# Lab Session 06

## *(PL/SQL)*

**By: Ammara Yaseen**

# PL/SQL

PL/SQL stands for procedural language-standard query language. It is a significant member of Oracle programming tool set which is extensively used to code server side programming. Similar to SQL language PL/SQL is also a **case-insensitive** programming language.

It contains the standard programming constructs you would expect from such a language, such as:

- Block Structure
- Variable & types
- Conditional Logic
- Loops
- Cursors, which holds the results returned by a query
- Procedures
- Functions
- Packages, which may be used to group procedures and functions together in one unit

## Block Structures

Generally a program written in PL/SQL language is divided into blocks. **We can say blocks are basic programming units in PL/SQL programming language.**

PL/SQL blocks are divided into 3 different sections which are:

1. The Declaration Section
2. The Execution Section and
3. The Exception-handling Section

The Execution Section is the only mandatory section of block whereas Declaration and Exception Handling sections are optional.

## Basic prototype of PL/SQL Block:

```
DECLARE
Declaration Statements
BEGIN
Executable statements
Exception
Exception handling statements
END;
```

## Declaration Section:

This is the first section of PL/SQL block which contains definition of PL/SQL identifiers such as variables, Constants, cursors and so on. You can say this is the place where all local variables used in the program are defined and documented.

```
DECLARE
Var_first_name VARCHAR2(30);
Var_last_name   VARCHAR2(30);
Con_flag        CONSTANT        NUMBER:=0;
```

## Execution Section:

This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section. The content of this section must be complete to allow the block to compile. By complete,mean complete set of instruction for the PL/SQL engine must be between BEGIN and END keyword.

The execution Section of any PL/SQL block always begins with the Keyword BEGIN and ends with the Keyword END.

This is the only mandatory section in PL/SQL block. This section supports all DML commands and SQL*PLUS built-in functions.

```
BEGIN
SELECT first_name, last_name INTO var_first_name,
var_last_name
FROM employees WHERE employee_id =100;
DBMS_OUTPUT.PUT_LINE
('Employee Name '||var_first_name||' '||var_last_name);
END;
```

**Exception-Handling Section:**

This is the last section of PL/SQL block which is optional like the declaration block. This section contains statements that are executed when a runtime error occurs within the block.

Runtime error occurs while the program is running and cannot be detected by the PL/SQL compiler. When a runtime error occurs, controlled is pass to the exception handling section of the block the error is evaluated and specific exception is raised.

```
EXCEPTION
WHEN NO_DATA_FOUND THEN     DBMS_OUTPUT.PUT_LINE ('No Employee Found
with '||employee_id);
```

## Variables and Types:

Variable declared within the DECLARE section may only be referenced within that block. A variable declaration has both a name and a type.

- **Syntax:**

  [Name of the variable] [Type];

- **Example:**

```
SET SERVEROUTPUT ON;
DECLARE
Test_var1 NUMBER;  — Declaring variable Test_var
BEGIN
Test_var1:= 10;
DBMS_OUTPUT.PUT_LINE (Test_var1);
END;
```

- **USING %TYPE keyword:**

It is used to declare a variable of the same type as a specified column in a table:

SAL EMP.SAL**%TYPE**;

## Example 1

```
SET SERVEROUTPUT ON;
DECLARE
Width INTEGER;
Height INTEGER:= 2;
```

```
Area INTEGER;
BEGIN
Area:= 6;
Width: = area/height;
DBMS_OUTPUT.PUT_LINE (‘width = ' || width);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE ('Division by Zero');
END;
/
```

## Example 2

```
DECLARE
ENO EMP.EMPNO%TYPE;
NAME EMP.ENAME%TYPE;
BEGIN
ENO:= 2;
SELECT ENAME INTO NAME FROM EMP WHERE
EMPNO=ENO;
DBMS_OUTPUT.PUT_LINE ('NAME OF THE EMPLOYEE IS '||NAME);
END;
/
```

## Example 3

```
DECLARE
ENO EMP.EMPNO%TYPE;
NAME EMP.ENAME%TYPE;
BEGIN
ENO:= 2;
```

```
SELECT ENAME INTO NAME FROM EMP WHERE
EMPNO=ENO;
DBMS_OUTPUT.PUT_LINE ('NAME OF THE EMPLOYEE IS '||NAME);
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN DBMS_OUTPUT.PUT_LINE ('NO DATA FOUND');
END;
```

## Example 4(Taking Input)

```
SET SERVEROUTPUT ON
DECLARE
ENO EMP.EMPNO%TYPE;
NAME EMP.ENAME%TYPE;
BEGIN
ENO: = &ENO;
SELECT ENAME INTO NAME FROM EMP WHERE
EMPNO=ENO;
DBMS_OUTPUT.PUT_LINE ('NAME OF THE EMPLOYEE IS '||NAME);
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN DBMS_OUTPUT.PUT_LINE ('NO DATA FOUND');
END;
/
```

## Types Of Conditional Control Statement in PL/SQL:

In Oracle PL/SQL we have two types of conditional control statements which are

1. IF statements and
2. CASE statements

Both these statements can be further divided into different forms. For example IF statements has 3 different forms

1. IF THEN
2. IF THEN ELSE
3. IF THEN ELSEIF

**Syntax:**

```
IF CONDITION 1 THEN
STATEMENT 1;
ELSIF CONDITION 2 THEN
STATEMENT 2;
ELSIF CONDITION 3 THEN
STATEMENT 3;
...
ELSE
STATEMENT N; END IF;
```

**Example:**

```
DECLARE
 v_Place VARCHAR2(30) := '&Enter Place';
BEGIN
 IF v_Place = 'Metropolis' THEN
 DBMS_OUTPUT.PUT_LINE('This City Is Protected By Superman');
ELSIF v_Place = 'Gotham' THEN
 DBMS_OUTPUT.PUT_LINE('This City is Protected By Batman');
ELSIF v_Place = 'Amazon' THEN
DBMS_OUTPUT.PUT_LINE('This City is protected by Wonder Woman');
ELSE
DBMS_OUTPUT.PUT_LINE('Please Call Avengers');
END IF;
DBMS_OUTPUT.PUT_LINE('Thanks For Contacting us');END;
```

# Types of Loops in Oracle PL/SQL:

There are 4 types of Loops in Oracle PL/SQL

1. Simple Loop
2. While Loop
3. Numeric For Loop and
4. Cursor For loop

## SIMPLE LOOP:

**Syntax:**

```
LOOP
Statement 1;
Statement 2;
...
Statement 3;
END LOOP;
```

**Example:**

```
DECLARE
  v_counter   NUMBER :=0;
  v_result  NUMBER;
BEGIN
  LOOP
  v_counter := v_counter+1;
  v_result := 19*v_counter;
  DBMS_OUTPUT.PUT_LINE('19'||' x '||v_counter||' = '|| v_result);
IF v_counter >=10 THEN
  EXIT;
END IF;
  END LOOP;
END;
```

```
DECLARE
  v_counter   NUMBER :=0;
  v_result  NUMBER;
BEGIN
  LOOP
  v_counter := v_counter+1;
  v_result := 19*v_counter;
  DBMS_OUTPUT.PUT_LINE('19'||' x '||v_counter||' = '|| v_result);
EXIT WHEN i_counter>=10;
  END LOOP;
END;
```

**WHILE LOOP:**

**Syntax:**

```
  WHILE condition LOOP
  Statement 1;
  Statemen 2;
  ...
  Statement 3;
  END LOOP;
```

**Example:**

```
DECLARE
  v_counter  NUMBER :=1;
  v_result NUMBER ;
BEGIN
  WHILE  v_counter <= 10
LOOP
  v_result := 9  *v_counter;
DBMS_OUTPUT.PUT_LINE('9'||' x '||v_counter||' = '||v_result);
  v_counter  := v_counter+1;
END LOOP;
DBMS_OUTPUT.PUT_LINE('out');
END;
/
```

**FOR LOOP:**

**Syntax:**

```
FOR loop_counter IN [REVERSE] lower limit.. upper_limit LOOP
  Statement 1;
  Statement 2;
  …
  Statement 3;
END LOOP;
```

**Example 1:**

```
SET SERVEROUTPUT ON;
BEGIN
  FOR v_counter IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
  END LOOP;
END;
```

**Example 2:**

```
BEGIN
  FOR v_counter IN REVERSE 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
  END LOOP;
END;
/
```

**Example 3:**

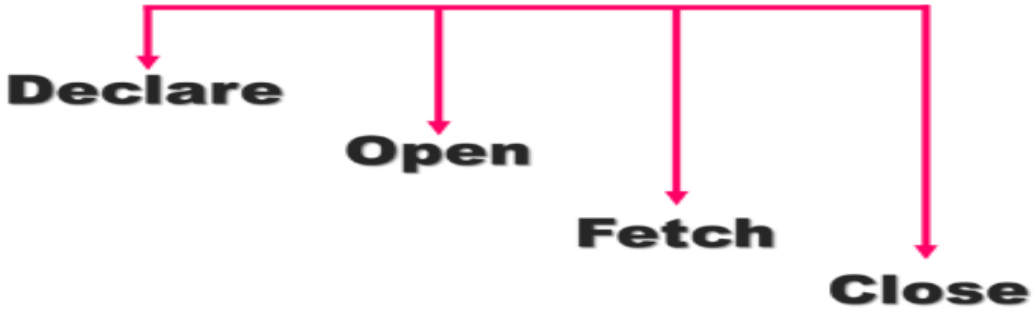
```
DECLARE
  v_result  NUMBER;
BEGIN
  FOR v_counter IN  1..10 LOOP
    v_result:= 19*v_counter;
    DBMS_OUTPUT.PUT_LINE(v_result);
  END LOOP;
END;
/
```

# Cursors

Cursor is a pointer to a memory area called context area. This context area is a memory region inside the Process Global Area or PGA assigned to hold the information about the processing of a SELECT statement or DML Statement such as INSERT, DELETE, UPDATE or MERGE.

- You use cursor when you have a SELECT statement that returns more than one row from the database. A cursor is basically a set of rows that you can access one at a time.

Steps for creating Cursor:




**STEP 1 : Declare variables to store column values:**

These variables must be compatible with the column types.

**DECLARE**

v_Empno emp.empno%TYPE;

v_Ename emp.ename%TYPE;

v_Job emp.job%TYPE;

**STEP 2: Declare the cursor:**

```
CURSOR cursor_name IS select_statement;
```

**STEP 3: Open the Cursor**

This step runs the SELECT statement. It must be placed in the executable section of the block (BEGIN).

```
OPEN cursor_name;
```

**STEP 4: Fetch the rows from the cursor**

**Syntax:**

```
FETCH cursor name INTO PL/SQL variable;
```

**STEP 5: Close the Cursor**

Closing your cursors frees up system resources.

```
CLOSE cursor_name;
```

Basic Programming Structure of the Cursor:

```
DECLARE
    CURSOR cursor_name IS select_statement;
BEGIN
     OPEN cursor_name;
     FETCH cursor_name INTO PL/SQL variable [PL/SQL record];
CLOSE cursor_name;
END;
/
```

Example:

```
SET SERVEROUTPUT ON;
DECLARE
  v_name VARCHAR2(30);
  --Declare Cursor
  CURSOR cur_emp IS
  SELECT ename FROM EMP
  WHERE empno > 1050;
BEGIN
  OPEN cur_emp;
  LOOP
    FETCH cur_emp INTO v_name;
    DBMS_OUTPUT.PUT_LINE (v_name);
    EXIT WHEN cur_emp%NOTFOUND;
  END LOOP;--Simple Loop End
  CLOSE cur_emp;
END;
/
```

## Cursors and FOR Loops

You can use the power of FOR loop to access the rows in a cursor. When you use a FOR loop, you don't have to explicitly open and close the cursor------ the FOR loop does this automatically.

**Syntax**:

```
FOR loop_index IN cursor_name
    LOOP
        Statements…
    END LOOP;
```

Example 1:

```
SET SERVEROUTPUT ON;
DECLARE
 CURSOR cur_emp1 IS
 SELECT ename,job FROM emp
 WHERE empno >2000;
BEGIN
  FOR L_IDX IN cur_emp1
  LOOP
    DBMS_OUTPUT.PUT_LINE(L_IDX.ename||' '||L_IDX.job);
  END LOOP;
END;
/
```

Example 2:

```
SET SERVEROUTPUT ON;
BEGIN
  FOR  L_IDX  IN (SELECT ename, job FROM emp
  WHERE empno >2000)
  LOOP
     DBMS_OUTPUT.PUT_LINE (L_IDX.ename||' '||L_IDX.job);
  END LOOP;
END;
/
```

## Explicit Cursor Attributes

Attributes that Obtain status information about a cursor. The following table illustrates explicit cursors you can use with cursors:

| Attribute | Type | Description |
|---|---|---|
| %ISOPEN | Boolean | Evaluates to TRUE if the cursor is open |
| %NOTFOUND | Boolean | Evaluates to TRUE if the most recent fetch does not return a row |
| %FOUND | Boolean | Complement of %NOTFOUND |
| %ROWCOUNT | Number | Evaluates to the total number of rows returned so far |

# Procedures

A **procedure** is a group of PL/SQL statements that you can call by name.

**Syntax:**

```
CREATE [OR REPLACE] PROCEDURE pro_name (Parameter - List)
IS
    Declare statements
BEGIN
    Executable statements
END procedure name;
/
```

**Example:**

```
CREATE OR REPLACE PROCEDURE pr_example IS
  var_name VARCHAR2 (30):= 'Ammara';
  var_web VARCHAR2 (30) := 'fastnu.com';
BEGIN
  DBMS_OUTPUT.PUT_LINE('Whats Up? I am '||var_name||' from '||var_web);
END Pr_example;
/
```

**Invoking the Procedure**

```
begin
pr_example;
end;
```

OR

```
execute pr_example;
```

OR

```
exec pr_example;
```

# Functions

In Oracle Database we can define a PL/SQL function as a self-contained sub-program that is meant to do some specific well defined task. Functions are named PL/SQL block which means they can be stored into the database as a database object and can be reused.

**Syntax:**

```
CREATE [OR REPLACE] FUNCTION function_name
(Parameter 1, Parameter 2…)
RETURN datatype
IS
    Declare variable, constant etc.
BEGIN
    Executable Statements
    Return (Return Value);
END;
```

---

**Example:**

```
--Function Header
CREATE OR REPLACE FUNCTION circle_area (radius NUMBER)
RETURN NUMBER IS
--Declare a constant and a variable
pi      CONSTANT NUMBER(7,2) := 3.141;
area    NUMBER(7,2);
BEGIN
  --Area of Circle pi*r*r;
  area := pi * (radius * radius);
  RETURN area;
END;
```

**CALLING FUNCTION:**

```
begin
dbms_output.put_line(circle_area(25);
end;
/
```

OR

```
DECLARE
A NUMBER(45);
Begin
A:=circle_ares(29);
dbms_output.put_line(A);
end;
/
```

# Difference between stored procedures and functions:

| Sr.No. | User Defined Function | Stored Procedure |
|---|---|---|
| 1 | Function must return a value. | Stored Procedure may or not return values. |
| 2 | Will allow only Select statements, it will not allow us to use DML statements. | Can have select statements as well as DML statements such as insert, update, delete and so on |
| 3 | It will allow only input parameters, doesn't support output parameters. | It can have both input and output parameters. |
| 4 | It will not allow us to use try-catch blocks. | For exception handling we can use try catch blocks. |
| 5 | Transactions are not allowed within functions. | Can use transactions within Stored Procedures. |
| 6 | We can use only table variables, it will not allow using temporary tables. | Can use both table variables as well as temporary table in it. |
| 7 | Stored Procedures can't be called from a function. | Stored Procedures can call functions. |
| 8 | Functions can be called from a select statement. | Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure. |

# Views

A view is a virtual table.A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

**Syntax**

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

**Example:**

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table.

The view is created with the following SQL:

```
CREATE VIEW Current_Product_List AS
SELECT Product_ID,Product_Name
FROM Products
WHERE Discontinued='NO';
```

```
SELECT * FROM Current_Product _List;
```

**Example:**

```
CREATE VIEW Products_Above_Average_Price AS
SELECT Product_Name,Unit_Price
FROM Products
WHERE Unit_Price>(SELECT AVG(Unit_Price) FROM Products)
```

## View from View

```
CREATE VIEW Category_Sales_For_1997 AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS Category_Sales
FROM Product_Sales _for _1997
GROUP BY Category_Name;
```

# Exercise

1. Write a PL/SQL code that takes two inputs from user, add them and store the sum in new variable and show the output.

2. Write a PL/SQL code that takes two inputs, lower boundary and upper boundary, then print the sum of all the numbers between the boundaries INCLUSIVE.

3. Write a PL/SQL code to retrieve the employee name, hire date, and the department name in which he works, whose number is input by the user.

4. Write a PL/SQL code to check whether the given number is palindrome or not.

5. Write a PL/SQL code that takes all the required inputs from the user for the Employee table and then insert it into the Employee and Department table in the database.

6. Write a PL/SQL code to find the first employee who has a salary over $2500 and is higher in the chain of command than employee 7499. Note: For chain, use of LOOP is necessary.

7. Write a PL/SQL code to print the sum of first 100 numbers.

**Views:**

8. Create a view, that stores information of only those employees who belongs to Accounts department.

**Cursors:**

9. Write a PL/SQL code to print out the employee information who earns more than 2000 salary.
10. Write a PL/SQL program displaying top 10 employee details

**Procedures:**

11. To Write a pl/sql program for creating a procedure for calculating sum of two numbers. Execute it as well.