Practical 4 – PL/SQL

Go through the given explanation, try out all the examples given and attempt all exercises.

1. Introduction

1.1. What is PL/SQL?

PL/SQL stands for Procedural Language extension of SQL.

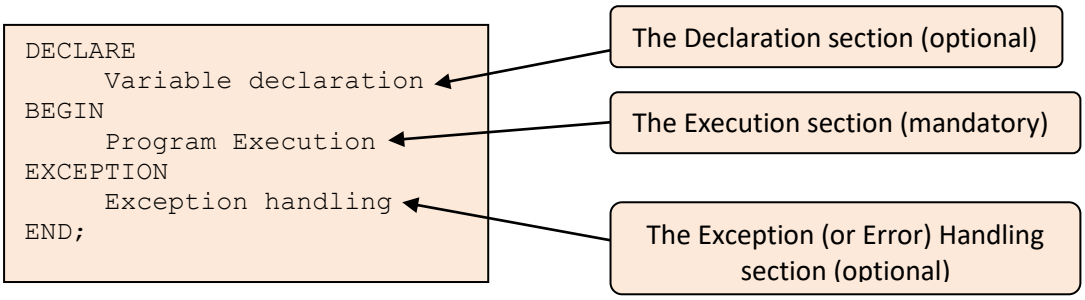
PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

1.2. The PL/SQL Engine

Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).

1.3. A Simple PL/SQL Block

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block. A PL/SQL Block consists of three sections:



1.3.1. Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

1.3.2. Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is

written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

1.3.3. Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

1.4. PL/SQL Statement

Every statement in the above three sections must end with a semicolon '; '. PL/SQL blocks can be nested within other PL/SQL blocks.

1.5. Comments

Comments can be used to document code.

```
DECLARE
   /* Multi-line comments are not required
      to actually use multiple lines.
   */
BEGIN  -- This is a single line comment
  NULL;
END;
```

1.6. Example 01 : Your first PL/SQL Program

```
SQL> DECLARE
  2  BEGIN
  3     DBMS_OUTPUT.PUT_LINE('This is my first PL/SQL Program');
  4  END;
  5  /
```

Try out the above PL/SQL. To see the output of the above PL/SQL, set the server out put on as follows.

```
SQL> set severoutput on
```

1.7. DBMS_OUTPUT package

This is a built-in package, which offers a number of ways to generate output from within your PL/SQL program. This package contain several procedures that a user may use.
e.g. PUT_LINE(), PUT(),GET_LINE(),GET_LINES(),NEW_LINE(), etc..

2. PL/SQL Placeholders

Placeholders are temporary storage area. Placeholders can be any of Variables, Constants and Records. Oracle defines placeholders to store data temporarily, which are used to manipulate data during the execution of a PL SQL block.

2.1. PL/SQL Variables

These are placeholders that store the values that can change through the PL/SQL Block.

The General Syntax to declare a variable is:

```
variable_name datatype [NOT NULL := value ];
```

- variable_name is the name of the variable.
- datatype is a valid PL/SQL datatype.
- NOT NULL is an optional specification on the variable.
- value or DEFAULT value is also an optional specification, where you can initialize a variable.
- Each variable declaration is a separate statement and must be terminated by a semicolon.

e.g.

```
DECLARE
      salary number(4);
      var_emp_id number(6) := 1116;
      dept varchar2(10) NOT NULL := "HR Dept";
```

2.2. Example 02 : Using variables

```
DECLARE
 var_cname varchar(12);
 var_clno  char(3) := 'c01';
BEGIN
   SELECT c.name INTO var_cname
   FROM  client c
   WHERE c.clno = var_clno;

   DBMS_OUTPUT.PUT_LINE('Name of the client with clno : '
                          || var_clno || ' is '  || var_cname  );
END;
/
```

Try the above example

Note:

- Refer the tables you created in practical 1.
- || - two pipe signs are used to concatenate.

2.3. Exercise 01

Write a PL/SQL to get the current price of a stock belongs to company 'IBM'. Store the company name ('IBM') in a variable.

2.4. PL/SQL Scalar Data types

- PL/SQL Number Types

BINARY_DOUBLE, BINARY_FLOAT, BINARY_INTEGER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NATURAL, NATURALN, NUMBER, NUMERIC, PLS_INTEGER, POSITIVE, POSITIVEN, REAL, SIGNTYPE, SMALLINT

- PL/SQL Character and String Types and PL/SQL National Character Types

CHAR, CHARACTER, LONG, LONG RAW, NCHAR, NVARCHAR2, RAW, ROWID, STRING, UROWID, VARCHAR, VARCHAR2

- PL/SQL Boolean Types

BOOLEAN

- PL/SQL Date, Time, and Interval Types

DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, TIMESTAMP WITH LOCAL TIMEZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND

2.5. Scope of Variables

- Local variables - These are declared in a inner block and cannot be referenced by outside Blocks.
- Global variables - These are declared in a outer block and can be referenced by its itself and by its inner blocks.

```
1> DECLARE
2>  var_num1 number;
3>  var_num2 number;
4> BEGIN
5>  var_num1 := 100;
6>  var_num2 := 200;
7>  DECLARE
8>   var_mult number;
9>   BEGIN
10>    var_mult := var_num1 * var_num2;
11>   END;
12> END;
13> /
```

2.6. PL/SQL Constants

The General Syntax to declare a constant is:

```
constant_name CONSTANT datatype := VALUE;
```

- constant_name is the name of the constant i.e. similar to a variable name.
- The word CONSTANT is a reserved word and ensures that the value does not change.
- VALUE - It is a value which must be assigned to a constant when it is declared. You cannot assign a value later.

```
DECLARE
salary_increase CONSTANT number (3) := 10;
```

3. PL/SQL Records

Records are another type of datatypes which oracle allows to be defined as a placeholder. Records are composite datatypes, which means it is a combination of different scalar datatypes like char, varchar, number etc. Each scalar data types in the record holds a value. A record can be visualized as a row of data. It can contain all the contents of a row.

3.1. Declaring a record:

To declare a record, you must first define a composite datatype.

```
DECLARE
TYPE employee_type IS RECORD
(employee_id number(5),
 employee_first_name varchar2(25),
 employee_last_name varchar2(25),
 employee_dept char(3));
```

Then declare a record for that type.

```
employee_rec employee_type;
```

If all the fields of a record are based on the columns of a table, we can declare the record as follows:

```
DECLARE
 employee_rec employee%ROWTYPE;
```

3.2. Example 03 : Accessing a Record

```
DECLARE
   client_rec client%ROWTYPE;
   var_clno client.clno%TYPE := 'c01';
BEGIN
   select * into client_rec
   from client c
   WHERE c.clno = var_clno ;

   DBMS_OUTPUT.PUT_LINE('Client No : ' || client_rec.clno );
   DBMS_OUTPUT.PUT_LINE('Name : ' || client_rec.name );
   DBMS_OUTPUT.PUT_LINE('Address : ' || client_rec.address );
END;
/
```

Try the above example

Note:

`var_clno client.clno%TYPE` – Declare a variable which is of the same type as the clno column in client table.

4. Conditional Statements in PL/SQL

PL/SQL supports programming language features like conditional statements

`IF condition THEN`
` statement 1;`
`ELSE`
` statement 2;`
`END IF;`
	`IF condition 1 THEN`
	`  statement 1;`
	`  statement 2;`
	`ELSIF condition 2 THEN`
	`  statement 3;`
	`ELSE`
	`  statement 4;`
	`END IF;`

`IF condition 1 THEN`
` statement 1;`
` statement 2;`
`ELSIF condtion2 THEN`
	`IF condition1 THEN`
	`  IF condition2 THEN`
	`  statement1;`
	`  END IF;`

```
 statement 3;
ELSE
 statement 4;
END IF;
```

```
ELSIF condition3 THEN
  statement2;
END IF;
```

4.1. Exercise 02

Write a PL/SQL to display a message for a given stock of 'IBM' company according to the following criteria.

Current price < 45 - 'Current price is very low !'
45<=Current price < 55 - 'Current price is low !'
55<=Current price < 65 - 'Current price is medium !'
65<=Current price < 75 - 'Current price is medium high !'
75<=Current price - 'Current price is high !'

Store the company name in a variable.

5. Iterative Statements in PL/SQL

An iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times. These are similar to those in

There are three types of loops in PL/SQL:

- Simple Loop
- While Loop
- For Loop

5.1. Simple Loop

A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

The General Syntax to write a Simple Loop is:

```
LOOP
   statements;
   EXIT;      {or EXIT WHEN condition;}
END LOOP;
```

5.2. While Loop

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

The General Syntax to write a WHILE LOOP is:

```
WHILE <condition> LOOP
      statements;
END LOOP;
```

5.3.FOR Loop

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reaches the value of the end integer.

The General Syntax to write a FOR LOOP is:

```
FOR counter IN [REVERSE] val1..val2 LOOP
      statements;
END LOOP;
```

E.G.

```
for i in 1..1000 loop
    insert into a values(i,i*2);
end loop;
```

```
for i in reverse 1..1000 loop
    insert into a values(i,i*2);
end loop;
```

5.4.Labels

Each of the loops can be labelled. When a loop is labelled, the exit statement can then refer to that label.

```
begin
  <<i_loop>> for i in 1 .. 10 loop
    <<j_loop>> for j in 1 .. 10 loop
      dbms_output.put(to_char(j, '999'));
      exit j_loop when j=i;
    end loop;
    dbms_output.new_line;
  end loop;
end;
/
```

5.5.Exercise 03

Write three PL/SQLs to draw the following shape. Use all three types of loops in each PL/SQL.

```
 9 9 9 9 9 9 9 9 9
 8 8 8 8 8 8 8 8
 7 7 7 7 7 7 7
 6 6 6 6 6 6
 5 5 5 5 5
 4 4 4 4
 3 3 3
 2 2
 1
```

6. Cursors

A cursor can hold a collection of rows retrieved from the database as result of a select statement. It is a temporary work area which is used to store the data retrieved from the database, and to manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are two types of cursors in PL/SQL:

- Implicit cursors:
  These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.
- Explicit cursors:
  They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

6.1.Implicit Cursors

When you execute DML statements like DELETE, INSERT, UPDATE and SELECT statements, implicit cursors are created to process these statements.

Oracle provides few attributes called as cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

For example, when you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected. When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

6.1.1. Example 4 : Implicit Cursor

```
DECLARE
   var_rows number(5);
BEGIN
   UPDATE purchase p
   SET p.qty  = p.qty + 100;

   IF SQL%NOTFOUND THEN
      dbms_output.put_line('None of the quantities were updated');
   ELSIF SQL%FOUND THEN
      var_rows := SQL%ROWCOUNT;
      dbms_output.put_line('Quantities for ' || var_rows ||
                                        ' purchases were updated');
   END IF;
END;
/
```

In the above PL/SQL Block, the quantities of all the purchases in the ‘purchase’ table are updated. If none of the quantities are updated we get a message 'None of the quantities were updated'. Else we get a message like for example, 'Quantities for 10purchases were updated ' if there are 10 rows in ‘purchase’ table.

6.1.2. Example 5 : Implicit Cursor with For Loops

When using FOR LOOP you need not declare a record or variables to store the cursor values, need not open, fetch and close the cursor. These functions are accomplished by the FOR LOOP automatically.

General Syntax for using FOR LOOP:

```
FOR record_name IN cusror_name
LOOP
    process the row...
END LOOP;
```

```
DECLARE
   CURSOR stock_cur IS
      SELECT s.company, s.price
      FROM stock s;
   stock_rec stock_cur%rowtype;
BEGIN
   FOR stock_rec in stock_cur LOOP
      dbms_output.put_line('Company : ' || stock_rec.company
                  ||' - Current Price : ' ||stock_rec.price);
   END LOOP;
END;
/
```

The above PL/SQL Block, retrieve the stock name and current price into a cursor. Access each row at a time and display.

6.1.3. Exercise 04

The companies have decided to offer a bonus to the loyal clients who have purchased stocks of their company. A client who has purchased stocks before 1st January 2002 gets an additional 50 stocks as bonus. Those who have purchased stocks before 1st January 2001 get 100 stocks and for the purchases done before 1st January 2000 will be given 150 stocks as bonus.

Write a PL/SQL block to update the purchase table. Increase the quantities as given above.

You must use a cursor for loop for this.

6.2.Explicit Cursors

An explicit cursor is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

The General Syntax for creating a cursor is as given below:

```
CURSOR cursor_name IS select_statement;
```

- cursor_name – A suitable name for the cursor.
- select_statement – A select query which returns multiple rows.

6.2.1. How to use Explicit Cursor?

There are four steps in using an Explicit Cursor.

- DECLARE the cursor in the declaration section.
- OPEN the cursor in the Execution Section.
- FETCH record from the cursor into PL/SQL variables or records at a time and procee it in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

6.2.2. Declaring a Cursor in the Declaration Section:

```
DECLARE
   CURSOR emp_cur IS
   SELECT *
   FROM emp_tab
   WHERE salary > 5000;
```

6.2.3. Open the cursor.

General Syntax to open a cursor is:

```
OPEN cursor_name;
```

e.g.

```
   OPEN emp_cur;
```

6.2.4. Fetch records from a cursor

General Syntax to fetch records from a cursor is:

```
FETCH cursor_name INTO record_name;
```

OR

```
FETCH cursor_name INTO variable_list;
```

6.2.5. Close a cursor

General Syntax to close a cursor is:

```
CLOSE cursor_name;
```

6.2.6. Example 06: Explicit cursors with simple loops.

```
DECLARE
   CURSOR stock_cur IS
      SELECT s.company, s.price
      FROM stock s;
   stock_rec stock_cur%rowtype;
BEGIN
   IF NOT stock_cur%ISOPEN THEN
      OPEN stock_cur;
   END IF;
   LOOP
      FETCH stock_cur INTO stock_rec;
      EXIT WHEN stock_cur%NOTFOUND;
      dbms_output.put_line('Company : ' || stock_rec.company
                  ||' - Current Price : ' ||stock_rec.price);
   END LOOP;
END;
/
```

The above example is same as example 05. There we are using two cursor attributes %ISOPEN and %NOTFOUND. The attribute %ISOPEN is used to check if the cursor is open, if the condition is true the program does not open the cursor again, it directly moves to next statement.
The cursor attribute %NOTFOUND is used to check whether the fetch returned any row. If there is no rows found the program would exit, a condition which exists when you fetch the cursor after the last row, if there is a row found the program continues. We can use %FOUND in place of %NOTFOUND and vice versa. If we do so, we need to reverse the logic of the program. So use these attributes in appropriate instances.

6.2.7. Exercise 05

Write a PL/SQL block with an explicit cursor to perform the same set of actions mentioned in exercise 04.

You must use an explicit cursor and a while loop.

6.2.8. Cursor Attributes.

Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors. We use these attributes to avoid errors while accessing cursors through OPEN, FETCH and CLOSE Statements.

When does an error occur while accessing an explicit cursor?

- When we try to open a cursor which is not closed in the previous operation.
- When we try to fetch a cursor after the last operation.

These are the attributes available to check the status of an explicit cursor.

Attributes	Return values	Example
%FOUND	TRUE, if fetch statement returns at least one row. FALSE, if fetch statement doesn't return a row.	Cursor_name%FOUND
%NOTFOUND	TRUE, , if fetch statement doesn't return a row. FALSE, if fetch statement returns at least one row.	Cursor_name%NOTFOUND
%ROWCOUNT	The number of rows fetched by the fetch statement If no row is returned, the PL/SQL statement returns an error.	Cursor_name%ROWCOUNT
%ISOPEN	TRUE, if the cursor is already open in the program FALSE, if the cursor is not opened in the program.	Cursor_name%ISNAME