**PRACTICAL NO 2**

**CONTROL STRUCTURE IN PL/SQL**

**SIMPLE LOOP**

* Loops allows a certain part of the code in a program to get executed for the desired number of times.

DECLARE

i NUMBER(2):= 1;

BEGIN

LOOP

dbms\_output.put\_line(i);

EXIT WHEN i >= 10;

i := i+1;

END LOOP;

END;

/

Loop Control Statements

Loop control statements are those that actually control the flow of execution inside the loop.

### **CONTINUE**

* This keyword sends an instruction to the PL/SQL engine that whenever PL/SQL engine encounters this keyword inside the loop, then it will skip the remaining code in the execution block of the code, and next iteration will start immediately.
* This will be mainly used if the code inside the loop wants to be skipped for certain iteration values.

### **EXIT / EXIT WHEN**

* This keyword sends an instruction to the PL/SQL engine that whenever PL/SQL engine encounters this keyword, then it will immediately exit from the current loop.
* If the PL/SQL engine encounters the EXIT in a nested loop, then it will come out of the loop in which it has been defined, i.e. in a nested loops, giving EXIT in the inner loop will only exit the control from inner loop but not from the outer loop.
* ‘EXIT WHEN’ is followed by an expression which gives a Boolean result.
* If the result is TRUE, then the control will EXIT.

### **GOTO**

This statement will transfer the control to the labeled statement (“GOTO <label> ;”). This has the following restrictions

* Transfer of control can be done only within the subprograms.
* Transfer of control cannot be done from exception handling part to the execution part

**USING WHILE LOOP**

A **WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

WHILE condition LOOP

sequence\_of\_statements

END LOOP;

## Example

DECLARE

a number(2) := 10;

BEGIN

WHILE a < 20 LOOP

dbms\_output.put\_line('value of a: ' || a);

a := a + 1;

END LOOP;

END;

/

**USING FOR LOOP**

A **FOR LOOP** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

## Syntax

FOR counter IN initial\_value .. final\_value LOOP

sequence\_of\_statements;

END LOOP;

## Example

DECLARE

a number(2);

BEGIN

FOR a IN 10 .. 20 LOOP

dbms\_output.put\_line('value of a: ' || a);

END LOOP;

END;

/

## Reverse FOR LOOP Statement

DECLARE

a number(2) ;

BEGIN

FOR a IN REVERSE 10 .. 20 LOOP

dbms\_output.put\_line('value of a: ' || a);

END LOOP;

END;

/

FOR LOOP :

EXAMPLE :- PRINT TABLE OF 5 USING FOR LOOP

Set serveroutput on;

DECLARE

ans NUMBER(4);

BEGIN

ans := 5;

FOR num IN 1**..**10 LOOP

DBMS\_OUTPUT.PUT\_LINE('5 x’ || num ||’=’|| ans);

ans := ans + 5;

END LOOP;

END;

/

**USING GOTO STATEMENT**

A **GOTO** statement in PL/SQL programming language provides an unconditional jump from the GOTO to a labelled statement in the same subprogram.

**NOTE** − The use of GOTO statement is not recommended in any programming language because it makes it difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a GOTO can be rewritten so that it doesn't need the GOTO.

### **Syntax**

GOTO label;

..

..

<< label >>

statement;

### **Example**

DECLARE

a number(2) := 10;

BEGIN

<<loopstart>>

-- while loop execution

WHILE a < 20 LOOP

dbms\_output.put\_line ('value of a: ' || a);

a := a + 1;

IF a = 15 THEN

a := a + 1;

GOTO loopstart;

END IF;

END LOOP;

END;

/

**USING CONTINUE**

## Example

DECLARE

a number(2) := 10;

BEGIN

-- while loop execution

WHILE a < 20 LOOP

dbms\_output.put\_line ('value of a: ' || a);

a := a + 1;

IF a = 15 THEN

-- skip the loop using the CONTINUE statement

a := a + 1;

CONTINUE;

END IF;

END LOOP;

END;

/

**Using Continue –When in Loop**

SQL> BEGIN

FOR num IN 1 .. 10 LOOP

CONTINUE WHEN MOD(num,2) != 0;

DBMS\_OUTPUT.PUT\_LINE(num);

END LOOP;

END;

/