2

Writing Executable Statements

Objectives

After completing this lesson, you should be able to do the following:

- Describe the significance of the executable section
- Use identifiers correctly
- Write statements in the executable section
- Describe the rules of nested blocks
- Execute and test a PL/SQL block
- Use coding conventions

PL/SQL Block Syntax and Guidelines

- Statements can continue over several lines.
- Lexical units can be classified as:
  - Delimiters
  - Identifiers
  - Literals
  - Comments

ORACLE

Identifiers

- Can contain up to 30 characters
- Must begin with an alphabetic character
- Can contain numerals, dollar signs, underscores, and number signs
- Cannot contain characters such as hyphens, slashes, and spaces
- Should not have the same name as a database table column name
- Should not be reserved words

ORACLE

PL/SQL Block Syntax and Guidelines

- **Literals**
  - **Character and date literals must be enclosed in single quotation marks.**

```
v_name := 'Henderson';
```

  - **Numbers can be simple values or scientific notation.**
- **A slash (/) runs the PL/SQL block in a script file or in some tools such as *i*SQL*PLUS.**

Commenting Code

- Prefix single-line comments with two dashes (`--`).
- Place multiple-line comments between the symbols `/*` and `*/`.

Example:

```
DECLARE
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := :g_monthly_sal * 12;
END;     -- This is the end of the block
```

SQL Functions in PL/SQL

- Available in procedural statements:
  - Single-row number
  - Single-row character
  - Data type conversion
  - Date
  - Timestamp
  - GREATEST and LEAST
  - Miscellaneous functions

  } Same as in SQL

- Not available in procedural statements:
  - DECODE
  - Group functions

SQL Functions in PL/SQL: Examples

- Build the mailing list for a company.

```
v_mailing_address := v_name||CHR(10)||
                     v_address||CHR(10)||v_state||
                     CHR(10)||v_zip;
```

- Convert the employee name to lowercase.

```
v_ename       := LOWER(v_ename);
```

Data Type Conversion

- Convert data to comparable data types.
- Mixed data types can result in an error and affect performance.
- Conversion functions:
  - TO_CHAR
  - TO_DATE
  - TO_NUMBER

```
DECLARE
   v_date DATE := TO_DATE('12-JAN-2001', 'DD-MON-YYYY');
BEGIN
   . . .
```

Data Type Conversion

This statement produces a compilation error if the variable `v_date` is declared as a `DATE` data type.

```
v_date := 'January 13, 2001';
```

Data Type Conversion

To correct the error, use the `TO_DATE` conversion function.

```
v_date := TO_DATE ('January 13, 2001',
                   'Month DD, YYYY');
```

Nested Blocks and Variable Scope

- PL/SQL blocks can be nested wherever an executable statement is allowed.
- A nested block becomes a statement.
- An exception section can contain nested blocks.
- The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.

Nested Blocks and Variable Scope

Example:

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
      y:= x;
  END;
  ...
END;
```

Scope of *x*

Scope of *y*

Identifier Scope

An identifier is visible in the regions where you can reference the identifier without having to qualify it:

- A block can look up to the enclosing block.
- A block cannot look down to enclosed blocks.

Qualify an Identifier

- The qualifier can be the label of an enclosing block.
- Qualify an identifier by using the block label prefix.

```
<<outer>>
   DECLARE
      birthdate DATE;
   BEGIN
         DECLARE
             birthdate DATE;
         BEGIN
            ...
            outer.birthdate :=
                              TO_DATE('03-AUG-1976',
                                      'DD-MON-YYYY');
         END;
    ....
    END;
```

Determining Variable Scope

Class Exercise

```
<<outer>>
DECLARE
  v_sal       NUMBER(7,2) := 60000;
  v_comm      NUMBER(7,2) := v_sal * 0.20;
  v_message   VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
       v_sal          NUMBER(7,2) := 50000;
       v_comm         NUMBER(7,2) := 0;
       v_total_comp   NUMBER(7,2) := v_sal + v_comm;
  BEGIN
       v_message := 'CLERK not'||v_message;
       outer.v_comm := v_sal * 0.30;
  END;
    v_message := 'SALESMAN'||v_message;
END;
```

1

2

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

} Same as in SQL

- Exponential operator (**)

Operators in PL/SQL

Examples:

- Increment the counter for a loop.

```
v_count       := v_count + 1;
```

- Set the value of a Boolean flag.

```
v_equal       := (v_n1 = v_n2);
```

- Validate whether an employee number contains a value.

```
v_valid       := (v_empno IS NOT NULL);
```

Programming Guidelines

Make code maintenance easier by:

- Documenting code with comments
- Developing a case convention for the code
- Developing naming conventions for identifiers and other objects
- Enhancing readability by indenting

ORACLE

Indenting Code

For clarity, indent each level of code.

Example:

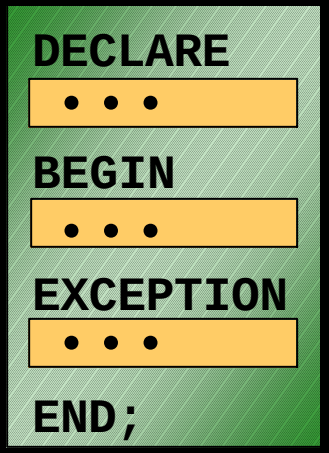
```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
```

```
DECLARE
  v_deptno        NUMBER(4);
  v_location_id   NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    v_deptno,
          v_location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

Summary

In this lesson you should have learned that:

- PL/SQL block syntax and guidelines
- How to use identifiers correctly
- PL/SQL block structure: nesting blocks and scoping rules
- PL/SQL programming:
  - Functions
  - Data type conversions
  - Operators
  - Conventions and guidelines



Practice 2

PL/SQL Block

```
DECLARE
    v_weight      NUMBER(3) := 600;
    v_message   VARCHAR2(255) := 'Product 10012';
BEGIN

      DECLARE
            v_weight     NUMBER(3) := 1;
            v_message   VARCHAR2(255) := 'Product 11001';
            v_new_locn  VARCHAR2(50) := 'Europe';
      BEGIN
            v_weight := v_weight + 1;
            v_new_locn := 'Western ' || v_new_locn;
```

1 →

```
      END;
    v_weight := v_weight + 1;
    v_message := v_message || ' is in stock';
    v_new_locn := 'Western ' || v_new_locn;
```

2 →

```
      END;
     /
```

1. Evaluate the PL/SQL block above and determine the data type and value of each of the following variables according to the rules of scoping.
   a. The value of `V_WEIGHT` at position 1 is:
   b. The value of `V_NEW_LOCN` at position 1 is:
   c. The value of `V_WEIGHT` at position 2 is:
   d. The value of `V_MESSAGE` at position 2 is:
   e. The value of `V_NEW_LOCN` at position 2 is:

Practice 2 (continued)

Scope Example

```
DECLARE
  v_customer          VARCHAR2(50) := 'Womansport';
  v_credit_rating     VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
      v_customer      NUMBER(7) := 201;
      v_name          VARCHAR2(25) := 'Unisports';
    BEGIN
      v_customer      v_name      v_credit_rating
      END;
      v_customer      v_name      v_credit_rating
END;
/
```

2. Suppose you embed a subblock within a block, as shown above. You declare two variables, `V_CUSTOMER` and `V_CREDIT_RATING`, in the main block. You also declare two variables, `V_CUSTOMER` and `V_NAME`, in the subblock. Determine the values and data types for each of the following cases.
   a. The value of `V_CUSTOMER` in the subblock is:
   b. The value of `V_NAME` in the subblock is:
   c. The value of `V_CREDIT_RATING` in the subblock is:
   d. The value of `V_CUSTOMER` in the main block is:
   e. The value of `V_NAME` in the main block is:
   f. The value of `V_CREDIT_RATING` in the main block is:

Practice 2 (continued)

3. Create and execute a PL/SQL block that accepts two numbers through *i*SQL*Plus substitution variables.

a. Use the `DEFINE` command to provide the two values.

```
DEFINE p_num1 = 2
DEFINE p_num2 = 4
```

b. Pass the two values defined in step a above, to the PL/SQL block through *i*SQL*Plus substitution variables. The first number should be divided by the second number and have the second number added to the result. The result should be stored in a PL/SQL variable and printed on the screen.

Note: `SET VERIFY OFF` in the PL/SQL block.

```
4.5
PL/SQL procedure successfully completed.
```

4. Build a PL/SQL block that computes the total compensation for one year.

a. The annual salary and the annual bonus percentage values are defined using the `DEFINE` command.

b. Pass the values defined in the above step to the PL/SQL block through *i*SQL*Plus substitution variables. The bonus must be converted from a whole number to a decimal (for example, 15 to .15). If the salary is `null`, set it to zero before computing the total compensation. Execute the PL/SQL block. *Reminder:* Use the `NVL` function to handle `null` values.

Note: Total compensation is the sum of the annual salary and the annual bonus.

To test the `NVL` function, set the `DEFINE` variable equal to `NULL`.

```
DEFINE p_salary = 50000
DEFINE p_bonus = 10
```

```
PL/SQL procedure successfully completed.
```

G_TOTAL
55000