

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in PL/SQL has a specific data type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

The name of a PL/SQL variable consists of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters. By default, variable names are not case-sensitive. You cannot use a reserved PL/SQL keyword as a variable name.

PL/SQL programming language allows to define various types of variables, which we will cover in subsequent chapters like date time data types, records, collections, etc. For this chapter, let us study only basic variable types.

## Variable Declaration in PL/SQL

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

The syntax for declaring a variable is:

```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]
```

Where, *variable\_name* is a valid identifier in PL/SQL, *datatype* must be a valid PL/SQL data type or any user defined data type which we already have discussed in last chapter. Some valid variable declarations along with their definition are shown below:

```
sales number(10, 2);  
pi CONSTANT double precision := 3.1415;  
name varchar2(25);  
address varchar2(100);
```

When you provide a size, scale or precision limit with the data type, it is called a **constrained declaration**. Constrained declarations require less memory than unconstrained declarations. For example:

```
sales number(10, 2);  
name varchar2(25);  
address varchar2(100);
```

## Initializing Variables in PL/SQL

Whenever you declare a variable, PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following:

- The **DEFAULT** keyword
- The **assignment** operator

For example:

```
counter binary_integer := 0;  
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

You can also specify that a variable should not have a **NULL** value using the **NOT NULL** constraint. If you use the NOT NULL constraint, you must explicitly assign an initial value for that variable.

It is a good programming practice to initialize variables properly otherwise, sometimes program would produce unexpected result. Try the following example which makes use of various types of variables:

```
DECLARE
    a integer := 10;
    b integer := 20;
    c integer;
    f real;
BEGIN
    c := a + b;
    dbms_output.put_line('Value of c: ' || c);
    f := 70.0/3.0;
    dbms_output.put_line('Value of f: ' || f);
END;
```

When the above code is executed, it produces the following result:

```
Value of c: 30
Value of f: 23.333333333333333333
PL/SQL procedure successfully completed.
```

## Variable Scope in PL/SQL

PL/SQL allows the nesting of Blocks, i.e., each program block may contain another inner block. If a variable is declared within an inner block, it is not accessible to the outer block. However, if a variable is declared and accessible to an outer Block, it is also accessible to all nested inner Blocks. There are two types of variable scope:

- **Local variables** - variables declared in an inner block and not accessible to outer blocks.
- **Global variables** - variables declared in the outermost block or a package.

Following example shows the usage of **Local** and **Global** variables in its simple form:

```
DECLARE
    -- Global variables
    num1 number := 95;
    num2 number := 85;
BEGIN
    dbms_output.put_line('Outer Variable num1: ' || num1);
    dbms_output.put_line('Outer Variable num2: ' || num2);
    DECLARE
        -- Local variables
        num1 number := 195;
        num2 number := 185;
    BEGIN
        dbms_output.put_line('Inner Variable num1: ' || num1);
        dbms_output.put_line('Inner Variable num2: ' || num2);
    END;
END;
```

When the above code is executed, it produces the following result:

```
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185
```

PL/SQL procedure successfully completed.

## Assigning SQL Query Results to PL/SQL Variables

You can use the SELECT INTO statement of SQL to assign values to PL/SQL variables. For each item in the SELECT list, there must be a corresponding, type-compatible variable in the INTO list. The following example illustrates the concept: Let us create a table named CUSTOMERS:

**(For SQL statements please look at the [SQL tutorial](#))**

```
CREATE TABLE CUSTOMERS(  
  ID      INT NOT NULL,  
  NAME    VARCHAR (20) NOT NULL,  
  AGE     INT NOT NULL,  
  ADDRESS CHAR (25),  
  SALARY  DECIMAL (18, 2),  
  PRIMARY KEY (ID)  
);
```

Table Created

Next, let us insert some values in the table:

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );  
  
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );  
  
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );  
  
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );  
  
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );  
  
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

The following program assigns values from the above table to PL/SQL variables using the SELECT INTO clause of SQL:

```
DECLARE  
  c_id customers.id%type := 1;  
  c_name customers.name%type;  
  c_addr customers.address%type;  
  c_sal customers.salary%type;  
BEGIN  
  SELECT name, address, salary INTO c_name, c_addr, c_sal  
  FROM customers  
  WHERE id = c_id;  
  
  dbms_output.put_line  
  ('Customer ' || c_name || ' from ' || c_addr || ' earns ' || c_sal);  
END;  
/
```

When the above code is executed, it produces the following result:

Customer Ramesh from Ahmedabad earns 2000

PL/SQL procedure completed successfully