Subprogram

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.

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 a value.
- Procedures − These subprograms do not return a value directly; mainly used to perform an action.

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;
```

Example

```
DECLARE
  a number;
  b number;
  c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
  IF x < y THEN
    z:= x;
  ELSE
    z:= y;
  END IF;
END;
```

```
BEGIN
  a:= 23;
  b:= 45;
  findMin(a, b, c);
  dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
/
```

Creating Function

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Example

```
Select * from customers;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM customers;

  RETURN total;
END;
/
```

Calling a Function

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

PL/SQL - Cursors

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

There are two types of cursors −

- Implicit cursors
- Explicit cursors

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

1. **Electricity bill calculation using PLSQL**

Objective

Create an electricity billing system, rent rs 20/-
Slab 1 : 1-40 units-0
Slab2: 40-80 units -40
Slab3: >80 -1.40+excess of 80

Program

```
SQL> create table electricity(cons_id varchar(4) primary key, c_name varchar(20), rent
number(2) check (rent=20), unit number(6));
Table created
SQL> insert into electricity values (‘E001’,’deepika’,20,35);
1 row created
SQL> @
1 row created
SQL> insert into electricity values (‘E003’,’arun’,20,80);
1 row created
SQL> insert into electricity values (‘E004’,’rahul’,20,90);
1 row created

SQL> alter table electricity add (total number (6,2));
Table altered

SQL>DECLARE
   v_total electricity.total%TYPE;
   CURSOR c IS SELECT * FROM electricity;
BEGIN
   FOR i IN c LOOP
      BEGIN -- Start inner block for exception handling (if needed)
         IF i.unit <= 40 THEN
            v_total := i.rent;
         ELSIF i.unit <= 80 THEN
            v_total := i.rent + (i.unit - 40) * 0.40;
         ELSE
            v_total := i.rent + (40 * 0.40) + (i.unit - 80) * 1.40;
         END IF;
```

```
            UPDATE electricity
            SET total = v_total
            WHERE cons_id = i.cons_id;
        END; -- End inner block
    END LOOP;

    COMMIT; -- Ensures all updates are saved permanently
END;
/
```

```
PL/SQL procedure successfully completed
```

```
SQL> select* from electricity;
Cons   c_name    rent   unit   total
E001   deepika     20      35      20
E002    varna       20      61            28.4
E003    arun       20             80        36
E004     rahul     20        90       50
```

2. **Student Result calculation**

Objective

An examination has been conducted to a class of 5 students and four scores of each student have been provided in the data along with register number and name. Write a PL/SQL block to do the following

Assign a letter grade to each student based on the average score;

Average Score	Grade
90-100	a
75-89	b
60-74	c
50-59	d
0-49	e

Program

```
SQL> create table studres(regno number(4) primary key, name varchar(20), paper1 number(2),
paper2 number(2), paper3 number(2), paper4 number(2));
Table created
SQL> insert into studres values(1001,’mini’, 23, 49, 44, 46);
1 row created
SQL> insert into studres values(1002,’safeer’, 40, 30, 20, 10);
1 row created
SQL> insert into studres values(1003,’baby’, 49, 39, 46, 45);
1 row created
 SQL> insert into studres values(1004,’danish’, 40, 10, 20, 22);
1 row created

SQL> insert into studres values(1005,’swetha’, 20, 18, 20, 15);
1 row created

SQL> alter table studres add( averg number(5,2), grade varchar(2));
Table altered

SQL> DECLARE
   CURSOR c IS SELECT * FROM studres;
   v_avg   number;
   v_tot   number;
   v_grade varchar(3);
BEGIN
   FOR i IN c LOOP
      v_tot := i.paper1 + i.paper2 + i.paper3 + i.paper4;
      v_avg := v_tot / 4; -- Assuming average is total divided by 4 subjects

      IF v_avg >= 85 THEN
         v_grade := 'A';
      ELSIF v_avg >= 65 THEN
         v_grade := 'B';
      ELSIF v_avg >= 40 THEN
         v_grade := 'C';
      ELSIF v_avg >= 25 THEN
         v_grade := 'D';
      ELSE
         v_grade := 'E';
      END IF;
```

```
        UPDATE studres
        SET averg = v_avg, grade = v_grade
        WHERE regno = i.regno;
    END LOOP;
END;
/
```

```
PL/SQL procedure successfully completed
```

```
SQL> select* from studres;
```

3. **Salary Calculation**

Objective

A salary statement contains Name, Basic pay , allowance total , deduction (include , IT), gross pay, and net pay .

Allowance = 20% of basic pay
gross pay = Basic pay + Allowance.
Deduction = 10% of basic pay
income tax is calculated on the basis of annual income under the following condition.

annual salary	Income tax
< =300,000	Nil
>30,000 but <55,000	30% of excess over the amount Rs = 30,000/-
>=55,000	50% of excess over the amount Rs = 55,000/-

program:

```
SQL > Create table salary (empno number (5) primary key , name varchar(20), basis pay
number (10,2));
Table Created.
```

```
SQL> Insert into . salary values (1001 , 'Baby' ,15,000);
1 row Created
SQL > Insert into salary values (1002, 'Hanna', 20,000);
1 row Created
SQL>insert into salary values (1003, 'chinnu',6000);
1 row Created
SQL > insert into salary values (1004, 'megha', 400,000);
1 row Created.
SQL > Insert into salary values (1005, 'swetha', 5200);
1 row Created.
SQL > ALTER TABLE salary add (allowance number (10,2) , deduction number (10,2),gross
pay number (10,2), net pay number (10,2), income tax number (10,2));
Table Altered.


SQL >DECLARE
   v_allw  salary.ALLOWANCE%TYPE;
   v_gp    salary.GROSSPAY%TYPE;
   v_ded   salary.DEDUCTION%TYPE;
   v_net   salary.NETPAY%TYPE;
   v_inc   salary.INCOMETAX%TYPE;
   BASISPAY salary.BASISPAY%TYPE;
   an_in   NUMBER(10,2);

   CURSOR c IS SELECT * FROM salary;
BEGIN
   FOR i IN c LOOP
      v_allw := (20 * i.BASISPAY) / 100;
      v_gp   := i.BASISPAY + v_allw;
      v_ded  := (10 * i.BASISPAY) / 100;
      v_net  := v_gp - v_ded;
      an_in  := v_net * 12;

      IF an_in <= 30000 THEN
         v_inc := 0;
      ELSIF an_in BETWEEN 30000 AND 55000 THEN
         v_inc := (an_in - 30000) * 30 / 100;
      ELSE
         v_inc := ((an_in - 55000) * 50 / 100) + ((25 * 30000) / 100);
      END IF;
```

```
        UPDATE salary
        SET allowance = v_allw,
            grosspay = v_gp,
            deduction = v_ded,
            netpay = v_net,
            incometax = v_inc
        WHERE empno = i.empno;
    END LOOP;
END;
/
```

```
            PL/ SQL  Procedure successfully comleted.

            SQL > select * from salary ;
```

PL/SQL - Triggers

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).

Creating Triggers

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
```

```
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

```
Select * from customers;


+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

```
/


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );

UPDATE customers SET salary = salary + 500 WHERE id = 2;
```