Database Programming with PL/SQL

Using Variables in PL/SQL

ACADEMY

Objectives

This lesson covers the following objectives:

- List the uses of variables in PL/SQL
- Identify the syntax for variables in PL/SQL
- Declare and initialize variables in PL/SQL
- Assign new values to variables in PL/SQL

Purpose

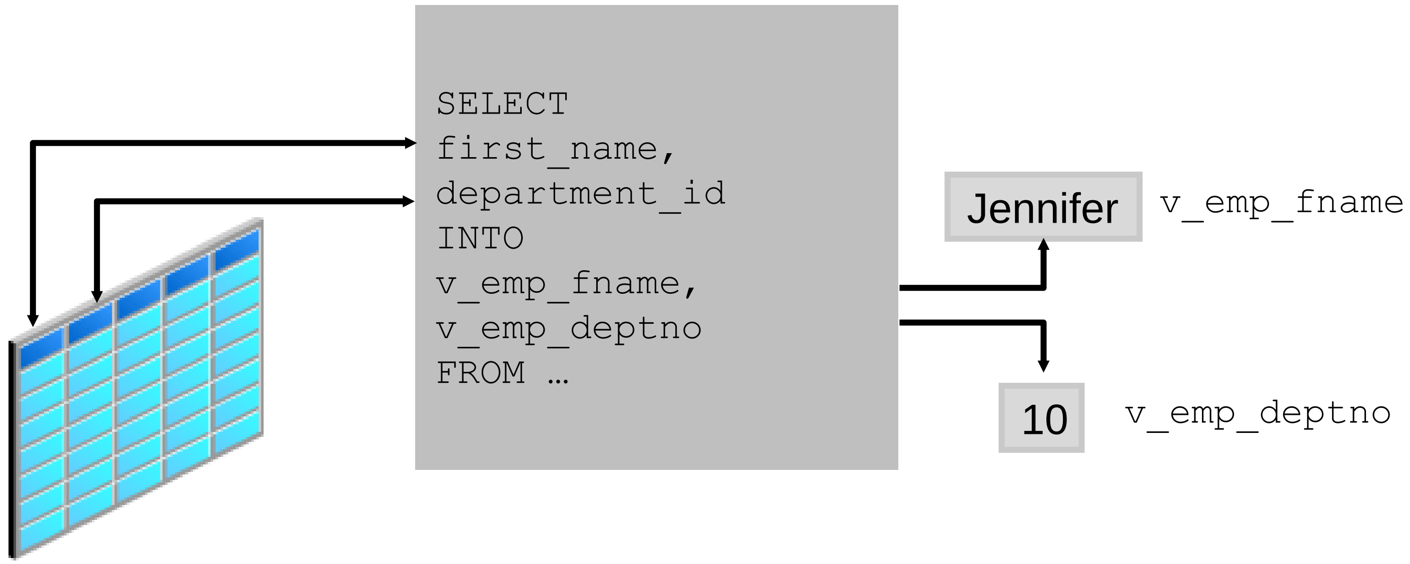
You use variables to store and manipulate data. In this lesson, you learn how to declare and initialize variables in the declarative section of a PL/SQL block. With PL/SQL, you can declare variables and then use them in SQL and procedural statements.

Variables can be thought of as storage containers that hold something until it is needed.

Use of Variables

Use variables for:

- Temporary storage of data
- Manipulation of stored values
- Reusability



```
SELECT
first_name,
department_id
INTO
v_emp_fname,
v_emp_deptno
FROM …
```

Handling Variables in PL/SQL

Variables are:

- Declared and initialized in the declarative section
- Used and assigned new values in the executable section

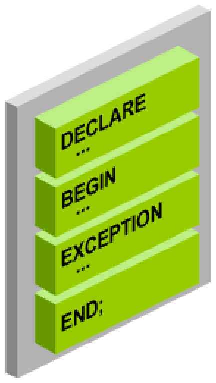
Variables can be:

- Passed as parameters to PL/SQL subprograms
- Assigned to hold the output of a PL/SQL subprogram

Declaring Variables

All PL/SQL variables must be declared in the declaration section before referencing them in the PL/SQL block.

- The purpose of a declaration is to allocate storage space for a value, specify its data type, and name the storage location so that you can reference it.
- You can declare variables in the declarative part of any PL/SQL block, subprogram, or package.



Declaring Variables: Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
      [:= expr | DEFAULT expr];
```

Initializing Variables

Variables are assigned a memory location inside the `DECLARE` section. Variables can be assigned a value at the same time. This process is called initializing.

```
DECLARE
v_counter  INTEGER := 0;
BEGIN
    v_counter := v_counter + 1;
    DBMS_OUTPUT.PUT_LINE(v_counter);
END;
```

Declaring and Initializing Variables Example 1

```
DECLARE
  fam_birthdateDATE;
  fam_size      NUMBER(2) NOT NULL := 10;
  fam_location      VARCHAR2(13) := 'Florida';
  fam_bank      CONSTANT NUMBER := 50000;
  fam_population   INTEGER;
  fam_name      VARCHAR2(20) DEFAULT 'Roberts';
  fam_party_size   CONSTANT PLS_INTEGER := 20;
```

Declaring and Initializing Variables Example 2

```
DECLARE
  v_emp_hiredate   DATE;
  v_emp_deptno     NUMBER(2) NOT NULL := 10;
  v_location       VARCHAR2(13) := 'Atlanta';
  c_comm           CONSTANT NUMBER := 1400;
  v_population     INTEGER;
  v_book_type      VARCHAR2(20) DEFAULT 'fiction';
  v_artist_nameVARCHAR2(50);
  v_firstname      VARCHAR2(20):='Rajiv';
  v_lastname       VARCHAR2(20) DEFAULT 'Kumar';
  c_display_no     CONSTANT PLS_INTEGER := 20;
…
```

Assigning Values in the Executable Section

After a variable is declared, you can use it in the executable section of a PL/SQL block. For example, in the following block, the variable `v_myname` is declared in the declarative section of the block. You can access this variable in the executable section of the same block. What do you think the block will print?

```
DECLARE
  v_myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
  v_myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
END;
```

Assigning Values in the Executable Section Example 1

In this example, the value `John` is assigned to the variable in the executable section. The value of the variable is concatenated with the string `My name is:` .

The output is:

```
My name is:
My name is:  John

Statement process.
```

Assigning Values in the Executable Section Example 2

In this block, the variable `v_myname` is declared and initialized in the declarative section. `v_myname` holds the value `John` after initialization. This value is manipulated in the executable section of the block.

```
DECLARE
  v_myname VARCHAR2(20):= 'John';
BEGIN
  v_myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||v_myname);
END;
```

The output is:

```
My name is:  Steven

Statement processed.
```

Passing Variables as Parameters to PL/SQL Subprograms

Parameters are values passed to a program by the user or by another program to customize the program. In PL/SQL, subprograms can take parameters. You can pass variables as parameters to procedures and functions.

In the following example, the parameter `v_date` is being passed to the procedure `PUT_LINE`, which is part of the package, `DBMS_OUTPUT`.

```
DECLARE
  v_date VARCHAR2(30);
BEGIN
  SELECT TO_CHAR(SYSDATE) INTO v_date FROM dual;
  DBMS_OUTPUT.PUT_LINE(v_date);
END;
```

Assigning Variables to PL/SQL Subprogram Output

You can use variables to hold the value that is returned by a function.

```
--function to return number of characters in string
FUNCTION num_characters (p_string IN VARCHAR2) RETURN INTEGER IS
  v_num_characters INTEGER;
BEGIN
  SELECT LENGTH(p_string) INTO v_num_characters FROM dual;
  RETURN v_num_characters;
END;
```

```
--anonymous block: assign variable to function output
DECLARE
  v_length_of_string INTEGER;
BEGIN
  v_length_of_string := num_characters('Oracle Corporation');
  DBMS_OUTPUT.PUT_LINE(v_length_of_string);
END;
```

Terminology

Key terms used in this lesson included:

- Parameters
- Variables

Summary

In this lesson, you should have learned how to:

- List the uses of variables in PL/SQL
- Identify the syntax for variables in PL/SQL
- Declare and initialize variables in PL/SQL
- Assign new values to variables in PL/SQL