Database Programming with PL/SQL

2-4
Using Scalar Data Types
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;
  ...
```
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.

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

Date data types include **DATE, TIMESTAMP, and**
**TIMESTAMP WITH TIME ZONE.**

```sql
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;
```
Declaring **BOOLEAN** Variables

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

```plsql
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;
```

This code declares three variables of the **BOOLEAN** type and uses an `IF` statement to print either 'Test is TRUE' or 'Test is FALSE' based on the value of `v_valid1`. The `DBMS_OUTPUT.PUT_LINE` function is used to output the results to the console.
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.
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;
```
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:

```plsql
v_first_name   VARCHAR2(20);

or

v_first_name   employees.last_name%TYPE;
```
Using the \%TYPE Attribute

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

```plsql
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;
```
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;
...```
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
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