Stanford
nâng tầm tri thức

9

PL/SQL Programming In Oracle

Oracle for Base

ORACLE

Objectives

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

- Overview of PL/SQL
- Basic Syntax
- Conditions
- Loops
- Procedures
- Functions

Lesson Stanford

- **Overview of PL/SQL**:
  - PL/SQL (**P**rocedural **L**anguage/**S**tructured **Q**uery **L**anguage), the Oracle procedural extension of SQL, is a completely portable, high-performance transaction-processing language.
  - PL/SQL is a combination of SQL a long with the procedural features of programming languages.
  - PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

Lesson Stanford

- **Overview of PL/SQL**:
  - The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are notable facts about PL/SQL:
    - *PL/SQL is a completely portable, high-performance transaction-processing language.*
    - *PL/SQL provides a built-in interpreted and OS independent programming environment.*
    - *PL/SQL can also directly be called from the command-line SQL*Plus interface.*

Lesson Stanford

- **Overview of PL/SQL**:
  - The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are notable facts about PL/SQL:
    - *Direct call can also be made from external programming language calls to database.*
    - *PL/SQL's general syntax is based on that of ADA and Pascal programming language.*
    - *Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.*

Lesson Stanford

- **Advantages of PL/SQL** :
  - SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. Dynamic SQL is SQL allows embedding DDL statements in PL/SQL blocks.
  - PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
  - PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
  - PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.

Lesson Stanford

- **Advantages of PL/SQL** :
  - Applications written in PL/SQL are fully portable.
  - PL/SQL provides high security level.
  - PL/SQL provides access to predefined SQL packages.
  - PL/SQL provides support for Object-Oriented Programming.
  - PL/SQL provides support for Developing Web Applications and Server Pages.

Basic Syntax

S.N	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword EXCEPTION. This section is again optional and contains exception(s) that handle errors in the program.

Basic Syntax

- Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Here is the basic structure of a PL/SQL block:

```
DECLARE
>
BEG<declarations sectionIN
<executable command(s)>
EXCEPTION
<exception handling>
END;
```

Basic Syntax

- **Example:**

```
set serveroutput on

DECLARE
message  varchar2(30):= ‘Stanford – Day lap trinh';
BEGIN
dbms_output.put_line(message);
END;
/
```

Basic Syntax

- **The PL/SQL Comments**

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by PL/SQL compiler. The PL/SQL single-line comments start with the delimiter --(double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
-- variable declaration
message  varchar2(30):= 'Stanford – Day lap trinh';
BEGIN
/*
* PL/SQL executable statement(s)
*/
dbms_output.put_line(message);
END;
```

Basic Syntax

- **PL/SQL Program Units**

A PL/SQL unit is any one of the following:

- PL/SQL block
- Function
- Package
- Package body
- Procedure
- Trigger
- Type
- Type body

Basic Syntax

- **Variable Declaration in PL/SQL**

PL/SQL variables must be declared in the declaration section or in a package as a global variable. When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

The syntax for declaring a variable is:

```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT
initial_value]
```

Basic Syntax

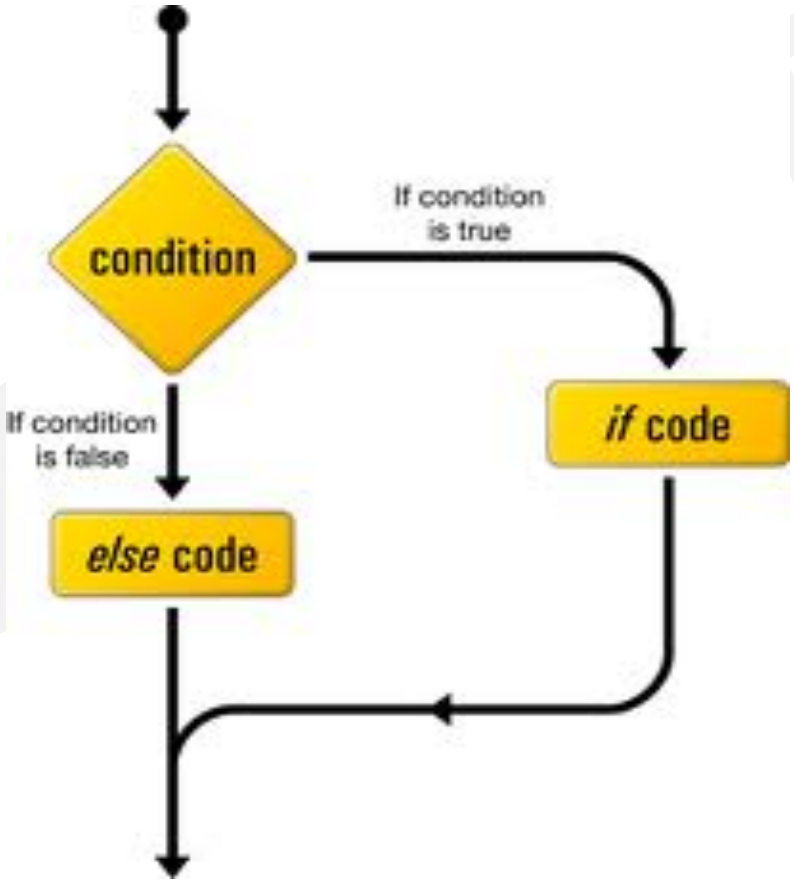
- **Variable Declaration in PL/SQL**

Where, variable_name is a valid identifier in PL/SQL, datatype must be a valid PL/SQL data type or any user defined data type. Some valid variable declarations along with their definition are shown below:

```
sales number(10, 2);
pi CONSTANT double precision := 3.1415;
name varchar2(25);
address varchar2(100);
/
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

Conditions

- **IF - THEN statement**



Conditions

- **IF - THEN statement**

Syntax for IF-THEN statement is

```
IF condition THEN
S;
END IF;
```

Example:

```
IF (a <= 20) THEN
c:= c+1;
END IF;
```

Conditions

- **IF - THEN statement**

Example 1:

```
DECLARE
a number(2) := 10;
BEGIN
a:= 10;
-- check the boolean condition using if statement
IF( a < 20 ) THEN
-- if condition is true then print the following
dbms_output.put_line('a is less than 20 ' );
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```

Conditions

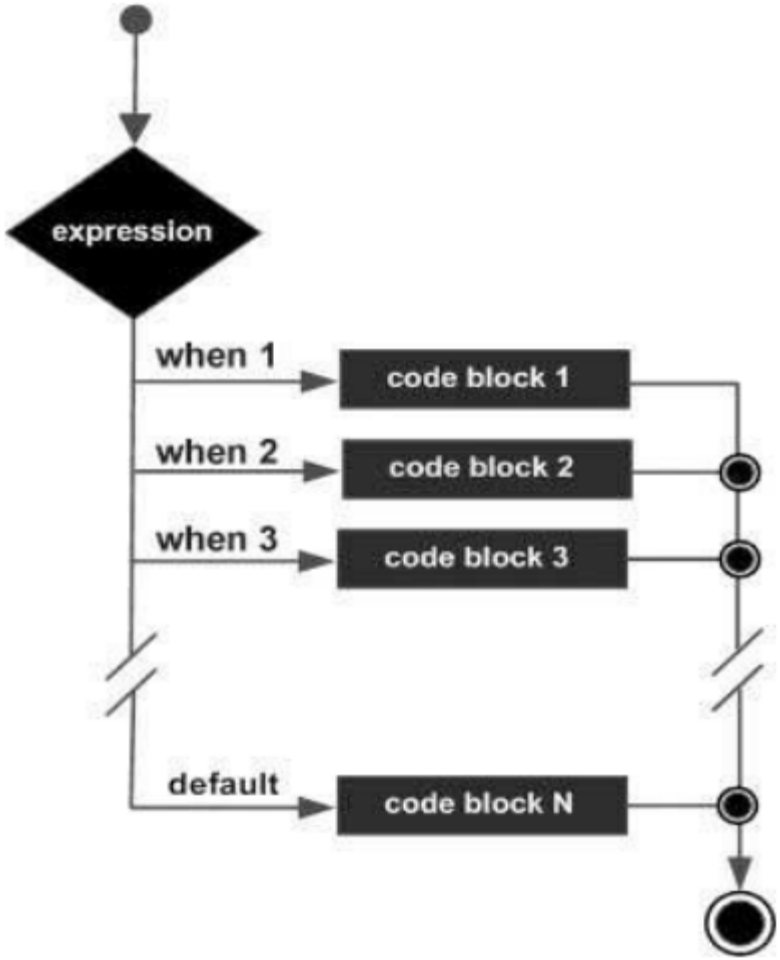
- **IF - THEN statement**

Example 2: Consider we have a table and few records in the table

```
DECLARE
c_id customers.id%type := 1;
c_sal customers.salary%type;
BEGIN
SELECT  salary
INTO  c_sal
FROM customers
WHERE id = c_id;
IF (c_sal <= 2000) THEN
UPDATE customers
SET salary =  salary + 1000
WHERE id = c_id;
dbms_output.put_line ('Salary updated');
END IF;
END;
```

Conditions

- **Case statement**



Conditions

- **Case statement**

The syntax for case statement in PL/SQL is:

```
CASE selector
WHEN 'value1' THEN S1;
WHEN 'value2' THEN S2;
WHEN 'value3' THEN S3;
...
ELSE Sn;  -- default case
END CASE;
```

Conditions

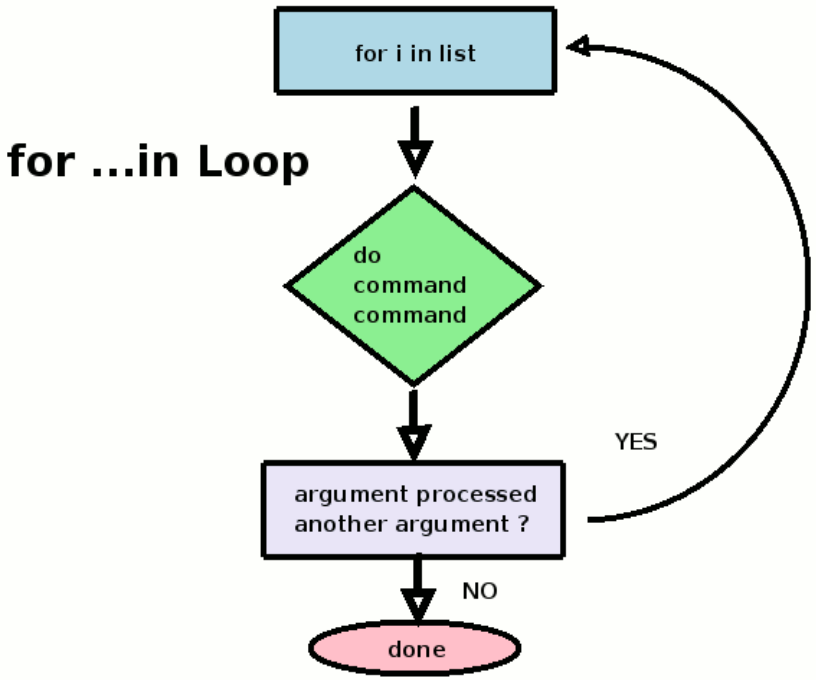
- **Case statement**

Example:

```
DECLARE
grade char(1) := 'A';
BEGIN
CASE grade
when 'A' then dbms_output.put_line('Excellent');
when 'B' then dbms_output.put_line('Very good');
when 'C' then dbms_output.put_line('Well done');
when 'D' then dbms_output.put_line('You passed');
when 'F' then dbms_output.put_line('Better try again');
else dbms_output.put_line('No such grade');
END CASE;
END;
/
```

Loops

- **Loops statement**



Loops

- **Loops statement**

Loop Type	Description
PL/SQL Basic LOOP	In this loop structure, sequence of statements is enclosed between the LOOP and END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.
PL/SQL WHILE LOOP	Repeats a statement or group of statements until a given condition is true. It tests the condition before executing the loop body.
PL/SQL FOR LOOP	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable
Nested loops in PL/SQL	You can use one or more loop inside any another basic loop, while or for loop.

Loops

- **PL/SQL Basic LOOP**

Basic loop structure encloses sequence of statements in between the LOOP and END LOOP statements. With each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

Syntax:

The syntax of a basic loop in PL/SQL programming language is:

```
LOOP
Sequence of statements;
END LOOP;
```

Loops

- **PL/SQL Basic LOOP**

Example:

```
DECLARE
x number := 10;
BEGIN
LOOP
dbms_output.put_line(x);
x := x + 10;
IF x > 50 THEN
exit;
END IF;
END LOOP;
-- after exit, control resumes here
dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

Loops

- **PL/SQL WHILE LOOP**

A WHILE LOOP statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

```
WHILE condition LOOP
sequence_of_statements
END LOOP;
```

Loops

- **PL/SQL WHILE LOOP**

Example:

```
DECLARE
a number(2) := 10;
BEGIN
WHILE a < 20 LOOP
dbms_output.put_line('value of a: ' || a);
a := a + 1;
END LOOP;
END;
/
```

Loops

- **PL/SQL FOR LOOP**

A FOR LOOP is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

```
FOR counter IN initial_value .. final_value LOOP
sequence_of_statements;
END LOOP;
```

Loops

- **PL/SQL FOR LOOP**

Example:

```
DECLARE
      a number(2);
      BEGIN
      FOR a in 10 .. 20 LOOP
      dbms_output.put_line('value of a: ' || a);
      END LOOP;
END;
/
```

Loops

- **Nested loops in PL/SQL**

PL/SQL allows using one loop inside another loop. Following section shows few examples to illustrate the concept.

The syntax for a nested basic LOOP statement in PL/SQL is as follows:

Syntax:

```
LOOP
      Sequence of statements1
      LOOP
      Sequence of statements2
      END LOOP;
END LOOP;
```

Loops

- **Nested loops in PL/SQL**

Example: The following program uses a nested basic loop to find the prime numbers from 2 to 100:

```
DECLARE
      i number(3);
      j number(3);
```

Loops

- **Nested loops in PL/SQL**

Example:

```
BEGIN
      i := 2;
      LOOP
            j:= 2;
            LOOP
                  exit WHEN ((mod(i, j) = 0) or (j = i));
                  j := j +1;
            END LOOP;
      IF (j = i ) THEN
            dbms_output.put_line(i || ' is prime');
      END IF;
      i := i + 1;
      exit WHEN i = 50;
      END LOOP;
END;
```

Procedures

- **Procedures in PL/SQL**

A subprogram is a program unit/module that performs a particular task.

- PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:
  - **Functions**: these subprograms return a single value, mainly used to compute and return a value.
  - **Procedures**: these subprograms do not return a value directly, mainly used to perform an action.

Procedures

- **Procedures in PL/SQL**

Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
< procedure_body >
END procedure_name;
```

Procedures

- **Procedures in PL/SQL**

Example:

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line('Hello World!');
END;
/
```

- **Executing a Standalone Procedure**
  - Using the **EXECUTE** keyword
  - Calling the name of the procedure from a PL/SQL block

  Ex: EXECUTE greetings;

Procedures

- **Procedures in PL/SQL**

```
DROP PROCEDURE procedure-name;
```

```
BEGIN
DROP PROCEDURE greetings;
END;
/
```

Procedures

- **Procedures in PL/SQL**

```
DECLARE
      a number; b number; c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
      IF x < y THEN
      z:= x;
      ELSE
      z:= y;
      END IF;
END;
BEGIN
      a:= 23;
      b:= 45;
      findMin(a, b, c);
      dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
```

Procedures

- **Procedures in PL/SQL**

Example 2:

```
DECLARE
a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
x := x * x;
END;
BEGIN
a:= 23;
squareNum(a);
dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

Functions

- **Functions in PL/SQL**

A PL/SQL function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
< function_body >
END [function_name];
```

Functions

- **Functions in PL/SQL**

Example: The following example illustrates creating and calling a standalone function. This function returns the total number of CUSTOMERS in the customers table.

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
total number(2) := 0;
BEGIN
SELECT count(*) into total
FROM customers;
RETURN total;
END;
/
```

Functions

- **Functions in PL/SQL**

Calling a Function

Example:

```
DECLARE
c number(2);
BEGIN
c := totalCustomers();
dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

Functions

- **Functions in PL/SQL**

Example 2:

```
DECLARE
      a number; b number;
      c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
z number;
BEGIN
IF x > y THEN
      z:= x;
ELSE
      z:= y;
END IF;
RETURN z;
END;
```

Functions

- **Functions in PL/SQL**

Example 2:

```
BEGIN
a:= 23;
b:= 45;
c := findMax(a, b);
dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

THANK YOU !

Stanford – Dạy kinh nghiệm lập trình