
ORACLE
Academy

Database Programming with PL/SQL

2-4

Using Scalar Data Types

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Declare and use scalar data types in PL/SQL
  - Define guidelines for declaring and initializing PL/SQL variables
  - Identify the benefits of anchoring data types with the %TYPE attribute

Purpose

- Most of the variables you define and use in PL/SQL have scalar data types
- A variable can have an explicit data type, such as VARCHAR2, or it can automatically have the same data type as a table column in the database
- You will learn the benefits of basing some variables on table columns

Declaring Character Variables

- All variables must be declared
- The data itself will determine what data type you assign to each variable
- Commonly used character data types include CHAR and VARCHAR2
- Columns that may exceed the 32,767 character limit of a VARCHAR2 could be defined using LONG, but should be defined using CLOB

```
DECLARE
 v_country_id          CHAR(2);
 v_country_name        VARCHAR2(70);
 v_country_rpt         CLOB;
```

ORACLE Academy

The internationally recognized two-character country abbreviations (ex. BR, CA, EG, IN, RO, UK, etc.) can be defined using CHAR(2) since they are all two-characters in length.

Since the length of country names may vary, that variable should be defined using VARCHAR2.

Although the v_country_report could be defined as a LONG data type, the CLOB data type was introduced in the Oracle8i database and should now be used for documents that use the database character set exclusively.

Declaring Number Variables

- Number data types include NUMBER, INTEGER, PLS_INTEGER, BINARY_FLOAT and several others
- Adding the keyword CONSTANT constrains the variable so that its value cannot change
- Constants must be initialized

```
DECLARE
 v_employee_id    NUMBER(6,0);
 v_loop_count        INTEGER := 0;
 c_tax_rate       CONSTANT NUMBER(3,2) := 8.25;
...
```

ORACLE Academy

Data types such as BINARY_FLOAT, BINARY_DOUBLE, PLS_INTEGER, and BINARY_INTEGER are used primarily for high-speed scientific computation, as they are faster in calculations.

PLS_INTEGER and BINARY_INTEGER are PL/SQL-only data types that are more efficient than NUMBER or INTEGER for integer calculations.

Declaring Date Variables

- Date data types include DATE, TIMESTAMP, and TIMESTAMP WITH TIMEZONE

```
DECLARE
 v_date1    DATE := '05-Apr-2015';
 v_date2    DATE := v_date1 + 7;
 v_date3    TIMESTAMP := SYSDATE;
 v_date4       TIMESTAMP WITH TIME ZONE := SYSDATE;
BEGIN
 DBMS_OUTPUT.PUT_LINE(v_date1);
 DBMS_OUTPUT.PUT_LINE(v_date2);
 DBMS_OUTPUT.PUT_LINE(v_date3);
 DBMS_OUTPUT.PUT_LINE(v_date4);
END;
```

- Choosing between DATE, TIMESTAMP, etc., is determined by what data you need to know in the future

Declaring BOOLEAN Variables

- BOOLEAN is a data type that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL

```
DECLARE
 v_valid1          BOOLEAN := TRUE;
 v_valid2          BOOLEAN;
 v_valid3          BOOLEAN NOT NULL := FALSE;
BEGIN
 IF v_valid1 THEN
  DBMS_OUTPUT.PUT_LINE('Test is TRUE');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Test is FALSE');
 END IF;
END;
```


A BOOLEAN variable is NULL by default and may remain NULL (unless the NOT NULL constraint is included, in which case the variable must be initialized to either TRUE or FALSE).

Notice that when a BOOLEAN variable is initialized, the value is NOT a string and is not enclosed in single quotes.

Using BOOLEAN Variables

- When using BOOLEAN variables:
  - Only the values TRUE, FALSE, and NULL can be assigned to a BOOLEAN variable
  - Conditional expressions use the logical operators AND and OR, and the operator NOT to check the variable values
  - The variables always yield TRUE, FALSE, or NULL
  - You can use arithmetic, character, and date expressions to return a BOOLEAN value

Guidelines for Declaring PL/SQL Variables

- Use meaningful and appropriate variable names
- Follow naming conventions
- Use v_name to represent a variable and c_name to represent a constant
- Declare one identifier per line for better readability, code maintenance, and easier commenting
- Use the NOT NULL constraint when the variable must hold a value
- Use the CONSTANT constraint when the variable value should not change within the block

NUMBER variables are typically initialized to "0" so as to avoid calculations using a NULL value.

BOOLEAN variables are typically initialized to FALSE and tested for TRUE.

Guidelines for Declaring PL/SQL Variables

- Set initial values for BOOLEANs and NUMBERs
- Avoid using column names as identifiers

```
DECLARE
 first_name VARCHAR2(20);
BEGIN
 SELECT first_name
  INTO first_name
  FROM employees
  WHERE last_name = 'Vargas';
 DBMS_OUTPUT.PUT_LINE(first_name);
END;
```


In this example, the code works because the Oracle server knows the identifier following the SELECT keyword must be a column name and the identifier following the INTO keyword must be a variable name. Likewise, the argument for the PUT_LINE function also must be a variable name.

Although this code works, and is simple enough that we know what is happening, more complex code could be confusing to maintain or modify if variable names are identical to column names.

Defining Variables with the %TYPE Attribute

- Variables derived from database fields should be defined using the %TYPE attribute, which has several advantages
- For example, in the EMPLOYEES table, the column first_name is defined as VARCHAR2(20)
- In a PL/SQL block, you could define a matching variable with either:

```
v_first_name  VARCHAR2(20);
```

- Or

```
v_first_name  employees.last_name%TYPE;
```


The %TYPE reference is resolved at the time the code is compiled. It also establishes a dependency between the code and the table column referenced. This means that if the table column is changed, the code that references it is marked invalid. No changes are required to the invalid code, but the invalid code must be recompiled to update the reference.

Defining a data type using the %TYPE attribute is referred to as an "anchored data type."

Using the %TYPE Attribute

- Look at this partial table definition from the EMPLOYEES table
- Then look at the code in the next slide

Column Name	Data Type
EMPLOYEE_ID	NUMBER(6,0)
FIRST_NAME	VARCHAR2(20)
LAST_NAME	VARCHAR2(25)
EMAIL	VARCHAR2(25)

Using the %TYPE Attribute

- This PL/SQL block stores the correct first name in the v_first_name variable
- But what if the table column is later altered to be VARCHAR2(25) and a name longer than 20 characters is added?

```
DECLARE
 v_first_name    VARCHAR2(20);
BEGIN
 SELECT first_name
  INTO v_first_name
  FROM employees
  WHERE last_name = 'Vargas';
 DBMS_OUTPUT.PUT_LINE(v_first_name);
END;
```

ORACLE Academy

Using the %TYPE attribute to define an "anchored data type" for v_first_name would solve the problem. Otherwise, a programmer would have to find and modify every place in every program with a variable defined to hold an employee's first name.

With the %TYPE attribute, the code would look like this:

```
DECLARE
 v_first_name   employees.first_name%TYPE;
BEGIN
 SELECT first_name
  INTO v_first_name
  FROM employees
  WHERE last_name = 'Vargas';
 DBMS_OUTPUT.PUT_LINE(v_first_name);
```

```
END;
```

Using the %TYPE Attribute

- Using the %TYPE attribute to define an "anchored data type" for v_first_name would solve the problem
- Otherwise, a programmer would have to find and modify every place in every program with a variable defined to hold an employee's first name

```
DECLARE
  v_first_name         employees.first_name%TYPE;
BEGIN
  SELECT first_name
    INTO v_first_name
    FROM employees
    WHERE last_name = 'Vargas';
  DBMS_OUTPUT.PUT_LINE(v_first_name);
END;
```

Using the %TYPE Attribute

- The %TYPE attribute:
  - Is used to automatically give a variable the same data type and size as:
    - A database column
    - Another declared variable
  - Is prefixed with either of the following:
    - The database table name and column name
    - The name of the other declared variable

Using the %TYPE Attribute

- Syntax:

```
identifier    table_name.column_name%TYPE;
identifier    identifier%TYPE;
```

- Examples:

```
DECLARE
 v_first_name    employees.first_name%TYPE;
 v_salary        employess.salary%TYPE;
 v_old_salary    v_salary%TYPE;
 v_new_salary    v_salary%TYPE;
 v_balance          NUMBER(10,2);
 v_min_balance    v_balance%TYPE := 1000;
...
```

ORACLE Academy

In the examples,

v_first_name will have the same data type as the column first_name in the employees table.

v_salary will have the same data type as the column salary in the employees table.

v_old_salary will have the same data type as the variable v_salary.

v_new_salary will have the same data type as the variable v_salary.

v_min_balance will have the same data type as the variable v_balance.

A NOT NULL database column constraint does not apply to variables that are declared using %TYPE. Therefore, if you declare a variable using the %TYPE attribute that uses a database column defined as NOT NULL, that database constraint does NOT pass to the variable.

Advantages of the %TYPE Attribute

- Advantages of the %TYPE attribute are:
  - You can avoid errors caused by data type mismatch or wrong precision
  - You need not change the variable declaration if the table column definition changes
  - Otherwise, if you have already declared some variables for a particular table column without using the %TYPE attribute, then the PL/SQL block can return errors if the table column is altered

Advantages of the %TYPE Attribute

- Advantages of the %TYPE attribute are:
  - When you use the %TYPE attribute, PL/SQL determines the data type and size of the variable when the block is compiled
  - This ensures that such a variable is always compatible with the column that is used to populate it

Terminology

- Key terms used in this lesson included:
  - %TYPE
  - BOOLEAN

- %TYPE – Attribute used to declare a variable according to another previously declared variable or table column.
- BOOLEAN – A datatype that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.

Summary

- In this lesson, you should have learned how to:
  - Declare and use scalar data types in PL/SQL
  - Define guidelines for declaring and initializing PL/SQL variables
  - Identify the benefits of anchoring data types with the %TYPE attribute

ORACLE

Academy