Experiment No.8
Procedures and Functions
Date of Performance:21/3/25
Date of Submission:27/3/25

Aim: To implement Functions and procedure.

Objective: The function must return a value but in Stored Procedure it is optional. Even a procedure can return zero or n values. Functions can have only input parameters for it whereas Procedures can have input or output parameters

Theory:

Procedure:

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
      BEGIN
      < procedure_body >
      END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.

The AS keyword is used instead of the IS keyword for creating a standalone procedure

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
```

{IS | AS}
BEGIN
  < function_body >
END [function_name];

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Implementation:

```
mysql> CREATE DATABASE SE2;
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> USE SE2;
Database changed
mysql>
mysql> CREATE TABLE Employees_SE2 (
    ->     EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    ->     EmployeeName VARCHAR(100) NOT NULL,
    ->     DepartmentID INT NOT NULL,
    ->     Salary DECIMAL(10,2) NOT NULL
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO Employees_SE2 (EmployeeName, DepartmentID, Salary) VALUES
    -> ('Alice', 1, 5000.00),
    -> ('Bob', 2, 6000.00),
    -> ('Charlie', 1, 5500.00),
    -> ('David', 3, 7000.00),
    -> ('Emma', 2, 6200.00);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetEmployeesByDept(IN dept_id INT)
    -> BEGIN
    ->     SELECT * FROM Employees_SE2 WHERE DepartmentID = dept_id;
    -> END
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetEmployeesByDept(2);
+------------+--------------+--------------+---------+
| EmployeeID | EmployeeName | DepartmentID | Salary  |
+------------+--------------+--------------+---------+
|          2 | Bob          |            2 | 6000.00 |
|          5 | Emma         |            2 | 6200.00 |
+------------+--------------+--------------+---------+
2 rows in set (0.00 sec)

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

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetTotalSalaryByDept(dept_id INT) RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE total_salary DECIMAL(10,2);
    ->
    ->     SELECT SUM(Salary) INTO total_salary
    ->     FROM Employees_SE2
    ->     WHERE DepartmentID = dept_id;
    ->        RETURN total_salary;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT GetTotalSalaryByDept(2) AS TotalSalary;
+-------------+
| TotalSalary |
+-------------+
|    12200.00 |
+-------------+
1 row in set (0.00 sec)
```

Conclusion: The experiment successfully implemented SQL functions and procedures, showcasing their distinction and versatility in database operations. Functions demonstrated the ability to return values and handle input parameters, while procedures highlighted flexibility by utilizing input, output, and mixed parameters, enabling efficient execution of complex tasks and data manipulation in relational databases.