TATA CONSULTANCY SERVICES

TATA

Oracle PL/SQL – Variables

PL/SQL Variables

- Variables are memory regions used in a PL/SQL block to hold data.
- Defined in the DECLARATION section of the block, where they are assigned a specific data type, size and are often initialized with a value.
- When a variable is declared, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

PL/SQL Variables Rules

Rules to be followed while naming a variable

- Variable name should not exceed 30 characters
- Reserved PL/SQL keywords cannot be used as variable names
- Variable name cannot start with a number
- Symbols other than &,_,# cannot be used in variable names
- Two variables can have the same name, provided they are in different blocks.
- The variable name (identifier) should not be the same as the name of table columns used in the block.
- The variable names should be meaningful.

Declaring PL/SQL Variables

Syntax

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

Example

```
DECLARE

    v_hiredate   DATE;
    v_deptno     NUMBER(2) NOT NULL := 10;
    v_location   VARCHAR2(13) := 'Atlanta';
    c_comm       CONSTANT NUMBER := 1400;
     v_salary    NUMBER DEFAULT 0;
```

Initializing Variables

A variable can be initialized with a value using

- The DEFAULT keyword
- The assignment operator (:=)
- Values can be directly assigned from database columns to variables using SELECT..INTO statement.

Syntax

```
variable_name := expr;
```

Example:

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Happy Learning';
```

Variable Scope

- PL/SQL allows nesting of blocks within blocks.

- Scope refers to the accessibility and availability of a variable within a block.

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 and can be accessed from all the inner blocks

Variable Scope Example

```
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
        num3 number := 195;
        num2 number := 185;
    BEGIN
        dbms_output.put_line('Inner Variable num1: ' || num1);
        dbms_output.put_line('Inner Variable num2: ' || num2);
        dbms_output.put_line('Inner Variable num3: ' || num3);
    END;
END;
/
```

Variable Scope Example Output

Output:

```
Outer Variable num1:  95
Outer Variable num2:  85
Inner Variable num1:  95
Inner Variable num2:  185
Inner Variable num3:  195

PL/SQL procedure successfully completed
```

Here, num2 is declared as a global and a local variable. Inside the sub block, num2 is local variable and outside the block it would take the value of the global variable.

%TYPE

- **%TYPE** can be used to declare a variable with a data type that directly maps to the data type of a
  - column of a table or
  - another variable

- **%TYPE** is mainly useful when declaring variables that will hold database values. When the data type of a table column changes, all variables mapped to that column type will automatically change.

Syntax:

```
variable_name TABLE_NAME.COLUMN_NAME%TYPE;
```

%TYPE Example

- In the given example, Employee is the table name and ename is the column name. If ename is of the type varchar2(30) then the variable v_maiden_name will also have the same data type.

- Similarly the variable v_books_sold will always have the same data type as that of the variable v_books_printed.

```
v_maiden_name      Employee.ename%TYPE ;

v_books_printed    NUMBER(6);
v_books_sold       v_books_printed%TYPE;
```

- The advantages of using %TYPE is that there is no need to know the exact data type of the column **ename** and if the data type of ename changes, the variable type will also change automatically.

%ROWTYPE

- %ROWTYPE attribute provides a record type that represents a row in the table or a record of the query output. It maps the variable to all columns in the table.
- When any of the table column data type is modified, the change will be reflected in the structure of the variable the next time it is run or compiled.
- Table columns in a row and corresponding fields in the record type will have the same name and data type.

Syntax:

```
variable_name TABLE_NAME%ROWTYPE;
```

%ROWTYPE Example

In the below example, Employee is a table with below fields

• empid	number(10)
• empname	varchar2(30)
• salary	number(7)

v_emp_row is a record type variable which has the same column definition of Employee record. The fields of the variable would be accessed as shown below

```
v_emp_row Employee%ROWTYPE;

v_emp_row.empid
v_emp_row.empname
v_emp_row.salary
```

%ROWTYPE Example

In the below example, Employee is a table with below fields

empid	number(10)
empname	varchar2(30)
salary	number(7)

v_emp_row is a record type variable which has the same column definition of Employee record. The fields of the variable would be accessed as shown below

```
v_emp_row Employee%ROWTYPE;

v_emp_row.empid
v_emp_row.empname
v_emp_row.salary
```

TATA CONSULTANCY SERVICES

TATA

Thank You