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.

PL/SQL Functionalities

PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers. It can support Array and handle exceptions (runtime errors). After the implementation of version 8 of Oracle database have included features associated with object orientation. You can create PL/SQL units like procedures, functions, packages, types and triggers, etc. which are stored in the database for reuse by applications.

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow of control statements to process the data.

The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages. It inherits the robustness, security, and portability of the Oracle Database.

PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals. A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

* Delimeters
* Identifiers
* Literals
* Comments

# **PL/SQL Variables**

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. It is nothing except a name given to a storage area. Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

#### **1. 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.

**Syntax for declaring variable:**

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:

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

After the execution, this will produce the following result:

Value of c: 70

Value of f: 33.333333333333333333

PL/SQL procedure successfully completed.

## Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If you declare a variable within an inner block, it is not accessible to an outer block. There are two types of variable scope:

* 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. /

After the execution, this will produce the following result:

Outer Variable num1: 95

Outer Variable num2: 85

Inner Variable num1: 195

Inner Variable num2: 185

PL/SQL procedure successfully completed.

### **Variable Attributes:**

When you declare a PL/SQL variable to hold the column values, it must be of correct data types and precision, otherwise error will occur on execution. Rather than hard coding the data type and precision of a variable. PL/SQL provides the facility to declare a variable without having to specify a particular data type using %TYPE and %ROWTYPE attributes. These two attributes allow us to specify a variable and have that variable data type be defined by a table/view column or a PL/SQL package variable.

A % sign servers as the attribute indicator. This method of declaring variables has an advantage as the user is not concerned with writing and maintaining code.

**Following are the types of Variable Attributes in PL/SQL.**

* **%TYPE:**

The %TYPE attribute is used to declare variables according to the already declared variable or database column. It is used when you are declaring an individual variable, not a record. The data type and precision of the variable declared using %TYPE attribute is the same as that of the column that is referred from a given table. This is particularly useful when declaring variables that will hold database values. When using the %TYPE keyword, the name of the columns and the table to which the variable will correspond must be known to the user. These are then prefixed with the variable name. If some previously declared variable is referred then prefix that variable name to the %TYPE attribute.

**The syntax for declaring a variable with %TYPE is:**

1. <var\_name> <tab\_name>.<column\_name>%TYPE;

Where <column\_name> is the column defined in the <tab\_name>.

**Consider a declaration.**

SALARY EMP.SAL % TYPE;

This declaration will declare a variable SALARY that has the same data type as column SAL of the EMP table.

**Example:**

1. **DECLARE**
2. SALARY EMP.SAL % TYPE;
3. ECODE EMP.empno % TYPE;
4. **BEGIN**
5. Ecode :=&Ecode;
6. **Select** SAL **into** SALARY **from** EMP **where** EMPNO = ECODE;
7. dbms\_output.put\_line('Salary of ' || ECODE || 'is = || salary');
8. **END**;

After the execution, this will produce the following result:

Enter value for ecode: 7499

Salary of 7499 is = 1600

PL/SQL procedure successfully completed.

* **%ROWTYPE:**

The %ROWTYPE attribute is used to declare a record type that represents a row in a table. The record can store an entire row or some specific data selected from the table. A column in a row and corresponding fields in a record have the same name and data types.

**The syntax for declaring a variable with %ROWTYPE is:**

1. <var\_name> <tab\_name>.ROW%TYPE;

Where <variable\_name> is the variable defined in the <tab\_name>.

**Consider a declaration.**

EMPLOYEE EMP. % ROW TYPE;

This declaration will declare a record named EMPLOYEE having fields with the same name and data types as that of columns in the EMP table. You can access the elements of EMPLOYEE record as

EMPLOYEE.SAL := 10000;

EMPLOYEE.ENAME := ‘KIRAN’;

**Example:**

1. **DECLARE**
2. EMPLOYEE EMP. % ROW TYPE;
3. **BEGIN**
4. EMPLOYEE.EMPNO := 2092;
5. 5   EMPLOYEE.ENAME := 'Sanju';
6. **Insert** **into** EMP **where** (EMPNO, ENAME) **Values** (employee.empno, employee.ename);
7. dbms\_output.put\_line('Row Inserted');
8. **END**;

After the execution, this will produce the following result:

Row Inserted

PL/SQL procedure successfully completed.

## Advantages:

* If you don’t know the data type at the time of declaration. The data type assigned to the associated variables will be determined dynamically at run time.
* If the data type of the variable you are referencing changes the %TYPE or %ROWTYPE variable changes at run time without having to rewrite variable declarations. For example: if the ENAME column of an EMP table is changed from a VARCHAR2(10) to VRACHAR2(15) then you don’t need to modify the PL/SQL code.

# **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:**

27.9M

644

Features of Java - Javatpoint

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' |