What is PL/SQL:

PL/SQL stands for Procedural Language extension of SQL. It was developed by Oracle Corporation in the late 1980s to enhance the capabilities of SQL. It is the procedural extension language for SQL.

## Advantages of PL/SQL:

1. PL/SQL is a procedural language.  
   2. PL/SQL is a block structure language.  
   3. PL/SQL handles the exceptions.  
   4. PL/SQL engine can process the multiple SQL statements simultaneously as a single block hence reduce network traffic and provides better performance.

PL/SQL block structure:

|  |
| --- |
| DECLARE  Declaration statements;  BEGIN  Execution statements;  EXCEPTION  Exception handling statements;  END;  / |

PL/SQL Block sections:

1. Declaration section (optional).  
2. Execution section (mandatory).  
3. Exception handling section (optional).

Declaration section:

It is an optional section and starts with DECLARE keyword. It is used to declare the variables, constants, records and cursors etc.

Execution section:

Execution section starts with BEGIN keyword and ends with END keyword. It is a mandatory section. It is used to write the program logic code.  
Note: Execution section must have one statement.

Exception handling section:

Execution section starts with EXCEPTION keyword. It is an optional section. It is used to handle the exceptions occurred in execution section.

Important points:

1. Every PL/SQL statement will be followed by semicolon (;).
2. 2. PL/SQL blocks can be nested.

Variable:

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs.

A variable should not exceed 30 characters.

It needs to declare the variable first in the declaration section of a PL/SQL block before using it.

2. By default, variable names are not case sensitive. A reserved PL/SQL keyword cannot be used as a variable name.

## How to declare variable in PL/SQL

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

Following is the syntax for declaring variable:

1. variable\_name [CONSTANT] datatype [NOT NULL] [:= | **DEFAULT** initial\_value]

Here, variable\_name is a valid identifier in PL/SQL and datatype must be valid PL/SQL data type. A data type with size, scale or precision limit is called a constrained declaration. The constrained declaration needs less memory than unconstrained declaration.

**Example:**

Radius Number := 5;

Date\_of\_birth date;

Declaration Restrictions:

In PL/SQL while declaring the variable some restrictions hold.

Forward references are not allowed i.e. you must declare a constant or variable before referencing it in another statement even if it is a declarative statement.

val number := Total - 200;

Total number := 1000;

The first declaration is illegal because the TOTAL variable must be declared before using it in an assignment expression.

Variables belonging to the same datatype cannot be declared in the same statement.

N1, N2, N3 Number;

It is an illegal declaration.

Naming rules for PL/SQL variables

The variable in PL/SQL must follow some naming rules like other programming languages.

* The variable\_name should not exceed 30 characters.
* Variable name should not be the same as the table table's column of that block.
* The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v\_data and V\_DATA refer to the same variables.
* You should make your variable easy to read and understand, after the first character, it may be any number, underscore (\_) or dollar sign ($).
* NOT NULL is an optional specification on the variable.

## Initializing Variables in PL/SQL

Evertime you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.

* The DEFAULT keyword
* The assignment operator

1. counter binary\_integer := 0;
2. greetings varchar2(20) **DEFAULT** 'Hello JavaTpoint';

You can also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable.

You must have a good programming skill to initialize variable properly otherwise, sometimes program would produce unexpected result.

Example of initilizing variable

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

**DECLARE**

1. a **integer** := 30;
2. b **integer** := 40;
3. c **integer**;
4. f **real**;
5. **BEGIN**
6. c := a + b;
7. dbms\_output.put\_line('Value of c: ' || c);
8. f := 100.0/3.0;
9. dbms\_output.put\_line('Value of f: ' || f);
10. **END**;

After the execution, this will produce the following result:

Value of c: 70

Value of f: 33.333333333333333333

PL/SQL procedure successfully completed.

* Local Variable: Local variables are the inner block variables which are not accessible to outer blocks.
* Global Variable: Global variables are declared in outermost block.

Example of Local and Global variables

Let's take an example to show the usage of Local and Global variables in its simple form:

1. **DECLARE**
2. -- Global variables
3. num1 number := 95;
4. num2 number := 85;
5. **BEGIN**
6. dbms\_output.put\_line('Outer Variable num1: ' || num1);
7. dbms\_output.put\_line('Outer Variable num2: ' || num2);
8. **DECLARE**
9. -- Local variables
10. num1 number := 195;
11. num2 number := 185;
12. **BEGIN**
13. dbms\_output.put\_line('Inner Variable num1: ' || num1);
14. dbms\_output.put\_line('Inner Variable num2: ' || num2);
15. **END**;
16. **END**;
17. /

Variable Scope in PL/SQL:

As we discussed in earlier tutorial that PL/SQL allows the nesting of blocks i.e. blocks with blocks. Based on the nesting structure PL/SQL variables can be divide into following categories:  
**Local variables** – Those variables which are declared in an inner block and not accessible to outer blocks are known as local variables.  
**Global variables** – Those variables which are declared in the outer block or a package and accessible to itself and inner blocks are known as global variables.

Example:

|  |
| --- |
| DECLARE  -- Global variables  num1 number := 10;  num2 number := 20;  BEGIN  dbms\_output.put\_line('Outer Variable num1: ' || num1);  dbms\_output.put\_line('Outer Variable num2: ' || num2);  DECLARE  -- Local variables  num3 number := 30;  num4 number := 40;  BEGIN  dbms\_output.put\_line('Outer variable in inner block num1: ' || num1);  dbms\_output.put\_line('Outer variable in inner block num2: ' || num2);  dbms\_output.put\_line('Inner Variable num3: ' || num3);  dbms\_output.put\_line('Inner Variable num4: ' || num4);  END;  END;  / |

Output:

|  |
| --- |
| Outer Variable num1: 10  Outer Variable num2: 20  Outer variable in inner block num1: 10  Outer variable in inner block num2: 20  Inner Variable num3: 30  Inner Variable num4: 40 |

## PL/SQL Constants:

A constant holds a value used in a PL/SQL block that does not change throughout the program. It is a user-defined literal value.

Let's take an example to explain it well:

Suppose, you have to write a program which will increase the salary of the employees upto 30%, you can declare a constant and use it throughout the program. Next time if you want to increase the salary again you can change the value of constant than the actual value throughout the program.

## Syntax to declare a constant:

|  |
| --- |
| constant\_name CONSTANT datatype := VALUE; |

## Where:

**constant\_name** – is a valid identifier name.  
**CONSTANT** – is a keyword.  
**VALUE** – is a value which must be assigned to a constant when it is declared. You cannot assign a value later.

## Example:

* NUMBER(8,2)------ 8 digits in total
* 2 of which are after the decimal point

|  |
| --- |
| DECLARE  -- constant declaration  pi constant number := 3.141592654;  -- other declarations  --ex:555.22  radius number(5,2);  dia number(5,2);  circumference number(7, 2);  area number (10, 2);  BEGIN  -- processing  radius := 10.5;  dia := radius \* 2;  circumference := 2.0 \* pi \* radius;  area := pi \* radius \* radius;  -- output  dbms\_output.put\_line('Radius: ' || radius);  dbms\_output.put\_line('Diameter: ' || dia);  dbms\_output.put\_line('Circumference: ' || circumference);  dbms\_output.put\_line('Area: ' || area);  END;  / |

## Output:

|  |
| --- |
| Radius: 10.5  Diameter: 21  Circumference: 65.97  Area: 346.36 |

## PL/SQL Literals:

Literals is an explicit numeric, character, string or Boolean values which are not represented by identifiers i.e. TRUE, NULL, w3spoint etc.  
**Note: PL/SQL literals are case-sensitive.**

## Types of literals in PL/SQL:

1. Numeric Literals (765, 23.56 etc.).  
2. Character Literals (‘A’ ‘%’ ‘9’ ‘ ‘ ‘z’ etc.).  
3. String Literals (tutorialspointexamples.com etc.).  
4. BOOLEAN Literals (TRUE, FALSE and NULL).  
5. Date and Time Literals (‘2016-12-25’ ‘2016-02-03 12:10:01’ etc.).

Hello worldprogram in plsql:

DECLARE

-- variable declaration

message varchar2(20):= 'Hello World!';

BEGIN

--output

dbms\_output.put\_line(message);

END;

/

## PL/SQL If statement:

If statement is used to execute a block of statements if specified condition is true.  
**Commonly used PL/SQL If statement:**

## IF-THEN statement:

***Syntax:***

|  |
| --- |
| **IF** condition  THEN  *//Block of statements1*  END **IF**; |

Block of statements1 executes when the specified condition is true.

## IF-THEN-ELSE statement:

***Syntax:***

|  |
| --- |
| **IF** condition  THEN  *//Block of statements1*  **ELSE**  *//Block of statements2*  END **IF**; |

Block of statements1 executes when the specified condition is true otherwise Block of statements2 executes.

## IF-THEN-ELSIF statement:

***Syntax:***

|  |
| --- |
| **IF** condition1  THEN  *//Block of statements1*  ELSIF condition2  *//Block of statements2*  **ELSE**  *//Block of statements3*  END **IF**; |

Block of statements1 executes when condition1 is true if false codition2 is checked and Block of statements2 executes if condition2 is true and so on. Block of statements in ELSE block executes when no condition is true.

## Example:

|  |
| --- |
| DECLARE  var number(3) := 50;  BEGIN  **IF** (var = 10) THEN  dbms\_output.put\_line('Value of var is 10');  ELSIF (var = 20) THEN  dbms\_output.put\_line('Value of var is 20');  ELSIF (var = 30) THEN  dbms\_output.put\_line('Value of var is 30');  **ELSE**  dbms\_output.put\_line('None of the above condition is true.');  END **IF**;  dbms\_output.put\_line('Exact value of var is: '|| var);  END;  / |

***Output:***

|  |
| --- |
| None of the above condition is **true**.  Exact value of var is: 50 |

## PL/SQL Case Statement:

Switch statement is used to execute a block of statement based on the switch expression value. An expression must be of type int, short, byte or char. A case value should be a constant literal value and cannot be duplicated. Expression value is compared with each case value. If a match found corresponding block of statements will be executed. A break statement is used to terminate the execution of statements. If no case value matches with expression value then default block of statements will be executed. If break statement is not used within case, all matching cases will be executed.  
***Syntax:***

|  |
| --- |
| **CASE** [expression]  WHEN condition1 THEN Block of statements1  WHEN condition2 THEN Block of statements2  ...  WHEN conditionn THEN Block of statementsn  **ELSE** Block of statements  END |

## Example:

|  |
| --- |
| DECLARE  nameChar **char**(1) := 'J';  BEGIN  **CASE** nameChar  when 'B' then dbms\_output.put\_line('Bharat');  when 'R' then dbms\_output.put\_line('Richi');  when 'S' then dbms\_output.put\_line('Sahdev');  when 'V' then dbms\_output.put\_line('Vinod');  when 'H' then dbms\_output.put\_line('Harish');  when 'M' then dbms\_output.put\_line('Mahesh');  when 'V' then dbms\_output.put\_line('Vivek');  when 'A' then dbms\_output.put\_line('Anil');  when 'J' then dbms\_output.put\_line('Jai');  **else** dbms\_output.put\_line('No such name');  END **CASE**;  END;  / |

## Output:

## Pl sql exit loop:

The pl sql loop repeatedly executes a block of statements until it reaches a loop exit. The EXIT and EXIT WHEN statements are used to terminate a loop.

## Where:

**EXIT:** The EXIT statement is used to terminate the loop unconditionally and normally used with IF statement.  
**EXIT WHEN:** The EXIT WHEN statement is used to terminate the loop conditionally. It terminates the loop when the specified condition is true.

## PL/SQL LOOP statement syntax with EXIT:

|  |
| --- |
| LOOP  *//block of statements*  EXIT;  END LOOP; |

## PL/SQL LOOP statement syntax with EXIT WHEN:

|  |
| --- |
| LOOP  *//block of statements*  EXIT WHEN condition;  END LOOP; |

## PL/SQL LOOP statement example with EXIT:

|  |
| --- |
| DECLARE  num NUMBER := 1;  BEGIN  LOOP  DBMS\_OUTPUT.PUT\_LINE(num);  **IF** num = 10 THEN  EXIT;  END **IF**;  num := num+1;  END LOOP;  END; |

## Output:

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10 |

## PL/SQL LOOP statement example with EXIT WHEN:

|  |
| --- |
| DECLARE  num NUMBER := 1;  BEGIN  LOOP  DBMS\_OUTPUT.PUT\_LINE(num);  EXIT WHEN num = 10;  num := num+1;  END LOOP;  END; |

## Output:

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10 |

## PL SQL WHILE LOOP:

The pl sql while loop repeatedly executes a block of statements until a particular condition is true. It first check the condition and executes a block of statements if condition is true.

## PL SQL WHILE LOOP syntax:

|  |
| --- |
| **WHILE** condition  LOOP  *//block of statements;*  END LOOP; |

## PL SQL WHILE LOOP example:

|  |
| --- |
| DECLARE  num NUMBER := 1;  BEGIN  **WHILE** num <= 10  LOOP  DBMS\_OUTPUT.PUT\_LINE(num);  num := num+1;  END LOOP;  END; |

## Output:

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10 |

## Pl sql for in loop:

The pl sql for in loop repeatedly executes a block of statements for a fixed number of times. The loop iteration occurs between the start and end integer values. The counter is always incremented by 1 and loop terminates when the counter reaches the value of the end integer.

## Pl sql for in loop syntax:

|  |
| --- |
| **FOR** loop\_counter IN [REVERSE] start\_value .. end\_value  LOOP  *//block of statements.*  END LOOP; |

**Note:**  
1. The double dot (..) specifies the range operator.  
2. By default iteration is from start\_value to end\_value but we can reverse the iteration process by using REVERSE keyword.  
3. No need to declare the counter variable explicitly because it is declared implicitly in the declaration section.  
4. The counter variable is incremented by 1 and does not need to be incremented explicitly.  
5. The EXIT and EXIT WHEN statements can be used.

## Pl sql for in loop example:

|  |
| --- |
| DECLARE  BEGIN  **FOR** var IN 1..10  LOOP  DBMS\_OUTPUT.PUT\_LINE(var);  END LOOP;  END; |

## Output:

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10 |

## Pl sql for in loop reverse example:

|  |
| --- |
| DECLARE  BEGIN  **FOR** var IN REVERSE 1..10  LOOP  DBMS\_OUTPUT.PUT\_LINE(var);  END LOOP;  END;    Output:  10  9  8  7  6  5  4  3  2  1 |

## Pl sql continue statement:

The pl sql continue statement is a control statement which is used to skip the following statement in the body of the loop and continue with the next iteration of the loop.

## Pl sql continue syntax:

|  |
| --- |
| **continue**; |

## Pl sql continue statement example:

|  |
| --- |
| DECLARE  num NUMBER := 0;  BEGIN  **WHILE** num < 10  LOOP  num := num +1;  **IF** num = 5 THEN  **CONTINUE**;  END **IF**;  DBMS\_OUTPUT.PUT\_LINE(num);  END LOOP;  END; |

## Output:

|  |
| --- |
| 1  2  3  4  6  7  8  9  10 |

## Pl sql goto statement:

The pl sql goto statement provides an unconditional jump from the GOTO to a labeled statement in the same subprogram. A label can be declare with the << label >> syntax.

## pl sql goto statement syntax:

|  |
| --- |
| **GOTO** label\_name;  *//Other statements*  <<label\_name>>  Statement; |

## Pl sql goto statement example:

|  |
| --- |
| DECLARE  num number := 1;  BEGIN  <<loop1>>  -- **while** loop execution  **WHILE** num <= 10 LOOP  dbms\_output.put\_line ('Value of num: ' || num);  num := num + 1;  **IF** num = 5 THEN  num := num + 1;  **GOTO** loop1;  END **IF**;  END LOOP;  END;  / |

## Output:

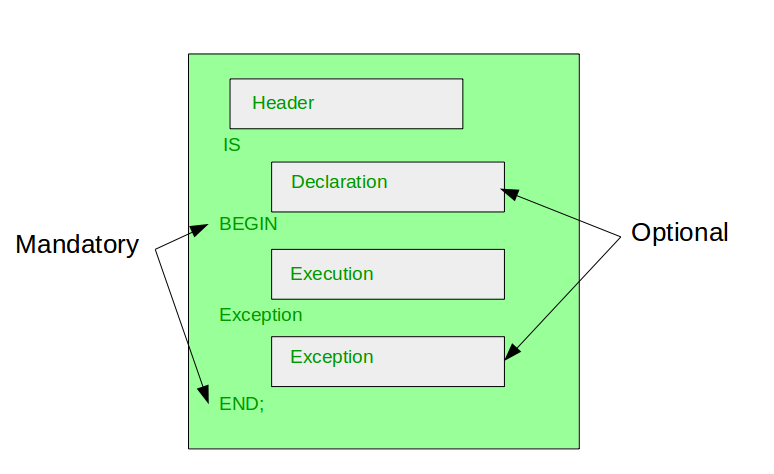
|  |
| --- |
| Value of num: 1  Value of num: 2  Value of num: 3  Value of num: 4  Value of num: 6  Value of num: 7  Value of num: 8  Value of num: 9  Value of num: 10 |

## Pl sql goto statement restrictions:

1. GOTO statement cannot transfer control into an IF statement, CASE statement, LOOP statement or sub-block.  
2. GOTO statement cannot transfer control from one IF statement clause to another or from one CASE statement WHEN clause to another.  
3. GOTO statement cannot transfer control from an outer block into a sub-block.  
4. GOTO statement cannot transfer control out of a subprogram.  
5. GOTO statement cannot transfer control into an exception handler.

# Blocks in PL/SQL:

In PL/SQL, All statements are classified into units that is called Blocks. PL/SQL blocks can include variables, SQL statements, loops, constants, conditional statements and exception handling. Blocks can also build a function or a procedure or a package.



Broadly, PL/SQL blocks are two types: Anonymous blocks and

* 1. **Anonymous blocks:** In PL/SQL, That’s blocks which is not have header are known as anonymous blocks. These blocks do not form the body of a function or triggers or procedure.

1. Example:

DECLARE

-- declare variable a, b and c

-- and these three variables datatype are integer

a number;

b number;

c number;

BEGIN

a:= 10;

b:= 100;

--find largest number

--take it in c variable

IF a > b THEN

c:= a;

ELSE

c:= b;

END IF;

dbms\_output.put\_line(' Maximum number in 10 and 100: ' || c);

END;

/

-- Program End

**2. Named blocks:** That’s PL/SQL blocks which having header or labels are known as Named blocks. These blocks can either be subprograms like functions, procedures, packages or Triggers.

Example: Here a code example of find greatest number with Named blocks means using function.

DECLARE

a number;

b number;

c number;

--Function return largest number of

-- two given number

BEGIN

a:= 10;

b:= 100;

c := findMax(a, b);

dbms\_output.put\_line(' Maximum number in 10 and 100 is: ' || c);

END;

/

Function:

create or replace FUNCTION findMax(x IN number, y IN number)

RETURN number

IS

z number;

BEGIN

IF x > y THEN

z:= x;

ELSE

Z:= y;

END IF;

RETURN z;

END;

/