# Lab 13: Stored Functions and Triggers

Objective(s):

To learn the Stored Functions and Triggers

## Functions

Subprograms are named PL/SQL blocks that can accept parameters and be invoked from a calling environment.

**Types of Subprograms**

PL/SQL has two types of subprograms, procedures and functions.

**Function Specification**

- The header is relevant for named blocks only and determines the way that the program unit is called or invoked.
- The header determines:
- The PL/SQL subprogram type, that is, either a procedure or a function
- The name of the subprogram
- The parameter list, if one exists
- The RETURN clause, which applies only to functions
- The IS or AS keyword is mandatory.

**Function Body**

- The declaration section of the block between IS|AS and BEGIN.
- The keyword DECLARE that is used to indicate the start of the declaration section in anonymous blocks is not used here.
- The executable section between the BEGIN and END keywords is mandatory, enclosing the body of actions to be performed.
- There must be at least one statement existing in the executable section. There should be at least one NULL; statement, which is considered an executable statement.
- The exception section between EXCEPTION and END is optional.

**Example:**

Consider the following example where the function QUERY_CALL_SQL queries the SALARY column of the EMPLOYEE table:

```
CREATE OR REPLACE FUNCTION query_call_sql(a NUMBER) RETURN NUMBER
IS
        s NUMBER;
BEGIN
        SELECT salary INTO s FROM employees
        WHERE employee_id = 170;
        RETURN (s + a);
END;
```

The above function, when invoked from the following UPDATE statement, returns the error message.

```
UPDATE employees
SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

# Triggers

***DATABASE TRIGGERS*** execute implicitly when:

- A data event such as ***DML*** on a table (an INSERT, UPDATE, or DELETE triggering statement),
- An ***INSTEAD OF trigger*** on a view,
- ***Data Definition Language (DDL)*** statements such as CREATE and ALTER are issued, no matter which user is connected or which application is used.

**Note:**

- Database triggers can be defined on ***tables*** and on ***views.***
- If a DML operation is issued on a view, the INSTEAD OF trigger defines what actions take place.
- If these actions include DML operations on tables, then any triggers on the base tables are fired.

**Example:**

Create a ROW LEVEL TRIGGER, which first insert a DEPTNO which is not present in the DEPT table.

SQL> *CREATE OR REPLACE TRIGGER emp_trigger*

*BEFORE INSERT ON emp*

*FOR EACH ROW*

*BEGIN*

*INSERT INTO dept(deptno) VALUES(:new.deptno);*

*END;*

# Exercise

**FUNCTIONS:**

1. Create a stored function MANAGER without input parameters. It must return the total salary of all the managers in the EMP table to PL/SQL anonymous block and must be displayed in the same anonymous block.
2. Create a stored function MANAGER2 with parameters. It must take empno as an input and must return its manager name.
   Write a SELECT statement to display all employees' names and their manager names. Manager names must be displayed using MANAGER2 stored function.
3. Create a stored function MANAGER3 with parameters. It must take MANAGER NAME as an input and must display all employees' names working as CLERK under that manager.
   MANAGER3 must be called from a PL/SQL anonymous block.

**TRIGGERS:**

1. Create a new table which is the replica of EMP table without records. Now write a trigger on EMP table so that whenever any DML operation is performed on EMP, original record of the EMP table must be maintained in the new table before getting changed in it.

2. Create a trigger on EMP table which stops any person from INSERT or UPDATE of the record if the new job of the affected record is CLERK and new salary is less than 5000.