LAB CYCLE 3

Date : 02/04/2024

Experiment No: 5

Familiarisation of Stored Procedure, Function, Cursor and Triggers

- A procedur (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database.

  ```
  Syntax: CREATE PROCEDURE procedure_name(parameter_list)
  BEGIN
          statements;
  END //
  ```
- A stored function is a special kind stored program that returns a single value.

  ```
  Syntax: DELIMITER $$
  CREATE FUNCTION function_name(param1,param2,...)
  RETURNS datatype
  [NOT] DETERMINISTIC
  BEGIN
  -- statements
  END $$
  DELIMITER ;
  ```
- A cursor in database is a construct which allows you to iterate/traversal the records of a table. In MySQL you can use cursors with in a stored program such as procedures, functions etc.

  ```
  Syntax: DECLARE cursor_name CURSOR FOR select_statement;
  ```
- A trigger in MySQL is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event.

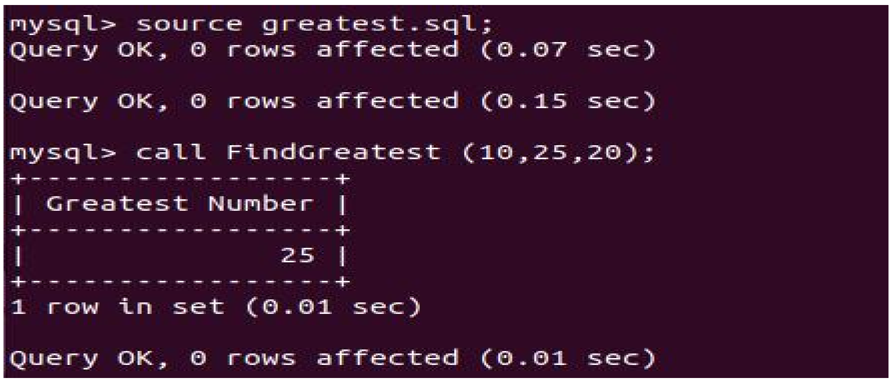
  ```
  Syntax: CREATE TRIGGER trigger_name
  {BEFORE | AFTER} {INSERT | UPDATE| DELETE }
  ON table_name FOR EACH ROW
  trigger_body;
  ```

1. Write a stored procedure to read three numbers and find the greatest among them.

SQL :

```
DELIMITER //

DROP PROCEDURE IF EXISTS FindGreatest;

CREATE PROCEDURE  FindGreatest(IN n1 INT,in n2 INT,in n3 INT)

BEGIN

        DECLARE largest INT;

        IF n1<n2 AND n3<n2 THEN

                SET largest=n2;

        ELSEIF n1<n3 AND n2<n3 THEN

                SET largest=n3;

        ELSE

                SET  largest=n1;

        END IF;

        SELECT largest AS Greatest Number;

END  //

DELIMITER ;
```

OUTPUT :



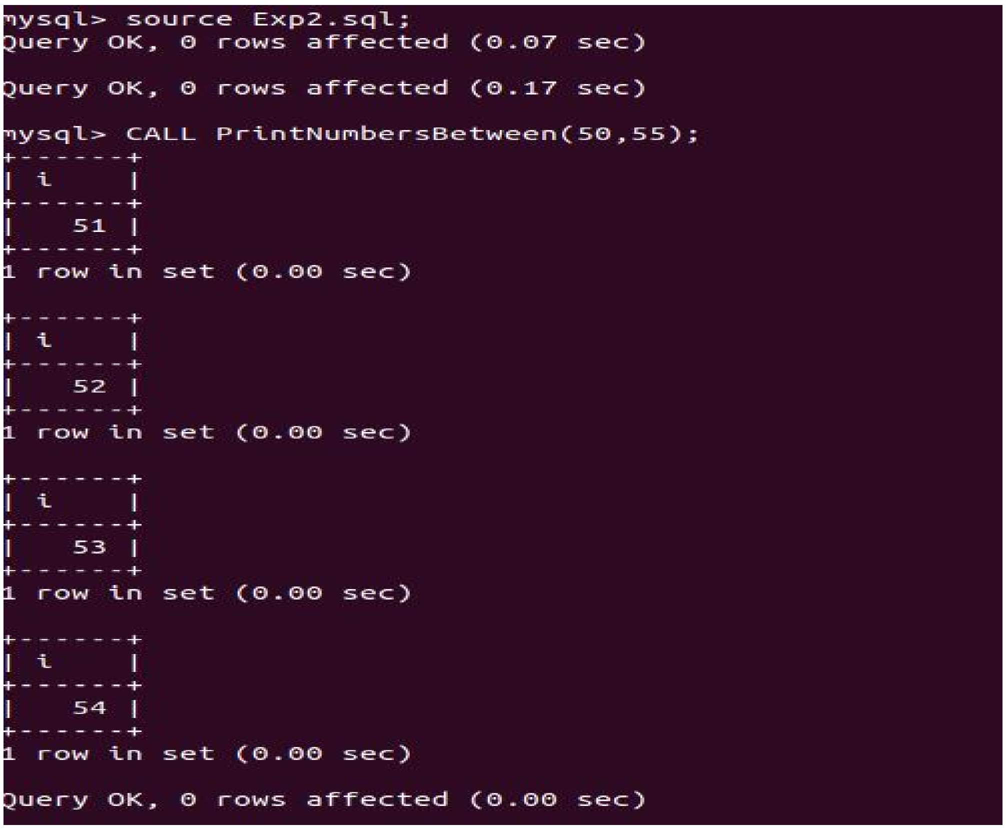
```
mysql> source greatest.sql;
Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.15 sec)

mysql> call FindGreatest (10,25,20);
+-----------------+
| Greatest Number |
+-----------------+
|              25 |
+-----------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

2.Write a stored procedure to read two numbers and print all the numbers between them.

SQL :

```
DELIMITER //

CREATE PROCEDURE PrintNumbersBetween(num1 INT, num2 INT)
BEGIN
   DECLARE i INT;

   SET i = LEAST(num1, num2) + 1;

   WHILE i < GREATEST(num1, num2) DO
      SELECT i;
      SET i = i + 1;
   END WHILE;
END //

DELIMITER ;
```

OUTPUT :



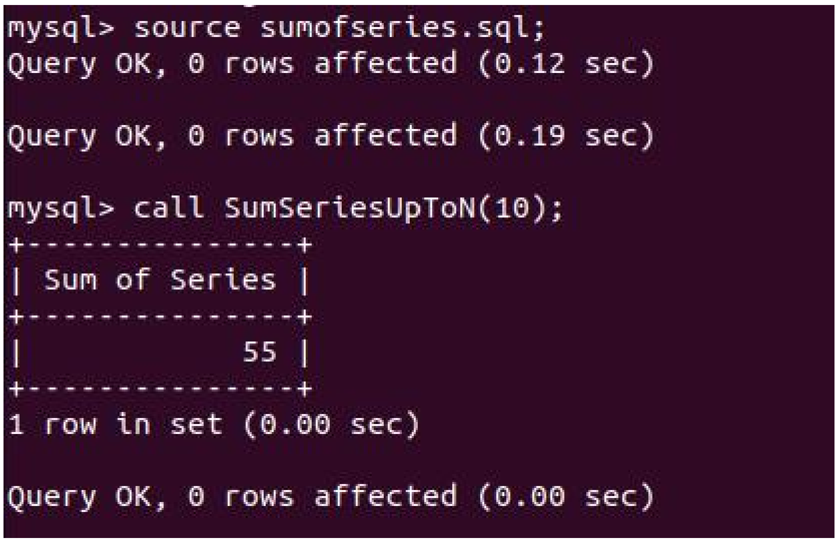
```
nysql> source Exp2.sql;
Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.17 sec)

nysql> CALL PrintNumbersBetween(50,55);
+------+
| i    |
+------+
|   51 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|   52 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|   53 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|   54 |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

3.Write a stored procedure to read N and find the sum of the series 1+2+3 +... N

SQL :

```
DROP PROCEDURE IF EXISTS SumSeriesUpToN;
DELIMITER //
CREATE PROCEDURE SumSeriesUpToN(IN n INT)
BEGIN
        DECLARE sum INT DEFAULT 0;
        DECLARE i INT DEFAULT 1;
        WHILE i<=n DO
                SET sum=sum+i;
                SET i=i+1;
        END WHILE;
        SELECT sum AS sum  of  series;
END //
DELIMITER ;
```

OUTPUT :



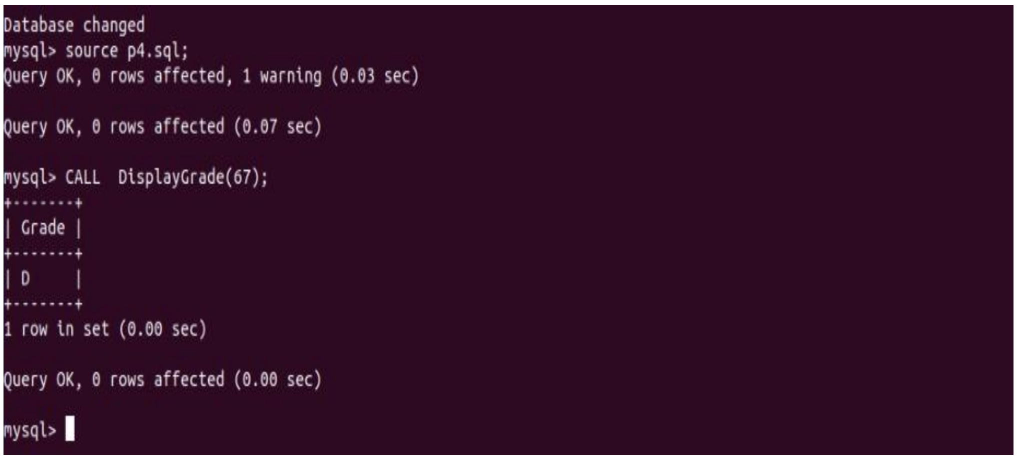
```
mysql> source sumofseries.sql;
Query OK, 0 rows affected (0.12 sec)

Query OK, 0 rows affected (0.19 sec)

mysql> call SumSeriesUpToN(10);
+---------------+
| Sum of Series |
+---------------+
|            55 |
+---------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

4.Write a stored procedure to read a mark and display the grade

SQL :

```
DELIMITER //
DROP PROCEDURE IF EXISTS DisplayGrade;
CREATE PROCEDURE DisplayGrade(IN mark INT)
BEGIN
        DECLARE grade VARCHAR(10);
        IF mark >= 90 THEN
                SET grade='A+';
        ELSEIF mark >= 80 THEN
                SET grade='A';
        ELSEIF mark >= 70 THEN
                SET grade='B';
        ELSEIF mark >= 60 THEN
                SET grade='C';
        ELSEIF mark >= 50 THEN
                SET grade='D';
        ELSE
                SET grade='F';
        END IF //
        SELECT grade as Grade;
END //
DELIMITER ;
```

OUTPUT :



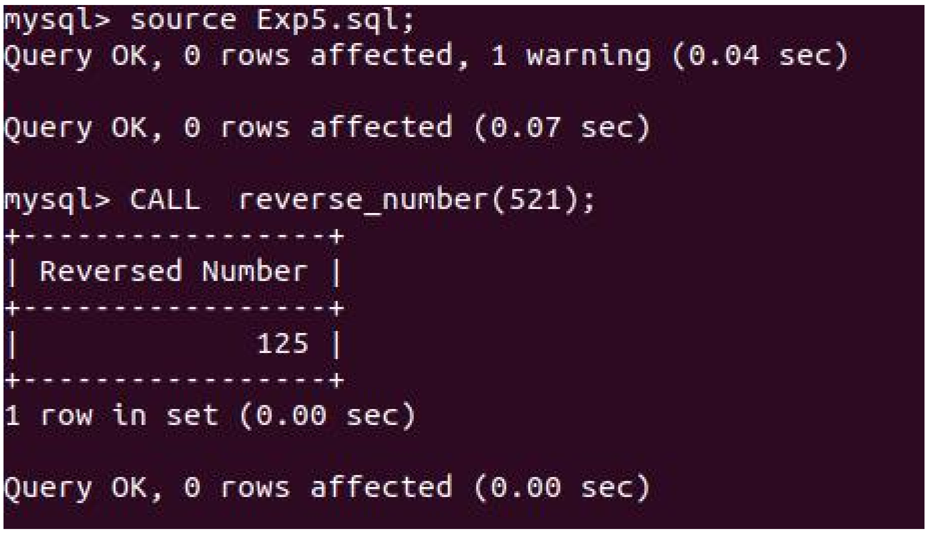
```
Database changed
mysql> source p4.sql;
Query OK, 0 rows affected, 1 warning (0.03 sec)

Query OK, 0 rows affected (0.07 sec)

mysql> CALL  DisplayGrade(67);
+-------+
| Grade |
+-------+
| D     |
+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> 
```

5.Write a stored procedure to read a number and invert the given number

SQL :

```
DELIMITER //
DROP PROCEDURE IF EXISTS reverse_number;
CREATE PROCEDURE reverse_number(IN num INT)
BEGIN
   DECLARE inverted INT DEFAULT 0;
   DECLARE remainder INT;
   WHILE num > 0 DO
      SET remainder = num % 10;
      SET inverted = inverted * 10 + remainder;
      SET num = FLOOR(num / 10);
   END WHILE;
   SELECT inverted AS Reversed Number;
END //
DELIMITER ;
```

OUTPUT :



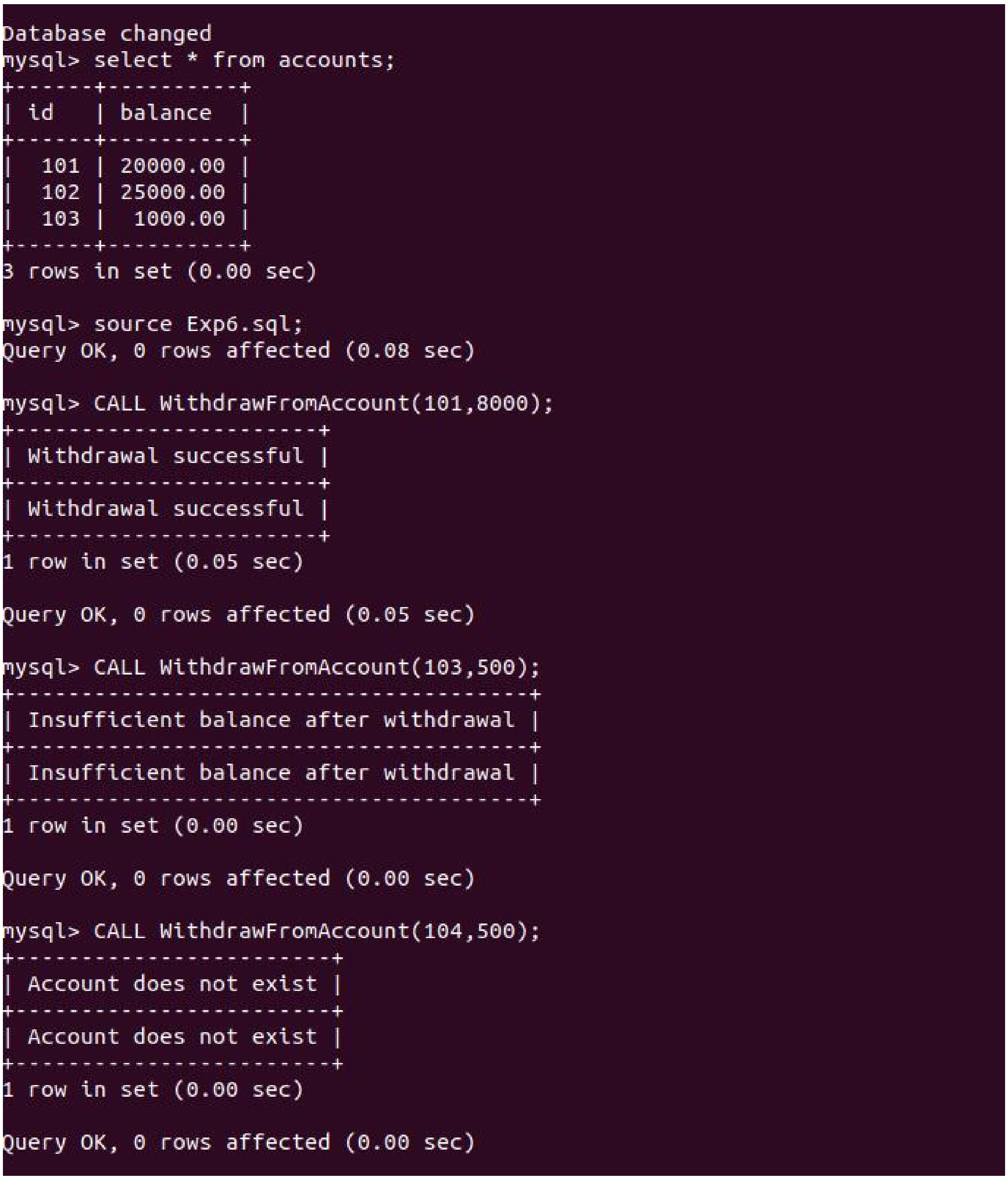
```
mysql> source Exp5.sql;
Query OK, 0 rows affected, 1 warning (0.04 sec)

Query OK, 0 rows affected (0.07 sec)

mysql> CALL  reverse_number(521);
+-----------------+
| Reversed Number |
+-----------------+
|             125 |
+-----------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

6. 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,balance int);
insert into account values(101,20000);
insert into account values(102,25000);
insert into account values(103,1000);

DELIMITER //

CREATE PROCEDURE WithdrawFromAccount(account_id INT, amount DECIMAL(10, 2))
BEGIN
   DECLARE current_balance DECIMAL(10, 2);

   SELECT balance INTO current_balance FROM accounts WHERE id = account_id;

   IF current_balance IS NULL THEN
      SELECT 'Account does not exist';
   ELSEIF current_balance - amount < 1000 THEN
      SELECT 'Insufficient balance after withdrawal';
   ELSE
      UPDATE accounts SET balance = balance - amount WHERE id = account_id;
      SELECT 'Withdrawal successful';
   END IF;
END //

DELIMITER ;
```

Output:



```
Database changed
mysql> select * from accounts;
+------+----------+
| id   | balance  |
+------+----------+
|  101 | 20000.00 |
|  102 | 25000.00 |
|  103 |  1000.00 |
+------+----------+
3 rows in set (0.00 sec)

mysql> source Exp6.sql;
Query OK, 0 rows affected (0.08 sec)

mysql> CALL WithdrawFromAccount(101,8000);
+-----------------------+
| Withdrawal successful |
+-----------------------+
| Withdrawal successful |
+-----------------------+
1 row in set (0.05 sec)

Query OK, 0 rows affected (0.05 sec)

mysql> CALL WithdrawFromAccount(103,500);
+--------------------------------------+
| Insufficient balance after withdrawal |
+--------------------------------------+
| Insufficient balance after withdrawal |
+--------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL WithdrawFromAccount(104,500);
+-----------------------+
| Account does not exist |
+-----------------------+
| Account does not exist |
+-----------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

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

SQL:

create table customer (customer_id int,name varchar(25),credits int,city varchar(20));

insert into customer values(100,"John",5500,’Kottayam’);

insert into customer values(101,"Abraham",4000,’Kottayam’);

insert into customer values(102,"George",400,’Kottayam’);

DELIMITER //

CREATE PROCEDURE Display_Customers(IN Cust_name VARCHAR(25))

BEGIN

    SELECT * FROM customer WHERE name = Cust_name;

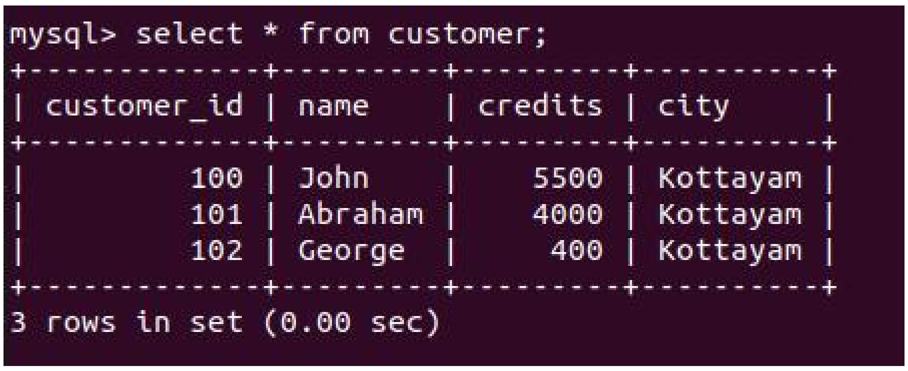
END //

DELIMITER ;

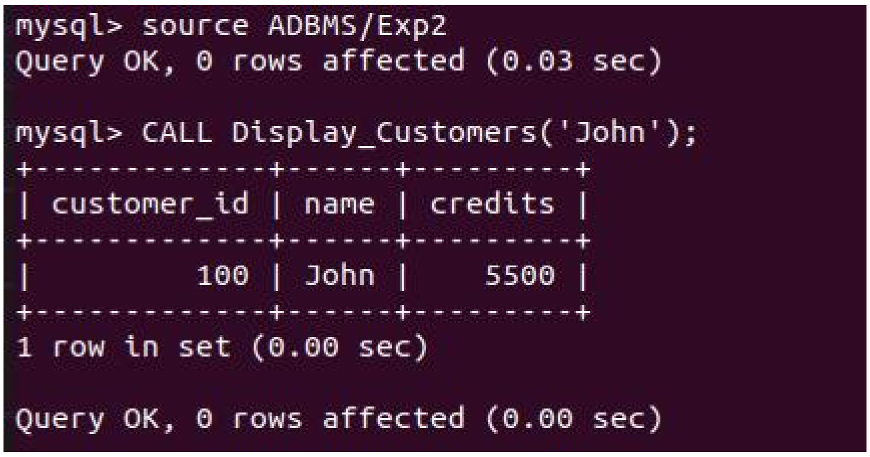
Source ADBMS/Exp2

Call Display_Customers(‘john’)

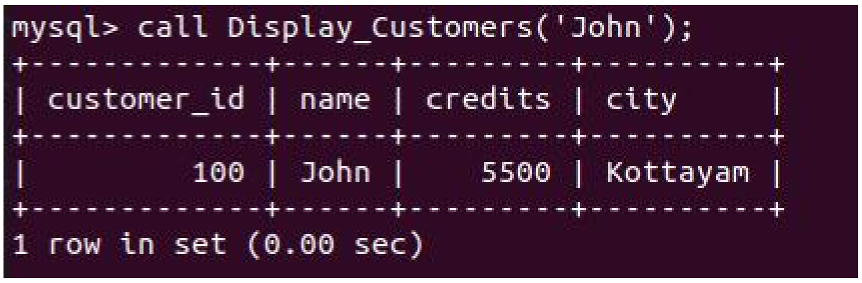
OUTPUT:



```
mysql> select * from customer;
+-------------+---------+---------+----------+
| customer_id | name    | credits | city     |
+-------------+---------+---------+----------+
|         100 | John    |    5500 | Kottayam |
|         101 | Abraham |    4000 | Kottayam |
|         102 | George  |     400 | Kottayam |
+-------------+---------+---------+----------+
3 rows in set (0.00 sec)
```



```
mysql> source ADBMS/Exp2
Query OK, 0 rows affected (0.03 sec)

mysql> CALL Display_Customers('John');
+-------------+------+---------+
| customer_id | name | credits |
+-------------+------+---------+
|         100 | John |    5500 |
+-------------+------+---------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```



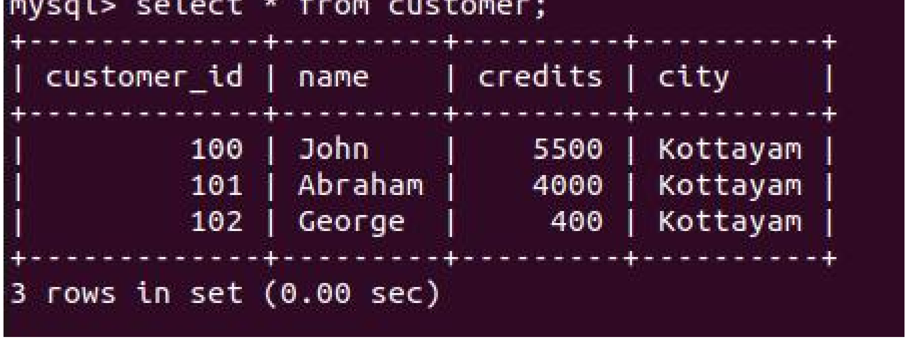
```
mysql> call Display_Customers('John');
+-------------+------+---------+----------+
| customer_id | name | credits | city     |
+-------------+------+---------+----------+
|         100 | John |    5500 | Kottayam |
+-------------+------+---------+----------+
1 row in set (0.00 sec)
```

8. 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]

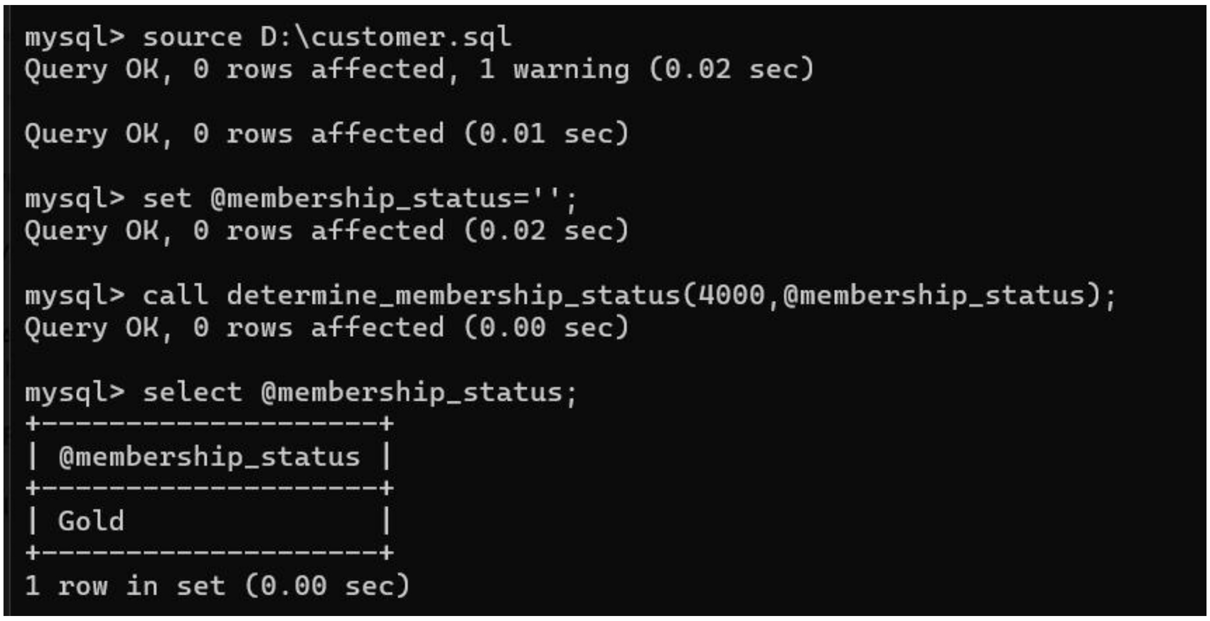
SQL:

```
DELIMITER //
CREATE procedure Membership(IN id INT,OUT membershiplevel varchar(50))
BEGIN
        DECLARE customercredits INT;
        select credits INTO customercredits from customer where customer_id = id;
        IF customercredits > 5000 THEN
              SET membershiplevel='Membership Platinum';
        ELSEIF customercredits BETWEEN 1000 AND 5000 THEN
              SET membershiplevel='Gold';
        ELSE
              SET membershiplevel='Silver';
        END IF;
END//
DELIMITER ;
Source ADBMS/Exp3
CALL Membership(101,@membership_statusl);
select @membership_status;
```

OUTPUT:



```
mysql> select * from customer;
+-------------+---------+---------+----------+
| customer_id | name    | credits | city     |
+-------------+---------+---------+----------+
|         100 | John    |    5500 | Kottayam |
|         101 | Abraham |    4000 | Kottayam |
|         102 | George  |     400 | Kottayam |
+-------------+---------+---------+----------+
3 rows in set (0.00 sec)
```



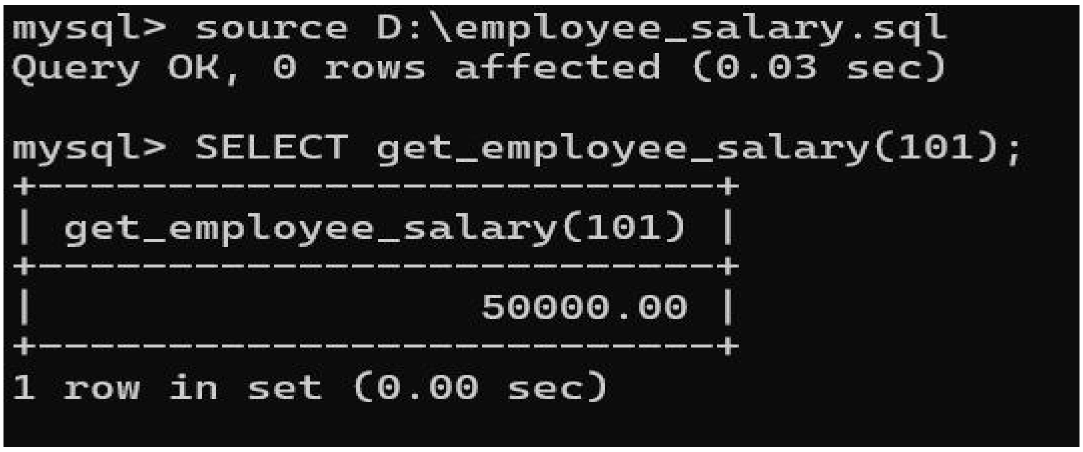
```
mysql> source D:\customer.sql
Query OK, 0 rows affected, 1 warning (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> set @membership_status='';
Query OK, 0 rows affected (0.02 sec)

mysql> call determine_membership_status(4000,@membership_status);
Query OK, 0 rows affected (0.00 sec)

mysql> select @membership_status;
+--------------------+
| @membership_status |
+--------------------+
| Gold               |
+--------------------+
1 row in set (0.00 sec)
```

9. Create a function to accept the Id of an employee and return his salary

SQL:

```
DELIMITER //
CREATE FUNCTION get_employee_salary(employee_id INT) RETURNS DECIMAL(10,2)
BEGIN
   DECLARE emp_salary DECIMAL(10,2);
   SELECT salary INTO emp_salary FROM employees WHERE id = employee_id;
   RETURN emp_salary;
END //
DELIMITER ;
SELECT get_employee_salary(101);
```

OUTPUT :



```
mysql> source D:\employee_salary.sql
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT get_employee_salary(101);
+--------------------------+
| get_employee_salary(101) |
+--------------------------+
|                 50000.00 |
+--------------------------+
1 row in set (0.00 sec)
```

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

SQL:

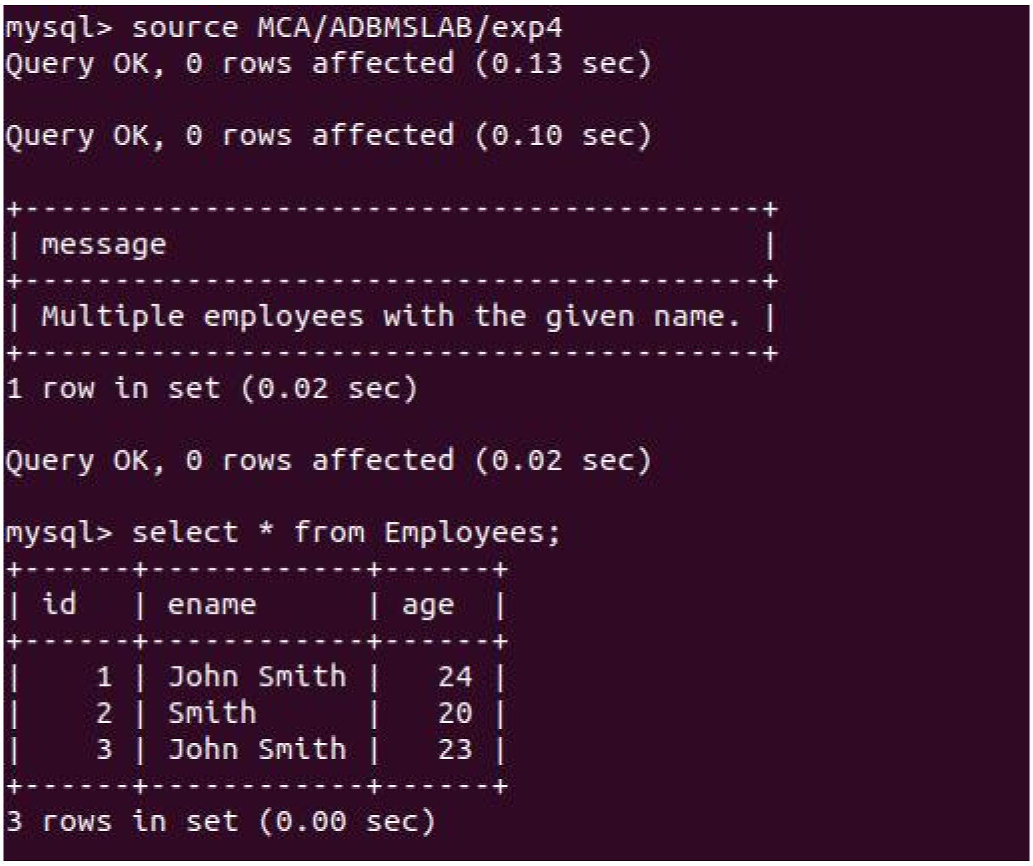
```
CREATE TABLE Employees (EmployeeID INT PRIMARY KEY,Name VARCHAR(100));
INSERT INTO Employees (EmployeeID, Name)VALUES(1, 'John Smith'),(2, 'Jane Doe'),
(3, 'Michael Johnson');

DELIMITER //
CREATE FUNCTION GetEmployeeCountByName(employeeName VARCHAR(100))
RETURNS INT DETERMINISTIC
BEGIN
        DECLARE empCount INT;
        SELECT COUNT(*) INTO empCount FROM Employees WHERE ename
            employeeName;
          RETURN empCount;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE UpdateEmployeeDetails(IN employeeName VARCHAR(100))
BEGIN
       DECLARE empCount INT;
       SET empCount = (SELECT GetEmployeeCountByName(employeeName));
        IF empCount = 1 THEN
             UPDATE Employees SET age = 24 WHERE ename = employeeName;
             SELECT 'Employee details updated successfully.' AS message;
        ELSEIF empCount = 0 THEN
             SELECT 'No Employee with the given name.' AS message;
        ELSE
             SELECT 'Multiple employees with the given name.' AS message;
          END IF;
END //
```

DELIMITER ;

Source ADBMS/Exp4

CALL UpdateEmployeeDetails('John Smith');

OUTPUT :



```
mysql> source MCA/ADBMSLAB/exp4
Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.10 sec)

+--------------------------------------+
| message                              |
+--------------------------------------+
| Multiple employees with the given name. |
+--------------------------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)

mysql> select * from Employees;
+------+------------+------+
| id   | ename      | age  |
+------+------------+------+
|    1 | John Smith |   24 |
|    2 | Smith      |   20 |
|    3 | John Smith |   23 |
+------+------------+------+
3 rows in set (0.00 sec)
```

11. 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

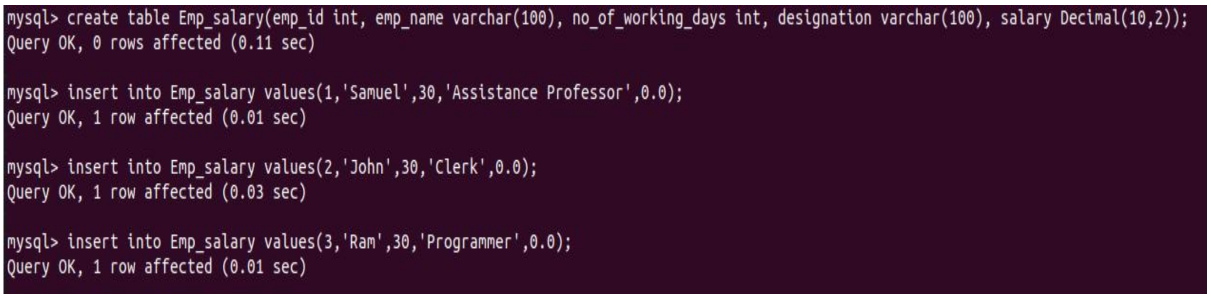
SQL:

create table Emp_salary(emp_id int, emp_name varchar(100), no_of_working_days int, designation varchar(100), salary Decimal(10,2));

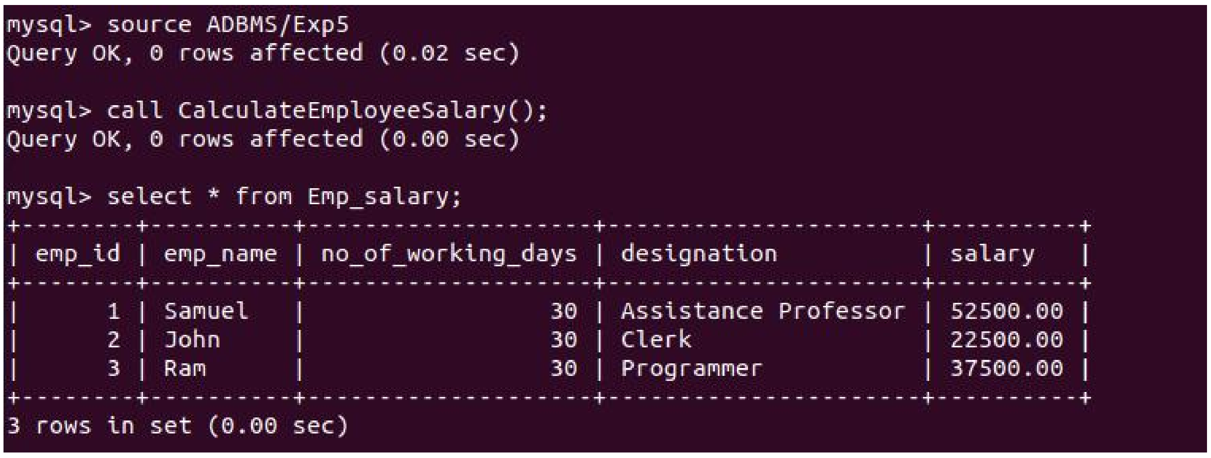
insert into Emp_salary values(1,'Samuel',30,'Assistance Professor',0.0);

```
insert into Emp_salary values(2,'John',30,'Clerk',0.0);
insert into Emp_salary values(3,'Ram',30,'Programmer',0.0);

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;
                UPDATE Emp_salary SET salary = sal WHERE emp_id = empId;
                END LOOP;
        CLOSE employeeCursor;
END //
DELIMITER ;
Source ADBMS/Exp5
call CalculateEmployeeSalary();
```

Output:



```
mysql> create table Emp_salary(emp_id int, emp_name varchar(100), no_of_working_days int, designation varchar(100), salary Decimal(10,2));
Query OK, 0 rows affected (0.11 sec)

mysql> insert into Emp_salary values(1,'Samuel',30,'Assistance Professor',0.0);
Query OK, 1 row affected (0.01 sec)

mysql> insert into Emp_salary values(2,'John',30,'Clerk',0.0);
Query OK, 1 row affected (0.03 sec)

mysql> insert into Emp_salary values(3,'Ram',30,'Programmer',0.0);
Query OK, 1 row affected (0.01 sec)
```



```
mysql> source ADBMS/Exp5
Query OK, 0 rows affected (0.02 sec)

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

mysql> select * from Emp_salary;
+--------+----------+--------------------+----------------------+----------+
| emp_id | emp_name | no_of_working_days | designation          | salary   |
+--------+----------+--------------------+----------------------+----------+
|      1 | Samuel   |                 30 | Assistance Professor | 52500.00 |
|      2 | John     |                 30 | Clerk                | 22500.00 |
|      3 | Ram      |                 30 | Programmer           | 37500.00 |
+--------+----------+--------------------+----------------------+----------+
3 rows in set (0.00 sec)
```

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

    a) For first 100 units Rs:2 per unit.

    b) 101 to 200 units Rs:2.5 per unit.

    c) 201 to 300 units Rs: 3 per unit.

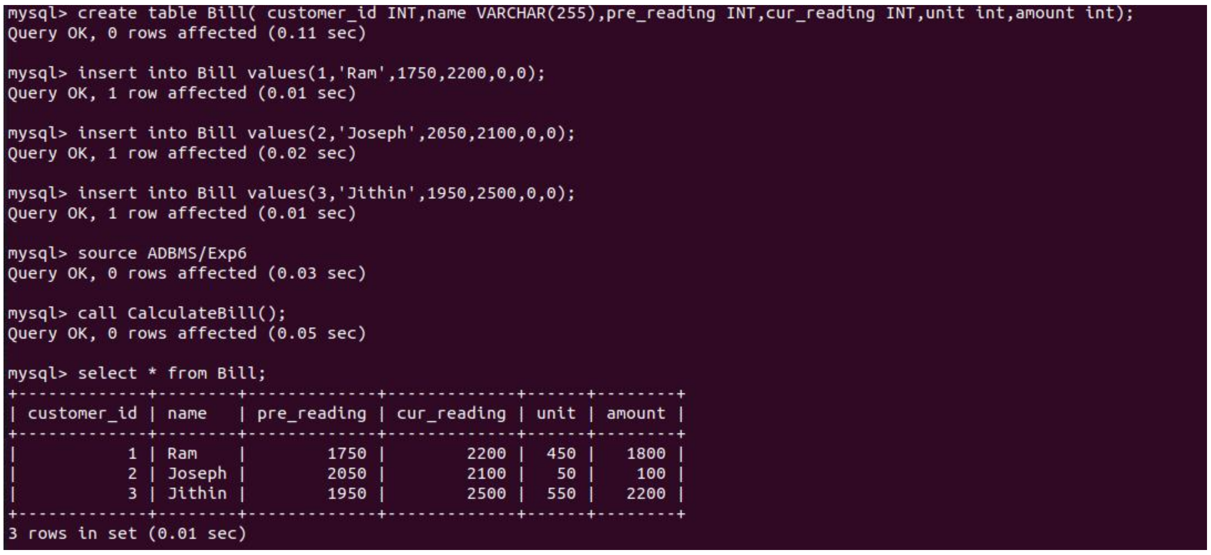
    d) Above 300 units Rs: 4 per unit

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

SQL:

```
create table Bill( customer_id INT,name VARCHAR(255),pre_reading INT,cur_reading INT,unit int,amount int);
DELIMITER //
CREATE PROCEDURE CalculateBill()
BEGIN
        DECLARE done INT DEFAULT FALSE;
        DECLARE cust_id INT;
        DECLARE cust_name VARCHAR(255);
        DECLARE pre_reading_val INT;
        DECLARE cur_reading_val INT;
        DECLARE units_consumed INT;
        DECLARE bill_amount DECIMAL(10, 2);
        DECLARE cur CURSOR FOR SELECT customer_id, name, pre_reading,
        cur_reading FROM Bill;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
        OPEN cur;
            read_loop: LOOP
            FETCH cur INTO cust_id, cust_name, pre_reading_val, cur_reading_val;
            IF done THEN
                LEAVE read_loop;
            END IF;
            SET units_consumed = cur_reading_val - pre_reading_val;
            IF units_consumed <= 100 THEN
                SET bill_amount = units_consumed * 2;
            ELSEIF units_consumed <= 200 THEN
                SET bill_amount = units_consumed * 2.5;
            ELSEIF units_consumed <= 300 THEN
                SET bill_amount = units_consumed * 3;
            ELSE
                SET bill_amount = units_consumed * 4;
            END IF;
            UPDATE Bill SET unit = units_consumed, amount = bill_amount WHERE
            customer_id = cust_id;
          END LOOP;
        CLOSE cur;
    END //
    DELIMITER ;
    Source ADBMS/Exp6
    call CalculateBill();
```

OUTPUT :



```
mysql> create table Bill( customer_id INT,name VARCHAR(255),pre_reading INT,cur_reading INT,unit int,amount int);
Query OK, 0 rows affected (0.11 sec)

mysql> insert into Bill values(1,'Ram',1750,2200,0,0);
Query OK, 1 row affected (0.01 sec)

mysql> insert into Bill values(2,'Joseph',2050,2100,0,0);
Query OK, 1 row affected (0.02 sec)

mysql> insert into Bill values(3,'Jithin',1950,2500,0,0);
Query OK, 1 row affected (0.01 sec)

mysql> source ADBMS/Exp6
Query OK, 0 rows affected (0.03 sec)

mysql> call CalculateBill();
Query OK, 0 rows affected (0.05 sec)

mysql> select * from Bill;
+-------------+--------+-------------+-------------+------+--------+
| customer_id | name   | pre_reading | cur_reading | unit | amount |
+-------------+--------+-------------+-------------+------+--------+
|           1 | Ram    |        1750 |        2200 |  450 |   1800 |
|           2 | Joseph |        2050 |        2100 |   50 |    100 |
|           3 | Jithin |        1950 |        2500 |  550 |   2200 |
+-------------+--------+-------------+-------------+------+--------+
3 rows in set (0.01 sec)
```

13. 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_history (employee_id int,employee_name varchar(50),employee_department varchar(50),date_of_deletion date);

Create table employee (employee_id int, employee_name varchar(50), employee_department varchar(50));

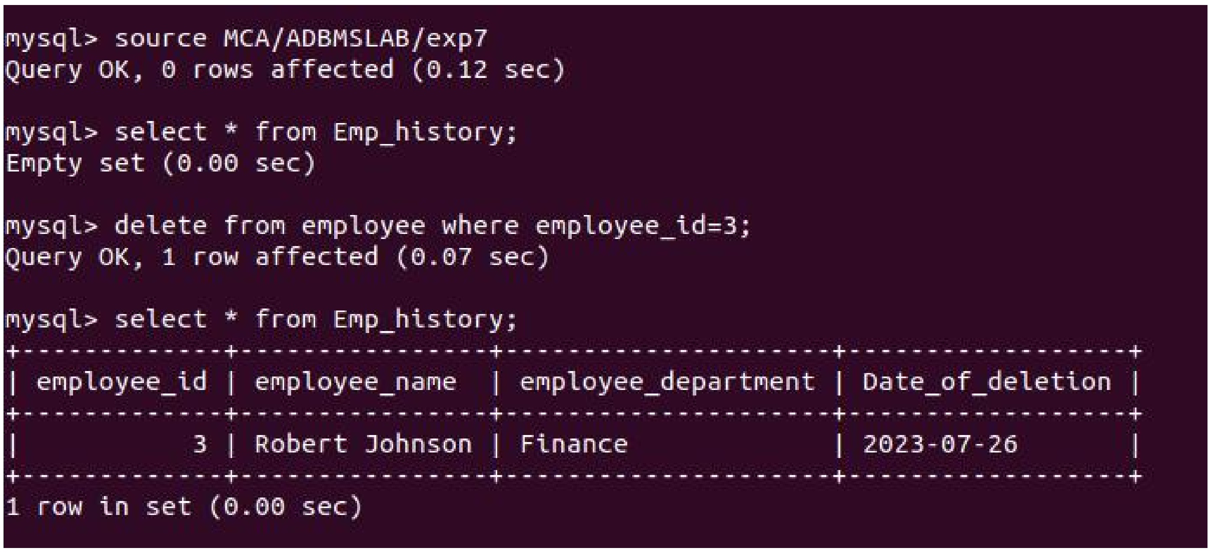
Insert into employee (employee_id, employee_name, employee_department) values(1, 'john doe', 'sales'),(2, 'jane smith', 'marketing'),(3, 'robert johnson', 'finance');

```
DELIMITER //
CREATE TRIGGER trg_employee_delete
AFTER DELETE ON employee
FOR EACH ROW
BEGIN
        INSERT INTO Emp_history (employee_id, employee_name, employee_department,
        Date_of_deletion)VALUES (OLD.employee_id, OLD.employee_name,
        OLD.employee_department, CURDATE());
END //
DELIMITER ;
```

Source ADBMS/Exp7

Delete from employee where employee_id=1;

OUTPUT:



```
mysql> source MCA/ADBMSLAB/exp7
Query OK, 0 rows affected (0.12 sec)

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

mysql> delete from employee where employee_id=3;
Query OK, 1 row affected (0.07 sec)

mysql> select * from Emp_history;
+-------------+---------------+---------------------+-----------------+
| employee_id | employee_name  | employee_department | Date_of_deletion |
+-------------+---------------+---------------------+-----------------+
|           3 | Robert Johnson | Finance             | 2023-07-26       |
+-------------+---------------+---------------------+-----------------+
1 row in set (0.00 sec)
```

14. 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]

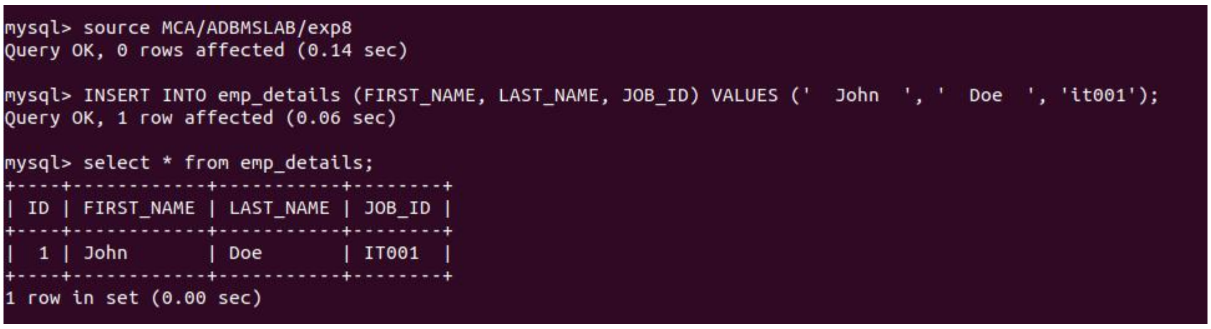
SQL: Create table emp_details (id int primary key auto_increment,first_name varchar(50),last_name varchar(50),job_id varchar(50));

```
DELIMITER $$
CREATE TRIGGER before_insert_emp_details
BEFORE INSERT ON emp_details
FOR EACH ROW
BEGIN
   SET NEW.FIRST_NAME = TRIM(NEW.FIRST_NAME);
   SET NEW.LAST_NAME = TRIM(NEW.LAST_NAME);
   SET NEW.JOB_ID = UPPER(NEW.JOB_ID);
END$$
DELIMITER ;
```

Source ADBMS/Exp8

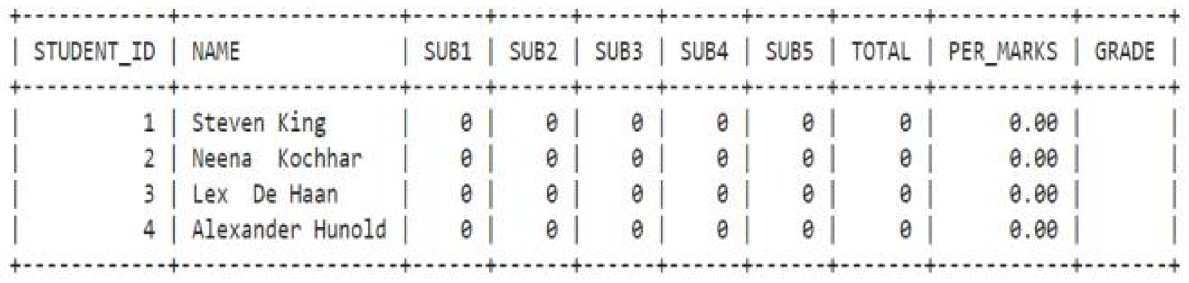
Insert into emp_details (first_name, last_name, job_id) values ('   john   ', '   doe   ', 'it001');

OUTPUT:



```
mysql> source MCA/ADBMSLAB/exp8
Query OK, 0 rows affected (0.14 sec)

mysql> INSERT INTO emp_details (FIRST_NAME, LAST_NAME, JOB_ID) VALUES ('  John  ', '  Doe  ', 'it001');
Query OK, 1 row affected (0.06 sec)

mysql> select * from emp_details;
+----+------------+-----------+--------+
| ID | FIRST_NAME | LAST_NAME | JOB_ID |
+----+------------+-----------+--------+
|  1 | John       | Doe       | IT001  |
+----+------------+-----------+--------+
1 row in set (0.00 sec)
```

15. 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 |       |
+------------+------------------+------+------+------+------+------+-------+-----------+-------+
```

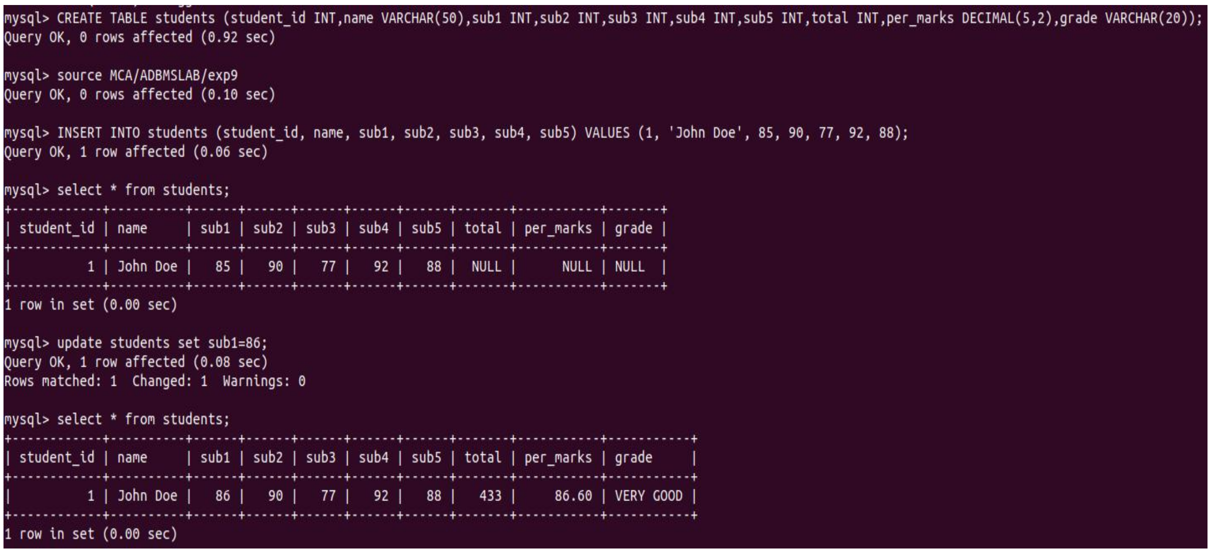
For this sample calculation, the following conditions are assumed:
Total Marks (will be stored in TOTAL column) : TOTAL = SUB1 + SUB2 + SUB3 + SUB4 +
SUB5.
Percentage of Marks (will be stored in PER_MARKS column): PER_MARKS = (TOTAL)/5
Grade (will be stored in GRADE column):
- If PER_MARKS>=90 -> 'EXCELLENT'
- 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,name VARCHAR(50),sub1 INT,sub2 INT,sub3 INT,sub4 INT,sub5 INT,total INT,per_marks DECIMAL(5,2),grade VARCHAR(20));

DELIMITER //
CREATE TRIGGER calculate_marks
BEFORE UPDATE ON students
```

```
FOR EACH ROW
BEGIN
        SET NEW.total = NEW.sub1 + NEW.sub2 + NEW.sub3 + NEW.sub4 + NEW.sub5;
        SET NEW.per_marks = NEW.total / 5;
        IF NEW.per_marks >= 90 THEN
                SET NEW.grade = 'EXCELLENT';
         ELSEIF NEW.per_marks >= 75 AND NEW.per_marks < 90 THEN
                SET NEW.grade = 'VERY GOOD';
         ELSEIF NEW.per_marks >= 60 AND NEW.per_marks < 75 THEN
                SET NEW.grade = 'GOOD';
         ELSEIF NEW.per_marks >= 40 AND NEW.per_marks < 60 THEN
                SET NEW.grade = 'AVERAGE';
         ELSE
                SET NEW.grade = 'NOT PROMOTED';
         END IF;
END //
DELIMITER ;
Source ADBMS/Exp9
INSERT INTO students (student_id, name, sub1, sub2, sub3, sub4, sub5) VALUES (1, 'John
Doe', 85, 90, 77, 92, 88);
update students set sub1=86;
```

OUTPUT:



```
mysql> CREATE TABLE students (student_id INT,name VARCHAR(50),sub1 INT,sub2 INT,sub3 INT,sub4 INT,sub5 INT,total INT,per_marks DECIMAL(5,2),grade VARCHAR(20));
Query OK, 0 rows affected (0.92 sec)

mysql> source MCA/ADBMSLAB/exp9
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO students (student_id, name, sub1, sub2, sub3, sub4, sub5) VALUES (1, 'John Doe', 85, 90, 77, 92, 88);
Query OK, 1 row affected (0.06 sec)

mysql> select * from students;
+------------+----------+------+------+------+------+------+-------+-----------+-------+
| student_id | name     | sub1 | sub2 | sub3 | sub4 | sub5 | total | per_marks | grade |
+------------+----------+------+------+------+------+------+-------+-----------+-------+
|          1 | John Doe |   85 |   90 |   77 |   92 |   88 |  NULL |      NULL | NULL  |
+------------+----------+------+------+------+------+------+-------+-----------+-------+
1 row in set (0.00 sec)

mysql> update students set sub1=86;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from students;
+------------+----------+------+------+------+------+------+-------+-----------+-----------+
| student_id | name     | sub1 | sub2 | sub3 | sub4 | sub5 | total | per_marks | grade     |
+------------+----------+------+------+------+------+------+-------+-----------+-----------+
|          1 | John Doe |   86 |   90 |   77 |   92 |   88 |   433 |     86.60 | VERY GOOD |
+------------+----------+------+------+------+------+------+-------+-----------+-----------+
1 row in set (0.00 sec)
```