Lab Sheet : *PL/SQL PROGRAMMING*

NOTE: Create and Login in ORACLE @ **https://livesql.oracle.com/** for PL/SQL practices.

OBJECTIVES:

- **PL/SQL introduction**
  - **block Structure**
  - **Conditional logic**
  - **Loops**

Introduction :

The combination of SQL language along with the procedural features of programming languages known as PL/SQL. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL enables you to write programs that contain SQL statements.

Advantage of PL/SQL over SQL:

SQL enables us to create, organize, retrieve and maintain data stored in database it does not provide the features which a typical programming language offers, such as control-of-flow construct (sequence, selection and iteration construct) , or the facility to declare and use variables.

PL/SQL allows us to create functions and procedures, trap exceptions, use branching and looping constructs, put code in packages, do row by row processing of databases, and in general create a modular program.

BLOCK STRUCTURE:

PL/SQL programs are divided up into structures known as blocks, with each block containing PL/SQL and SQL statements. A PL/SQL block has the following structure:

```
[ DECLARE
      declaration_statements
  ]
 BEGIN
    executable_statements
  [ EXCEPTION
       exception_handling_statements
  ]
  END;
 /
```

- **declaration_statements:** declare the variables used in the rest of the PL/SQL block. DECLARE blocks are optional.
- **executable_statements :** are the actual executable SQL statements, which may include loops, conditional logic, and so on.
- **exception_handling_statements:** are statements that handle any execution errors that might occur when the block is run. EXCEPTION blocks are optional.

NOTE: Every statement is terminated by a semicolon (;), and a PL/SQL block is terminated using the forward slash (/) character. (You may also skip / character)

PL/SQL Variables : They are placeholders [temp storage area] that store the values that can change through the PL/SQL Block.

Declare variables as:

i) variable_name datatype [NOT NULL := value];

- NOT NULL is an optional specification on the variable.
- *value* or DEFAULT *value* is also an optional specification, where you can initialize a variable.
- Each variable declaration is a separate statement and must be terminated by a semicolon.

```
Example:    DECLARE
            salary number(4);
            dept varchar2(10) NOT NULL := "CSE MNIT ";
```

- **When a variable is specified as NOT NULL, you must initialize the variable when it is declared.**

ii) %TYPE: This attribute used in the declaration of a variable when the variables attributes must be picked up from a table column. Useful when declaring variables that will hold database values that has the same datatype as column of the table.

```
Variable_name   Table_name.Column_name%type;
   Example:     my_title        books.title%TYPE;
```

Note: we need not know the exact data type of the table column say of title of books table and if we changes the database definitions of column say of *title* the data type of *my_title* changes accordingly at run time.

iii) %ROWTYPE: A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable.

Columns in a row and corresponding fields in a record have the same names and data types.

Example :

```
DECLARE
   dept_rec dept%ROWTYPE;          -- declare record variable
```

Use **dot notation** to reference fields, as the following example shows:

```
my_deptno := dept_rec.deptno;   //here deptno is column in dept table
```

Two ways values can be assigned to variables:

1) Directly assign values to variables: **variable_name:= value;**
2) Assign values to variables directly from the database columns by using a **SELECT….. INTO** statement as shown below

```
SELECT column_name INTO variable_name  FROM table_name  [WHERE condition];
```

Displaying user messages on the screen: Whenever start the Oracle at that time required to write the input command "set serveroutput on".

To redirect messages in the DBMS_OUTPUT message buffer to standard output, we need to specify SET SERVEROUTPUT ON.

For setting the server output on execute command:

SET SERVEROUTPUT ON; **//no need to use @https://livesql.oracle.com**

Then write the following code to display a message:

DBMS_OUTPUT.PUT_LINE('Displaying own message');

above message show only on when you execute first " SET SERVEROUTPUT ON" command.

Comments in PL/SQL can take one of two forms:

- **Multi-line comments :** are delimited with **/*....*/** and
- **Single line comments:** starts with two dashes **--**.

Scope of variables in PS/SQL

- **Local variables** - Declared in the Inner block which cannot be referenced by outside Blocks.
- **Global variables** - Declared in the outer block which can be referenced by itself and by its inner blocks.

Task 1

Example1:

```
--SET SERVEROUTPUT ON;  //no need to use skip it
 DECLARE
        --assign value into num1 and num2 variable
        num1 number := 10;
        num2 number := 20;
BEGIN
        DECLARE
        /* declaring variable num_addition */
               num_addition number;
        BEGIN
         num_addition := num1 + num2;
        dbms_output.put_line('Addition is: ' || num_addition);
        END;
END;
/
```

NOTE: The backward slash '/' in the above program indicates to execute the above PL/SQL Block.
(you may skip backslash)

Example2.

```
DECLARE
   num1 int;
   num2 number;
BEGIN
   num1 := 100;
```

```
            num2 := 200;
               DECLARE
                 mult number;
               BEGIN
                  mult := num1 *num2;
    dbms_output.put_line('InnerBlock:'||num1||'*'||num2||'is'||
                          mult);

                END;
          END;
```

Note: In the above code variable 'mult' is declared in the inner block, so cannot be accessed in the outer block i.e. it cannot be accessed after second last END. The variables 'num1' and num2' can be accessed anywhere in the block.

// Now use **mult** variable in outer block and try to store in it multiplication/addition of num1 and num2 and print sum. Observer the effect.

Task 2: First create sailor/boat/reserve tables and then use them

To get the rating of sailor with sid '71' and display it on the screen.

```
DECLARE
   var_rating number(4);  tmpsid number(4) := 71;
  BEGIN
   SELECT rating INTO var_rating
            FROM sailors   WHERE sid = tmpsid;
   dbms_output.put_line(var_rating);
  dbms_output.put_line('The Sailor'||tmpsid||'has rating'||var_rating);
  END;
  /
```

PL/SQL Constants: *constant* is a value used in a PL/SQL Block that remains unchanged throughout the program. A constant is a user-defined literal value. You can declare a constant and use it instead of actual value.

```
constant_name CONSTANT datatype :=VALUE;
```

- *constant_name* is the name of the constant i.e. similar to a variable name.
- The word *CONSTANT* is a reserved word and ensures that the value does not change.
- *VALUE* - It is a value which must be assigned to a constant when it is declared. You cannot assign a value later.

```
Example:     DECLARE
             salary_increase CONSTANT number (3) := 10;
```

Note: You must **assign a value to a constant at the time you declare it.** If you do not assign a value to a constant while declaring it and try to assign a value in the execution section, you will get an error.

-Execution of below Pl/SQL block you will get error.

Example:

```
--SET SERVEROUTPUT ON;
DECLARE
 salary_increase CONSTANT number(3);
BEGIN
 salary_increase := 100;
 dbms_output.put_line (salary_increase);
END;
/
```

Task 3: The following example calculates the width of a rectangle given its area and height: -

```
-- SET SERVEROUTPUTON;
DECLARE
        v_width  INTEGER;
        v_height INTEGER := 2;
        v_area  INTEGER := 6;
BEGIN
        v_width := v_area / v_height;
        DBMS_OUTPUT.PUT_LINE('v_width = ' || v_width);
EXCEPTION
 WHEN ZERO_DIVIDE THEN
          DBMS_OUTPUT.PUT_LINE('Division  by   zero');
END;
/
```

EXPLANATION:

- The SET SERVEROUTPUT ON command turns the server output on so you can see the lines produced by DBMS_OUTPUT.
- PUT_LINE() on the screen when you run the script in SQL*Plus.
- The DECLARE block contains declarations for three INTEGER variables named v_width, v_height, and v_area **(always put v_ at the start of variable names).** The v_height and v_area variables are initialized to 2 and 6 respectively.
- **You may also specify a variable's type using the %TYPE keyword, which tells PL/SQL to use the same type as a specified column in a table. Suppose we have the following table structure:**

```
create table Employee(
                empno  number,
                sal number,
                comm.  number
                );
v_empno   employee.empno%TYPE;
```

- Next comes the BEGIN block, which contains three lines. The first line sets v_width to v_area divided by v_height; this means v_width is set to 3 (= 6 / 2). The third line calls DBMS_OUTPUT.PUT_LINE() to display the value of v_width on the screen. DBMS_OUTPUT is a built-in package of code that comes with the Oracle database; among other items, DBMS_OUTPUT contains procedures that allow you to output values to thescreen.
- Next, the EXCEPTION block handles any attempts to divide a number by zero. It does this by "catching" the ZERO_DIVIDE exception; in the example, no attempt is actually made to divide by zero, but if you change v_height to 0 and run the script you'll see the exception.
- At the very end of the script, the forward slash character (/) marks the end of the PL/SQL block.

- **CONDITIONAL LOGIC:**

We can use the IF, THEN, ELSE, ELSIF, and END IF keywords to perform conditional logic:

```
IF condition1 THEN
        statements1
ELSIF condition2 THEN
        statements2
ELSE
        statements3
END IF;
```

where

- condition1 and condition2 are Boolean expressions that evaluate to true or false.
- statements1, statements2, and statements3 are PL/SQL statements

FLAVOURS of CONDITIONAL LOGIC:

IF condition
THEN
statements;
ELSE
statements;
END IF;
	IF condition
	THEN
	statements;
	statements;
	ELSIF condtion THEN
	statements;
	ELSE
	statements;
	END IF
		IF condition THEN
		ELSE
		IF condition THEN
		statements;
		END IF;
		ELSIF condition THEN
		statements;
		END IF;

Task 4:

```
-- SET SERVEROUTPUT ON;
DECLARE
        v_message VARCHAR2(50);
        v_count   INTEGER := 2;
        v_area    INTEGER := 6;
BEGIN
        IF v_count > 0 THEN
                v_message := 'v_count is positive';
         IF v_area > 0 THEN
                v_message := 'v_count and v_area are positive';
        END IF;
        ELSIF v_count = 0 THEN
                v_message := 'v_count is zero';
        ELSE
                v_message := 'v_count is negative';
        END IF;
        DBMS_OUTPUT.PUT_LINE('v_message = ' || v_message);
        END;
        /
```

- **LOOPS:** You use a loop to run statements zero or more times. There are three types of loops in PL/SQL:
  - **Simple loops** run until you explicitly end the loop.
  - **WHILE loops** run until a specified condition occurs.
  - **FOR loops** run a predetermined number of times.

➢ **Simple loops:** A simple loop runs until you explicitly end the loop. The syntax for a simple loop is as follows:

```
LOOP
        statements
END LOOP;
```

To end the loop, you use either an EXIT or an EXIT WHEN statement. The EXIT statement ends a loop immediately; the EXIT WHEN statement ends a loop when a specified condition occurs.

Task 5:

```
-- SET SERVEROUTPUTON;
DECLARE
        v_counter NUMBER := 0;
BEGIN
        LOOP
                v_counter := v_counter + 1;
                DBMS_OUTPUT.PUT_LINE('v_counter = ' || v_counter);
        EXIT WHEN       v_counter = 5;
        END LOOP;
END;
/
```

➢ **WHILE LOOPS:** A WHILE loop runs until a specified condition occurs. The syntax for a WHILE loop is as follows:

```
WHILE  condition LOOP
        Statements
END LOOP;
```

Task 6:

```
-- SET SERVEROUTPUTON;
DECLARE
        v_counter NUMBER := 0;
BEGIN
        WHILE v_counter < 6 LOOP
                v_counter := v_counter + 1;
                DBMS_OUTPUT.PUT_LINE('v_counter = ' || v_counter);
        END LOOP;
END;
/
```

Task 7: Create a table **numtab(a,b)** where a,b is number and insert few rows into it. Afterward execute the following PL/SQl Block:

```
--SET SERVEROUTPUT ON;
    DECLARE
      x number;   y number;
    BEGIN
      SELECT a, b INTO x , y
                  FROM numtab WHERE a>3;
     IF x=3 THEN
        INSERT INTO numtab VALUES(y,x);
     ELSE
        INSERT INTO numtab VALUES( y+10,x+20);
        dbms_output.put_line('inserted into table");

      ENDIF;
    END;
    /
```

➢ **FOR LOOPS:** A FOR loop runs a predetermined number of times; you determine the number of times the loop runs by specifying the lower and upper bounds for a loop variable. The loop variable is then incremented (or decremented) each time around the loop. The syntax for a FOR loop is as follows:

FOR loop_variable **IN [REVERSE]** lower_bound..upper_bound **LOOP**
Statements
END LOOP;

where:

- **loop_variable** is the loop variable. You can use a variable that already exists as the loop variable, or you can just have the loop create a new variable for you (this occurs if the variable you specify doesn't exist). The loop variable value is increased (or decreased if you use the REVERSE keyword) by 1 each time through the loop.
- **REVERSE** means that the loop variable value is to be decremented each time through the loop. The loop variable is initialized to the upper boundary, and is decremented by 1 until the loop variable reaches the lower boundary. You must specify the lower boundary before the upper boundary.
- **lower_bound** is the loop's lower boundary. The loop variable is initialized to this lower boundary provided REVERSE is not used.
- **upper_bound** is the loop's upper boundary. If REVERSE is used, the loop variable is initialized to this upper boundary.

Task 8:

```
--SET SERVEROUTPUT ON;
    DECLARE
       v_counter NUMBER;
     BEGIN
       FOR  v_counter   IN 1..5 LOOP
         DBMS_OUTPUT.PUT_LINE( 'v_counter :' ||v_counter);
       END LOOP;
     END;
      /
```

Task 9: In above replace LOOP BLOCK BY FOLLWING CODE AND OBSERVE THE OUTPUT:

```
FOR v_counter IN REVERSE 1..5 LOOP
   DBMS_OUTPUT.PUT_LINE('v_counter :' ||v_counter);
END LOOP;
```

FLAVOURS of While LOGIC:

LOOP statements; EXIT; {or EXIT WHEN condition;} END LOOP; - Initialise a variable before the loop body. - Increment the variable in the loop. - Use a EXIT WHEN statement to exit from the Loop. - If we use a EXIT statement without WHEN condition, the statements in the loop is executed only once.	WHILE <condition> LOOP statements; END LOOP; -Initialise a variable before the loop body. - Increment the variable in loop. - EXIT WHEN statement and EXIT statements can be used in while loops but it's not done oftenly.	FOR counter IN start..end LOOP statements; END LOOP; • start - Start integer value. • end - End integer value. -counter variable is incremented by 1 and does not need to be incremented explicitly. - EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done oftenly.

Exercise

1: Write a PL /SQL block to inverse a given number and display the inverted number as output using for and while loop.

Hint: ***substr(string, start_position, [length])***

string is the source string.
start_position is the position for extraction. The first position in the string is always 1.
length is optional. It is the number of characters to extract. If this parameter is omitted, *substr* will return the entire string.

length(string1) : *string1* is the string to return the length for. If *string1* is NULL, then the function returns NULL.

symbol "||" in statement used for string concatenation: **eg.** *InvertedNO= invertedNO||substr(…);*

2: Write a PL/SQL code block to calculate the area of a circle for a value of radius varying 4 to 10.Store the radius and the corresponding values of calculated area in an empty table named **AREAS**, consisting of two columns Radius and Area and also display the calculated areas.

3: Write the PL/SQL block considering *SAILORS* relation of previous labs, which displays the sid,sname,rating and age of sailor with current date as the below given formats and in case rating of sailor greater than 7 then complete information of the that sailor will be stored in ***HigherRating*** table.

Sample output display format

```
-----------------------------------------------
sid          sname     rating   age      Date
-----------------------------------------------
 ...          ...         ...        ...
 ...          ...         ...        ...
-----------------------------------------------
```