ASSIGNMENT NO.05

TITLE: UNNAMED PL/SQL CODE BLOCK USING CONTROL STRUCTURE AND EXCEPTION HANDLING

PROBLEM STATEMENT: Write a PL/SQL block of code for the following requirements:- Schema:

1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)

Accept roll_no & name of book from user. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5 per day. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. After submitting the book, status will change from I to R. If condition of fine is true, then details will be stored into fine table.

OBJECTIVES:

- To learn how to use PLSQL functions
- To understand working of decision structures and looping actions in PLSQL.
- To manage errors with exception handling

OUTCOMES:

- Students will be able define, declare, initialize and manage variable values.
- Students will be able to write and execute PLSQL unnamed blocks and use control structures.
- Students will be able to manage errors with exception handling

SOFTWARE & HARDWARE REQUIREMENTS:

1. 64-bit Open source Linux or its derivative
2. Oracle Server

THEORY:

PL SQL basically stands for "Procedural Language SQL". This is the extension of Structured Query Language (SQL) that is used in Oracle. PL SQL provides data manipulation power of SQL with processing power of procedural language. It allows the programmers to instruct the compiler 'what to do' through SQL and 'how to do' through its procedural way. It gives more control to the programmers by the use of loops, conditions and object oriented concepts.

PL/SQL program units organize the code into blocks. A block without a name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.

An anonymous block is an only one-time use and useful in certain situations such as creating test units. The following illustrates anonymous block syntax:

```
[DECLARE]
Declaration statements;
BEGIN
Execution statements;
[EXCEPTION]
Exception handling statements;
END;
```

The anonymous block has three basic sections that are the declaration, execution, and exception handling. Only the execution section is mandatory and the others are optional.
The anonymous block has no name associated with it. In fact, the anonymous block is missing the header section altogether. Instead it simply uses the DECLARE reserved word to mark the beginning of its optional declaration section.

- The declaration section allows you to define data types, structures, and variables. You often declare variables in the declaration section by giving those names, data types, and initial values.
- The execution section is required in a block structure and it must have at least one statement. The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section.
- The exception handling section is starting with the EXCEPTION keyword. The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.

Without a name, the anonymous block cannot be called by any other block -- it doesn't have a handle for reference. Instead, anonymous blocks serve as scripts that execute PL/SQL statements, including calls to procedures and functions. Anonymous blocks can also serve as nested blocks inside procedures, functions, and other anonymous blocks.

To display database's output on the screen, you need to:
First, use the SET SERVEROUTPUT ON command to instruct SQL to echo database's output after executing the PL/SQL block.
Second, use the DBMS_OUTPUT.PUT_LINE procedure to output a string on the screen.

The following example displays a message Hello PL/SQL on a screen:

```
SET SERVEROUTPUT ON
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('Hello PL/SQL');
END;
```

Variable Declaration:

Variable Declaration specifies the variable name, data type of variable and size. In PL/SQL variable declaration you can also specifies initial value of declared variables.

The general syntax to declare a variable is:

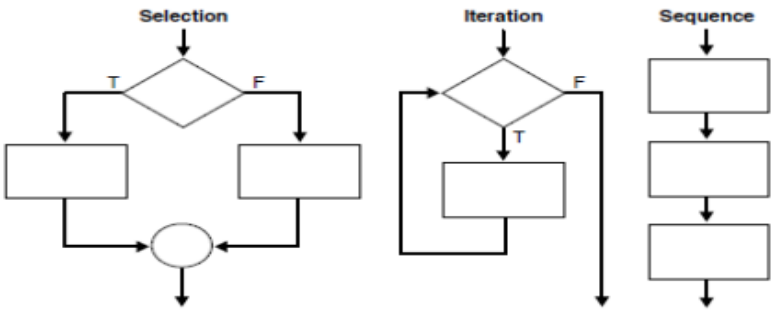
```
variable_name Datatype[Size] [NOT NULL] := [ value ];
```

Following is an example of unnamed PLSQL block for addition of two numbers.

```
DECLARE
        num1 number := 10;
        num2 number := 20;
BEGIN
        DECLARE
                num_addition number;
        BEGIN
                num_addition := num1 + num2;
                dbms_output.put_line('Addition is: ' ||num_addition);
        END;  -- End of access num_addition variable
END;
```

PL/SQL Control Structures

Procedural computer programs use the basic control structures.



- The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).
- The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.
- The sequence structure simply executes a sequence of statements in the order in which they occur.

Conditional Control: IF and CASE Statements: The IF statement lets you execute a sequence of statements conditionally. That is, whether the sequence is executed or not depends on the value of a condition. There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF. The CASE statement is a compact way to evaluate a single condition and choose between many alternative actions.

IF-THEN Statement

```
IF condition THEN
   sequence_of_statements
END IF;
```

IF-THEN-ELSE Statement

```
IF condition THEN
   sequence_of_statements1
ELSE
   sequence_of_statements2
END IF;
```

IF-THEN-ELSIF Statement

```
IF condition1 THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

CASE Statement

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions.

```
[<<label_name>>]
CASE selector
   WHEN expression1 THEN sequence_of_statements1;
   WHEN expression2 THEN sequence_of_statements2;
   ...
   WHEN expressionN THEN sequence_of_statementsN;
  [ELSE sequence_of_statementsN+1;]
END CASE [label_name];
```

Iterative Control: LOOP and EXIT Statements

LOOP statements let you execute a sequence of statements multiple times. There are three forms of LOOP statements: LOOP, WHILE-LOOP, and FOR-LOOP.

LOOP

```
LOOP
   sequence_of_statements
END LOOP;
```

EXIT

The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

Example:

```
LOOP
  ...
  IF credit_rating < 3 THEN
    ...
    EXIT;  -- exit loop immediately
  END IF;
END LOOP;
-- control resumes here
```

EXIT-WHEN

The EXIT-WHEN statement lets a loop complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement after the loop.

Example:

```
LOOP
  FETCH c1 INTO ...
  EXIT WHEN c1%NOTFOUND;  -- exit loop if condition is true
  ...
END LOOP;
CLOSE c1;
```

WHILE-LOOP

The WHILE-LOOP statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP, as follows:

```
WHILE condition LOOP
   sequence of statements
END LOOP;
```

FOR-LOOP

Whereas the number of iterations through a WHILE loop is unknown until the loop completes, the number of iterations through a FOR loop is known before the loop is entered. FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed by the keywords FOR and LOOP. A double dot (..) serves as the range operator.

```
FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
   sequence_of_statements
END LOOP;
```

Sequential Control: GOTO and NULL Statements

Unlike the IF and LOOP statements, the GOTO and NULL statements are not crucial to PL/SQL programming. The structure of PL/SQL is such that the GOTO statement is seldom needed.

GOTO Statement: The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block. In the following example, you go to an executable statement farther down in a sequence of statements:

```
BEGIN
  ...
  GOTO insert_row;
  ...
  <<insert_row>>
  INSERT INTO emp VALUES ...
END;
```

NULL Statement: The NULL statement does nothing other than pass control to the next statement. In a conditional construct, the NULL statement tells readers that a possibility has been considered, but no action is necessary. In the following example, the NULL statement shows that no action is taken for unnamed exceptions:

```
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    ROLLBACK;
  WHEN VALUE_ERROR THEN
    INSERT INTO errors VALUES ...
    COMMIT;
  WHEN OTHERS THEN
    NULL;
END;
```

PL/SQL EXCEPTION

In PL/SQL, any kind of errors is treated as exceptions. An exception is defined as a special condition that change the program execution flow. The PL/SQL provides you with a flexible and powerful way to handle such exceptions.

PL/SQL catches and handles exceptions by using exception handler architecture. Whenever an exception occurs, it is raised. The current PL/SQL block execution halts and control is passed to a separate section called exception section.

In the exception section, you can check what kind of exception has been occurred and handle it appropriately. This exception handler architecture enables separating the business logic and exception handling code hence make the program easier to read and maintain.

There are two types of exceptions:

- System exception: the system exception is raised by PL/SQL run-time when it detect an error. For example, NO_DATA_FOUND exception is raised if you select a non-existing record from the database.
- Programmer-defined exception: the programmer-defined exception is defined by you in a specific application. You can map exception names with specific Oracle errors using the EXCEPTION_INIT pragma. You can also assign a number and description to the exception using RAISE_APPLICATION_ERROR.

Defining PL/SQL Exception

An exception must be defined before it can be raised. Oracle provides many predefined exceptions in the STANDARD package.

```
EXCEPTION_NAME EXCEPTION;
```

To raise exception that you've defined you use the RAISE statement as follows:

```
RAISE EXCEPTION_NAME;
```

CONCLUSION: