Lab 7

Objectives:

At the end of this lab, you should be able to:
Write simple Procedural Language/SQL (PL/SQL) program.

Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for relational databases.
PL/SQL Block Structure

PL/SQL is a block-structured language, meaning that programs can be divided into logical blocks. A PL/SQL block consists of up to three sections:

- **Declarative (optional):** contains all variables, constants, coursers that are used in the executable section.
- **Executable (required):** contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.
- **Exception handling (optional):** specifies the section to perform when errors and abnormal conditions arise in the executable section.

```
Declarative          DECLARE
                      V       VARCHAR2(5);

                     BEGIN
                      SELECT      column_name
Executable              INTO      v
                        FROM      table_name;

                     EXCEPTION
Exception handling    WHEN exception_name THEN
                      ...
                     END;
```

Declaring PL/SQL Variables

```
Declare
        v_hiredate          DATE;
        v_deptno            NUMBER(2) NOT NULL := 10;
        v_location          VARCHAR2(13) := 'Atlanta';
       c_comm               CONSTANT NUMBER := 1400;
```

Initialize the variable to an expression with the assignment operator (:=) or, equivalently, with the DEFAULT reserved word. If you do not assign an initial value, the new variable contains NULL by default until you assign it later.

The %TYPE Attribute

You can use the %TYPE attribute to declare a variable according to another previously declared variable or database column

```
...
  v_ename                emp.ename%TYPE;
  v_balance              NUMBER(7,2);
  v_min_balance          v_balance%TYPE := 10;
...
```

The %ROWTYPE Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table.
- Fields in the record take their names and datatypes from the columns of the table or view.

For example:

dept_record   dept%ROWTYPE;

To access a field of the variable of dept_record, you use dot.

dept_record.depno=50;

Boolean Variables

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.
- The variables are connected by the logical operators **AND**, **OR**, and **NOT.**

V_sal Boolean :=true;

Writing Executable Statements

Because PL/SQL is an extension of SQL, the general syntax rules that apply to SQL also apply to the PL/SQL language.

1. Commenting Code

Comment the PL/SQL code with two dashes (--) if the comment is on a single line, or enclose the comment between the symbols /* and */ if the comment spans several lines

2. Retrieving Data Using PL/SQL

- Use the SELECT statement to retrieve data from the database with into clause. The INTO clause is mandatory and occurs between the SELECT and FROM clauses.
- You must give one variable for each item selected, and their order must correspond to the items selected.
- Queries Must Return One and Only One Row.

```
DECLARE
  v_deptno   NUMBER(2);
  v_loc      VARCHAR2(15);
BEGIN
  SELECT     deptno, loc
  INTO       v_deptno, v_loc
  FROM       dept
  WHERE      dname = 'SALES';
...
END;
```

To display information from a PL/SQL block you can use *DBMS_OUTPUT.PUT_LINE*. DBMS_OUTPUT is an Oracle-supplied package, and PUT_LINE is a procedure within that package. Within a PL/SQL block, reference DBMS_OUTPUT.PUT_LINE and, in parentheses, the information you want to print to the screen. The package must first be enabled in your SQL*Plus session. To do this, execute the SQL*Plus command *SET SERVEROUTPUT ON.*

Example:

```
SET SERVEROUTPUT ON
DECLARE
 v  NUMBER(6,2) ;
BEGIN
SELECT AVG(SAL) INTO v FROM EMP;

 DBMS_OUTPUT.PUT_LINE ('The average salary is ' || TO_CHAR(v));
END;
```

Manipulating Data Using PL/SQL

You can issue the DML commands INSERT, UPDATE, and DELETE without restriction in PL/SQL. Including COMMIT or ROLLBACK statements

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE     emp
  SET        sal = sal + v_sal_increase
  WHERE      job = 'ANALYST';
END;
```

Note: PL/SQL variable assignments always use := and SQL column assignments always use =.

Writing Control Structures

1. IF Statements

Syntax :

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

For example:
write a program that displays number of employees whose salaries is less than 5000 or display a message " No employee found"

```
Declare
            x  number(3):=0;
Begin
        Select count(*) into x  from emp where sal<5000;
        If x=0 then
                Dbms_output.put_line('No employee found');
        Else
                Dbms_output.put_line(x);
        End if;
End;
```

Exception

```
Declare
            x  number(3):=0;
Begin
        Select deptno into x from dept where dname='IT';
Exception
When no_data_found then
        Dbms_output.put_line(' No such department');
End;
```

PL/SQL - Procedures

A **subprogram** is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

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.

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts:

- **Declarative (optional):** contains all variables, constants, coursers that are used in the executable section.
- **Executable (required):** contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.
- **Exception handling (optional):** specifies the sction to perform when errors and abnormal conditions arise in the executable section.

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

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

No Parameters

```
CREATE OR REPLACE PROCEDURE pr_class IS
      var_code VARCHAR2 (30):= 'CMPS352'
      var_name VARCHAR2 (50) := 'Fundamentals of DB Systems LAB;'
   BEGIN
      DBMS_OUTPUT.PUT_LINE('This course is '||var_name||' which is  '||var_code);
   END pr_class;
```

--Stored Procedure for Department Wide Salary Raise

```
CREATE OR REPLACE PROCEDURE emp_sal( dep_id NUMBER, sal_raise
NUMBER)
IS
BEGIN
  UPDATE EMP SET SAL = SAL * sal_raise WHERE DEPTNO = dep_id;
  DBMS_OUTPUT.PUT_LINE ('salary updated successfully');
END;
```

Testing

```
EXEC emp_sal;(20,0.5)
SELECT * FROM EMP;
```

-- Adjusting Salaries Employees Table

```
--This procedure might need you to disable a trigger in order to work.
CREATE OR REPLACE PROCEDURE adjust_salary(
    in_employee_id IN EMP.EMPNO%TYPE,
    in_percent IN NUMBER
)IS
BEGIN
 --   update employee's salary
   UPDATE emp
   SET sal = sal + sal * in_percent / 100
   WHERE empno = in_employee_id;
END;

--Testing
 --before adjustment
SELECT sal FROM emp WHERE empno= 7788;
 --call procedure
exec adjust_salary(7788,5)
 --after adjustment
SELECT sal FROM emp WHERE empno= 7788;
```

PL/SQL - Functions

A 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];
```

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Example

The following example illustrates how to create and call a standalone function. This function returns the total number of EMPLOYEES in the emp table.

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

Calling a Function

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