PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL). PL/SQL is a combination of SQL along with the procedural features of programming languages.

Oracle created PL/SQL that extends some limitations of SQL to provide a more comprehensive solution for building mission-critical applications running on the Oracle database

Main advantages to use PL/SQL over SQL

- SQL is executed single articulation at once while PL/SQL is executed a square of code.
- SQL is definitive, i.e., it instructs the database anyway not how to do it. While PL/SQL is procedural, i.e., it reports the database how to get things done.
- SQL is a type of structured query language that we use for the database. PL/SQL is a type of programming language that acts as an extension to SQL for the database.
- SQL language does not contain any if control, FOR loop, and structures similar to these. PL/SQL language consists of if controls, while loop, FOR loop, and various other structures similar to these.
- SQL does not consist of any variables. PL/SQL consists of variables as well as data types and more.
- SQL type of language is data-oriented. PL/SQL type of language is application-oriented.
- One can make use of SQL for writing the queries, creating and executing the statements of DDL and DML. One can make use of PL/SQL for writing the program blocks, procedures, functions, packages, and triggers.

Need to start with

SET SERVEROUTPUT ON;

To comment a more than one line start with /* and end with */

To comment a single line give two hyphen -- before starting the comment

/* PL/SQL %TYPE Attribute The %TYPE attribute allow you to declare a constant, variable, or parameter to be of the same data type as previously declared variable, record, nested table, or database column.*/

There are 4 types of PL/SQL Loops.

1.Basic Loop / Exit Loop

PL/SQL exit loop is used when a set of statements is to be executed at least once before the termination of the loop. There must be an EXIT condition specified in the loop, otherwise the loop will get into an infinite number of iterations. After the occurrence of EXIT condition, the process exits the loop.

```
1. DECLARE
2. i NUMBER := 1;
3. BEGIN
4. LOOP
5. EXIT WHEN i>10;
6. DBMS_OUTPUT.PUT_LINE(i);
7. i := i+1;
8. END LOOP;
9. END;
```

O/P: 1 2 3 4 5 6 7 8 9 10

```
1. DECLARE
2. VAR1 NUMBER;
3. VAR2 NUMBER;
4. BEGIN
5. VAR1:=100;
6. VAR2:=1;
7. LOOP
8. DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
9. IF (VAR2=10) THEN
10. EXIT;
11. END IF;
12. VAR2:=VAR2+1;
```

```
13. END LOOP;
14. END;
```

O/P: 100 200 300 400….1000

2.While Loop

PL/SQL while loop is used when a set of statements has to be executed as long as a condition is true, the While loop is used. The condition is decided at the beginning of each iteration and continues until the condition becomes false.

```
1. DECLARE
2. i INTEGER := 1;
3. BEGIN
4. WHILE i <= 10 LOOP
5. DBMS_OUTPUT.PUT_LINE(i);
6. i := i+1;
7. END LOOP;
8. END;
```

O/P:: 1 2 3 …10

3.For Loop

PL/SQL for loop is used when when you want to execute a set of statements for a predetermined number of times. The loop is iterated between the start and end integer values. The counter is always incremented by 1 and once the counter reaches the value of end integer, the loop end

>>**SUM OF odd NUMBERS USING USER INPUT...for loop**

```
DECLARE
N NUMBER;
SUM1 NUMBER DEFAULT 0;
ENDVALUE NUMBER;
BEGIN
ENDVALUE:=&ENDVALUE;
N:=1;
FOR N IN 1.. ENDVALUE
LOOP
IF MOD(N,2)=1
```

```
THEN
SUM1:=SUM1+N;
END IF;
END LOOP;
DBMS_OUTPUT.PUT_LINE('SUM = ' || SUM1);
END;
```

USING **CONTINUE**

```
DECLARE
N NUMBER;
SUM1 NUMBER DEFAULT 0;
ENDVALUE NUMBER;
BEGIN
ENDVALUE:=&ENDVALUE;
N:=1;
FOR N IN 1.. ENDVALUE
LOOP
IF MOD(N,2)=1
THEN
CONTINUE;
END IF;
SUM1:=SUM1+N;
END LOOP;
DBMS_OUTPUT.PUT_LINE('SUM = ' || SUM1);
END;
```

```
DECLARE
N NUMBER;
SUM1 NUMBER DEFAULT 0;
ENDVALUE NUMBER;
BEGIN
ENDVALUE:=&ENDVALUE;
N:=1;
FOR N IN 1.. ENDVALUE
LOOP
CONTINUE
```

WHEN MOD(N,2)=0;

```
SUM1:=SUM1+N;
END LOOP;
DBMS_OUTPUT.PUT_LINE('SUM = ' || SUM1);
END;
```

Using IN REVERSE

```
DECLARE
N NUMBER;
SUM1 NUMBER DEFAULT 0;
ENDVALUE NUMBER;
BEGIN
ENDVALUE:=&ENDVALUE;
N:=1;
FOR N IN REVERSE 1.. ENDVALUE
LOOP
IF MOD(N,2)=1
THEN
SUM1:=SUM1+N;
END IF;
END LOOP;
DBMS_OUTPUT.PUT_LINE('SUM = ' || SUM1);
END;
```

>> CALCULATION OF NET SALARY

```
DECLARE
ENAME VARCHAR2(15);
BASIC NUMBER;
DA NUMBER;
HRA NUMBER;
PF NUMBER;
NETSALARY NUMBER;
BEGIN
```

```
ENAME:=&ENAME;
BASIC:=&BASIC;
DA:=BASIC * (41/100);
HRA:=BASIC * (15/100);
IF (BASIC < 3000)
THEN
PF:=BASIC * (5/100);
ELSIF (BASIC >= 3000 AND BASIC <= 5000)
THEN
PF:=BASIC * (7/100);
ELSIF (BASIC >= 5000 AND BASIC <= 8000)
THEN
PF:=BASIC * (8/100);
ELSE
PF:=BASIC * (10/100);
END IF;
NETSALARY:=BASIC + DA + HRA -PF;
DBMS_OUTPUT.PUT_LINE('EMPLOYEE NAME : ' || ENAME);
DBMS_OUTPUT.PUT_LINE('PROVIDEND FUND : ' || PF);
DBMS_OUTPUT.PUT_LINE('NET SALARY : ' || NETSALARY);
END;
```

4.Cursor For Loop

Procedure and Function::

A subprogram is a program unit/module that performs a particular task.

A subprogram can be created −

- At the schema level
- Inside a package
- Inside a PL/SQL block

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.

A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP

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

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

It can run two ways one

```
EXECUTE greetings;
```

Another in pl/sql block

```
BEGIN
   greetings;
END;
/
```

```
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;
```

Methods for Passing Parameters

Actual parameters can be passed in three ways −

Positional notation

Named notation

Mixed notation

Positional Notation

In positional notation, you can call the procedure as − findMin(a, b, c, d);

Named Notation- findMin(x => a, y => b, z => c, m => d);

Mixed Notation - findMin(a,y=>b,c);

Using Function::

```
DECLARE
   a number;
   b number;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
```

```
   ELSE
      z:= y;
   END IF;
   RETURN z;
END;
BEGIN
   a:= 23;
   b:= 45;
   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

CURSORS::

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement.

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.

It could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

Implicit cursors:: Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

%FOUND

Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

%NOTFOUND

The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

%ROWCOUNT

Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

```
 create table Empl(ename varchar2(10), dept varchar2(10), salary
number(6,2));
insert ALL
into Empl values('Harry', 'Database', 1000)
into Empl values('Ron', 'Networking', 2000)
into Empl values('Hermoine', 'OOP', 6000)
into Empl values('Annabeth', 'OS', 5000)
into Empl values('Percy', 'ML', 3000)
```

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
```

```
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
/
```

Explicit cursors : Explicit cursors are programmer-defined cursors for gaining more control over the context area.

```
CREATE TABLE CUSTOMERS( ID NUMBER(1), NAME VARCHAR2(15), ADDRESS
VARCHAR2(15));

INSERT INTO CUSTOMERS VALUES(1,'RAMESH','KOLKATA');
INSERT INTO CUSTOMERS VALUES(2,'KHILAN','DELHI');
INSERT INTO CUSTOMERS VALUES(3,'RAJESH','MUMBAI');
```

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```