerk'),
    (3, 'Employee3', 22, 'Programmer'),
    (4, 'Employee4', 18, 'Assistance Professor'),
    (5, 'Employee5', 23, 'Clerk');
```

Procedure

```
SQL> DELIMITER //
CREATE PROCEDURE CalculateEmployeeSalary()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE empId INT;
DECLARE workingDays INT;
   DECLARE position VARCHAR(100);
   DECLARE sal DECIMAL(10, 2);
   DECLARE employeeCursor CURSOR FOR
      SELECT emp_id, no_of_working_days, designation, salary FROM
Emp_salary;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done =
TRUE;
   OPEN employeeCursor;
   read_loop: LOOP
      FETCH employeeCursor INTO empId, workingDays, position, sal;
      IF done THEN
         LEAVE read_loop;
      END IF;
      IF position = 'Assistance Professor' THEN
         SET sal = workingDays * 1750;
      ELSEIF position = 'Clerk' THEN
         SET sal = workingDays * 750;
      ELSE
         SET sal = workingDays * 1250;
      END IF;
```

emp_id	emp_name	no_of_working_days	designation	salary
1	Employee1	20	Assistance Professor	35000.00
2	Employee2	25	Clerk	18750.00
3	Employee3	22	Programmer	27500.00
4	Employee4	18	Assistance Professor	31500.00
5	Employee5	23	Clerk	17250.00

5 rows in set (0.0006 sec)

```
UPDATE Emp_salary SET salary = sal WHERE emp_id = empId;
   END LOOP;
   CLOSE employeeCursor;
END //
DELIMITER ;
SQL> CALL CalculateEmployeeSalary();
SQL> select * from emp_salary;
```

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.

```
CREATE TABLE Bill (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    pre_reading INT,
    cur_reading INT,
    unit INT,
    amount DECIMAL(10, 2)
);
```

```
INSERT INTO Bill (customer_id, name, pre_reading, cur_reading, unit, amount)
VALUES
    (1, 'Customer1', 100, 200, 0, 0),
    (2, 'Customer2', 150, 350, 0, 0),
    (3, 'Customer3', 100, 400, 0, 0),
    (4, 'Customer4', 150, 600, 0, 0);
```

Procedure

```
SQL> DELIMITER //
CREATE PROCEDURE CalculateElectricityBill()
BEGIN
   DECLARE cust_id INT;
   DECLARE pre_read INT;
   DECLARE cur_read INT;
   DECLARE consumed_units INT;
   DECLARE bill_rate DECIMAL(10, 2);
   DECLARE customer_cursor CURSOR FOR
      SELECT customer_id, pre_reading, cur_reading, (cur_reading -
pre_reading) AS units_consumed
      FROM Bill;
   OPEN customer_cursor;
   read_loop: LOOP
      FETCH customer_cursor INTO cust_id, pre_read, cur_read,
consumed_units;
      IF cust_id IS NULL THEN
         LEAVE read_loop;
      END IF;
      IF consumed_units <= 100 THEN
         SET bill_rate = consumed_units * 2.00;
      ELSEIF consumed_units <= 200 THEN
         SET bill_rate = (100 * 2.00) + ((consumed_units - 100) * 2.50);
      ELSEIF consumed_units <= 300 THEN
         SET bill_rate = (100 * 2.00) + (100 * 2.50) + ((consumed_units - 200) *
3.00);
      ELSE
```

customer_id	name	pre_reading	cur_reading	unit	amount
1	Customer1	100	200	100	200.00
2	Customer2	150	350	200	450.00
3	Customer3	100	400	300	750.00
4	Customer4	150	600	450	1350.00

4 rows in set (0.0004 sec)

```
SET bill_rate = (100 * 2.00) + (100 * 2.50) + (100 * 3.00) + ((consumed_units -
300) * 4.00);

      END IF;

      UPDATE Bill SET amount = bill_rate, unit = (cur_read - pre_read)
WHERE customer_id = cust_id;

   END LOOP;

   CLOSE customer_cursor;

END;

//

DELIMITER ;


SQL> call CalculateElectricityBill();

SQL> select * from bill;
```

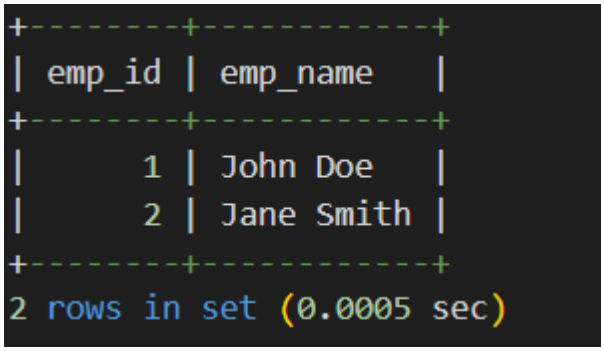
7. Create a trigger on employee table such that whenever a row is deleted, it is moved to history table named ‘Emp_history’ with the same structure as employee table. ‘Emp_history’ will contain an additional column “Date_of_deletion” to store the date on which the row is removed.[After Delete Trigger]

```
SQL> CREATE TABLE Emp_table (
    emp_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL
);

SQL> CREATE TABLE Emp_history (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL,
    Date_of_deletion DATE NOT NULL
);

SQL> INSERT INTO Emp_table (emp_name)
VALUES
    ('John Doe'),
    ('Jane Smith'),
    ('Mike Johnson'),
    ('Emily Davis'),
    ('Robert Brown')
```

Procedure

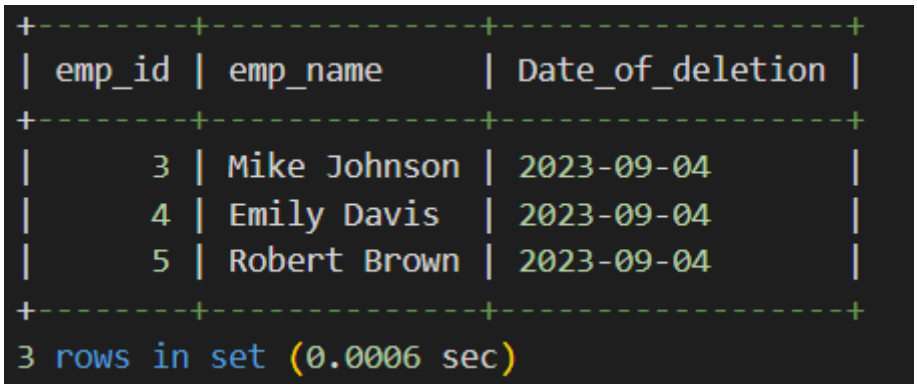
```
DELIMITER //
CREATE TRIGGER MoveToEmpHistory
```

i.



```
+--------+------------+
| emp_id | emp_name   |
+--------+------------+
|      1 | John Doe   |
|      2 | Jane Smith |
+--------+------------+
2 rows in set (0.0005 sec)
```

ii.



```
+--------+--------------+------------------+
| emp_id | emp_name     | Date_of_deletion |
+--------+--------------+------------------+
|      3 | Mike Johnson | 2023-09-04       |
|      4 | Emily Davis  | 2023-09-04       |
|      5 | Robert Brown | 2023-09-04       |
+--------+--------------+------------------+
3 rows in set (0.0006 sec)
```

```
AFTER DELETE ON Emp_table
FOR EACH ROW
BEGIN
   INSERT INTO Emp_history (emp_id, emp_name, Date_of_deletion)
   VALUES (OLD.emp_id, OLD.emp_name, NOW());
END;
//
DELIMITER ;
```

i. SQL> select * from emp_table;

ii.SQL> select * from emp_history;

```
+--------+------------+-----------+----------+
| emp_id | FIRST_NAME | LAST_NAME | JOB_ID   |
+--------+------------+-----------+----------+
|      1 | John       | Doe       | MANAGER  |
|      2 | Mike       | Brown     | ENGINEER |
+--------+------------+-----------+----------+
2 rows in set (0.0005 sec)
```

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 TABLE emp_details (
   emp_id INT AUTO_INCREMENT PRIMARY KEY,
   FIRST_NAME VARCHAR(255),
   LAST_NAME VARCHAR(255),
   JOB_ID VARCHAR(255)
);
DELIMITER //

SQL> CREATE TRIGGER BeforeInsert_emp_details
BEFORE INSERT ON emp_details
FOR EACH ROW
BEGIN
   SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
   SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
   SET NEW.JOB_ID = UPPER(NEW.JOB_ID);
END;
//
DELIMITER ;

SQL> INSERT INTO emp_details (FIRST_NAME, LAST_NAME, JOB_ID)
VALUES ('        John ', ' Doe ', 'manager');

select * from emp_details;
```

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]

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	

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<90 -> 'VERY GOOD'

- If PER_MARKS>=60 AND PER_MARKS<75 -> 'GOOD'

- If PER_MARKS>=40 AND PER_MARKS<60 -> 'AVERAGE'

- If PER_MARKS<40-> 'NOT PROMOTED'

```
SQL> CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    sub1 INT,
    sub2 INT,
    sub3 INT,
    sub4 INT,
    sub5 INT,
```

```
total INT,
    per_marks DECIMAL(5, 2),
    grade VARCHAR(20)
);
SQL> INSERT INTO students (name, sub1, sub2, sub3, sub4, sub5)
VALUES
    ('John Doe', 50, 60, 70, 80, 0),
    ('Mike Brown', 0, 0, 0, 0, 0);

SQL> DELIMITER //
CREATE TRIGGER calculate_marks_and_grade
BEFORE UPDATE ON students
FOR EACH ROW
BEGIN
    DECLARE total_marks INT;
    DECLARE percentage DECIMAL(5, 2);
    DECLARE student_grade VARCHAR(20);
    SET total_marks = NEW.sub1 + NEW.sub2 + NEW.sub3 + NEW.sub4 +
NEW.sub5;
    SET percentage = total_marks / 5;
    IF percentage >= 90 THEN
        SET student_grade = 'EXCELLENT';
    ELSEIF percentage >= 75 AND percentage < 90 THEN
        SET student_grade = 'VERY GOOD';
    ELSEIF percentage >= 60 AND percentage < 75 THEN
        SET student_grade = 'GOOD';
    ELSEIF percentage >= 40 AND percentage < 60 THEN
        SET student_grade = 'AVERAGE';
```

```
+------------+------------+------+------+------+------+------+-------+-----------+-------+
| student_id | name       | sub1 | sub2 | sub3 | sub4 | sub5 | total | per_marks | grade |
+------------+------------+------+------+------+------+------+-------+-----------+-------+
|          1 | John Doe   |   50 |   60 |   70 |   80 |  100 |   360 |     72.00 | GOOD  |
|          2 | Mike Brown |    0 |    0 |    0 |    0 |    0 |  NULL |      NULL | NULL  |
+------------+------------+------+------+------+------+------+-------+-----------+-------+
2 rows in set (0.0004 sec)
```

```
ELSE
      SET student_grade = 'NOT PROMOTED';
   END IF;
   SET NEW.total = total_marks;
   SET NEW.per_marks = percentage;
   SET NEW.grade = student_grade;
END;
//
DELIMITER ;
```

```
SQL> update students set sub5 = 100 where student_id = 1;
SQL> select * from students;
```

Result:

SQL query executed successfully and output is verified.

Lab Cycle 4 **Date: 27/06/23**

Experiment No :5

AIM:

1: Create a Table ‘Order_Details’ with the given data and perform the following operations

mysql->CREATE TABLE Order_Details (Order_ID INT,Product_Name

VARCHAR(100),Order_Num INT,Order_Date DATE);

INSERT INTO Order_Details (Order_ID, Product_Name, Order_Num, Order_Date)

VALUES (1, 'Laptop', 5544, '2020-02-01'),

(2, 'Mouse', 3322, '2020-02-11'),

(3, 'Desktop', 2135, '2020-01-05'),

(4, 'Mobile', 3432, '2020-02-22'),

(5, 'Anti-Virus', 5648, '2020-03-10');

a) Start a new transaction and insert 2 rows into the table Order_Details.

mysql->START TRANSACTION;

INSERT INTO Order_Details (Order_ID, Product_Name, Order_Num, Order _Date) VALUES (6, 'FireWall', 1006, '2023-07-22'), (7, 'Keyboard', 1007, '2023-07 22');

b)

```
mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
|        4 | Mobile       |      3432 | 2020-02-22 |
|        5 | Anti-Virus   |      5648 | 2020-03-10 |
|        6 | FireWall     |      1006 | 2023-07-22 |
|        7 | Keyboard     |      1007 | 2023-07-22 |
+----------+-------------+-----------+------------+
```

c)

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
|        4 | Mobile       |      3432 | 2020-02-22 |
|        5 | Anti-Virus   |      5648 | 2020-03-10 |
+----------+-------------+-----------+------------+
```

b) Display the details of Order_Details.

mysql-> select * from Order_Details;

c) Rollback the current transaction and check the contents of the table Order_Details.

mysql-> ROLLBACK;

select*from Order_Details;

e)

```
mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
+----------+-------------+-----------+------------+
3 rows in set (0.00 sec)
```

f.

```
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
+----------+-------------+-----------+------------+
3 rows in set (0.00 sec)
```

g)

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.01 sec)

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

h)

```
mysql> Update Order_Details set Product_Name='ASUS Laptop' where Order_Num=5544;
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

d) Start a new transaction and delete 2 rows from the table Order_Details.

```
mysql-> START TRANSACTION;
        delete from Order_Details where Order_ID in(4,5);
```

e) Display the details of Order_Details.

```
mysql->select * from Order_Details;
```

f) Commit the current transaction and check the details of the table Order_Details.

```
mysql->COMMIT;
        select * from Order_Details;
```

g) Disable autocommit and create a savepoint named ‘Check_Updates’.

```
mysql->SET autocommit = 0;
        SAVEPOINT Check_Updates;
```

h) Update details of the order with Order_Num 5544.

```
mysql->update Order_Details set Product_Name='ASUS Laptop' where
    Order_Num=5544;
```

i) Insert 2 more rows into the table.

```
mysql->INSERT INTO Order_Details (Order_ID, Product_Name, Order_Num,
Order_Date) VALUES(6, 'FireWall', 1006, '2023-07-22'),(7, 'Laptop', 1007,
'2023-07-22');
```

j) Create a savepoint named ‘Check_delete’

```
mysql-> SAVEPOINT
Check_delete;
```

k) Delete order details of a product named ‘Laptop’.

```
mysql->delete from Order_Details where Product_Name='Laptop';
```

l)

```
mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | ASUS Laptop  |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
|        6 | FireWall     |      1006 | 2023-07-22 |
+----------+-------------+-----------+------------+
4 rows in set (0.00 sec)
```

m)

```
mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | ASUS Laptop  |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
|        6 | FireWall     |      1006 | 2023-07-22 |
|        7 | Laptop       |      1007 | 2023-07-22 |
+----------+-------------+-----------+------------+
5 rows in set (0.00 sec)
```

n)

```
mysql> ROLLBACK TO SAVEPOINT Check_Updates;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from Order_Details;
+----------+-------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+-------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
+----------+-------------+-----------+------------+
```

l) Display the current details of the table Order_Details.

mysql-> select * from Order_Details;

m) Rollback to savepoint ‘Check_delete’ and check the details of the table Order_Details.

mysql->ROLLBACK TO SAVEPOINT Check_delete;

select*from Order_Details;

n) Rollback to savepoint ‘Check_Updates’ and check the details of the table Order_Details.

mysql->ROLLBACK TO SAVEPOINT Check_Updates;

select*from Order_Details;

o)

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

mysql> SET autocommit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from Order_Details;
+----------+--------------+-----------+------------+
| Order_ID | Product_Name | Order_Num | Order_Date |
+----------+--------------+-----------+------------+
|        1 | Laptop       |      5544 | 2020-02-01 |
|        2 | Mouse        |      3322 | 2020-02-11 |
|        3 | Desktop      |      2135 | 2020-01-05 |
+----------+--------------+-----------+------------+
3 rows in set (0.00 sec)
```

o) Commit the changes and enable autocommit.

mysql-> COMMIT;

SET autocommit=1;

RESULT:

Program run successfully and output is obtained

Lab Cycle 5 **Date: 11/07/23**

Experiment No :6

AIM:

1. Build sample collections/documents to perform query operation.

i.Create a database (Eg : MyCev)

test> use MyCev

ii.Create a collection (Eg: db_mca)

MyCev> db.createCollection("db_mca")

iii.Create a collection (Eg: db_cs)

MyCev> db.createCollection("db_cs")

iv.Insert 10 data to the collection

MyCev> db.db_mca.insertMany([{ name: "ram", rollno: 15,branch :"mca",mark:70 } , {name: "rani", rollno: 16,branch :"mca",mark:72 },{name: "mahi", rollno: 17,branch :"mca",mark:65} ,{name: "anandhan", rollno: 05,branch :"mca",mark:75 },{name: "sreekutty", rollno: 18,branch :"mca",mark:76},{name: "devika", rollno: 10,branch :"mca",mark:78 },{name: "adithya", rollno: 03,branch :"mca",mark:80 },{name: "maya", rollno: 20,branch :"mca",mark:82 },{name: "nidhi", rollno: 22,branch :"mca",mark:84},{name: "arathy", rollno: 07,branch :"mca",mark:88 }]);

MyCec> db.db_cs.insertMany([{name: "John", rollno: 101, branch: "CS", mark: 70},{name: "Alice", rollno: 102, branch: "CS", mark: 72},{name: "Bob", rollno: 103, branch: "CS", mark: 65},{name: "Eva", rollno: 104, branch: "CS", mark:

v) MyCev> db.db_mca.find().limit(5);

```
[
  {
    _id: ObjectId("64ede662964e5f10ee8afc61"),
    name: 'ram',
    rollno: 15
    branch:’cs’
  },
  {
    _id: ObjectId("64ede684964e5f10ee8afc62"),
    name: 'maya',
    rollno: 25,
    branch: 'mca',mark: 70},
{
    _id: ObjectId("64ede9df964e5f10ee8afc6b"),
    name: 'anandhan',
    rollno: 5,
    branch: 'mca',
    mark: 75
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6c"),
    name: 'sreekutty',
    rollno: 18,
    branch: 'mca',
    mark: 76
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6d"),
    name: 'devika',
    rollno: 10,
    branch: 'mca',
    mark: 78
  }]
```

vi) MyCev> db.db_mca.find().skip(2)

```
[
   {
    _id: ObjectId("64ede9df964e5f10ee8afc6a"),
    name: 'mahi',
    rollno: 17,
    branch: 'mca',
    mark: 65
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6b"),
    name: 'anandhan',
    rollno: 5,
    branch: 'mca',
    mark: 75
```

75},{name: "David", rollno: 105, branch: "CS", mark: 76},{name: "Sarah", rollno: 106, branch: "CS", mark: 78},{name: "Michael", rollno: 107, branch: "CS", mark: 80},{name: "Emily", rollno: 108, branch: "CS", mark: 82},{name: "James", rollno: 109, branch: "CS", mark: 84},{name: "Olivia", rollno: 110, branch: "CS", mark: 88}]);

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

MyCev> db. db_mca.find().limit(5);

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

MyCev> db.db_mca.find().skip(2)

```
},
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6c"),
    name: 'sreekutty',
    rollno: 18,
    branch: 'mca',
    mark: 76
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6d"),
    name: 'devika',
    rollno: 10,
    branch: 'mca',
    mark: 78
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc6e"),
    name: 'adithya',
    rollno: 3,
    branch: 'mca',
    mark: 80
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc70"),
    name: 'nidhi',
    rollno: 22,
    branch: 'mca',
    mark: 84
  },
  {
    _id: ObjectId("64ede9df964e5f10ee8afc71"),
    name: 'arathy',
    rollno: 7,
    branch: 'mca',
    mark: 88
  }]

vii)
  MyCev> db.db_mca.find().sort({ field_name: 1 });
[
  {
    _id: ObjectId("64ede7a1964e5f10ee8afc63"),
    name: 'ram',
    rollno: 15,
    branch: 'mca',
    mark: 70
  },
  {
    _id: ObjectId("64ede7a1964e5f10ee8afc64"),
    name: 'rani',
    rollno: 16,
    branch: 'mca',
    mark: 72
  },
  {
```

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

```
MyCev> db. db_mca.find().sort({ field_name: 1 });
```

viii.Delete data from the collection

```
MyCev> db. db_cs.deleteMany({ key:1 });
```

```
_id: ObjectId("64ede808964e5f10ee8afc67"),
  name: 'mahi',
  rollno: 17,
  branch: 'mca',
  mark: 65
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc6b"),
  name: 'anandhan',
  rollno: 5,
  branch: 'mca',
  mark: 75
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc6c"),
  name: 'sreekutty',
  rollno: 18,
  branch: 'mca',
  mark: 76
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc6d"),
  name: 'devika',
  rollno: 10,
  branch: 'mca',
  mark: 78
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc6e"),
  name: 'adithya',
  rollno: 3,
  branch: 'mca',
  mark: 80
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc6f"),
  name: 'maya',
  rollno: 20,
  branch: 'mca',
  mark: 82
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc70"),
  name: 'nidhi',
  rollno: 22,
  branch: 'mca',
  mark: 84
},
{
  _id: ObjectId("64ede9df964e5f10ee8afc71"),
  name: 'arathy',
  rollno: 7,
  branch: 'mca',
```

```
  mark: 88
 }]
```

ix.Drop the collection (db_cs)

MyCev> db. db_cs.drop();

x.Drop Database

MyCev> db.dropDatabase()

iv. [

{ "_id": ObjectId("64f2c64864b3ab60751038b1"), "name": "maya", "rollno": 15, "subject": "english", "mark": 70 },

{ "_id": ObjectId("64f2c64864b3ab60751038b2"), "name": "ardra", "rollno": 16, "subject": "maths", "mark": 75 },

{ "_id": ObjectId("64f2c64864b3ab60751038b3"), "name": "mahima", "rollno": 17, "subject": "botany", "mark": 65 },

{ "_id": ObjectId("64f2c64864b3ab60751038b4"), "name": "anandhan", "rollno": 5, "subject": "zoology", "mark": 75 },

{ "_id": ObjectId("64f2c64864b3ab60751038b5"), "name": "sreekutty", "rollno": 18, "subject": "commerce", "mark": 76 },

{ "_id": ObjectId("64f2c64864b3ab60751038b6"), "name": "devika", "rollno": 10, "subject": "geography", "mark": 78 },

{ "_id": ObjectId("64f2c64864b3ab60751038b7"), "name": "adithya", "rollno": 3, "subject": "history", "mark": 80 },

{ "_id": ObjectId("64f2c64864b3ab60751038b8"), "name": "maya", "rollno": 20, "subject": "malayalam", "mark": 82 },

{ "_id": ObjectId("64f2c64864b3ab60751038b9"), "name": "theertha", "rollno": 22, "subject": "maths", "mark": 84 },

{ "_id": ObjectId("64f2c64864b3ab60751038ba"), "name": "akshaya", "rollno": 7, "subject": "english", "mark": 88 }

]

2. Design Databases using MongoDB and perform CRUD operations.

i.Create a database Myclass.

test> use Myclass

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

Myclass> db.createCollection("db_students")

iii.Insert details of 10 students in a class

Myclass>db.students.insertMany([{name:"maya",rollno:15,subject:"english",mark:70},{name:"ardra",rollno: 16,subject :"maths",mark:75 },{name: "mahima",rollno:17,subject:"botany",mark:65},{name:"anandhan",rollno: 05,subject:"zoology",mark:75},{name:"sreekutty",rollno:18,subject:"commerce",mark:76},{name:"devika",rollno: 10,subject:"geography",mark:78 },{name: "adithya", rollno: 03,subject:"history",mark:80 },{name: "maya", rollno:20,subject:"malayalam",mark:82},{name:"theertha",rollno:22,subject:"maths",mark:84},{name:"akshaya",rollno:07,subject:"english",mark:8 8 }]);

iv.List the entire students in the class

Myclass> db.students.find();

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

Myclass> db.students.updateOne({name: "theertha"},{ $set:{

v)

```
Myclass> db.students.updateOne({name: "theertha"},{ $set:{ mark:90}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

vi)

```
Myclass> db.students.deleteOne({})
{ acknowledged: true, deletedCount: 1 }
```

mark:90}})

vi.Delete the data of first student in the collection

Myclass> db.students.deleteOne({})

```
[
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aaad"),
    emp_name: 'Abraham',
    designation: 'superwiser',
    salary: 35000
  },
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aab1"),
    emp_name: 'Samuel',
    designation: 'superwiser',
    salary: 35000
  }
]
```

v)

```
Employee> db.employee.find({designation:"sales"});
[
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aaab"),
    emp_name: 'Sharath',
    designation: 'sales',
    salary: 15000
  },
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aaae"),
    emp_name: 'Muhammed',
    designation: 'sales',
    salary: 15000
  },
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aaaf"),
    emp_name: 'Rohith',
    designation: 'sales',
    salary: 20000
  },
  {
    _id: ObjectId("64f303cdbf0c07e03ec2aab2"),
    emp_name: 'Johns',
    designation: 'sales',
    salary: 15000
  }
]
```

3.{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}

i. Create a database Employee

test> use Employee

ii. Create a collection “db_employee”

MyCev> db.createCollection("db_employee")

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

Employee> db.employee.insertMany([{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}]);

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

vi)

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

vii)

```
[ { _id: null, totalSalary: 65000 } ]
```

Employee>db.employee.find({salary:{$gt:15000},designation:"superwiser"});

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

Employee> db.employee.find({designation:"sales"});

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

Employee>db.db_employee.updateOne({emp_name:"Sharath"},{

$set:{emp_name:"Abhijith"}});

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

Employee>db.employee.aggregate([{$match:{designation:"sales" }},

{$group: { _id: null,totalSalary:{$sum:"$salary" } }}]);

RESULT:

Program run successfully and output is obtained