Assignment No. 9

Aim: Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use.

Objective:

- To study and implement PL/SQLprocedure.
- To study and implement PL/SQLfunction.

Theory :

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

A subprogram can be created −

- At the schemalevel
- Inside apackage
- Inside a PL/SQLblock

At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return avalue.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

S.No	Parts & Description

1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local tothe

	subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

Creating a 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 theprocedure.
- [OR REPLACE] option allows the modification of an existingprocedure.
- 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 theprocedure.
- *procedure-body* contains the executablepart.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
```

```
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is −

```
DROP PROCEDURE procedure-name;
```

You can drop the greetings procedure by using the following statement −

```
DROP PROCEDURE greetings;
```

Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms −

S.No	Parameter Mode & Description
1	**IN** An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed byreference**.
2	**OUT** An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**.
	IN OUT

	3	An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read. The actual parameter corresponding to an IN OUT formal parameter must be a	

	variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed byvalue.**

Functions:

A function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functionstoo.

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 thefunction.
- [OR REPLACE] option allows the modification of an existingfunction.
- 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 theprocedure.
- The function must contain a **return**statement.
- The *RETURN* clause specifies the data type you are going to return from thefunction.
- *function-body* contains the executablepart.
- The AS keyword is used instead of the IS keyword for creating a standalonefunction.

Conclusion:-

We have studied and executed PL/SQL stored procedures and functions.

OUTPUT -

⍰ Create Simple Procedures

```
mysql> CREATE PROCEDURE PRINT()
-> BEGIN
-> SELECT 'HELLO' AS OUTPUT;
-> END;
-> $
Query OK, 0 rows affected (0.06 sec)

mysql> CALL PRINT();
-> $
+--------+
| OUTPUT |
+--------+
| HELLO |
+--------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE STUDENT(ROLL_NO INT PRIMARY KEY, NAME VARCHAR(20), CLASS
VARCHAR(20), MARKS INT);
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO STUDENT VALUES (1, 'Shreyas Patil',null,900),(2,'Piyush
Rajput',null,1001),(3,'Jagruti Patil',null,600),(4,'Manisha
Yemul',null,800),(5,'Aditya Pandit',null,200),(6,'Harsh',null,50);
Query OK, 6 rows affected (0.07 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM STUDENT;
+---------+---------------+-------+-------+
| ROLL_NO | NAME | CLASS | MARKS |
+---------+---------------+-------+-------+
| 1 | Shreyas Patil | NULL | 900 |
| 2 | Piyush Rajput | NULL | 1001 |
| 3 | Jagruti Patil | NULL | 600 |
| 4 | Manisha Yemul | NULL | 800 |
| 5 | Aditya Pandit | NULL | 200 |
| 6 | Harsh | NULL | 50 |
+---------+---------------+-------+-------+
6 rows in set (0.00 sec)
```

⍰ Set Delimiter:

```
mysql> DELIMITER $
```

⍰ Create Procedure to Show Data

```
mysql> CREATE PROCEDURE GETDATA(IN ID INT)
-> BEGIN
-> SELECT * FROM STUDENT WHERE ROLL_NO=ID;
-> END;
-> $
Query OK, 0 rows affected (0.08 sec)
```

⍰ Create procedure to get the value

```
mysql> CREATE PROCEDURE GETSAL(IN CID INT, OUT SALARY FLOAT)
-> BEGIN
-> SELECT CUST_SALARY INTO SALARY FROM CUSTOMER WHERE CUST_ID = CID;
-> END;
-> $
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM CUSTOMER;
-> $
+---------+---------------+-------------+------------+---------------+---------+
| CUST_ID | CUST_NAME | CUST_SALARY | CONTACT | CITY | COUNTRY |
+---------+---------------+-------------+------------+---------------+---------+
| 1 | PIYUSH RAJPUT | 50000 | 8877665544 | PUNE | INDIA |
| 2 | SHREYAS PATIL | 500000 | 8873475544 | JALGAON | INDIA |
| 3 | HARRY POTTER | 5000000 | 9073475544 | SAN FRANSISCO | US |
+---------+---------------+-------------+------------+---------------+---------+
3 rows in set (0.02 sec)

mysql> CALL GETSAL(1, @SAL)$
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @SAL$
+-------+
| @SAL |
+-------+
| 50000 |
+-------+
1 row in set (0.00 sec)
```

⍰ Create procedure to Update the data.

```
mysql> CREATE PROCEDURE UPDATECLASS()
-> BEGIN
-> UPDATE STUDENT SET CLASS = 'DISTINCTION' WHERE MARKS<=1500 AND MARKS>=990;
-> UPDATE STUDENT SET CLASS = 'FIRST CLASS' WHERE MARKS<=989 AND MARKS>=900;
-> UPDATE STUDENT SET CLASS = 'SECOND CLASS' WHERE MARKS<=899 AND MARKS>=825;
```

```
-> UPDATE STUDENT SET CLASS = 'POOR' WHERE MARKS<=824;
-> END;
-> $
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE PROCEDURE ASSTATUS()
-> BEGIN
-> UPDATE CUSTOMER SET STATUS = 'Platinum' WHERE SALARY > 75000;
-> UPDATE CUSTOMER SET STATUS = 'Gold' WHERE SALARY > 45000 AND SALARY < 75000;
-> UPDATE CUSTOMER SET STATUS = 'Silver' WHERE SALARY < 45000;
-> END;
-> $
Query OK, 0 rows affected (0.00 sec)

mysql> CALL ASSTATUS()$
Query OK, 1 row affected (0.19 sec)

mysql> SELECT * FROM CUSTOMER$
+------+---------+--------+----------+
| ID | NAME | SALARY | STATUS |
+------+---------+--------+----------+
| 1 | Shreyas | 50000 | Gold |
| 2 | Piyush | 50000 | Gold |
| 3 | Aditya | 90000 | Platinum |
| 4 | Harsh | 10000 | Silver |
+------+---------+--------+----------+
4 rows in set (0.00 sec)
```

Call Procedures

```
mysql> CALL UPDATECLASS()$
Query OK, 4 rows affected (0.14 sec)

mysql> SELECT * FROM STUDENT$

+---------+--------------+-------------+-------+
| ROLL_NO | NAME | CLASS | MARKS |
+---------+--------------+-------------+-------+
| 1 | Shreyas Patil | FIRST CLASS | 900 |
| 2 | Piyush Rajput | DISTINCTION | 1001 |
| 3 | Jagruti Patil | POOR | 600 |
| 4 | Manisha Yemul | POOR | 800 |
| 5 | Aditya Pandit | POOR | 200 |
| 6 | Harsh | POOR | 50 |
+---------+--------------+-------------+-------+
6 rows in set (0.00 sec)

mysql> CALL GETDATA(1)$
+---------+--------------+-------------+-------+
| ROLL_NO | NAME | CLASS | MARKS |
+---------+--------------+-------------+-------+
```

```
| 1 | Shreyas Patil | FIRST CLASS | 900 |
+---------+---------------+-------------+-------+
1 row in set (0.00 sec)
```

Create and call functions:

```
mysql> CREATE FUNCTION calcProfit(cost FLOAT, price FLOAT) RETURNS DECIMAL(9,2)
-> BEGIN
-> DECLARE profit DECIMAL(9,2);
-> SET profit = price-cost;
-> RETURN profit;
-> END$$
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *, calcProfit(prod_cost,prod_price) AS profit FROM products$
+---------+--------------+-----------+------------+--------+
| prod_id | prod_name | prod_cost | prod_price | profit |
+---------+--------------+-----------+------------+--------+
| 1 | Basic Widget | 5.95 | 8.35 | 2.40 |
| 2 | Micro Widget | 0.95 | 1.35 | 0.40 |
| 3 | Mega Widget | 99.95 | 140 | 40.05 |
+---------+--------------+-----------+------------+--------+
3 rows in set (0.02 sec)
```

Show List of Procedures of Specific Database:

```
MySQL> SHOW PROCEDURE STATUS WHERE DB = 'SHREYAS'$
+---------+-------------+-----------+----------------+---------------------+------------------
---+---------------+---------+----------------------+----------------------+------------------
--+
| Db | Name | Type | Definer | Modified | Created
| Security_type | Comment | character_set_client | collation_connection | Database Collation |
+---------+-------------+-----------+----------------+---------------------+------------------
---+---------------+---------+----------------------+----------------------+------------------
--+
| college | ASSTATUS | PROCEDURE | root@localhost | 2019-08-26 11:44:34 | 2019-08-26
11:44:34 | DEFINER | | latin1 | latin1_swedish_ci |
latin1_swedish_ci |
| college | GETDATA | PROCEDURE | root@localhost | 2019-08-26 11:44:18 | 2019-08-26
11:44:18 | DEFINER | | latin1 | latin1_swedish_ci |
latin1_swedish_ci |


| college | GETSAL | PROCEDURE | root@localhost | 2019-08-26 11:44:04 | 2019-08-26
11:44:04 | DEFINER | | latin1 | latin1_swedish_ci |
```

```
latin1_swedish_ci |
| college | UPDATECLASS | PROCEDURE | root@localhost | 2019-08-26 11:44:40 | 2019-08-26
11:44:40 | DEFINER | | latin1 | latin1_swedish_ci |
latin1_swedish_ci |
| college | UPDATESAL | PROCEDURE | root@localhost | 2019-08-26 11:43:37 | 2019-08-26
11:43:37 | DEFINER | | latin1 | latin1_swedish_ci |
latin1_swedish_ci |
+---------+-------------+-----------+----------------+---------------------+------------------
---+---------------+---------+----------------------+----------------------+------------------
--+
5 rows in set (0.152 sec)
```