PL/SQL:

PL/SQL is a block structured language that can have multiple blocks in it.

Our PL/SQL tutorial includes all topics of PL/SQL language such as conditional statements, loops, arrays, string, exceptions, collections, records, triggers, functions, procedures, cursors etc.

What is PL/SQL

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

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. /

# PL/SQL Constants

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value. It can be declared and used instead of actual values.

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:**

20.5M

457

How to find Nth Highest Salary in SQL

1. constant\_name CONSTANT datatype := VALUE;

* **Constant\_name:**it is the name of constant just like variable name. The constant word is a reserved word and its value does not change.
* **VALUE:**it is a value which is assigned to a constant when it is declared. It can not be assigned later.

## Example of PL/SQL constant

Let's take an example to explain it well:

1. **DECLARE**
2. -- constant declaration
3. pi constant number := 3.141592654;
4. -- other declarations
5. radius number(5,2);
6. dia number(5,2);
7. circumference number(7, 2);
8. area number (10, 2);
9. **BEGIN**
10. -- processing
11. radius := 9.5;
12. dia := radius \* 2;
13. circumference := 2.0 \* pi \* radius;
14. area := pi \* radius \* radius;
15. -- output
16. dbms\_output.put\_line('Radius: ' || radius);
17. dbms\_output.put\_line('Diameter: ' || dia);
18. dbms\_output.put\_line('Circumference: ' || circumference);
19. dbms\_output.put\_line('Area: ' || area);
20. **END**;
21. /

After the execution of the above code at SQL prompt, it will produce the following result:.

1. Radius: 9.5
2. Diameter: 19
3. Circumference: 59.69
4. Area: 283.53
6. Pl/SQL **procedure** successfully completed.

## PL/SQL Literals

Literals are the explicit numeric, character, string or boolean values which are not represented by an identifier. For example: TRUE, NULL, etc. are all literals of type boolean. PL/SQL literals are case-sensitive. There are following kinds of literals in PL/SQL:

* Numeric Literals
* Character Literals
* String Literals
* BOOLEAN Literals
* Date and Time Literals

## Example of these different types of Literals:

|  |  |
| --- | --- |
| **Literals** | **Examples** |
| Numeric | 75125, 3568, 33.3333333 etc. |
| Character | 'A' '%' '9' ' ' 'z' '(' |
| String | Hello JavaTpoint! |
| Boolean | TRUE, FALSE, NULL etc. |
| Date and Time | '26-11-2002' , '2012-10-29 12:01:01' |

# 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):**

OOPs Concepts in Java

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**;

# 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:

19.6M

421

HTML Tutorial

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**;

# 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:

23.7M

532

C++ vs Java

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**;

# 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**;

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**;

# PL/SQL Continue Statement

The continue statement is used to exit the loop from the reminder if its body either conditionally or unconditionally and forces the next iteration of the loop to take place, skipping any codes in between.

The continue statement is not a keyword in Oracle 10g. It is a new feature encorporated in oracle 11g.

For example: If a continue statement exits a cursor FOR LOOP prematurely then it exits an inner loop and transfer control to the next iteration of an outer loop, the cursor closes (in this context, CONTINUE works like GOTO).

**Syntax:**

1. **continue**;

## Example of PL/SQL continue statement

Let's take an example of PL/SQL continue statement.

1. **DECLARE**
2. x NUMBER := 0;
3. **BEGIN**
4. LOOP -- After CONTINUE statement, control resumes here
5. DBMS\_OUTPUT.PUT\_LINE ('Inside loop:  x = ' || TO\_CHAR(x));
6. x := x + 1;
7. IF x < 3 **THEN**
8. **CONTINUE**;
9. **END** IF;
10. DBMS\_OUTPUT.PUT\_LINE
11. ('Inside loop, after CONTINUE:  x = ' || TO\_CHAR(x));
12. EXIT **WHEN** x = 5;
13. **END** LOOP;
15. DBMS\_OUTPUT.PUT\_LINE (' After loop:  x = ' || TO\_CHAR(x));
16. **END**;
17. /

# PL/SQL GOTO Statement

In PL/SQL, GOTO statement makes you able to get an unconditional jump from the GOTO to a specific executable statement label in the same subprogram of the PL/SQL block.

Here the label declaration which contains the label\_name encapsulated within the << >> symbol and must be followed by at least one statement to execute.

**Syntax:**

1. **GOTO** label\_name;

Here the label declaration which contains the label\_name encapsulated within the << >> symbol and must be followed by at least one statement to execute.

11.8M

188

Triggers in SQL (Hindi)

1. **GOTO** label\_name;
2. ..
3. ..
4. <<label\_name>>
5. Statement;

## Example of PL/SQL GOTO statement

Let's take an example of PL/SQL GOTO statement.

1. **DECLARE**
2. a number(2) := 30;
3. **BEGIN**
4. <<loopstart>>
5. -- while loop execution
6. WHILE a < 50 LOOP
7. dbms\_output.put\_line ('value of a: ' || a);
8. a := a + 1;
9. IF a = 35 **THEN**
10. a := a + 1;
11. **GOTO** loopstart;
12. **END** IF;
13. **END** LOOP;
14. **END**;
15. /

# PL/SQL Procedure:

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

* **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
* **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

1. **IN parameters:**The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:**The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:**The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A procedure may or may not return any value.

## Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

|  |  |
| --- | --- |
| **S.No** | **Parts & Description** |
| 1 | **Declarative Part**  It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution. |
| 2 | **Executable Part**  This is a mandatory part and contains statements that perform the designated action. |
| 3 | **Exception-handling**  This is again an optional part. It contains the code that handles run-time errors. |

## Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

Where,

* *procedure-name* specifies the name of the procedure.
* [OR REPLACE] option allows the modification of an existing procedure.
* The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* *procedure-body* contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone procedure.

### Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

CREATE OR REPLACE PROCEDURE greetings

AS

BEGIN

dbms\_output.put\_line('Hello World!');

END;

/

When the above code is executed using the SQL prompt, it will produce the following result −

Procedure created.

**Syntax for creating procedure:**

1. **CREATE** [OR REPLACE] **PROCEDURE** procedure\_name
2. [ (parameter [,parameter]) ]
3. **IS**
4. [declaration\_section]
5. **BEGIN**
6. executable\_section
7. [EXCEPTION
8. exception\_section]
9. **END** [procedure\_name];

Example:

**Table creation:**

**create** **table** user(id number(10) **primary** **key**,**name** varchar2(100));

Now write the procedure code to insert record in user table.

1. **create** or replace **procedure** "INSERTUSER"
2. (id IN NUMBER,
3. **name** IN VARCHAR2)
4. **is**
5. **begin**
6. **insert** **into** user **values**(id,**name**);
7. **end**;
8. /

PL/SQL program to call procedure

Let's see the code to call above created procedure.

1. **BEGIN**
2. insertuser(101,'Rahul');
3. dbms\_output.put\_line('record inserted successfully');
4. **END**;
5. /
6. **DROP** **PROCEDURE** procedure\_name;

Example of drop procedure

1. **DROP** **PROCEDURE** pro1;

### IN & OUT Mode Example 1

This program finds the minimum of two values. Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.

DECLARE

a number;

b number;

c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS

BEGIN

IF x < y THEN

z:= x;

ELSE

z:= y;

END IF;

END;

BEGIN

a:= 23;

b:= 45;

findMin(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c);

END;

/

In and out parameters:

DECLARE

a number;

PROCEDURE squareNum(x IN OUT number) IS

BEGIN

x := x \* x;

END;

BEGIN

a:= 23;

squareNum(a);

dbms\_output.put\_line(' Square of (23): ' || a);

END;

/

PLSQL Functions:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax:

**Syntax to create a function:**

**CREATE** [OR REPLACE] **FUNCTION** function\_name [parameters]

1. [(parameter\_name [IN | **OUT** | IN **OUT**] type [, ...])]
2. **RETURN** return\_datatype
3. {**IS** | **AS**}
4. **BEGIN**
5. < function\_body >

## PL/SQL Trigger

[PL/SQL Trigger](https://www.javatpoint.com/pl-sql-trigger)

## Interview Questions

[PL/SQL Interview](https://www.javatpoint.com/pl-sql-interview-questions)

[SQL Interview](https://www.javatpoint.com/sql-interview-questions)

## Quiz

[SQL QUIZ](https://www.javatpoint.com/sql-quiz)

JavaTpoint

|  |
| --- |
| **[Next →](https://www.javatpoint.com/pl-sql-cursor)**  **[← Prev](https://www.javatpoint.com/pl-sql-procedure)** PL/SQL Function The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.  **Syntax to create a function:**  **CREATE** [OR REPLACE] **FUNCTION** function\_name [parameters]   1. [(parameter\_name [IN | **OUT** | IN **OUT**] type [, ...])] 2. **RETURN** return\_datatype 3. {**IS** | **AS**} 4. **BEGIN** 5. < function\_body > 6. **END** [function\_name]; |

**Here:**

* **Function\_name:** specifies the name of the function.
* **[OR REPLACE]** option allows modifying an existing function.
* The **optional parameter list** contains name, mode and types of the parameters.
* **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

### The function must contain a return statement.

* RETURN clause specifies that data type you are going to return from the function.
* Function\_body contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone function.

## PL/SQL Function Example

Let's see a simple example to **create a function**.

1. **create** or replace **function** adder(n1 in number, n2 in number)
2. **return** number
3. **is**
4. n3 number(8);
5. **begin**
6. n3 :=n1+n2;
7. **return** n3;
8. **end**;
9. /
10. **DECLARE**
11. n3 number(2);
12. **BEGIN**
13. n3 := adder(11,22);
14. dbms\_output.put\_line('Addition is: ' || n3);
15. **END**;
16. /

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return the maximum of two values.

**DECLARE**

1. a number;
2. b number;
3. c number;
4. **FUNCTION** findMax(x IN number, y IN number)
5. **RETURN** number
6. **IS**
7. z number;
8. **BEGIN**
9. IF x > y **THEN**
10. z:= x;
11. **ELSE**
12. Z:= y;
13. **END** IF;
15. **RETURN** z;
16. **END**;
17. **BEGIN**
18. a:= 23;
19. b:= 45;
21. c := findMax(a, b);
22. dbms\_output.put\_line(' Maximum of (23,45): ' || c);
23. **END**;
24. /

Example:

1. **CREATE** OR REPLACE **FUNCTION** totalCustomers
2. **RETURN** number
3. **IS**
4. total number(2) := 0;
5. **BEGIN**
6. **SELECT** count(\*) **into** total
7. **FROM** customers;
8. **RETURN** total;
9. **END**;
10. /
11. **DECLARE**
12. c number(2);
13. **BEGIN**
14. c := totalCustomers();
15. dbms\_output.put\_line('Total no. of Customers: ' || c);
16. **END**;
17. /