Control Structures

PL/SQL Control Structures are used to control flow of execution.  PL/SQL provides different kinds of statements to provide such type of procedural capabilities.These statements are almost same as that of provided by other languages.  
 The flow of control statements can be classified into the following categories:

* Conditional Control
* Iterative Control
* Sequential Control

**Conditional Control :**

PL/SQL allows the use of an IF statement to control the execution of a block of code.   
 In PL/SQL, the IF -THEN - ELSIF - ELSE - END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed.

1) If (condition) then stmt.

2) If (condition) then stmt

Else stmt.

3) if (condition) then stmt

Elsif condtion then stmt

Elsif condtion then stmt

….

….

Else stmt.

4) case condition

When ‘value’ then stmt;

When ‘value’ then stmt;

..

..

Else stmt;

*Syntax:*

IF < Condition > THEN

  < Action >

ELSIF <Condition> THEN

< Action >

ELSE < Action >

END IF;

*Example:*

create file named "condi.sql"

DECLARE

   a Number := 30;       b Number;

BEGIN

       IF a > 40 THEN

      b := a - 40;

    DBMS\_OUTPUT.PUT\_LINE('b=' || b);

  elseif a = 30 then

  b := a + 40;

  DBMS\_OUTPUT.PUT\_LINE('b=' || b);

    ELSE

    b := 0;

  DBMS\_OUTPUT.PUT\_LINE('b=' || b);

 END IF;

END;

/

*Output:*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://condi.sql  
b=70  
  
PL/SQL successfully completed.

**Iterative Control :**

 Iterative control indicates the ability to repeat or skip sections of a code block.   
 A **loop** marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after the last statement in the sequence.   
Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops.  
 PL/SQL supports the following structures for iterative control:

**Simple loop :**

In simple loop, the key word loop should be placed before the first statement in the sequence and the keyword end loop should be written at the end of the sequence to end the loop.

*Syntax:*

**Loop**

**< Sequence of statements >**

**End loop;**

*Example:*

create file named it.sql

**DECLARE**

**i number := 0;**

**BEGIN**

**LOOP**

**dbms\_output.put\_line ('i = '||i);**

**i:=i+1;**

**EXIT WHEN i>=11;**

**END LOOP;**

**END;**

**/**

*Output:*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://it.sql  
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9  
i = 10  
PL/SQL successfully completed.

**WHILE loop**

The **while** loop executes commands in its body as long as the condtion remains true

*Syntax :*

**WHILE < condition >**

**LOOP**

**< Action >**

**END LOOP**

*Example :*

find reverse of given number using while loop

**DECLARE**

**num Number(3) :=123;**

**ans Number(3) :=0;**

**i Number(3) :=0;**

**BEGIN**

**WHILE num != 0**

**LOOP**

**i:=mod(num,10);**

**ans:=(ans \* 10 ) + i;**

**num:=floor(num/10);**

**END LOOP;**

**dbms\_output.put\_line('reverse of given number is: ' || ans);**

**END;**

**/**

*Output :*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://rev.sql  
  
reverse of given number is: 321  
PL/SQL successfully completed.

**The FOR Loop**

The  **FOR** loop can be used when the number of iterations to be executed are known.

*Syntax :*

FOR variable IN [REVERSE] start..end

LOOP

< Action >

END LOOP;

The variable in the For Loop need not be declared. Also the increment value cannot be specified. The For Loop variable is always incremented by 1.

*Example :*

**DECLARE**

**i number ;**

**BEGIN**

**FOR i IN 1 .. 10**

**LOOP**

**dbms\_output.put\_line ('i = '||i);**

**END LOOP;**

**END;**

**/**

*Output :*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://it.sql  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9  
i = 10  
PL/SQL successfully completed.

**Sequential Control :**

**The GOTO Statement**

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags «userdefined name». The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

*Syntax :*

GOTO jump;

....

<<jump>>

*Example :*

**DECLARE**

**BEGIN**

**dbms\_output.put\_line ('code starts');**

**dbms\_output.put\_line ('before GOTO statement');**

**GOTO down;**

**dbms\_output.put\_line ('statement will not get executed..');**

**<<down>>**

**dbms\_output.put\_line ('flow of execution jumped here.');**

**END;**

**/**

*Output :*

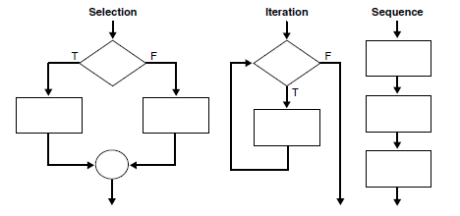
Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://a.sql  
code starts  
before gotostatements  
flow of execution jumped here.  
  
PL/SQL successfully completed.

NEW DATA

## ****Control Structures in PL/SQL****

Procedural computer programs use the basic control structures.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/control-structures.png)

* The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).
* The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.
* The sequence-structure simply executes a sequence of statements in the order in which they occur.

***Want to know more about SQL? Read this extensive***[***SQL Tutorial***](https://intellipaat.com/blog/tutorial/sql-tutorial/)***and enhance your knowledge!***

**Testing Conditions: IF and CASE Statements**  
The IF statement executes a sequence of statements depending on the value of a condition. There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.  
The CASE statement is a compact way to evaluate a single condition and choose between many alternative actions. It makes sense to use CASE when there are three or more alternatives to choose from.

* **Using the IF-THEN Statement**

The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF)  
The sequence of statements is executed only if the condition is TRUE. If the condition is FALSE or NULL, the IF statement does nothing. In either case, control passes to the next statement.  
**Example: Using a Simple IF-THEN Statement**

DECLARE

sales NUMBER(8,2) := 10100;

quota NUMBER(8,2) := 10000;

bonus NUMBER(6,2);

emp\_id NUMBER(6) := 120;

BEGIN

IF sales > (quota + 200) THEN

bonus := (sales - quota)/4;

UPDATE employees SET salary = salary + bonus WHERE employee\_id = emp\_id;

END IF;

END;

/

* **Using CASE Statements**

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.  
**Example: Using the CASE-WHEN Statement**

DECLARE

grade CHAR(1);

BEGIN

grade := 'B'; ‘&garde’ | ‘&character’

CASE grade

WHEN 'A' THEN DBMS\_OUTPUT.PUT\_LINE('Excellent');

WHEN 'B' THEN DBMS\_OUTPUT.PUT\_LINE('Very Good');

WHEN 'C' THEN DBMS\_OUTPUT.PUT\_LINE('Good');

WHEN 'D' THEN DBMS\_OUTPUT.PUT\_LINE('Fair');

WHEN 'F' THEN DBMS\_OUTPUT.PUT\_LINE('Poor');

ELSE DBMS\_OUTPUT.PUT\_LINE('No such grade');

END CASE;

END;

/

**Controlling Loop Iterations: LOOP and EXIT Statements**  
LOOP statements execute a sequence of statements multiple times. There are three forms of LOOP statements: LOOP, WHILE-LOOP, and FOR-LOOP.

* **Using the LOOP Statement**

The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

LOOP

sequence\_of\_statements

END LOOP;

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. You use an EXIT statement to stop looping and prevent an infinite loop. You can place one or more EXIT statements anywhere inside a loop, but not outside a loop. There are two forms of EXIT statements: EXIT and  
EXIT-WHEN.

***Go for this in-depth job-oriented***[***PL/SQL Training***](https://intellipaat.com/pl-sql-online-training/)***Course now!***

* **Using the EXIT Statement**

The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

* **Using the EXIT-WHEN Statement**

The EXIT-WHEN statement lets a loop complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement after the loop.

* **Labeling a PL/SQL Loop**

Like PL/SQL blocks, loops can be labeled. The optional label, an undeclared identifier enclosed by double angle brackets, must appear at the beginning of the LOOP statement. The label name can also appear at the end of the LOOP statement. When you nest labeled loops, use ending label names to improve readability.

* **Using the WHILE-LOOP Statement**

The WHILE-LOOP statement executes the statements in the loop body as long as a condition is true:

WHILE condition LOOP

sequence\_of\_statements

END LOOP;

***Come to***[***Intellipaat’s Community***](https://intellipaat.com/community/)***if you have queries!***

**Using the FOR-LOOP Statement**  
Simple FOR loops iterate over a specified range of integers. The number of iterations is known before the loop is entered. A double dot (..) serves as the range operator. The range is evaluated when the FOR loop is first entered and is never re-evaluated. If the lower bound equals the higher bound, the loop body is executed once.  
**Example: Using a Simple FOR LOOP Statement**

DECLARE

p NUMBER := 0;

BEGIN

FOR k IN 1..500 LOOP -- calculate pi with 500 terms

p := p + ( ( (-1) \*\* (k + 1) ) / ((2 \* k) - 1) );

END LOOP;

p := 4 \* p;

DBMS\_OUTPUT.PUT\_LINE( 'pi is approximately : ' || p ); -- print result

END;

/

**Sequential Control: GOTO and NULL Statements**  
The GOTO statement is seldom needed. Occasionally, it can simplify logic enough to warrant its use. The NULL statement can improve readability by making the meaning and action of conditional statements clear.  
Overuse of GOTO statements can result in code that is hard to understand and maintain. Use GOTO statements sparingly. For example, to branch from a deeply nested structure to an error-handling routine, raise an exception rather than use a GOTO statement.

* **Using the GOTO Statement**

The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block. The labeled statement or block can be down or up in the sequence of statements.  
**Example: Using a Simple GOTO Statement**

DECLARE

p VARCHAR2(30);

n PLS\_INTEGER := 37; -- test any integer > 2 for prime

BEGIN

FOR j in 2..ROUND(SQRT(n)) LOOP

IF n MOD j = 0 THEN -- test for prime

p := ' is not a prime number'; -- not a prime number

GOTO print\_now;

END IF;

END LOOP;

p := ' is a prime number';

<<print\_now>>

DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(n) || p);

END;

/

***Check out the top***[***PL/SQL Interview Questions***](https://intellipaat.com/blog/interview-question/pl-sql-interview-questions/)***to learn what is expected from PL/SQL professionals!***

* **Using the NULL Statement**

The NULL statement does nothing and passes control to the next statement. Some languages refer to such instruction as a no-op (no operation).  
**Example: Using the NULL Statement to Show No Action**

DECLARE

v\_job\_id VARCHAR2(10);

v\_emp\_id NUMBER(6) := 110;

BEGIN

SELECT job\_id INTO v\_job\_id FROM employees WHERE employee\_id = v\_emp\_id;

IF v\_job\_id = 'SA\_REP' THEN

UPDATE employees SET commission\_pct = commission\_pct \* 1.2;

ELSE

NULL; -- do nothing if not a sales representative

END IF;

END;

/