PL/SQL Placeholders

Placeholders are temporary storage area. Placeholders can be any of Variables, Constants and Records. Oracle defines placeholders to store data temporarily, which are used to manipulate data during the execution of a PL SQL block.   
  
Depending on the kind of data you want to store, you can define placeholders with a name and a datatype. Few of the datatypes used to define placeholders are as given below.   
Number (n,m) , Char (n) , Varchar2 (n) , Date , Long , Long raw, Raw, Blob, Clob, Nclob, Bfile

PL/SQL Variables

These are placeholders that store the values that can change through the PL/SQL Block.   
  
The General Syntax to declare a variable is:

*variable\_name datatype [NOT NULL := value ];*

* *variable\_name*is the name of the variable.
* *datatype*is a valid PL/SQL datatype.
* NOT NULL is an optional specification on the variable.
* *value*or DEFAULT *value* is also an optional specification, where you can initialize a variable.
* Each variable declaration is a separate statement and must be terminated by a semicolon.

For example, if you want to store the current salary of an employee, you can use a variable.

*DECLARE*

*salary number (6);*

\* “salary” is a variable of datatype number and of length 6.  
  
When a variable is specified as NOT NULL, you must initialize the variable when it is declared.  
  
For example: The below example declares two variables, one of which is a not null.

*DECLARE*

*salary number(4);*

*dept varchar2(10) NOT NULL := “HR Dept”;*

The value of a variable can change in the execution or exception section of the PL/SQL Block. We can assign values to variables in the two ways given below.

1) We can directly assign values to variables.   
    The General Syntax is:

*variable\_name:= value;*

2) We can assign values to variables directly from the database columns by using a SELECT.. INTO statement. The General Syntax is:

|  |
| --- |
| *SELECT column\_name*  *INTO variable\_name*  *FROM table\_name*  *[WHERE condition];* |

Example: The below program will get the salary of an employee with id '1116' and display it on the screen.

*DECLARE*

*var\_salary number(6);*

*var\_emp\_id number(6) = 1116;*

*BEGIN*

*SELECT salary*

*INTO var\_salary*

*FROM employee*

*WHERE emp\_id = var\_emp\_id;*

*dbms\_output.put\_line(var\_salary);*

*dbms\_output.put\_line('The employee '*

*|| var\_emp\_id || ' has salary ' || var\_salary);*

*END;*

*/*

**NOTE: The backward slash '/' in the above program indicates to execute the above PL/SQL Block.**

**Scope of Variables**

PL/SQL allows the nesting of Blocks within Blocks i.e, the Execution section of an outer block can contain inner blocks. Therefore, a variable which is accessible to an outer Block is also accessible to all nested inner Blocks. The variables declared in the inner blocks are not accessible to outer blocks. Based on their declaration we can classify variables into two types.

* *Local* variables - These are declared in a inner block and cannot be referenced by outside Blocks.
* *Global* variables - These are declared in a outer block and can be referenced by its itself and by its inner blocks.

For Example: In the below example we are creating two variables in the outer block and assigning thier product to the third variable created in the inner block. The variable 'var\_mult' is declared in the inner block, so cannot be accessed in the outer block i.e. it cannot be accessed after line 11. The variables 'var\_num1' and 'var\_num2' can be accessed anywhere in the block.

|  |
| --- |
| *1> DECLARE*  *2> var\_num1 number;*  *3>  var\_num2 number;*  *4> BEGIN*  *5>  var\_num1 := 100;*  *6>  var\_num2 := 200;*  *7>  DECLARE*  *8>   var\_mult number;*  *9>   BEGIN*  *10>    var\_mult := var\_num1 \* var\_num2;*  *11>   END;*  *12> END;*  *13> /* |