PL/SQL Practice

Learning Objectives

After completing this chapter, you should be able to

(i) Create of simple PL/SQL program which includes declaration section, executable section and EXCEPTION handling section.

(ii) Program development using WHILE LOOPS, numeric FOR LOOPS, nested LOOPs using ERROR handling, BUILT IN EXCEPTIONs, USER defined EXCEPTIONs, RAISE APPLICATION ERROR.

Syntax to write a PL/SQL program

```
DECLARE
    <declaration stmts>
BEGIN
   <executable stmts>
[EXCEPTION <EXCEPTIONal stmts>]----- optional
END;
/---END of buffer
```

Example: 1

Create a PL/SQL block to DECLARE and print a variable

Program

```
DECLARE
   v_note        varchar2(50):='HELLO WORLD';
BEGIN
   DBMS_OUTPUT.PUT_LINE(v_note');
END;
```

Loop syntax:

```
LOOP
        <statement(s);>
        Exit when <condition>;
END LOOP;
```

WHILE syntax:

```
WHILE <condition>
LOOP
        <statement(s);>
END LOOP;
```

For syntax:

```
FOR <var> IN <start_num> .. <end_num> LOOP
        <statement(s);>
END LOOP;
```

Example2:

A PL/SQL program using LOOP.

Program

```
DECLARE
   cnt number;
BEGIN
  DBMS_OUTPUT.PUT_LINE('This is a demo of FOR LOOP ');
  for cnt in 1..5 LOOP
     DBMS_OUTPUT.PUT_LINE('LOOP number ' || cnt);
  END LOOP;
END;
```

/OUTPUT:-

```
This ia a demo of FOR
LOOP LOOP number 1
LOOP number 2
LOOP number 3
LOOP number 4
LOOP number 5
```

Program using FOR statement

```
BEGIN
   for cnt in  1..10 LOOP
         DBMS_OUTPUT.PUT_LINE('LOOP counter ' || cnt);
   END LOOP;
END;
/
```

IF STATEMENT:

Other forms of **IF syntax** are:

1)

```
If <condition> then
        <action(s);>
END IF;
```

2)
```
IF <condition> then
        <action(s);>
ELSE
        <action(s);>
END IF;
```

3)
```
IF <condition> THEN
        <action(s);>
ElSIF <condition> THEN
        <action(s);>
ELSE
        <action(s);>
END IF;
```

Program

```
SET SERVEROUTPUT ON

DECLARE
   cnt number(2) := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE('This is a demo of LOOP
       LOOP');
  LOOP
      cnt := cnt + 1;
      exit when cnt > 10;
     DBMS_OUTPUT.PUT_LINE('LOOP counter ' || cnt);
  END LOOP;
END;
/

SET SERVEROUTPUT OFF
```

OUTPUT:-

```
This is the demo of LOOP LOOP
LOOP counter 1
LOOP counter 2
LOOP counter 3
LOOP counter 4
LOOP counter 5
LOOP counter 6
LOOP counter 7
LOOP counter 8
LOOP counter 9
LOOP counter 10
```

Program

```
SET SERVEROUTPUT ON

DECLARE
        cnt number(2) := 1;
BEGIN
   DBMS_OUTPUT.PUT_LINE('This is a demo of WHILE LOOP');
   WHILE cnt <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE('LOOP counter: ' ||  to_char(cnt, '999'));
      cnt := cnt + 1;
  END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

Example: 4

Write a program EMPDATA.SQL, to retrieve the employee details of an employee whose number is input by the user .

Program

```
-- PROGRAM TO RETRIEVE EMP DETAILS
SET SERVEROUTPUT ON

DECLARE
    v_name employees.first_name%type;
    v_sal employees.salary%type;
    v_job employees.job_id%type;
BEGIN
    SELECT first_name, salary, job_id
       INTO v_name, v_sal, v_job
    FROM employees
    WHERE employee_id = &n;
    DBMS_OUTPUT.PUT_LINE('Employee Details:');
    DBMS_OUTPUT.PUT_LINE('Name:  ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Salary:  ' ||  v_sal);
    DBMS_OUTPUT.PUT_LINE('job_id:    ' ||  v_job);
END; /
```

OUTPUT:-

```
SQL> SET SERVEROUTPUT ON

SQL> /

Enter value for n: 100
old   9:       where employee_id = &n;
new   9:       where employee_id = 100;
Employee Details:
Name:   Steven
Salary:  24000
job_id:  AD_PRES
```

Exercises:

2) Write a PL/SQL code, that prints the sum of 'n' natural numbers.

ANSWER:-

```
DECLARE
    isum number(2):=0;
    i number;
    n number:=&n;
BEGIN
   FOR i IN 1..n LOOP
        isum:=isum+i;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('sum is ' || isum);
END;
/
```

OUTPUT:-

```
Enter the number:7
sum is 28
```

Example: 1

Create a file (NEWINS.SQL), to insert into a new table, NEWEMP, the record of any employee whose number is input by the user.

1. Create the table NEWEMP <emp_no, emp_name, join_date, basic).
2. Open an editor and type the following program.

Program

```
DECLARE
    dno number(4);
    dname varchar2(30);
    ddate date;
    dbasic number(10);
BEGIN
     SELECT employee_id, first_name, hire_date, salary
          INTO dno, dname, ddate, dbasic
    FROM employees
    WHERE employee_id = &userno;
     IF sql%ROWCOUNT > 0 THEN
           INSERT INTO newemp
           VALUES (dno, dname, ddate, dbaisc);
    END IF;
END;
          /
```

3. Save the file as NEWINS
4. Execute the program as
   SQL> start newins

Example: 2

Create a file (NEWINS2.SQL), to insert into a new table, NEWEMP, the record of any employee whose number is input by the user. Also display on the screen the employee details and to handle errors like user entering a number which does not exist in the table.

Program

```
DECLARE
   dno number(4);
   dname   varchar2(30);
   ddate   date;
   dbasic  number(10);
BEGIN
   SELECT  employee_id, first_name, hire_date, salary
   INTO    dno, dname, ddate, dbasic
   FROM    employees
   WHERE   employee_id = &userno;
   IF sql%ROWCOUNT > 0 THEN
        INSERT INTO newemp
        VALUES (dno, dname, ddate, dbasic);
        DBMS_OUTPUT.PUT_LINE('Record inserted into NEWEMP');
        DBMS_OUTPUT.PUT_LINE(DNO || ' ' || DNAME || ' ' || DDATE || ' ' || DBASIC);
   END IF;
EXCEPTION
     WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE ('Record ' || &userno || ' does not exist');
END;
/
```

Example: 3

Create a file (CALCTAX.SQL), to calculate tax for a specIFic employee and display name and tax.

Program

```
DECLARE
        tot_basic       number(10, 2);
        tax             number(10, 2);
        name   varchar2(30);
BEGIN
    SELECT first_name, salary
    INTO   name, tot_basic
    FROM employees
    WHERE employee_id = &userno;
    IF tot_basic = 0 or tot_basic IS NULL THEN
        DBMS_OUTPUT.PUT_LINE('NO BASIC');

    ELSIF tot_basic <= 2000 THEN
         tax := tot_basic * .02;
        DBMS_OUTPUT.PUT_LINE(NAME || ' TOTAL BASIC: ' || TOT_BASIC);
        DBMS_OUTPUT.PUT_LINE(NAME || ' TOTAL TAX: ' || TAX);
    ELSE
        tax := tot_basic * .04;
        DBMS_OUTPUT.PUT_LINE(NAME || ' TOTAL BASIC: ' || TOT_BASIC);
        DBMS_OUTPUT.PUT_LINE(NAME || ' TOTAL TAX: ' || TAX);
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE ('Record ' || &userno || ' does not exist');
END;
/
```

EXECPTIONS:

When a program is executed certain errors are automatically recognized and certain error situations must be recognized by the program itself. Errors in general are referred to as Exceptions.

Exceptions can be either System defined or User defined.

Certain system EXCEPTIONs raise the following flags:

DUP_VAL_ON_INDEX – when user tries to insert a duplicate value into a unique column
INVALID_CURSOR – when user references an invalid cursor or attempts an illegal cursor operation
INVALID_NUMBER – when user tries to use something other than a number where one is called for
NO_DATA_FOUND – this flag becomes TRUE when SQL SELECT statement failed to retrieve any rows
TOO_MANY_ROWS – the flag becomes TRUE when SQL SELECT statement retrieves more than one row and it was supposed to retrieve only 1 row
VALUE_ERROR – user encounters an arithmetic, conversion, truncation or constraint error
ZERO_DIVIDE – flag becomes TRUE IF SQL SELECT statement tries to divide a number by 0
OTHERS – this flag is used to catch any error situations not coded by the programmer
In the EXCEPTION section and must appear last in the EXCEPTION section

User defined EXCEPTIONs must be DECLAREd in the DECLARE section with the reserved word, EXCEPTION.

Syntax for user defined EXCEPTION:

```
<EXCEPTION-name> EXCEPTION;
```

This EXCEPTION can be brought into action by the command,

```
RAISE <EXCEPTION-name>
```

When the EXCEPTION is raised, processing control is passed to the EXCEPTION section of the PL/SQL block.
The code for the EXCEPTION must be defined in the EXCEPTION section of the PL/SQL block.

```
WHEN <EXCEPTION-name> THEN
        <action>;
```

Exercises:

1) ***Write a PL/SQL code block that will accept an account number FROM the user and debit an amount of $2000 FROM the account. If the account has a minimum balance of 500 after amount is debited the process should set a freeze on the account by setting the status to F.***
   (use table schema Accounts (acno, balance, status))
2) ***Write a PL/SQL block of code to achieve the following:***
   If the price of a product is >4000 then change the price to 4000. The price change is to be recorded in the old price table along with product number and date on which the price was last changed. (use table schemas Product(pno, price) and Old_Price(pno, date_of_change, oldprice)

Example: 1

Create a PL/SQL program using cursors, to retrieve first record FROM the department relation.
(use the table dept(dno, dname, loc))

Program

```
DECLARE
     vdno  dept.deptno%type;
      vdname dept.dname%type;
      vloc  dept.loc%type;
      CURSOR c1 IS SELECT * FROM dept;
      -- or // cursor c1 is SELECT * FROM dept where rowno = 1;
BEGIN
   OPEN c1;
   FETCH c1  INTO vdno, vdname, vloc;
   DBMS_OUTPUT.PUT_LINE('vdno = ' || vdno || ' vdname = ' || vdname || ' vloc
        =   '|| vloc);
   CLOSE c1;
END;
/
```

PS:

Cursors are used when the SQL SELECT statement is expected to return more than 1 row.

A cursor must be DECLAREd and its definition contains a query and is defined in the DECLARE section of the program.

A cursor must be opened before processing and closed after processing. (Similar to how files are opened and closed in a C program).

Syntax to define a cursor:

CURSOR <CURSOR-NAME> IS <SELECT STATEMENT>

Syntax to open the cursor:

OPEN <CURSOR-NAME>

Syntax to store data in the cursor:

FETCH <CURSOR-NAME> INTO <VAR1>, <VAR2>, <VAR3>….

OR

FETCH <CURSOR-NAME> INTO <RECORD-NAME>

Syntax to close the cursor:

CLOSE <CURSOR-NAME>

Example: 2

Create a PL/SQL program using cursors, to retrieve each tuple FROM the departments relation.

Program

```
DECLARE
      vdept  departments%ROWTYPE;
         cursor  c1 is SELECT * FROM departments;
BEGIN
  FOR vdept IN c1 LOOP
       DBMS_OUTPUT.PUT_LINE ('vdno = ' || vdept.department_id ||
            ' vdname = ' || vdept.department_name || ' vloc = ' || vdept.location_id);
  END LOOP;
END;
/
```

PS:

The cursor for LOOP can be used to process multiple records. The advantage of cursor for LOOP is that the LOOP itself will open the cursor, read the records into the cursor FROM the table until END of file and close the cursor.

Syntax for cursor FOR LOOP:

```
        FOR <VARIABLE> IN <CURSOR-NAME> LOOP
              <STATEMENTs>
END LOOP;
```

Example: 3

Create a PL/SQL program using cursors, to display the number, name, salary of the three highest paid employees.

Program

```
DECLARE
    no employees.employee_id%type;
    name employees.first_name%type;
    salary employees.salary%type;
    CURSOR c1 IS
        SELECT employee_id, first_name, salary
        FROM employees order by salary desc;
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO no, name, salary;
      EXIT WHEN c1 %NOTFOUND;
      EXIT WHEN c1 %ROWCOUNT >3;
      DBMS_OUTPUT.PUT_LINE(no || ' ' || name || ' '||salary);
   END LOOP;
   CLOSE c1;
END;
//
```

PS:

Cursors Attributes:

There are 4 cursor attributes used to provide information on the status of a cursor.

%NOTFOUND – To determine if a row was retrieved
Used after FETCH
NOTFOUND is TRUE if row is not retrieved
NOTFOUND is FALSE if row is retrieved

%FOUND – To determine if a row was retrieved.
Used after FETCH
FOUND is TRUE IF row is retrieved
FOUND is FALSE IF row is not retrieved

%ROWCOUNT – To determine the number of rows retrieved
ROWCOUNT is 0 when cursor is opened
ROWCOUNT returns the number of rows retrieved

%ISOPEN – To determine the cursor is open
ISOPEN is TRUE IF a cursor is open
ISOPEN is FALSE IF a cursor is not open

Example: 4

Create a PL/SQL program using cursors, to delete the employees whose salary is more than 3000.

Program

```
DECLARE
   vrec employees%ROWTYPE;
   CURSOR c1 IS
      SELECT * FROM employees
      WHERE salary>3000 FOR UPDATE;
BEGIN
       OPEN c1;
   LOOP
      FETCH c1 INTO vrec;
      EXIT WHEN c1%NOTFOUND;
      DELETE FROM employees WHERE current of c1;
      DBMS_OUTPUT.PUT_LINE('Record deleted');
   END LOOP;
   CLOSE c1;
END;
/
```

PS:

In order to delete or update rows, the cursor must be defined with the FOR UPDATE clause.

Example: 5

Create a PL/SQL program using cursors, to update the salary of each employee by the avg salary IF their salary is less than avg salary.

Program

```
DECLARE
      vrec employees%ROWTYPE;
        avgsal number(10,2);
cursor c1 is SELECT * FROM employees for update;

BEGIN
        SELECT avg(salary)
        INTO avgsal
        FROM employees;
        FOR vrec IN c1 LOOP
            IF vrec.salary < avgsal THEN
                vrec.salary := avgsal;
                UPDATE employees
                    SET salary = vrec.salary
                WHERE current of c1;
                DBMS_OUTPUT.PUT_LINE('Record updated');
            END IF;
         END LOOP;
END;
/
```

PS:

Variable Attributes:

%TYPE - is used in PL/SQL to DECLARE a variable to be of the same type as a previously DECLAREd variable or to be of the same type as a column in a table.

```
TOTBASIC SALARY.BASIC%TYPE;
```

will DECLARE TOTBASIC of the same type as BASIC column FROM the table SALARY.

%ROWTYPE – DECLAREs a variable which is actually a record which has the same structure as a row FROM a table.

SALREC SALARY%ROWTYPE;
will DECLARE SALREC as a record variable equivalent to the row FROM the table
SALARY.

Example: 6

Create a PL/SQL program using cursors, to insert into a table, NEWEMP, the record of ALL MANAGERS. Also DISPLAY on the screen the NO, NAME, JOIN_DATE. Handle any user defined EXCEPTIONs.
(use table emp(emp_no, emp_name, join_date, desig))

Program

```
DECLARE
        ctr     number(2) := 0;     dno
        number(4);
        dname varchar2(30);
        ddate date;
        CURSOR cur_mgr IS
          SELECT employee_id,first_name, hire_date
          FROM employees
          WHERE UPPER (job_id) LIKE '%MGR';
          no_manager_found EXCEPTION;
BEGIN
   OPEN cur_mgr;
   LOOP
        FETCH cur_mgr
        INTO dno, dname, ddate;
        EXIT WHEN cur_mgr%NOTFOUND;
        ctr := ctr + 1;
        DBMS_OUTPUT.PUT_LINE(ctr || 'Record inserted into NEWEMP');
        DBMS_OUTPUT.PUT_LINE(dno || ' ' || dname || ' ' ||ddate);
        INSERT INTO newemp (emp_no,emp_name,join_date)
        VALUES (dno, dname, ddate);
    END LOOP;
```

```
        IF cur_mgr%ROWCOUNT = 0 THEN
            CLOSE cur_mgr;
            RAISE no_manager_found;
        END IF;
        DBMS_OUTPUT.PUT_LINE('TOTAL number of records' || ctr);
        CLOSE cur_mgr;
EXCEPTION
        WHEN no_manager_found THEN
                DBMS_OUTPUT.PUT_LINE('NO RECORS FOUND');
END;
        /
```