LAB NO: 6 **Date:**

PL/SQL BASICS

Objectives:
In this lab, student will be able to understand and use:

- PL/SQL anonymous block
- PL/SQL Conditional, Iterative and Sequential Control Statements
- Exception Handlers

PL/SQL

PL/SQL is the Oracle procedural extension of SQL. It is a portable, high-performance transaction-processing language.

Though SQL is the natural language of Oracle DB, it has some disadvantages when used as a programming language:

1. SQL does not have any procedural capabilities like condition checking, looping and branching.
2. SQL statements are passed on to the Oracle engine one at a time. This adds to the network traffic in a multi-user environment and decreases data processing speed.
3. SQL has no facility for programmed error handling.

Main Features of PL/SQL

1. PL/SQL combines the data-manipulating power of SQL with the processing power of procedural languages.
2. PL/SQL allows users to declare constants and variables, control program flow, define subprograms, and trap runtime errors.
3. Complex problems can be broken into easily understandable subprograms, which can be reused in multiple applications.
4. PL/SQL sends the entire block of SQL to the Oracle engine in one go, reducing network traffic and leading to faster query processing.
5. Variables in PL/SQL blocks can be used to store intermediate result of a query for later processing

PL/SQL Block Structure: The basic unit of a PL/SQL source program is the block, which groups related declarations and statements. The syntax for which is shown in Figure 7.1

```
<< label >> (optional)
DECLARE    -- Declarative part (optional)
  -- Declarations of local types, variables, & subprograms

BEGIN      -- Executable part (required)
  -- Statements (which can use items declared in declarative part)

[EXCEPTION -- Exception-handling part (optional)
  -- Exception handlers for exceptions (errors) raised in executable part]
END;
```

Fig 7.1 PL/SQL Block Structure

COMPONENTS OF A PL/SQL BLOCK

1. DECLARE

An optional declaration part in which variables, constants, cursors, and exceptions are defined and possibly initialised.

2. BEGIN …… END;

A mandatory executable part consists of a set of SQL and PL/SQL statements. Data manipulation, retrieval, looping and branching constructs are specified in this section.

3. EXCEPTION

An optional exception part deals with handling of errors that arise during execution of data manipulation statements in the PL/SQL code block. Errors can arise due to syntax, logic and/or validation rule violation.

Example: A PL/SQL block to display 'Hello, World'.

```
declare
       message varchar2(20):='Hello, World!';
begin
       dbms_output.put_line(message);
end;
/
```

'SET SERVEROUTPUT ON' command should be issued before executing the PL/SQL block. Alternatively, it can be included in the beginning of every PL/SQL block.

SQL DATA TYPES

The default data types that can be declare in PL/SQL are

1. Number
2. Char
3. Date
4. Boolean

Null values are allowed for Number, Char and Date but not Boolean data type.
E.g.: sname varchar2(30);

%TYPE

PL/SQL can use %Type to declare variables based on column definition in a table. Hence, if a column's attribute changes, the variable's attribute will change as well.

E.g.: sname student.name%TYPE;

%ROWTYPE

The %ROWTYPE attribute lets you declare a record that represents a row in a table or view. For each column in the referenced table or view, the record has a field with the same name and data type. To reference field in the record use *record_name*.*field_name*. The record fields do not inherit the constraints or default values of the corresponding columns. If the referenced item table or view changes, your declaration is automatically updated. You need not change your code when, for example, columns are added or dropped from the table or view.

E.g., srecord student%ROWTYPE;

Not Null: 'Not null' causes creation of a variable or a constant that cannot be assigned 'null' value. Attempt to assign null value to such a variable or constant will return an internal error.

VARIABLES: In PL/SQL a variable name must begin with a character with maximum length of 30. Space not allowed in variable names. Reserve words cannot be used as variable names unless enclosed within double quotes. Case is insignificant when declaring variables.

Values can be assigned to variables by:

1. Using the assignment operator **:=** (a colon followed by an 'equal to').
2. Selecting or fetching table data values INTO variables.

Constants can be declared with constant keyword.

E.g.: pi constant number **:=** 3.141592654;

DISPLAYING USER MESSAGES ON SCREEN

dbms_output is a package that includes procedures and functions that accumulate information in a buffer so that it can be retrieved later.

put_line is a procedure in dbms_output package used to display information in the buffer. SERVEROUTPUT should be set to ON before calling this procedure.

COMMENTS

Comment can be of two forms, as:

1. The comment line begins with a double hypen (--). The entire line is considered as a comment.
2. The comment line begins with a slash followed by an asterisk (/*****) and ends with asterisk followed by slash (*****/). All lines within is considered as comments.

CONDITIONAL CONTROL: IF-THEN-ELSIF-ELSE-END IF

Syntax:

```
IF < condition> THEN
       < action >

ELSIF <condition> THEN
       < action >

ELSE
```

```
       < action >
END IF;
```

Example: A PL/SQL block to display the grade for given letter grade.

```
DECLARE
       grade CHAR(1);
BEGIN
       grade := '&g';
IF grade = 'A' THEN
       DBMS_OUTPUT.PUT_LINE('Excellent');
ELSIF grade = 'B' THEN
       DBMS_OUTPUT.PUT_LINE('Very Good');
ELSIF grade = 'C' THEN
       DBMS_OUTPUT.PUT_LINE('Good');
ELSIF grade = 'D' THEN
       DBMS_OUTPUT. PUT_LINE('Fair');
ELSIF grade = 'F' THEN
       DBMS_OUTPUT.PUT_LINE('Poor');
ELSE
       DBMS_OUTPUT.PUT_LINE('No such grade');
END IF;
END;
/
```

ITERATIVE CONTROL: Simple Loop

Syntax:

```
LOOP
       <Sequence of statements>
END LOOP;
```

Once a loop begins to execute, it will go on forever. A conditional statement to control the number of times loop executes should accompany the simple loop construct.

Example: The following PL/SQL block traces the control flow in a simple LOOP construct.

```
DECLARE
       x NUMBER := 0;
BEGIN
       LOOP
              DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' ||
                                                  TO_CHAR(x));
              x := x + 1;
              IF x > 3 THEN  EXIT;
              END IF;
       END LOOP;
-- After EXIT, control resumes here
DBMS_OUTPUT.PUT_LINE(' After loop: x = ' || TO_CHAR(x));
END;
```

Output:

```
Inside loop: x = 0
Inside loop: x = 1
Inside loop: x = 2
Inside loop: x = 3
After loop: x = 4
```

ITERATIVE CONTROL: While Loop

Syntax:

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

Example: The following PL/SQL block traces the control flow in a while LOOP.

```
DECLARE
       x NUMBER := 0;
BEGIN
       WHILE x < 4
         LOOP
              DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' ||
                                                    TO_CHAR(x));
              X := x + 1;
         END LOOP
END;
/
```

ITERATIVE CONTROL: For Loop

Syntax:

```
FOR index IN [ REVERSE ] lower_bound..upper_bound LOOP
statements
END LOOP ;
```

With REVERSE, the value of index starts at upper_bound and decreases by one with each iteration of the loop until it reaches lower_bound.

Example: The following PL/SQL block traces the control flow in a FOR LOOP

```
BEGIN

       DBMS_OUTPUT.PUT_LINE ('lower_bound < upper_bound');
       FOR i IN 1..3 LOOP
              DBMS_OUTPUT.PUT_LINE (i);
       END LOOP;
       DBMS_OUTPUT.PUT_LINE ('lower_bound = upper_bound');
       FOR i IN 2..2 LOOP
```

```
                DBMS_OUTPUT.PUT_LINE (i);
        END LOOP;

        DBMS_OUTPUT.PUT_LINE ('lower_bound > upper_bound');
        FOR i IN 3..1 LOOP
                DBMS_OUTPUT.PUT_LINE (i);
        END LOOP;
END;
/
```

Output:

```
lower_bound < upper_bound
1
2
3
lower_bound = upper_bound
2
lower_bound > upper_bound
```

SEQUENTIAL CONTROL: GOTO statement

The GOTO statement transfers control to a label unconditionally. The label must be unique in its scope and must precede an executable statement or a PL/SQL block. When run, the GOTO statement transfers control to the labeled statement or block.

Syntax:

```
GOTO Label
The code block is marked using tags
<<Label>>
```

Example: A PL/SQL block to check whether a given number is prime number.

```
DECLARE
        p VARCHAR2(30);
        n PLS_INTEGER := 37;
BEGIN
        FOR j in 2..ROUND(SQRT(n)) LOOP
                IF n MOD j = 0 THEN
                        p := ' is not a prime number';
                        GOTO print_now;
                END IF;
        END LOOP;
        p := ' is a prime number';
<<print_now>>
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
```

```
/
```

ERROR HANDLING:
In PL/SQL, a warning or error condition is called an *exception*. Exceptions can be internally defined (by the run-time system) or user defined. Examples of internally defined exceptions include *division by zero* and *out of memory* etc.

When an error occurs, an exception is *raised*. That is, normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram. Internal exceptions are raised implicitly (automatically) by the run-time system.

To handle raised exceptions, you write separate routines called *exception handlers*. After an exception handler runs, the current block stops executing and the enclosing block resumes with the next statement. If there is no enclosing block, control returns to the host environment.

Syntax:

```
EXCEPTION
WHEN ex_name_1 THEN
       < Exception handler_1>
WHEN ex_name_2 OR ex_name_3 THEN
       < Exception handler_2>
WHEN OTHERS THEN
       < Exception handler_3>
END;
```

Some pre-defined exceptions:

Exception	Raised when ...
DUP_VAL_ON_INDEX	The program attempts to store duplicate values in a database column that is constrained by a unique index.
NO_DATA_FOUND	A SELECT INTO statement returns no rows
PROGRAM_ERROR	PL/SQL has an internal problem.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	The program attempts to divide a number by zero.

Example: Update all accounts with balance less than 0 to 0 in account(account_number, balance) table, populated with {(1, 100); (2, 3000); (3, 500)}

```
DECLARE
BEGIN
       update account set balance = 0 where balance<0
```

```
EXCEPTION

       WHEN NO_DATA_FOUND THEN dbms_output.put_line("No rows
found");

END;
```

User Defined Exceptions:

User can define exceptions of his own in the declarative part of any PL/SQL block, subprogram, or package. For example, an exception named `insufficient_balance` can be defined to flag overdrawn bank accounts. Unlike internal exceptions, user-defined exceptions *must* be given names and they must be raised explicitly by `RAISE` statements.

Example: Validate the resultant balance before supporting withdrawal of money from the account table such that a min. of 500 is maintained.

```
Declare
       Insufficient_Balance Exception;
       Amount account.balance%Type;
       Temp account.balance%Type;
       ANumber account.account_number%Type;
BEGIN
       ANumber := &Number;
       Amount := &Amount;
       Select balance into Temp from account where
                                    account_number = ANumber;
       Temp := Temp – Amount;
       IF (Temp >=500) THEN
       update account set balance = Temp where account_number =
       ANumber;
       ELSE
       RAISE Insufficient_Balanae;
EXCEPTION
       WHEN Insufficient_Balance THEN
              dbms_output.put_line("Insufficient Balance");

WHEN OTHERS THEN
              dbms_output.put_line("ERROR");

END;
```

LAB EXERCISE:

NOTE: Use a table StudentTable(RollNo, GPA) and populate the table with {(1, 5.8); (2, 6.5); (3, 3.4); (4,7.8); (5, 9.5)} unless a different DB schema is explicitly specified.

1. Write a PL/SQL block to display the GPA of given student.

Usage of IF –THEN:

2. Write a PL/SQL block to display the letter grade(0-4: F; 4-5: E; 5-6: D; 6-7: C; 7-8: B; 8-9: A; 9-10: A+} of given student.
3. Input the date of issue and date of return for a book. Calculate and display the fine with the appropriate message using a PL/SQL block. The fine is charged as per the table 8.1:

Late period	Fine
7 days	NIL
8 – 15 days	Rs.1/day
16 - 30 days	Rs. 2/ day
After 30 days	Rs. 5.00

Table 8.1

Simple LOOP:

4. Write a PL/SQL block to print the letter grade of all the students(RollNo: 1 - 5).

Usage of WHILE:

5. Alter StudentTable by appending an additional column LetterGrade Varchar2(2). Then write a PL/SQL block to update the table with letter grade of each student.

Usage of FOR:

6. Write a PL/SQL block to find the student with max. GPA without using aggregate function.

Usage of GOTO:

7. Implement lab exercise 4 using GOTO.

Exception Handling:

8. Based on the University database schema, write a PL/SQL block to display the details of the Instructor whose name is supplied by the user. Use exceptions to show appropriate error message for the following cases:

    a. Multiple instructors with the same name

    b. No instructor for the given name

9. Extend lab exercise 5 to validate the GPA value used to find letter grade. If it is outside the range, 0 – 10, display an error message, 'Out of Range' via an exception handler.

ADDITIONAL EXERCISE

Usage of IF –THEN:

1. Write a PL/SQL block to find out if a year is a leap year.

2. You went to a video store and rented DVD that is due in 3 days from the rental date. Input the rental date, rental month and rental year. Calculate and print the return date, return month, and return year.

Simple LOOP:

3. Write a simple loop such that message is displayed when a loop exceeds a particular value.

4. Write a PL/SQL block to print all odd numbers between 1 and 10.

Usage of WHILE:

5. Write a PL/SQL block to reverse a given string.

Usage of FOR:

6. Write a PL/SQL block of code for inverting a number 5639 or 9365.

Usage of GOTO:

7. Write a PL/SQL block of code to achieve the following: if the price of Product 'p00001' is less than 4000, then change the price to 4000. The Price change has to be recorded in the old_price_table along with Product_no and the date on which the price was last changed. Tables involved:
   Product_master(product_no, sell_price)
   Old_price_table(product_no,date_change, Old_price)

Exception:

8. Write a PL/SQL block that asks the user to input first number, second number and an arithmetic operator (+, -, *, /). If the operator is invalid, throw and handle a user-defined exception. If the second number is zero and the operator is /, handle the ZERO_DIVIDE predefined server exception.