EXPERIMENT-1

Aim- To study the DDL and DML commands.

Software Used- MySQL

DDL Commands-

1. **CREATE COMMAND-** This command is used to create the database or a table.

```
mysql> Create database Study;
Query OK, 1 row affected (0.02 sec)
```

```
mysql> Create Table Student( Rollno int, Name varchar(30) ,Age int );
Query OK, 0 rows affected (0.09 sec)
```

2. **ALTER COMMAND –** This command is used to make changes in the structure of the table.

```
mysql> alter table Student add Primary Key(Rollno);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc Student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Rollno | int         | NO   | PRI | NULL    |       |
| Name   | varchar(30) | YES  |     | NULL    |       |
| Age    | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> alter table Student rename Data;
Query OK, 0 rows affected (0.02 sec)

mysql> alter table Data add Address varchar(30);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table Data modify Address varchar(25);
Query OK, 4 rows affected (0.12 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> alter table Data drop Address;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. **DROP COMMAND-** This command is used to delete a whole database or just a table.

```
mysql> Drop Table Data;
Query OK, 0 rows affected (0.05 sec)

mysql> Select * from Data;
ERROR 1146 (42S02): Table 'study.data' doesn't exist
mysql>
```

4. **TRUNCATE COMMAND-** This command is used to delete the data inside the table not the whole table.

```
mysql> truncate table Student;
Query OK, 0 rows affected (0.06 sec)

mysql> select * from Student;
Empty set (0.00 sec)
```

5. **RENAME COMMAND**- This is used to rename an object existing in the database.

```
mysql> alter table Student rename Data;
Query OK, 0 rows affected (0.02 sec)
```

DML COMMANDS-

1. **INSERT COMMAND-** It is used to insert data into a table.

```
mysql> Insert into Student values(1,'Raunaq',20);
Query OK, 1 row affected (0.05 sec)

mysql> Insert into Student values(2,'Priyanka',25);
Query OK, 1 row affected (0.04 sec)

mysql> Insert into Student values(3,'Ram',30);
Query OK, 1 row affected (0.04 sec)

mysql> Insert into Student values(4,'Ajay',32);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> Select * from Student;
+--------+----------+------+
| Rollno | Name     | Age  |
+--------+----------+------+
|      1 | Raunaq   |   20 |
|      2 | Priyanka |   25 |
|      3 | Ram      |   30 |
|      4 | Ajay     |   32 |
+--------+----------+------+
4 rows in set (0.00 sec)
```

2. **UPDATE COMMAND -** It is used to update existing data within a table.

```
mysql> Update Student set age=26 where Rollno=2;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from Student;
+--------+----------+------+
| Rollno | Name     | Age  |
+--------+----------+------+
|      1 | Raunaq   |   20 |
|      2 | Priyanka |   26 |
|      3 | Ram      |   30 |
+--------+----------+------+
3 rows in set (0.00 sec)
```

3. **DELETE COMMAND-** It is used to delete records from a table .

```
mysql> Delete from Student;
Query OK, 3 rows affected (0.05 sec)

mysql> select * from Student;
Empty set (0.00 sec)
```

EXPERIMENT 2

Aim- To study Primary Key, Foreign Key and the various types of Joins in SQL.

Software Used- MySQL

Theory-

1. **Primary Key-** A primary key is a column (or set of columns) in a table that uniquely identifies each row in the table. It cannot contain null values and must be unique across all rows in the table. Only one primary key is allowed in a table.

```
mysql> create table employee(empid int not null primary key ,firstname varchar(30),lastname varchar(30),salary int);
Query OK, 0 rows affected (0.03 sec)

mysql> desc employee;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| empid     | int         | NO   | PRI | NULL    |       |
| firstname | varchar(30) | YES  |     | NULL    |       |
| lastname  | varchar(30) | YES  |     | NULL    |       |
| salary    | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

2. **Foreign Key-** The foreign key is a group of one or more columns in a database to uniquely identify another database record in some other table to maintain the referential integrity. It is also known as the referencing key that establishes a relationship between two different tables in a database. A foreign key always matches the primary key column in another table.

```
mysql> create table dept(deptid int not null primary key ,deptname varchar(30),empid int ,foreign key(empid) references employee(empi
d));
Query OK, 0 rows affected (0.07 sec)

mysql> desc dept;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| deptid   | int         | NO   | PRI | NULL    |       |
| deptname | varchar(30) | YES  |     | NULL    |       |
| empid    | int         | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Tables-

```
mysql> select * from employee;
+-------+-----------+----------+--------+
| empid | firstname | lastname | salary |
+-------+-----------+----------+--------+
|     1 | Raunaq    | Duggal   |  12000 |
|     2 | Amit      | Arora    |  18000 |
|     3 | Rahul     | Gupta    |  20000 |
|     4 | Ajay      | Rana     |  22000 |
+-------+-----------+----------+--------+
4 rows in set (0.00 sec)
```

```
mysql> select * from dept;
+--------+-----------+-------+
| deptid | deptname  | empid |
+--------+-----------+-------+
|     20 | Marketing |     3 |
|     21 | HR        |     2 |
|     23 | Software  |     1 |
|     24 | Finance   |     4 |
+--------+-----------+-------+
4 rows in set (0.00 sec)
```

3. **Joins –**
   a. **Outer Join-**
   1. **Left Outer Join-**

```
mysql> select employee.empid,employee.firstname,employee.lastname,employee.salary,dept.deptid,dept.deptname from employee left outer
join dept on employee.empid=dept.empid;
+-------+-----------+----------+--------+--------+-----------+
| empid | firstname | lastname | salary | deptid | deptname  |
+-------+-----------+----------+--------+--------+-----------+
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |
+-------+-----------+----------+--------+--------+-----------+
4 rows in set (0.04 sec)
```

   2. **Right Outer Join-**

```
mysql> select employee.empid,employee.firstname,employee.lastname,employee.salary,dept.deptid,dept.deptname from employee right outer
 join dept on employee.empid=dept.empid;
+-------+-----------+----------+--------+--------+-----------+
| empid | firstname | lastname | salary | deptid | deptname  |
+-------+-----------+----------+--------+--------+-----------+
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |
+-------+-----------+----------+--------+--------+-----------+
4 rows in set (0.00 sec)
```

   3. **Full Outer Join-**

```
mysql> select employee.empid,employee.firstname,employee.lastname,employee.salary,dept.deptid,dept.deptname from employee left outer
join dept on employee.empid=dept.empid union select employee.empid,employee.firstname,employee.lastname,employee.salary,dept.deptid,d
ept.deptname from employee right outer join dept on employee.empid=dept.empid;
+-------+-----------+----------+--------+--------+-----------+
| empid | firstname | lastname | salary | deptid | deptname  |
+-------+-----------+----------+--------+--------+-----------+
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |
+-------+-----------+----------+--------+--------+-----------+
4 rows in set (0.04 sec)
```

   b. **Inner Join-**

```
mysql> select employee.empid,firstname,lastname,salary from employee,dept where employee.empid=dept.empid;
+-------+-----------+----------+--------+
| empid | firstname | lastname | salary |
+-------+-----------+----------+--------+
|     1 | Raunaq    | Duggal   |  12000 |
|     2 | Amit      | Arora    |  18000 |
|     3 | Rahul     | Gupta    |  20000 |
|     4 | Ajay      | Rana     |  22000 |
+-------+-----------+----------+--------+
4 rows in set (0.00 sec)
```

   c. **Cross Join-**

```
mysql> select * from employee cross join dept;
+-------+-----------+----------+--------+--------+-----------+-------+
| empid | firstname | lastname | salary | deptid | deptname  | empid |
+-------+-----------+----------+--------+--------+-----------+-------+
|     4 | Ajay      | Rana     |  22000 |     20 | Marketing |     3 |
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |     3 |
|     2 | Amit      | Arora    |  18000 |     20 | Marketing |     3 |
|     1 | Raunaq    | Duggal   |  12000 |     20 | Marketing |     3 |
|     4 | Ajay      | Rana     |  22000 |     21 | HR        |     2 |
|     3 | Rahul     | Gupta    |  20000 |     21 | HR        |     2 |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |     2 |
|     1 | Raunaq    | Duggal   |  12000 |     21 | HR        |     2 |
|     4 | Ajay      | Rana     |  22000 |     23 | Software  |     1 |
|     3 | Rahul     | Gupta    |  20000 |     23 | Software  |     1 |
|     2 | Amit      | Arora    |  18000 |     23 | Software  |     1 |
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |     1 |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |     4 |
|     3 | Rahul     | Gupta    |  20000 |     24 | Finance   |     4 |
|     2 | Amit      | Arora    |  18000 |     24 | Finance   |     4 |
|     1 | Raunaq    | Duggal   |  12000 |     24 | Finance   |     4 |
+-------+-----------+----------+--------+--------+-----------+-------+
16 rows in set (0.00 sec)
```

d. Left Join-

```
mysql> select e.empid,e.firstname,e.lastname,e.salary,d.deptid,d.deptname from employee e left join dept d on e.empid=d.empid;
+-------+-----------+----------+--------+--------+-----------+
| empid | firstname | lastname | salary | deptid | deptname  |
+-------+-----------+----------+--------+--------+-----------+
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |
+-------+-----------+----------+--------+--------+-----------+
4 rows in set (0.00 sec)
```

e. Right Join-

```
mysql> select e.empid,e.firstname,e.lastname,e.salary,d.deptid,d.deptname from employee e right join dept d on e.empid=d.empid;
+-------+-----------+----------+--------+--------+-----------+
| empid | firstname | lastname | salary | deptid | deptname  |
+-------+-----------+----------+--------+--------+-----------+
|     3 | Rahul     | Gupta    |  20000 |     20 | Marketing |
|     2 | Amit      | Arora    |  18000 |     21 | HR        |
|     1 | Raunaq    | Duggal   |  12000 |     23 | Software  |
|     4 | Ajay      | Rana     |  22000 |     24 | Finance   |
+-------+-----------+----------+--------+--------+-----------+
4 rows in set (0.00 sec)
```

f. Full Join-

```
mysql> select * from employee full join dept on employee.empid=dept.empid;
```

EXPERIMENT 3

Aim- To study and perform constraints, Group By, Order By and Having Clauses.

Software Used- MySQL

Theory-

1. **Constraints-** Constraints are specific rules for data in a table. They can be specified when the table is created or by using ALTER TABLE statement.

```
mysql> Insert into Emp(Emp_no,Name,Salary,Age) Values('1', 'Nandini', '10000', '20'),('2', 'Tvisha', '10000', '20'),('3', 'Yuvraj', '10000', '20');
Query OK, 3 rows affected (0.18 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM emp;
+--------+---------+----------+-----+-----------------+
| Emp_no | Name    | salary   | age | GroupIdentifier |
+--------+---------+----------+-----+-----------------+
|      1 | Nandini | 10000.00 |  20 |           10000 |
|      2 | Tvisha  | 10000.00 |  20 |           10000 |
|      3 | Yuvraj  | 10000.00 |  20 |           10000 |
+--------+---------+----------+-----+-----------------+
3 rows in set (0.00 sec)
```

2. For the experiment, the following tables were designed to perform GROUP BY, ORDER BY and HAVING clause operations.

```
mysql> CREATE TABLE college ( name VARCHAR(100), year INT, subjects VARCHAR(255));
Query OK, 0 rows affected (0.16 sec)
```

```
mysql> insert into college (name,year,subjects) Values ('Nandini','3','English'),('Tvisha','3','Maths'),('Yuvraj','3','Science');
Query OK, 3 rows affected (0.44 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM college;
+---------+------+----------+
| name    | year | subjects |
+---------+------+----------+
| Nandini |    3 | English  |
| Tvisha  |    3 | Maths    |
| Yuvraj  |    3 | Science  |
+---------+------+----------+
3 rows in set (0.00 sec)
```

- **Group By:** The GROUP BY statement groups rows that have the same values as summary rows. They are often used with aggregate functions to group the result set by one or more columns.

```
mysql> SELECT subjects, year, COUNT(*) FROM college GROUP BY subjects, year;
+----------+------+----------+
| subjects | year | COUNT(*) |
+----------+------+----------+
| English  |    3 |        1 |
| Maths    |    3 |        1 |
| Science  |    3 |        1 |
+----------+------+----------+
3 rows in set (0.19 sec)
```

- **Order By:** The Order By clause in SQL, is used to sort fetched data in either ascending or descending according to one or more columns.

```
mysql> SELECT * FROM college ORDER BY year;
+---------+------+----------+
| name    | year | subjects |
+---------+------+----------+
| Nandini |    3 | English  |
| Tvisha  |    3 | Maths    |
| Yuvraj  |    3 | Science  |
+---------+------+----------+
3 rows in set (0.00 sec)
```

- **HAVING CLAUSE:** The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

```
mysql> ALTER TABLE Emp ADD GroupIdentifier INT;
Query OK, 0 rows affected (0.40 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE Emp SET GroupIdentifier = Salary;
Query OK, 3 rows affected (0.06 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

```
mysql> SELECT name, SUM(salary) AS TotalSalary FROM emp GROUP BY Name HAVING SUM(Salary) > 1000;
+---------+-------------+
| name    | TotalSalary |
+---------+-------------+
| Nandini |    10000.00 |
| Tvisha  |    10000.00 |
| Yuvraj  |    10000.00 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Conclusion: Constraints, Order By and Having Clauses were studied and performed.

EXPERIMENT -7

Aim- To Practice View Command.

Software Used- MySQL

Theory- A view is a virtual table based on the result of a SQL query. It is a stored query that can be used as a table, and it doesn't contain the actual data itself but provides a way to present data stored in other tables in a structured manner. Views are particularly useful for simplifying complex queries, restricting access to specific columns or rows of a table, or providing a consistent and more understandable interface to the underlying data.

Code-

```
mysql> SELECT * FROM employee;
+-------+-----------+----------+--------+
| empid | firstnamr | lastname | salary |
+-------+-----------+----------+--------+
|     1 | Nandini   | Sain     |  15000 |
|     2 | Angad     | Singh    |  10000 |
|     3 | Sristhi   | Iyer     |  10000 |
+-------+-----------+----------+--------+
3 rows in set (0.03 sec)

mysql> CREATE VIEW details AS SELECT first_namr, last_name FROM employee;
ERROR 1054 (42S22): Unknown column 'first_namr' in 'field list'
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| assignment         |
| database           |
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| study              |
| sys                |
| world              |
+--------------------+
9 rows in set (0.00 sec)

mysql> CREATE VIEW details AS SELECT firstnamr, lastname FROM employee;
Query OK, 0 rows affected (0.73 sec)

mysql> SELECT * FROM details;
+-----------+----------+
| firstnamr | lastname |
+-----------+----------+
| Nandini   | Sain     |
| Angad     | Singh    |
| Sristhi   | Iyer     |
+-----------+----------+
3 rows in set (0.03 sec)
```

```
mysql> SELECT * FROM Dept;
+---------+--------------+------+
| Dept_ID | Dept_Name    | EID  |
+---------+--------------+------+
|       1 | Department 1 | 1001 |
|       2 | Department 2 | 1002 |
|       3 | Department 3 | 1003 |
+---------+--------------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Dept;
+---------+--------------+------+
| Dept_ID | Dept_Name    | EID  |
+---------+--------------+------+
|       1 | Department 1 | 1001 |
|       2 | Department 2 | 1002 |
|       3 | Department 3 | 1003 |
+---------+--------------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM employee;
+-------+-----------+----------+--------+
| empid | firstnamr | lastname | salary |
+-------+-----------+----------+--------+
|     1 | Nandini   | Sain     |  15000 |
|     2 | Angad     | Singh    |  10000 |
|     3 | Sristhi   | Iyer     |  10000 |
+-------+-----------+----------+--------+
3 rows in set (0.00 sec)

mCREATE VIEW details2 AS SELECT employee.firstnamr, employee.salary, Dept.Dept_Name FROM employee, Dept WHERE employee.empid = Dept.Dept_ID;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM details2;
+-----------+--------+--------------+
| firstnamr | salary | Dept_Name    |
+-----------+--------+--------------+
| Nandini   |  15000 | Department 1 |
| Angad     |  10000 | Department 2 |
| Sristhi   |  10000 | Department 3 |
```

```
mysql> CREATE OR REPLACE VIEW details AS SELECT employee.firstnamr, employee.salary, Dept.Dept_Name FROM employee, Dept WHERE employee.empid = Dept.Dept_ID;

Query OK, 0 rows affected (0.05 sec)

mysql> SELECT * FROM details2;
+-----------+--------+--------------+
| firstnamr | salary | Dept_Name    |
+-----------+--------+--------------+
| Nandini   |  15000 | Department 1 |
| Angad     |  10000 | Department 2 |
| Sristhi   |  10000 | Department 3 |
+-----------+--------+--------------+
3 rows in set (0.00 sec)
```

EXPERIMENT-8

Aim- To practice PL/SQL commands.

Software Used- Oracle Apex

Code: Basics: Syntax, Comments, Variable Attributes, Conditionals: IF-THEN-ELSE, Case, Loops – For, While

1. **Syntax-**

```
1   DECLARE
2   message varchar2(30):= 'Hello, World! From Raunaq';
3   BEGIN
4   dbms_output.put_line(message);
5   END;
6   /
7
```

Results | Explain | Describe | Saved SQL | History

```
Hello, World! From Raunaq

Statement processed.

0.01 seconds
```

2. **Comments-**

```
1   DECLARE
2   -- variable declaration
3   message varchar2(20):= 'Hello, World!';
4    BEGIN
5   /*
6   * PL/SQL executable statement(s)
7   */
8   dbms_output.put_line(message);
9   END;
10  /
```

Results | Explain | Describe | Saved SQL | History

```
Hello, World!

Statement processed.

0.01 seconds
```

3. **Example-**

```
1   DECLARE
2   a integer := 30; b integer := 40; c integer;
3   f real;
4    BEGIN
5   c := a + b;
6   dbms_output.put_line('Value of c: ' || c);
7   f := 100.0/3.0;
8   dbms_output.put_line('Value of f: ' || f);
9   END;
10
```

Results | Explain | Describe | Saved SQL | History

```
Value of c: 70
Value of f: 33.333333333333333333333333333333333333
Statement processed.
```

4. **Variable Attributes-**

a. **% Type-**

```
DECLARE
  SALARY EMP.SAL%TYPE;
BEGIN
  SELECT SAL INTO SALARY FROM EMP WHERE EMPNO = :P1_ECODE;
  :P1_RESULT := 'Salary of ' || :P1_ECODE || ' is = ' || SALARY;
END;
```

b. **% Row Type-**

```
DECLARE
  EMPLOYEE EMP%ROWTYPE;
BEGIN
  EMPLOYEE.EMPNO := :P1_EMPNO; -- Assuming P1_EMPNO is an APEX item
  EMPLOYEE.ENAME := :P1_ENAME; -- Assuming P1_ENAME is an APEX item
  INSERT INTO EMP (EMPNO, ENAME)
  VALUES (EMPLOYEE.EMPNO, EMPLOYEE.ENAME);

  dbms_output.put_line('Row Inserted');
END;
```

5. **Conditionals-**

   a. **IF -THEN-ELSE-**

```
DECLARE
a number(3) := 500;
 BEGIN
-- check the boolean condition using if statement IF( a < 20 ) THEN
-- if condition is true then print the following dbms_output.put_line('a is less than 20 ' );
IF( a < 20 )
THEN
dbms_output.put_line('a is less than 20 ' );
-- if condition is true then print the following dbms_output.put_line('a is less than 20 ' );
ELSE
dbms_output.put_line('a is not less than 20 ' );
END IF;
dbms_output.put_line('value of a is : ' || a);
```

```
14      END;
15
```

Results | Explain | Describe | Sa

```
a is not less than 20
value of a is : 500

Statement processed.

0.00 seconds
```

   b. **CASE-**

```
DECLARE
grade char(1) := 'A';
BEGIN
CASE grade
when 'A' then dbms_output.put_line('Excellent'); when 'B' then dbms_output.put_line('Very good'); when 'C' then dbms_output.put_line('Good'); when 'D' the
when 'F' then dbms_output.put_line('Passed with Grade'); else dbms_output.put_line('Failed');
END CASE;
 END;
```

Results | Explain | Describe | Saved SQL | History

```
Excellent

Statement processed.
```

6. **Loop-**

   a. **For-**

```
DECLARE VAR1 NUMBER;
 BEGIN VAR1:=10;
FOR VAR2 IN 1..10 LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

```
10
20
30
40
50
60
70
80
90
100

Statement processed.
```

b. **While-**

```
DECLARE VAR1 NUMBER; VAR2 NUMBER;
BEGIN VAR1:=200; VAR2:=1;
WHILE (VAR2<=10) LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2); VAR2:=VAR2+1;
END LOOP;
END;
```

```
200
400
600
800
1000
1200
1400
1600
1800
2000

Statement processed.


0.00 seconds
```

Lab Assignment-1

Q.1 Create the following tables

Course(course_no char(4), course_name varchar(20))

Course_fee(course_no char(4), full_part char(1) (F/P), fees number(10))

course_no and full_part should be unique

Student(prospectus_no number(10), name varchar(20), address varchar(30), phone_no number(11), D_O_B date, total_amt number(10,2), amt_paid number(10,2), installment char(1) (I/F))

Installment(prospectus_no number(10) (foreign key) on delete cascade, installment_amt number(10,2), due_dt date, paid char(1) (P,U))

prospectus_no and due_dt should be unique

Course_taken(prospectus_no number(10) (foreign key), course_no char(4), start_dt date, full_part char(1) (F/P), time_slot char(2), performance varchar(20))

SQL Queries:

1. Retrieve name and course no of all the students.
2. List the names of students who have paid the full amount at the time of admission.
3. Find the names of students starting with A.
4. Print the names of students whose total amount is not equal to amount due.
5. Count the number of students who have joined in current year, current month.
6. Determine the maximum and minimum course fees.
7. Increase the fee of oracle by 50%.
8. Print the details of courses whose fees are between 5000 and 10000.
9. Display the admission date in Date, Month, Year format.
10. Find out in which course maximum number of students have taken admission.
11. Change the course_name from Unix to Unix Operating System,
12. Display the admission date in DD-MONTH-YYYY format.
13. Get the sum of amount to be collected from students in this month.
14. Find out in which course the maximum number of students have taken admission in the current month.
15. Select the students who have not yet paid full amount of fees.

```
mysql> Use assignment
Database changed
mysql> CREATE TABLE Course (course_no CHAR(4) PRIMARY KEY, course_name VARCHAR(20) NOT NULL);
Query OK, 0 rows affected (3.47 sec)

mysql> CREATE TABLE Course_fee (course_no CHAR(4), full_part CHAR(1) CHECK (full_part IN ('F', 'P')) NOT NULL, fees DECIMAL(10, 2) NOT NULL, PRIMARY KEY (co
urse_no, full_part), FOREIGN KEY (course_no) REFERENCES Course(course_no));
Query OK, 0 rows affected (1.96 sec)

mysql> CREATE TABLE Student ( prospectus_no INT(10) PRIMARY KEY, name VARCHAR(20) NOT NULL, address VARCHAR(30), phone_no BIGINT(11), D_O_B DATE, total_amt
DECIMAL(10, 2), amt_paid DECIMAL(10, 2), installment CHAR(1) CHECK (installment IN ('I', 'F')));
Query OK, 0 rows affected, 2 warnings (1.76 sec)

mysql> CREATE TABLE Installment ( prospectus_no INT(10), installment_amt DECIMAL(10, 2) NOT NULL, due_dt DATE NOT NULL, paid CHAR(1) CHECK (paid IN ('P', 'U
')) NOT NULL, PRIMARY KEY (prospectus_no, due_dt), FOREIGN KEY (prospectus_no) REFERENCES Student(prospectus_no) ON DELETE CASCADE);
Query OK, 0 rows affected, 1 warning (0.38 sec)

mysql> CREATE TABLE Course_taken ( prospectus_no INT(10), course_no CHAR(4), start_dt DATE, full_part CHAR(1) CHECK (full_part IN ('F', 'P')), time_slot CHA
R(2), performance VARCHAR(20), FOREIGN KEY (prospectus_no) REFERENCES Student(prospectus_no), FOREIGN KEY (course_no) REFERENCES Course(course_no));
Query OK, 0 rows affected, 1 warning (0.81 sec)

mysql> INSERT INTO Course (course_no, course_name) VALUES ('C001', 'Mathematics'), ('C002', 'Physics'), ('C003', 'Chemistry');
Query OK, 3 rows affected (2.81 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Course_fee (course_no, full_part, fees) VALUES ('C001', 'F', 5000.00), ('C001', 'P', 3000.00), ('C002', 'F', 5500.00), ('C002', 'P', 3200
.00), ('C003', 'F', 4800.00), ('C003', 'P', 2800.00);
Query OK, 6 rows affected (0.65 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Student (prospectus_no, name, address, phone_no, D_O_B, total_amt, amt_paid, installment) VALUES (1, 'Alice', '123 Main St', 1234567890,
'2000-05-15', 5000.00, 2000.00, 'F'), (2, 'Bob', '456 Elm St', 9876543210, '2001-03-20', 5500.00, 3200.00, 'F'), (3, 'Charlie', '789 Oak St', 5551234567, '1
999-12-10', 4800.00, 2800.00, 'I');
Query OK, 3 rows affected (0.81 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Installment (prospectus_no, installment_amt, due_dt, paid) VALUES (1, 1000.00, '2023-09-10', 'P'), (1, 2000.00, '2023-10-10', 'U'), (2, 2
000.00, '2023-09-15', 'P'), (2, 1200.00, '2023-10-15', 'U'), (3, 1000.00, '2023-09-20', 'P');
Query OK, 5 rows affected (0.07 sec)
```

```
mysql> INSERT INTO Course_taken (prospectus_no, course_no, start_dt, full_part, time_slot, performance) VALUES (1, 'C001', '2023-09-01', 'F', 'AM', 'Excelle
nt'), (2, 'C002', '2023-09-05', 'P', 'PM', 'Good'), (3, 'C001', '2023-09-03', 'F', 'AM', 'Average');
Query OK, 3 rows affected (0.51 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT s.name, ct.course_no FROM Student s JOIN Course_taken ct ON s.prospectus_no = ct.prospectus_no;
+---------+-----------+
| name    | course_no |
+---------+-----------+
| Alice   | C001      |
| Bob     | C002      |
| Charlie | C001      |
+---------+-----------+
3 rows in set (0.00 sec)

mysql> ^C
mysql> SELECT name FROM Student WHERE total_amt = amt_paid;
Empty set (0.10 sec)

mysql> SELECT name FROM Student WHERE name LIKE 'A%';
+-------+
| name  |
+-------+
| Alice |
+-------+
1 row in set (0.04 sec)

mysql> SELECT name FROM Student WHERE total_amt != amt_paid;
+---------+
| name    |
+---------+
| Alice   |
| Bob     |
| Charlie |
+---------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Student WHERE YEAR(D_O_B) = YEAR(CURRENT_DATE()) AND MONTH(D_O_B) = MONTH(CURRENT_DATE());
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.90 sec)

mysql> SELECT MAX(fees) AS max_fee, MIN(fees) AS min_fee FROM Course_fee;
+---------+---------+
| max_fee | min_fee |
+---------+---------+
| 5500.00 | 2800.00 |
+---------+---------+
1 row in set (0.00 sec)

mysql> UPDATE Course_fee SET fees = fees * 1.5 WHERE course_no = 'C001'; -- Assuming 'C001' represents the 'oracle' course
Query OK, 2 rows affected (0.11 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM Course_fee WHERE fees BETWEEN 5000 AND 10000;
+-----------+-----------+---------+
| course_no | full_part | fees    |
+-----------+-----------+---------+
| C001      | F         | 7500.00 |
| C002      | F         | 5500.00 |
+-----------+-----------+---------+
2 rows in set (0.00 sec)

mysql> SELECT DATE_FORMAT(D_O_B, '%d-%M-%Y') AS admission_date FROM Student;
+------------------+
| admission_date   |
+------------------+
| 15-May-2000      |
| 20-March-2001    |
| 10-December-1999 |
+------------------+
3 rows in set (1.36 sec)
```

```
mysql> SELECT course_no FROM ( SELECT course_no, COUNT(*) AS student_count FROM Course_taken GROUP BY course_no ORDER BY student_count DESC LIMIT 1 ) AS max
_students;
+-----------+
| course_no |
+-----------+
| C001      |
+-----------+
1 row in set (0.14 sec)

mysql> UPDATE Course SET course_name = 'Unix Operating System' WHERE course_name = 'Unix';
Query OK, 0 rows affected (0.56 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> SELECT DATE_FORMAT(D_O_B, '%d-%M-%Y') AS admission_date FROM Student;
+------------------+
| admission_date   |
+------------------+
| 15-May-2000      |
| 20-March-2001    |
| 10-December-1999 |
+------------------+
3 rows in set (0.00 sec)

mysql> SELECT SUM(total_amt - amt_paid) AS total_amount_to_collect FROM Student WHERE YEAR(D_O_B) = YEAR(CURRENT_DATE()) AND MONTH(D_O_B) = MONTH(CURRENT_DA
TE());
+-------------------------+
| total_amount_to_collect |
+-------------------------+
|                    NULL |
+-------------------------+
1 row in set (0.25 sec)

mysql> SELECT course_no FROM ( SELECT ct.course_no, COUNT(*) AS student_count FROM Course_taken ct JOIN Student s ON ct.prospectus_no = s.prospectus_no WHER
E YEAR(s.D_O_B) = YEAR(CURRENT_DATE()) AND MONTH(s.D_O_B) = MONTH(CURRENT_DATE()) GROUP BY ct.course_no ORDER BY student_count DESC LIMIT 1 ) AS max_student
s;
Empty set (0.56 sec)
```

```
mysql> SELECT name FROM Student WHERE total_amt != amt_paid;
+---------+
| name    |
+---------+
| Alice   |
| Bob     |
| Charlie |
+---------+
3 rows in set (0.00 sec)
```

Q2. Create the following tables and answer the queries: (Take appropriate data types and relationships to define the columns and then insert relevant data).

1. SUPPLIER(SNO, SNAME, STATUS, CITY)
2. PARTS(PNO, PNAME, COLOR, WEIGHT, CITY)
3. PROJECT(JNO, JNAME, CITY)
4. SPJ(SNO, PNO, JNO, QTY)

SQL Queries:

1. Get sno values for suppliers who supply project j1.
2. Get sno values for suppliers who supply project j1 with part p1.
3. Get jname values for projects supplied by supplier s1.
4. Get color values for parts supplied by supplier s1.
5. Get pno values for parts supplied to any project in London.
6. Get sno values for suppliers who supply project j1 with a red part.
7. Get sno values for suppliers who supply a London or Paris project with a red part.
8. Get pno values for parts supplied to any project by a supplier in the same city.
9. Get pno values for parts supplied to any project in London by a supplier in London.
10. Get jno values for projects supplied by at least one supplier not in the same city.
11. Get all pairs of city values such that a supplier in the first city supplies a project in the second city.
12. Get sno values for suppliers who supply the same part to all projects.
13. Get pno values for parts supplied to all projects in London.
14. Get sname values for suppliers who supplies at least one red part to any project.
15. Get total quantity of part p1 supplied by supplier s1.
16. Get the total number of projects supplied by supplier s3.
17. Change color of all red parts to orange.
18. Get sname values for suppliers who supply to both projects j1 and j2.
19. Get all city, pno, city triples such that a supplier in the first city supplies the specified part to a project in the second city.
20. Get jnames for those project which are supplied by supplier XYZ.

```
mysql> CREATE TABLE SUPPLIER ( SNO INT PRIMARY KEY, SNAME VARCHAR(255), STATUS VARCHAR(255), CITY VARCHAR(255) ); -- Insert data into the SUPPLIER table INS
ERT INTO SUPPLIER (SNO, SNAME, STATUS, CITY) VALUES (1, 'Supplier A', 'Active', 'New York'), (2, 'Supplier B', 'Inactive', 'London'), (3, 'Supplier C', 'Act
ive', 'Paris'), (4, 'Supplier D', 'Active', 'New York'), (5, 'Supplier E', 'Active', 'London');
Query OK, 0 rows affected (0.45 sec)

mysql> CREATE TABLE PARTS ( PNO INT PRIMARY KEY, PNAME VARCHAR(255), COLOR VARCHAR(255), WEIGHT DECIMAL(10, 2), CITY VARCHAR(255) ); -- Insert data into the
 PARTS table INSERT INTO PARTS (PNO, PNAME, COLOR, WEIGHT, CITY) VALUES (101, 'Part X', 'Red', 5.5, 'New York'), (102, 'Part Y', 'Blue', 4.0, 'London'), (10
3, 'Part Z', 'Red', 3.2, 'Paris'), (104, 'Part W', 'Green', 2.7, 'New York'), (105, 'Part V', 'Red', 6.1, 'London');
Query OK, 0 rows affected (1.26 sec)

mysql> CREATE TABLE PROJECT ( JNO INT PRIMARY KEY, JNAME VARCHAR(255), CITY VARCHAR(255) );
Query OK, 0 rows affected (2.16 sec)

mysql> INSERT INTO PROJECT (JNO, JNAME, CITY) VALUES (201, 'Project 1', 'New York'), (202, 'Project 2', 'London'), (203, 'Project 3', 'Paris'); -- Create th
e SPJ table CREATE TABLE SPJ ( SNO INT, PNO INT, JNO INT, QTY INT, PRIMARY KEY (SNO, PNO, JNO), FOREIGN KEY (SNO) REFERENCES SUPPLIER(SNO), FOREIGN KEY (PNO
) REFERENCES PARTS(PNO), FOREIGN KEY (JNO) REFERENCES PROJECT(JNO) ); -- Insert data into the SPJ table INSERT INTO SPJ (SNO, PNO, JNO, QTY) VALUES (1, 101,
 201, 100), (1, 102, 201, 50), (2, 102, 202, 75), (3, 103, 202, 60), (3, 103, 203, 40), (4, 104, 201, 120), (4, 105, 202, 90), (5, 105, 203, 30);
Query OK, 3 rows affected (0.30 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> CREATE TABLE SPJ ( SNO INT, PNO INT, JNO INT, QTY INT, PRIMARY KEY (SNO, PNO, JNO), FOREIGN KEY (SNO) REFERENCES SUPPLIER(SNO), FOREIGN KEY (PNO) REF
ERENCES PARTS(PNO), FOREIGN KEY (JNO) REFERENCES PROJECT(JNO));
Query OK, 0 rows affected (5.62 sec)

mysql> INSERT INTO PARTS (PNO, PNAME, COLOR, WEIGHT, CITY) VALUES     (101, 'Part 1', 'Red', 10, 'City A'), (102, 'Part 2', 'Blue', 20, 'City B'), (103, 'Pa
rt 3', 'Red', 15, 'City C'), (104, 'Part 4', 'Green', 30, 'City D'), (105, 'Part 5', 'Red', 25, 'City E');
Query OK, 5 rows affected (0.29 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> INSERT INTO SPJ (SNO, PNO, JNO, QTY) VALUES(1, 101, 201, 100), (1, 102, 201, 50), (2, 102, 202, 75), (3, 103, 202, 60), (3, 103, 203, 40), (4, 104, 2
01, 120), (4, 105, 202, 90), (5, 105, 203, 30);
Query OK, 8 rows affected (0.12 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> SELECT DISTINCT SNO FROM SPJ WHERE JNO = 201;
+-----+
| SNO |
+-----+
|   1 |
|   4 |
+-----+
2 rows in set (0.07 sec)

mysql> SELECT DISTINCT SNO FROM SPJ WHERE JNO = 201 AND PNO = 101;
+-----+
| SNO |
+-----+
|   1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT JNAME FROM PROJECT WHERE JNO IN (SELECT JNO FROM SPJ WHERE SNO = 1);
+-----------+
| JNAME     |
+-----------+
| Project 1 |
+-----------+
1 row in set (0.06 sec)
```

```
mysql> SELECT DISTINCT COLOR FROM PARTS WHERE PNO IN (SELECT PNO FROM SPJ WHERE SNO = 1);
+-------+
| COLOR |
+-------+
| Red   |
| Blue  |
+-------+
2 rows in set (0.00 sec)

mysql> SELECT DISTINCT PNO FROM SPJ WHERE JNO IN (SELECT JNO FROM PROJECT WHERE CITY = 'London');
+-----+
| PNO |
+-----+
| 102 |
| 103 |
| 105 |
+-----+
3 rows in set (0.00 sec)

mysql> SELECT DISTINCT SNO FROM SPJ WHERE JNO = 201 AND PNO IN (SELECT PNO FROM PARTS WHERE COLOR = 'Red');
+-----+
| SNO |
+-----+
|   1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT SNO FROM SPJ WHERE JNO IN (SELECT JNO FROM PROJECT WHERE CITY IN ('London', 'Paris')) AND PNO IN (SELECT PNO FROM PARTS WHERE COLOR =
 'Red');
+-----+
| SNO |
+-----+
|   3 |
|   4 |
|   5 |
+-----+
3 rows in set (0.00 sec)
```

Q3. Create the required Tables

1. 1-Display each employee name and hiredate of systems department.
2. Write query to calculate length of service of each employee.
3. Find the second maximum salary of all employees.
4. Display all employee name and department name in department name order.
5. 5-Find the name of lowest paid employee for each manager.
6. 6-Display the department that has no employee.
7. Find the employees who earn the maximum salary in each job type. Sort in descending order of salary.
8. In which year did most people joined the company? Display the year and number of employees.
9. Display the details of those employees who earn greater than average of their department.
10. List the employees having salary between 10000 and 20000
11. Display all employees hired during 1983. those employees who earn greater than average of their department.
12. Update the salaries of all employees in marketing department & hike it by 15%.
13. Get the gross salaries of all the employees.
14. Get the names of employees and their managers name.
15. Display the name, location and department name of all the employees earning more than 1500.
16. Show all the employees in Dallas.
17. List the employees name, job, salary, grade, and department for employees in the company except clerks. Sort on employee names.
18. Find the employees who earns the minimum salary for their job. Sort in descending order of salary.
19. Find the most recently hired employees in the department order by hiredate.
20. Find out the difference between highest and lowest salaries.

```
mysql> CREATE TABLE Employee ( emp_id INT PRIMARY KEY, emp_name VARCHAR(50),  hire_date DATE,  salary DECIMAL(10, 2),  job_type VARCHAR(30),  department_id
INT,  manager_id INT);
Query OK, 0 rows affected (5.20 sec)

mysql> CREATE TABLE Department ( department_id INT PRIMARY KEY, department_name VARCHAR(50), location VARCHAR(50));
Query OK, 0 rows affected (3.62 sec)
```

```
mysql> INSERT INTO Department (department_id, department_name, location) VALUES (1, 'HR', 'New York'), (2, 'Marketing', 'Los Angeles'), (3, 'Sales', 'Chicag
o'), (4, 'Systems', 'San Francisco');
Query OK, 4 rows affected (0.34 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Employee (emp_id, emp_name, hire_date, salary, job_type, department_id, manager_id) VALUES (1, 'John Smith', '2020-01-15', 60000.00, 'Man
ager', 1, NULL), (2, 'Jane Doe', '2019-06-22', 55000.00, 'HR Specialist', 1, 1), (3, 'Mike Johnson', '2021-03-10', 62000.00, 'Marketing Manager', 2, NULL),
(4, 'Emily Brown', '2020-12-05', 58000.00, 'Marketing Specialist', 2, 3), (5, 'David Lee', '2018-02-28', 70000.00, 'Sales Manager', 3, NULL), (6, 'Sarah Whi
te', '2021-08-14', 60000.00, 'Sales Representative', 3, 5), (7, 'Kevin Davis', '2019-11-03', 75000.00, 'Systems Manager', 4, NULL), (8, 'Olivia Moore', '202
0-04-20', 68000.00, 'Systems Analyst', 4, 7), (9, 'Emma Wilson', '2022-02-17', 62000.00, 'Systems Analyst', 4, 7);
Query OK, 9 rows affected (0.26 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> SELECT emp_name, hire_date FROM Employee WHERE department_id = (SELECT department_id FROM Department WHERE department_name = 'Systems');
+--------------+------------+
| emp_name     | hire_date  |
+--------------+------------+
| Kevin Davis  | 2019-11-03 |
| Olivia Moore | 2020-04-20 |
| Emma Wilson  | 2022-02-17 |
+--------------+------------+
3 rows in set (0.06 sec)

mysql> SELECT emp_name, DATEDIFF(CURRENT_DATE(), hire_date) AS length_of_service FROM Employee;
+--------------+-------------------+
| emp_name     | length_of_service |
+--------------+-------------------+
| John Smith   |              1352 |
| Jane Doe     |              1559 |
| Mike Johnson |               932 |
| Emily Brown  |              1027 |
| David Lee    |              2038 |
| Sarah White  |               775 |
| Kevin Davis  |              1425 |
| Olivia Moore |              1256 |
| Emma Wilson  |               588 |
+--------------+-------------------+
9 rows in set (0.88 sec)
```

```
mysql> SELECT DISTINCT salary FROM Employee ORDER BY salary DESC LIMIT 1 OFFSET 1;
+----------+
| salary   |
+----------+
| 70000.00 |
+----------+
1 row in set (0.57 sec)

mysql> SELECT e.emp_name, d.department_name FROM Employee e JOIN Department d ON e.department_id = d.department_id ORDER BY d.department_name;
+--------------+-----------------+
| emp_name     | department_name |
+--------------+-----------------+
| John Smith   | HR              |
| Jane Doe     | HR              |
| Mike Johnson | Marketing       |
| Emily Brown  | Marketing       |
| David Lee    | Sales           |
| Sarah White  | Sales           |
| Kevin Davis  | Systems         |
| Olivia Moore | Systems         |
| Emma Wilson  | Systems         |
+--------------+-----------------+
9 rows in set (0.06 sec)

mysql> SELECT DISTINCT m.emp_name AS manager_name, (SELECT emp_name FROM Employee WHERE manager_id = m.emp_id ORDER BY salary LIMIT 1) AS lowest_paid_employ
ee FROM Employee m WHERE manager_id IS NOT NULL;
+--------------+----------------------+
| manager_name | lowest_paid_employee |
+--------------+----------------------+
| Jane Doe     | NULL                 |
| Emily Brown  | NULL                 |
| Sarah White  | NULL                 |
| Olivia Moore | NULL                 |
| Emma Wilson  | NULL                 |
+--------------+----------------------+
5 rows in set (1.28 sec)
```

Open Ended Experiment

Aim: To implement cursor in PL/SQL

Theory Required:

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time.

Commands:

```
-- Create the EMPLOYEE table
CREATE TABLE EMPLOYEE (
  empId NUMBER(4) PRIMARY KEY,
  name VARCHAR2(50) NOT NULL,
  dept VARCHAR2(50) NOT NULL,
  age NUMBER(3) NOT NULL,
  city VARCHAR2(50) NOT NULL,
  salary NUMBER(10) NOT NULL
);


INSERT INTO EMPLOYEE VALUES (1, 'Clark', 'Sales', 23, 'Noida', 10000);
INSERT INTO EMPLOYEE VALUES (2, 'Dave', 'Accounting', 33, 'Gurgaon', 20000);
INSERT INTO EMPLOYEE VALUES (3, 'Ava', 'Sales', 28, 'Delhi', 30000);
INSERT INTO EMPLOYEE VALUES (4, 'Emily', 'Marketing', 30, 'Bangalore', 40000);
INSERT INTO EMPLOYEE VALUES (5, 'William', 'IT', 27, 'Noida', 50000);

-- PL/SQL block to update salaries and count the employees updated
DECLARE
  total_rows NUMBER(2);
  CURSOR employee_cursor IS
    SELECT empId, salary
    FROM EMPLOYEE
    FOR UPDATE; -- Lock the rows for update
BEGIN
  total_rows := 0;

  FOR emp_record IN employee_cursor LOOP
    -- Update the salary for each employee
    UPDATE EMPLOYEE
    SET salary = emp_record.salary + 5000
    WHERE CURRENT OF employee_cursor;
```

```
    total_rows := total_rows + 1;
  END LOOP;

  IF total_rows = 0 THEN
    dbms_output.put_line('No employees updated');
  ELSE
    dbms_output.put_line(total_rows || ' employees updated');
  END IF;

  COMMIT; -- Commit the changes
END;
/
```

Output:

```
5 EMPLOYEE updated
PL/SQL procedure successfully completed
```

Conclusion: The cursor program is successfully created in PL/SQL.

Open Ended Experiment

Aim: To implement trigger in PL/SQL

Theory Required:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Commands:

```
-- Create a PL/SQL block to create a table and a trigger

DECLARE

  -- Declare variables for table and trigger names

  table_name VARCHAR2(30) := 'customers';

  trigger_name VARCHAR2(30) := 'display_salary_changes';



BEGIN

  -- Create the 'customers' table

  EXECUTE IMMEDIATE 'CREATE TABLE ' || table_name || ' (

    ID NUMBER PRIMARY KEY,

    NAME VARCHAR2(50),

    SALARY NUMBER

  )';
```

```
  -- Create the trigger
  EXECUTE IMMEDIATE '
    CREATE OR REPLACE TRIGGER ' || trigger_name || '
    BEFORE DELETE OR INSERT OR UPDATE ON ' || table_name || '
    FOR EACH ROW
    WHEN (NEW.ID > 0)
    DECLARE
      sal_diff NUMBER;
    BEGIN
      -- Calculate salary difference
      sal_diff := :NEW.salary - NVL(:OLD.salary, 0);

      -- Print the information using dbms_output
      dbms_output.put_line("Old salary: " || NVL(:OLD.salary, 0));
      dbms_output.put_line("New salary: " || :NEW.salary);
      dbms_output.put_line("Salary difference: " || sal_diff);
    END;
  ';

  -- Commit the changes
  COMMIT;
END;
/
```

Output:

Trigger Created.

Conclusion: The trigger program is successfully created in PL/SQL.