Lab 3. PL/SQL Concepts.

Databases II

3.1. PL/SQL

3.1.1 PL/SQL Block Structure

A block is the most basic unit in PL/SQL. All PL/SQL programs are combined into blocks. These blocks can also be nested within each other. Different tasks within a single program can be separated into blocks.

PL/SQL blocks can be divided into two groups: named and anonymous. Named PL/SQL blocks are used when creating subroutines. These subroutines are procedures, functions, and packages. The subroutines then can be stored in the database and referenced by their names later. In addition, subroutines such as procedures and functions can be defined within the anonymous PL/SQL block. These subroutines exist as long as this block executes and cannot be referenced outside the block. In other words, subroutines defined in one PL/SQL block cannot be called by another PL/SQL block or referenced by their names later. Anonymous PL/SQL blocks do not have names. As a result, they cannot be stored in the database and referenced later.

PL/SQL blocks contain three sections: the declaration section, the executable section, and the exception-handling section. The executable section is the only mandatory section of the block.

The declaration and exception-handling sections are optional. As a result, a PL/SQL block has the following structure:

```
DECLARE
    Declaration statements
 BEGIN
    Executable statements
 EXCEPTION
    Exception-handling statements
END;
```

DECLARATION SECTION

The declaration section is the first section of the PL/SQL block. It contains definitions of PL/SQL identifiers such as variables, constants, cursors, and so on.

Example:

```
DECLARE
   v_first_name VARCHAR2(35);
   v_last_name  VARCHAR2(35);
   c_counter    CONSTANT NUMBER := 0;
```

EXECUTABLE SECTION

The executable section is the next section of the PL/SQL block. This section contains executable statements that allow you to manipulate the variables that have been declared in the declaration section.

Example:

```
BEGIN
   SELECT first_name, last_name
     INTO v_first_name, v_last_name
     FROM student
    WHERE student_id = 123;
   DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '|| v_last_name);
END;
```

EXCEPTION-HANDLING SECTION

The exception-handling section is the last section of the PL/SQL block. This section contains statements that are executed when a runtime error occurs within the block. Runtime errors occur while the program is running and cannot be detected by the PL/SQL compiler. When a runtime error occurs, control is passed to the exception-handling section of the block. The error is then evaluated, and a specific exception is raised or executed.

Example:

```
BEGIN
   SELECT first_name, last_name
     INTO v_first_name, v_last_name
     FROM student
    WHERE student_id = 123;
   DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '|| v_last_name);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('There is no student with '||'student id 123');
END;
```

3.2. PL/SQL in SQLDeveloper

SQLDeveloper is an interactive tool that allows you to type SQL or PL/SQL statements. These statements are then sent to the database. After they are processed, the results are sent back from the database and are displayed on the screen. However, there are some differences between entering SQL and PL/SQL statements.

Consider the following example of a SQL statement:

Example:

```
SELECT first_name, last_name FROM student;
```

The semicolon terminates this SELECT statement. Therefore, as soon as you type the semicolon and press Enter, the result set is displayed.

Now, consider the example of the PL/SQL block:

Example:

```
DECLARE
   v_first_name VARCHAR2(35);
   v_last_name  VARCHAR2(35);
BEGIN
  SELECT first_name, last_name
      INTO v_first_name, v_last_name
      FROM student
   WHERE student_id = 123;
   DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '|| v_last_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('There is no student with '||'student id 123');
END;

.

/
```

Two additional lines at the end of the block contain . and /. The . marks the end of the PL/SQL block and is optional. The / executes the PL/SQL block and is required.

SUBSTITUTION VARIABLES

PL/SQL does not really have capabilities to accept input from a user. However, SQLDeveloper allows a PL/SQL block to receive input information with the help of substitution variables.

Substitution variables cannot be used to output values, because no memory is allocated for them. SQLDeveloper substitutes a variable before the PL/SQL block is sent to the database. Substitution variables usually are prefixed by the ampersand (`&`) or double ampersand (`&&`) characters.

Consider the following example:

Example:

```
DECLARE
   v_student_id NUMBER := &sv_student_id;
   v_first_name VARCHAR2(35);
   v_last_name  VARCHAR2(35);
BEGIN
   SELECT first_name, last_name
     INTO v_first_name, v_last_name
     FROM student
    WHERE student_id = v_student_id;
      DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '|| v_last_name);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('There is no such student');
END;
```

When this example is executed, the user is asked to provide a value for the student ID. The student's name is then retrieved from the STUDENT table if there is a record with the given student ID. If there is no record with the given student ID, the message from the exception-handling section is displayed on the screen. When a single ampersand is used throughout the PL/SQL block, the user is asked to provide a value for each occurrence of the substitution variable.

Consider the following example:

Example:

```
BEGIN
   DBMS_OUTPUT.PUT_LINE ('Today is '||'&sv_day');
   DBMS_OUTPUT.PUT_LINE ('Tomorrow will be '||'&sv_day');
END;
```

Consider an altered version of the example (changes are shown in bold):

Example:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Today is '||'&&sv_day');
  DBMS_OUTPUT.PUT_LINE ('Tomorrow will be '||'&sv_day');
END;
```

When a substitution variable is assigned to the string (text) datatype, it is a good practice to enclose it in single quotes. You cannot always guarantee that a user will provide text informtion in single quotes. This practice will make your program less error-prone. It is illustrated in the following code fragment:

```
v_course_no VARCHAR2(5) := '&sv_course_no';
```

As mentioned earlier, substitution variables usually are prefixed by the ampersand (`&`) or double ampersand (`&&`) characters. These are default characters that denote substitution variables. A special SET command option available in SQLDeveloper allows you to change the default character (`&`) to any other character or disable the substitution variable feature. This SET command has the following syntax:

```
SET DEFINE character/ SET DEFINE ON/ SET DEFINE OFF
```

The first SET command option changes the prefix of the substitution variable from an ampersand to another character. This character cannot be alphanumeric or white space. The second (ON option) and third (OFF option) control whether SQLDeveloper looks for substitution variables. In addition, the ON option changes back the value of the character to the ampersand.

DBMS_OUTPUT.PUT_LINE

DBMS_OUTPUT.PUT_LINE statement displays information on the screen. It is very helpful when you want to see how your PL/SQL block is executed. DBMS_OUTPUT.PUT_LINE writes information to the buffer for storage. Before you can see the output printed on the screen, one of the following statements must be entered before the PL/SQL block:

```
SET SERVEROUTPUT ON;/ SET SERVEROUTPUT ON SIZE 5000;
```

Similarly, if you do not want the DBMS_OUTPUT.PUT_LINE statement to display information on the screen, you can issue the following SET command prior to the PL/SQL block:

```
SET SERVEROUTPUT OFF;
```

Exercises:

1. In this exercise, you calculate the square of a number. The value of the number is provided with the help of a substitution variable. Then the result is displayed on the screen. Create the following PL/SQL script:

```
SET SERVEROUTPUT ON
DECLARE
   v_num    NUMBER := &sv_num;
   v_result NUMBER;
BEGIN
v_result := POWER(v_num, 2);
DBMS_OUTPUT.PUT_LINE ('The value of v_result is: '|| v_result);
END;
```

   Execute the script, what will happen if the value of **`v_num`** is equal to 10?

2. In this exercise, you determine the day of the week based on today's date. You then display the results on the screen. Create the following PL/SQL script:

```
SET SERVEROUTPUT ON
DECLARE
   v_day VARCHAR2(20);
BEGIN
   v_day := TO_CHAR(SYSDATE, 'Day');
  DBMS_OUTPUT.PUT_LINE ('Today is '||v_day);
END;
```

Execute the script, and then answer the following questions:

2.1. What is printed on the screen?
2.2. What is printed on the screen if the statement SET SERVEROUTPUT OFF is issued?
2.3. How would you change the script to display the time of day as well?
2.4. Rewrite the script from exercise 2. In the output produced by the script, extra spaces appear after the day of the week. The new script should remove these extra spaces.

Here's the current output:

```
Today is Sunday   , 20:39
```

The new output should have this format:

```
Today is Sunday, 20:39
```

3. To calculate the area of a circle, you must square the circle's radius and then multiply it by π. Write a program that calculates the area and the perimeter of a circle. The value for the radius should be provided with the help of a substitution variable. Use 3.14 for the value of π. After the area and perimeter of the circle is calculated, display it on the screen.
4. Calculate the minimum, the maximum for two given numbers.
5. Write a program to accept a year and check whether it is leap year or not.
6. Write a program to accept 4 numbers and check whether it is Ramanujan number or not(it is the smallest number expressible as the sum of two cubes in two different ways).
7. Write a program to accept a char and check it is a vowel or consonant.
8. Write a program to accept a number of days and find out the number of years and the number of days and months.