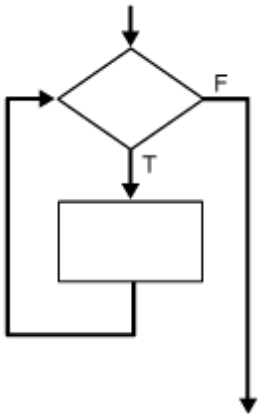
TATA CONSULTANCY SERVICES

# Oracle PL/SQL – Iterative Statements

## Iterative statements

- Iterative statements provide the ability to repeatedly execute a block of code until some condition is met.

- PL/SQL uses loops to accomplish iterative execution.




- Three types of iterative statements are

  - Simple Loop
  - FOR Loop
  - WHILE Loop

### 1. Simple Loops

- Encloses sequence of statements in between the LOOP and END LOOP keywords.
- The action statements can be a single or block of statements
- In each iteration, the sequence of statements within the loop is executed and then control resumes at the top of the loop
- An EXIT statement or an EXIT WHEN statement is required to break the loop and then the control passes to the next statement after the END LOOP.
- The condition can be a variable or expression which returns a boolean

value.

**Syntax:**

```
LOOP
     action_statements;
     EXIT [WHEN <condition>];
END LOOP;
```

## Using EXIT

- When the EXIT statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement after the END LOOP.
- For nested loops (one loop inside another loop), the EXIT statement will stop the execution of the innermost loop and start executing the next line of code after the block.

### Example: 1

In the below example, the loop is to print the numbers from 1 to 5. During the 6th iteration, when num is equal to 6 which is greater than 5, the loop terminates and the value num-1 (that is 5) is printed.

```
SET SERVEROUTPUT ON;
DECLARE
        num NUMBER(1) := 1;
BEGIN
        LOOP
            DBMS_OUTPUT.PUT_LINE(num);
            num := num + 1;
            IF num>5 THEN
               EXIT;
            END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('Total lines of output are ' || (num-1));
END;
/
```

**Output:**

```
1
2
3
4
5
Total lines of output are 5

PL/SQL procedure successfully completed
```

## Using EXIT WHEN:

- The EXIT-WHEN statement allows the condition in the WHEN clause to be evaluated.
- If the condition is true, the loop terminates and control passes to the statement immediately after END LOOP.
- Until the condition is true, the EXIT-WHEN statement acts like a NULL statement, except for evaluating the condition, and does not terminate the loop.
- A statement inside the loop must change the value of the condition. Otherwise,it may result in infinite loop.

### Example: 2

The below example is to print the numbers from 100 till 250 in increments of 25.

```
SET SERVEROUTPUT ON;
DECLARE
        a number:=100;
BEGIN
        -- begin loop
        LOOP
            a:=a+25;
        EXIT when a=250;
        DBMS_OUTPUT.PUT_LINE(a);
        END LOOP;
        -- end loop
END;
/
```

**Output:**

```
125
150
175
200
225

PL/SQL procedure successfully completed
```

## CONTINUE :

- The CONTINUE statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. In other words, it forces the next iteration of the loop to take place, skipping any code in between.

### 2. WHILE Loop:

- It repeatedly executes the statements in the loop body as long as a condition is true
- In WHILE loop the condition is tested at the top of the loop , so there are chances that the action statements may not be executed even once.
- If the condition is TRUE, the sequence of statements is executed and the control resumes at the top of the loop.
- If the condition is FALSE or NULL, the loop is skipped and control passes to the statement after the END LOOP.
- The number of iterations depends on the condition and is unknown until the loop completes.

**Syntax:**

```
WHILE <condition>
LOOP
    action_statements;
END LOOP;
```

### Example: 3

In the given example, the value for num1 is obtained from the user at run time. Here, num1 was entered as 5. The loop will be iterated till the value of num2 is less than num1.

```
set serveroutput on;
DECLARE
     num1 NUMBER:=&num1;
     num2 NUMBER:=1;
BEGIN
     WHILE (num2<=num1)
     LOOP
          DBMS_OUTPUT.PUT_LINE(num2);
          num2:=num2+1;
     END LOOP;
END;
/
```

### Output:

```
1
2
3
4
5

PL/SQL procedure successfully completed
```

**Note:**

'&' symbol is used to get the input from user during run time.

### 3. FOR Loop:

- FOR loops iterate over a specified range of integers (*lower_bound .. upper_bound*)

- The number of iterations are known before the loop is executed.
- The counter variable value or the range cannot be changed within the loop.
- The initial_value and final_value of the counter variable can be literals, variables, or expressions but must evaluate to numbers. Otherwise, PL/SQL raises the predefined exception VALUE_ERROR.
- The initial_value need not be 1; however, the loop counter increment (or decrement) must be 1.

**Syntax:**

```
FOR counter IN initial_value .. final_value
LOOP
   action_statements;
END LOOP;
```

Flow of control in a FOR loop is as follows:

- The initial step is executed first, and only once. This step allows you to declare and initialize any loop counter variables.

- Next, the condition, i.e., initial_value .. final_value is evaluated. It checks if the current counter value is within the range. If it is TRUE, the body of the loop is executed. If it is FALSE, the body of the loop does not execute and flow of control jumps to the next statement just after the END LOOP statement.

- After the body of the FOR loop executes (after each iteration), the value of the counter variable is increased or decreased by 1.

- The condition is now evaluated again. If it is TRUE, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). When the condition becomes FALSE, the FOR-LOOP terminates.

**Example: 4**

In the below example, the value for n (the number of rows to be inserted) is obtained from the user at run time. Here, n is given as 3 at run time and one row each will be inserted to employee table.

```
set serveroutput on;
DECLARE
     -- variable declaration
     n NUMBER:=&n;
     v_empid NUMBER(5);
     v_emp_name VARCHAR2(20):= 'Teresa';
     v_designation VARCHAR2(20):= 'ASE';
BEGIN
     FOR I IN 1..n
     LOOP
          -- assigning the counter value to v_empid
          v_empid := I;
          -- insert a row to Employee table for each iteration
          INSERT INTO Employee(emp_id,emp_name,designation)
                      VALUES(v_empid,v_emp_name,v_designation);
          COMMIT;
     END LOOP;
END;
/
```

**Output:**

**select * from employee;**

```
EMP_ID     EMP_NAME          DESIGNATION
-------------------------------------------------------------
   1          Teresa                    ASE
   2          Teresa                    ASE
   3          Teresa                    ASE
```

**4. REVERSE FOR Loop :**

- In REVERSE FOR loop, after each iteration the counter variable is decremented by 1 whereas in simple FOR, the counter is incremented by 1.

- By default, iteration proceeds from the lower_bound to the upper_bound. The order can be reversed by using the REVERSE keyword.

**Syntax:**

```
FOR counter REVERSE IN initial_value ..final_value
LOOP
   action_statements;
END LOOP;
```

**Example: 5**

```
DECLARE
       str VARCHAR2(200):= 'Result: ';
BEGIN
     FOR i IN REVERSE 1..10
     LOOP
         str:=str||' '||i;
     END LOOP;
     DBMS_OUTPUT.PUT_LINE(str);
END;
```

**Output:**

```
Result:  10 9 8 7 6 5 4 3 2 1

PL/SQL procedure successfully completed
```

### 5. Nested loops and labels

- PL/SQL allows nesting one loop within another loop. Loops can be nested to multiple levels.
- Loops can be labelled as per the requirement
- Looping can be done for Simple LOOP, WHILE and FOR loops.

**Syntax:**

```
LOOP
   Sequence_of_statements1
   LOOP
      Sequence_of_statements2
   END LOOP;
END LOOP;
```

### Example: 6

In the below example, names within << >> are label delimiters and names after the END LOOP (Loop1 and Loop2) are label names. Labels are optional.

```
DECLARE
        num NUMBER(3):=1;
BEGIN
        <<Loop1>> -- outer loop label
        LOOP
            <<Loop2>> -- inner loop label
            LOOP
                EXIT WHEN num>5;
                DBMS_OUTPUT.PUT_LINE(' Inner loop : ' ||num);
                num := num + 1;
            END LOOP Loop2; -- inner loop end here
            DBMS_OUTPUT.PUT_LINE(' Outer loop : ' || num);
            num := num + 1;
            EXIT WHEN num > 10;
        END LOOP Loop1;
END;
/
```

### Output:

```
Inner loop : 1
Inner loop : 2
Inner loop : 3
Inner loop : 4
Inner loop : 5
Outer loop : 6
Outer loop : 7
Outer loop : 8
Outer loop : 9
Outer loop : 10

PL/SQL procedure successfully completed
```