PL/SQL If

PL/SQL supports the programming language features like conditional statements and iterative statements. Its programming constructs are similar to how you use in programming languages like Java and C++.

**Syntax for IF Statement:**

There are different syntaxes for the IF-THEN-ELSE statement.

**Syntax: (IF-THEN statement):**

1. IF condition
2. **THEN**
3. Statement: {It **is** executed **when** condition **is** **true**}
4. **END** IF;

This syntax is used when you want to execute statements only when condition is TRUE.

**Syntax: (IF-THEN-ELSE statement):**

1. IF condition
2. **THEN**
3. {...statements **to** **execute** **when** condition **is** **TRUE**...}
4. **ELSE**
5. {...statements **to** **execute** **when** condition **is** **FALSE**...}
6. **END** IF;

This syntax is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

**Syntax: (IF-THEN-ELSIF statement):**

1. IF condition1
2. **THEN**
3. {...statements **to** **execute** **when** condition1 **is** **TRUE**...}
4. ELSIF condition2
5. **THEN**
6. {...statements **to** **execute** **when** condition2 **is** **TRUE**...}
7. **END** IF;

This syntax is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

**Syntax: (IF-THEN-ELSIF-ELSE statement):**

1. IF condition1
2. **THEN**
3. {...statements **to** **execute** **when** condition1 **is** **TRUE**...}
4. ELSIF condition2
5. **THEN**
6. {...statements **to** **execute** **when** condition2 **is** **TRUE**...}
7. **ELSE**
8. {...statements **to** **execute** **when** both condition1 and condition2 are **FALSE**...}
9. **END** IF;

It is the most advance syntax and used if you want to execute one set of statements when condition1 is TRUE, a different set of statement when condition2 is TRUE or a different set of statements when both the condition1 and condition2 are FALSE.

When a condition is found to be TRUE, the IF-THEN-ELSE statement will execute the corresponding code and not check the conditions any further.

If there no condition is met, the ELSE portion of the IF-THEN-ELSE statement will be executed.

ELSIF and ELSE portions are optional.

Example of PL/SQL If Statement

Let's take an example to see the whole concept:

1. **DECLARE**
2. a number(3) := 500;
3. **BEGIN**
4. -- check the boolean condition using if statement
5. IF( a < 20 ) **THEN**
6. -- if condition is true then print the following
7. dbms\_output.put\_line('a is less than 20 ' );
8. **ELSE**
9. dbms\_output.put\_line('a is not less than 20 ' );
10. **END** IF;
11. dbms\_output.put\_line('value of a is : ' || a);
12. **END**;

After the execution of the above code in SQL prompt, you will get the following result:

a is not less than 20

value of a is : 500

PL/SQL procedure successfully completed.

PL/SQL Case Statement

The PL/SQL CASE statement facilitates you to execute a sequence of satatements based on a selector. A selector can be anything such as variable, function or an expression that the CASE statement checks to a boolean value.

The CASE statement works like the IF statement, only using the keyword WHEN. A CASE statement is evaluated from top to bottom. If it get the condition TRUE, then the corresponding THEN calause is executed and the execution goes to the END CASE clause.

**Syntax for the CASE Statement:**

1. CASE [ expression ]
2. **WHEN** condition\_1 **THEN** result\_1
3. **WHEN** condition\_2 **THEN** result\_2
4. ...
5. **WHEN** condition\_n **THEN** result\_n
6. **ELSE** result
7. **END**

Example of PL/SQL case statement

Let's take an example to make it clear:

2

1. **DECLARE**
2. grade **char**(1) := 'A';
3. **BEGIN**
4. CASE grade
5. **when** 'A' **then** dbms\_output.put\_line('Excellent');
6. **when** 'B' **then** dbms\_output.put\_line('Very good');
7. **when** 'C' **then** dbms\_output.put\_line('Good');
8. **when** 'D' **then** dbms\_output.put\_line('Average');
9. **when** 'F' **then** dbms\_output.put\_line('Passed with Grace');
10. **else** dbms\_output.put\_line('Failed');
11. **END** CASE;
12. **END**;

After the execution of above code, you will get the following result:

Excellent

PL/SQL procedure successfully completed.

PL/SQL Loop

The PL/SQL loops are used to repeat the execution of one or more statements for specified number of times. These are also known as iterative control statements.

**Syntax for a basic loop:**

1. LOOP
2. **Sequence** **of** statements;
3. **END** LOOP;

Types of PL/SQL Loops

There are 4 types of PL/SQL Loops.

1. Basic Loop / Exit Loop
2. While Loop
3. For Loop
4. Cursor For Loop

PL/SQL Exit Loop (Basic 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.

**Syntax of basic loop:**

1. LOOP
2. **Sequence** **of** statements;
3. **END** LOOP;

**Syntax of exit loop:**

1. LOOP
2. statements;
3. EXIT;
4. {or EXIT **WHEN** condition;}
5. **END** LOOP;

Example of PL/SQL EXIT Loop

Let's take a simple example to explain it well:

Features of Java - Javatpoint

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**;

After the execution of the above code, you will get the following result:

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL Exit Loop.

* Initialize a variable before the loop body
* Increment the variable in the loop.
* You should use EXIT WHEN statement to exit from the Loop. Otherwise the EXIT statement without WHEN condition, the statements in the Loop is executed only once.

PL/SQL EXIT Loop Example 2

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**;

Output:

100

200

300

400

500

600

700

800

900

100

PL/SQL 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.

**Syntax of while loop:**

1. WHILE <condition>
2. LOOP statements;
3. **END** LOOP;

Example of PL/SQL While Loop

Let's see a simple example of PL/SQL WHILE loop.

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**;

After the execution of the above code, you will get the following result:

Triggers in SQL (Hindi)

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL WHILE Loop.

* Initialize a variable before the loop body.
* Increment the variable in the loop.
* You can use EXIT WHEN statements and EXIT statements in While loop but it is not done often.

PL/SQL WHILE Loop Example 2

1. **DECLARE**
2. VAR1 NUMBER;
3. VAR2 NUMBER;
4. **BEGIN**
5. VAR1:=200;
6. VAR2:=1;
7. WHILE (VAR2<=10)
8. LOOP
9. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
10. VAR2:=VAR2+1;
11. **END** LOOP;
12. **END**;

Output:

200

400

600

800

1000

1200

1400

1600

1800

2000

PL/SQL 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 ends.

**Syntax of for loop:**

1. **FOR** counter IN initial\_value .. final\_value LOOP
2. LOOP statements;
3. **END** LOOP;

* initial\_value : Start integer value
* final\_value : End integer value

PL/SQL For Loop Example 1

Let's see a simple example of PL/SQL FOR loop.

1. **BEGIN**
2. **FOR** k IN 1..10 LOOP
3. -- note that k was not declared
4. DBMS\_OUTPUT.PUT\_LINE(k);
5. **END** LOOP;
6. **END**;

After the execution of the above code, you will get the following result:

How to find Nth Highest Salary in SQL

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL WHILE Loop.

* You don't need to declare the counter variable explicitly because it is declared implicitly in the declaration section.
* The counter variable is incremented by 1 and does not need to be incremented explicitly.
* You can use EXIT WHEN statements and EXIT statements in FOR Loops but it is not done often.

PL/SQL For Loop Example 2

1. **DECLARE**
2. VAR1 NUMBER;
3. **BEGIN**
4. VAR1:=10;
5. **FOR** VAR2 IN 1..10
6. LOOP
7. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
8. **END** LOOP;
9. **END**;

Output:

10

20

30

40

50

60

70

80

90

100

PL/SQL For Loop REVERSE Example 3

Let's see an example of PL/SQL for loop where we are using REVERSE keyword.

1. **DECLARE**
2. VAR1 NUMBER;
3. **BEGIN**
4. VAR1:=10;
5. **FOR** VAR2 IN REVERSE 1..10
6. LOOP
7. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
8. **END** LOOP;
9. **END**;

Output:

100

90

80

70

60

50

40

30

20

10