Using Scalar Data Types
What Will I Learn?

In this lesson, you will learn to:

• Declare and use scalar data types

• Define guidelines for declaring and initializing PL/SQL variables

• Identify the benefits of anchoring data types with the \%TYPE attribute
Why Learn It?

Most of the variables we will define and use in PL/SQL will 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

Character data types include **CHAR**, **VARCHAR2** and **LONG**.

```sql
DECLARE
    v_emp_job    VARCHAR2(9);
    v_order_no   VARCHAR2(6);
    v_product_id VARCHAR2(10);
    v_rpt_body_part LONG;
    ...  
```
Declaring Number Variables

Number data types include `NUMBER`, `PLS_INTEGER`, `BINARY_INTEGER`, and `BINARY_FLOAT`. In the syntax, `CONSTANT` constrains the variable so that its value cannot change. Note also that constants must be initialized.

`INTEGER` is an alias for `NUMBER(38,0)`.

```sql
DECLARE
    v_dept_total_sal   NUMBER(9,2) := 0;
    v_count_loop      INTEGER := 0;
    c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
    ...
```
Declaring Date Variables

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

```sql
DECLARE
    v_orderdate DATE := SYSDATE + 7;
    v_natl_holiday DATE;
    v_web_sign_on_date TIMESTAMP;
...
```
Tell Me/Show Me

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_valid    BOOLEAN NOT NULL := TRUE;
    v_is_found BOOLEAN := FALSE;
    v_underage BOOLEAN;
...
```
Tell Me/Show Me

Declaring 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`.
- Arithmetic, character, and date expressions can be used to return a Boolean value.
Tell Me/Show Me

Guidelines for Declaring and Initializing PL/SQL Variables

- Use meaningful names and follow naming conventions.
- 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.
- Avoid using column names as identifiers.

```sql
DECLARE
    country_id NUMBER(6);
BEGIN
    SELECT country_id
    INTO country_id
    FROM countries
    WHERE country_name = 'Canada';
END;
```
Tell Me/Show Me

Anchoring Variables with the %TYPE Attribute

Rather than hard-coding the data type and precision of a variable, you can use the %TYPE attribute to declare a variable according to another previously declared variable or database column.

The %TYPE attribute is most often used when the value stored in the variable will be derived from a table in the database.

When you use the %TYPE attribute to declare a variable, you should prefix it with the database table and column name.
%TYPE Attribute

Look at this database table and the PL/SQL block which uses it:

```
CREATE TABLE myemps (  
    emp_name        VARCHAR2(6),
    emp_salary      NUMBER(6,2));

DECLARE  
    v_emp_salary   NUMBER(6,2);  
BEGIN  
    SELECT emp_salary INTO v_emp_salary  
        FROM myemps WHERE emp_name = 'Smith';
END;
```

This PL/SQL block will store the correct salary in the v_emp_salary variable. But what if the table column is altered later?
**Tell Me/Show Me**

**%TYPE Attribute**

The `%TYPE` attribute:

- Is used to automatically give a variable the same data type and size as:
  - A database column definition
  - Another declared variable
- Is prefixed with:
  - The database table and column, or
  - The name of the other declared variable
Tell Me/Show Me

Declaring Variables with the %TYPE Attribute

Syntax:

```
identifier  table.column_name%TYPE;
```

Examples:

```
... v_emp_lname    employees.last_name%TYPE;
v_balance       NUMBER(7,2);
v_min_balance   v_balance%TYPE := 1000;
... 
```
Advantages of the %TYPE Attribute

- You can avoid errors caused by data type mismatch or wrong precision.

- You need not change the variable declaration if the column definition changes. That is, if you have already declared some variables for a particular table without using the %TYPE attribute, then the PL/SQL block may return errors if the column for which the variable declared is altered.

- 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.
Tell Me/Show Me

%TYPE Attribute

Look again at the database table and the PL/SQL block:

```
CREATE TABLE myemps (  
    emp_name VARCHAR2(6),
    emp_salary NUMBER(6,2));

DECLARE  
    v_emp_salary myemps.emp_salary%TYPE;
BEGIN  
    SELECT emp_salary INTO v_emp_salary
    FROM myemps WHERE emp_name = 'Smith';
END;
```

Now the PL/SQL block will continue to work correctly even if the column data type is altered later.
Tell Me / Show Me

Terminology
Key terms used in this lesson include:

Boolean
%TYPE
Summary

In this lesson, you have learned how to:

• Declare and use scalar data types

• Define guidelines for declaring and initializing PL/SQL variables

• Identify the benefits of anchoring data types with the \%TYPE attribute
Try It/Solve It

This practice covers the following topics:

- Declaring and using scalar data types (character, number, date, and Boolean)
- Defining guidelines for declaring and initializing PL/SQL variables
- Identifying the benefits of anchoring data types with the %TYPE attribute