Name :- Rushikesh Kaobhaci Palve
Roll NO. 31258

Date : / /20 | Page No.

Assignment NO. 4

DOP :- 08-09-2021 DOS :- 14-09-2021

Title :- Unnamed PL/SQL code block : Use of Control structure and Exception handling is mandatory.

Problem Definition :-

Consider Tables :-

1. Borrower (Roll_No, Name, Date of Issue, Name of Book, Status)
2. Fine (Roll_no, Date, Amt)

- Accept Roll_no and 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 and for days less than 30, Rs 5 per day.
- After submitting the book, status will be changed from I to R.
- If condition of fine is true, the details will be stored into fine table.
- Also handles the exception by named exception handler or user defined exception handler.

Objectives :-

(a) Understand the control structure.
(b) Understand exception handling in PL/SQL.

Outcomes :-

Students will be able to understand the concepts of control structure and exception handling in PL/SQL.

Theory :-

Introduction :- PL/SQL

The development of database applications typically requires language constructs similar to those that can be found in programming languages such as, C, C++ or Pascal. These constructs are necessary in order to implement complex data structures and algorithms. A major restriction of the database language SQL, however, is that many tasks cannot be accomplished by using only the provided language elements.

PL/SQL (Procedural Language/SQL) is a procedural extension of Oracle-SQL that offers language constructs similar to those in imperative programming languages.

Or

A PL/SQL is procedural language extension to the SQL in which you can declare and use the variables, constants, do exception handling and you can also write the program modules in the form of PL/SQL subprograms. PL/SQL combines the feature of a procedural language with structured query language.

PL/SQL allows users and designers to develop complex database applications that require the

usage of control structures and procedural elements such as procedures, functions and modules.

The basic construct in PL/SQL is a block. Block allows designers to combine logically related (SQL-) statements into units. In a block, constants and variables can be declared and variables can be used to store query results. Statement in PL/SQL block include SQL statements, control structures (loops), condition statements (if then-else), exception handling, and calls of other PL/SQL blocks.

PL/SQL blocks that specify procedures and functions can be grouped into packages. A package is similar to a module and has an interface and an implementation part. Oracle offers several predefined packages, for example, input/output routines, file handling, job scheduling etc.

Another important feature of PL/SQL is that it offers a mechanism to process query results in a tuple-oriented way, that is, one tuple at a time. For this, cursors are used. A cursor basically is a pointer to a query result and is used to read attribute values of selected tuples into variables. A cursor typically is used in combination with a loop construct such that each tuple read by the cursor can be processed individually.

In summary, the major goals of PL/SQL are to

- Increase the expressiveness of SQL,
- Process query results in a tuple-oriented way,
- Optimize combined SQL statements,
- Develop modular database application programs,
- Reuse program code,
- Reduce the cost for maintaining & changing applications.

Advantages of PL/SQL :-

Following are some advantages of PL/SQL

1) Support to SQL :- PL/SQL is a procedural language extension to SQL supports all the functionalities of SQL.

2) Improved performance :- In SQL every statement individually goes to the ORACLE server, get processed and then execute. But in PL/SQL an entire block of statements can be sent to ORACLE server at one time, where SQL statements are processed one at time. PL/SQL block statements drastically reduce communication between the application and ORACLE. this helps in improving the performance.

3) Higher Productivity :- Users use procedural features to build applications. PL/SQL code is written in the form of PL/SQL block. PL/SQL blocks can also be used in other ORACLE Forms, ORACLE Reports. This code reusability increases the programmers productivity.

4) Portability :- Applications written in PL/SQL are portable. We can port them from one environment to any computer hardware and operating system environment running ORACLE.

5) Integration with ORACLE :- Both PL/SQL and ORACLE use SQL based. PL/SQL variables have datatypes native to the oracle RDBMS dictionary. This gives tight integration with ORACLE.

Features of PL/SQL :-

1) We can define and use variables and constants in PL/SQL.

2) PL/SQL provides control structures to control the flow of a program. The control structures supported by PL/SQL are If... Then, Loop, for... loop and others.

3) We can do row by row processing of data in PL/SQL. PL/SQL supports row by row processing using the mechanism called cursor.

4) We can handle pre-defined and user-defined error situations. Errors are warnings and called as exceptions in PL/SQL.

5) We can write modular application by using sub programs.

The structure of PL/SQL program :-

The basic unit of code in any PL/SQL program is a block. All PL/SQL programs are composed of blocks. These blocks can be written sequentially.

The structure of PL/SQL block :-

```
DECLARE
    Declaration section
BEGIN
    Executable section
EXCEPTION
    Exception handling section
END;
```

Where,

1) Declaration section

PL/SQL variables, types, cursors and local subprograms are defined here.

2) Executable section

Procedural and SQL statements are written here. This is the main section of the block. this section is required.

3) Exception handling section

Error handling code is written here this section is optional whether it is defined within body or outside body of program.

Conditional statements and loops used in PL/SQL:-

Conditional statements check the validity of a condition and accordingly execute a set of statements. The conditional statements supported by PL/SQL is

1) IF..THEN
2) IF..THEN...ELSE
3) IF..THEN...ELSEIF

1) IF..THEN :-

Syntax :-

```
IF condition THEN
statement list
END IF;
```

2) IF..THEN...ELSE :-

Syntax :-

```
IF condition THEN
    statement list
ELSE
    statements
END IF;
```

3) IF...THEN...ELSEIF :-

Syntax :-

```
IF condition THEN
    statement list
ELSEIF
    statement list
ELSE
    statement list
END IF;

END
```

2) CASE Expression :- CASE expression can also be used to control the branching logic within PL/SQL blocks. The general syntax is

```
CASE
WHEN <expression> THEN <statements>;
WHEN <expression> THEN <statements>;
:
ELSE
<statements>;
END CASE;
```

Here expression in WHEN clause is evaluated sequentially. When result of expression is TRUE, then corresponding set of statements are executed and program flow goes to END CASE.

ITERATIVE Constructs :-

Iterative constructs are used to execute a set of statements repetitively. The iterative constructs supported by PL/SQL are follows :

1) SIMPLE 'LOOP'
2) WHILE LOOP
3) FOR LOOP

1) The SIMPLE 'LOOP':- It is the simplest iterative construct and has syntax like:

```
LOOP
    statements
END LOOP;
```

The loop does not facilitate checking for a condition and so it is an endless loop. To end the iterations, the EXIT statement can be used.

```
LOOP
    <statement list>
        IF condition THEN
        EXIT;
        END IF;
END LOOP;
```

The statements here is executable statements, which will be executed repeatedly until the condition given in IF...THEN evaluates TRUE.

2) THE 'WHILE LOOP':- The WHILE..LOOP is a condition driven construct i.e. the condition is a part of the loop construct and not to be checked separately. The loop is executed as long as the condition evaluates to TRUE.

The syntax is :-

```
WHILE condition LOOP
    statements
END LOOP;
```

The condition is evaluated before each iteration of loop. If it evaluates to TRUE, sequence of statements are executed. If the condition is evaluated to FALSE or NULL, the loop is finished and the control resumes after the END LOOP statement.

3) **The FOR LOOP :-** The number of iterations for LOOP and WHILE LOOP is not known in advance. THE number of iterations depends on the loop condition. The FOR LOOP can be used to have a definite number of iterations.

The syntax is :-

```
FOR loop_counter IN [REVERSE] Low bound..
                                High bound LOOP
    statements;
END loop;
```

Where,

- loop_counter - is the implicitly declared index variable as BINARY_INTEGER.
- low bound and high bound specify the number of iterations.
- statements :- Are the contents of the loop.

EXCEPTIONS :-

Exceptions are errors or warnings in a PL/SQL program. PL/SQL implements error handling using exceptions and exception handlers.

Exceptions are the run time error that a PL/SQL program may encounter.
There are two types of exreptions
1) Predefined exceptions
2) User defined exreptions.

1) Predefined exceptions :- Peedefined exreptions are the error condition that are denned by ORACLE. Predefined exreptions are raised automatically whenever a PL/SQL program violates an ORACLE rule.

2) User Defined Exceptions :-
A user defined exceptions is an error or warning that is defined by the program. User defined exreptions ran be define in the declaration section of PL/SQL block. User defined exreptions are declared in the declarative section of a PL/SQL block. Exreptions have a type Exreption and srope.

Syntax :-

```
DECLARE
        <Exception Name> EXCEPTION;
BEGIN
        ---
        RAISE <Exreption Name>
        ---
EXCEPTION
        WHEN <Exreption Name> THEN
        <Action>
END;
```

Date: / /20 | Page No:

Exception Handling :-

→ A PL/SQL block may contain statements that sperify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception. One can distinguish between two types of exceptions.

- System Defined Exceptions
- User Defined Exceptions (which must be declared by the user in the declaration part of a block where the exception is used/implemented)

System Defined exceptions are always automatically raised whenever corresponding errors or warnings occur. User defined exceptions, in contrast, must be raised explicitly in a sequence of statements using raise <exception name>. After the keyword exception at the end of a block user defined exception handling routines are implemented. An implementation has the pattern

when <exception name> the <sequence of statements>;

Syntax :-

```
<Exception_name> Exception;
```

Handling Exceptions :- Exception handlers for all exceptions are written in the exception handling section of a PL/SQL block.

Syntax :-

```
Exception
    When exception_name then
        sequence_of_statements1;
    When exception_name then
        sequence_of_statement 2;
```

: Date :	Page No :
/ /20	

When exception_name then
sequence_of_statements ;
End ;

Example :-

```
Declare
    emp sal EMP.SAL %TYPE;
    emp no EMP.EMPNO %TYPE;
    too_high_sal exception;
begin
    select EMPNO, SAL into emp no, emp sal
    from EMP where ENAME = "KING";
    if emp sal * 1.05 > 4000 then raise too_high_sal
    else update EMP set SQL...
    end if;
exception
    when NO DATA FOUND -- no tuple selected
    then rollback;
    when too_high_sal then insert into high sal
        values (emp no);
    commit;
end;
```

After the keyword when a list of exception names connected with or can be specified. The last when clause in the exception part may contain the exception name others. This introduces the default exception handling routine, for example, a rollback. If a PL/SQL program is executed from the SQL* Plus shell, Exception handling routines may contain statements that display error or warning messages on the screen. For this, the procedure raise application error can be used. The procedure has two parameters

<error number> and <message text>. <error number> is a negative integer defined by the user and must range between -20000 and -20999.
<error message> is a string with a length up to 2048 characters.

The concatenation operator "||" can be used to concatenate single strings to one string. In order to display numeric variables, these variables must be converted to strings using the function to_char.
If the procedure raise application error is called from a PL/SQL block, processing the PL/SQL block terminates ~~all~~ and all database modifications are undone, that is, an implicit rollback is performed in addition to displaying the error message.

Example :-

```
if emp sal * 1.05 > 4000
then raise application error (-20010, "salary
increase for employee with ID " || to char
(EMP no) || ' is too high");
```

E.g.

```
Declare
    V_maxno number(2) := 20;
    V_rucno number(2);
    E_too_many_emp exception;
Begin
    select count(empno) into v_rucno from emp
    where deptno = 10;
    If v_rucno > 25 then
    raise e_too_many_Emp;
    End if;
Exception
    when e_too_many_emp then
    ---
end;
```

CONCLUSION :-

We successfully understood the concept of the control structure and also understood exception handling in PL/SQL.

OUTPUT :-

```
/* A]
  Problem statement :-
     Consider Tables:
          1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status)
          2. Fine(Roll_no,Date,Amt) Accept Roll_no & NameofBook from user.

      Check the number of days (from date of issue),
          If days are between 15 to 30 then fine amount will be Rs 5per 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.
          Also handles the exception by named exception handler or user define exception
          handler.
*/



  CREATE TABLE Borrower(
          Roll_no INT PRIMARY KEY,
          Student_Name VARCHAR(20),
          DateOfIssue DATE,
          Book_Name VARCHAR(20) NOT NULL,
          book_status CHAR(1)
      );



 INSERT INTO Borrower VALUES
     (1, 'TONY', '2021-08-31', 'DBMS', 'I'),
     (2, 'TIM', '2021-08-20', 'CNS', 'I'),
     (3, 'KIM', '2021-07-31', 'DSA', 'I'),
     (4, 'SAM', '2021-09-04', 'OOP', 'I'),
     (5, 'KEVIN', '2021-08-11', 'PPS', 'I');



 CREATE TABLE Fine(
          Roll_no INT NOT NULL REFERENCES Borrower(Roll_no),
          DateOfReturn DATE,
          TotalDays INT,
          Amount INT
     );
```

```
delimiter $
CREATE PROCEDURE ReturnBook(IN rno INT, IN Bname VARCHAR(30) )
    BEGIN
        DECLARE not_null INT DEFAULT 0;
        proc_exit:
            BEGIN
                DECLARE i_date
                DATE;

                DECLARE diff INT;
                DECLARE stat CHAR(1);
                DECLARE EXIT HANDLER FOR 1048 SET not_null = 1;

    SELECT DateOfIssue
        INTO i_date
        FROM borrower
        WHERE Roll_no = rno AND Book_Name = Bname;

    SELECT book_status
        INTO stat
        FROM Borrower
        WHERE Roll_no = rno AND Book_Name = Bname;

    IF stat = 'R' THEN
        SELECT 'Book Already Returned !!' AS MESSAGE;
        LEAVE proc_exit;
    ELSE
        UPDATE Borrower
        SET book_status = 'R'
        WHERE Roll_no = rno AND Book_Name = Bname;
    END IF;

    SELECT datediff(curdate(), i_date) INTO diff;

    IF(diff > 15 AND diff <= 30) THEN
        INSERT INTO Fine
        VALUES(rno, curdate(), diff, (diff*5) );

    ELSEIF (diff > 30) THEN
                INSERT INTO Fine
                VALUES(rno, curdate(), diff, (diff*50) );
            ELSE
                INSERT INTO Fine
                VALUES(rno, curdate(), diff, 0);
            END IF;
        END;
        IF not_null = 1 THEN
            SELECT 'Unsuccessful Insert - NULL value' AS ERROR;
        END IF;
END $
```

```
delimiter ;
CALL ReturnBook(1, 'DBMS');
CALL ReturnBook(2, 'CNS');
CALL ReturnBook(3, 'DSA');
CALL ReturnBook(4, 'OOP');
CALL ReturnBook(5, 'PPS');


SELECT * FROM Borrower;
SELECT * FROM Fine;
```

Roll_no	Student_Name	DateOfIssue	Book_Name	book_status
1	TONY	2021-08-31	DBMS	I
2	TIM	2021-08-20	CNS	I
3	KIM	2021-07-31	DSA	I
4	SAM	2021-09-04	OOP	I
5	KEVIN	2021-08-11	PPS	I

Roll_no	DateOfReturn	TotalDays	Amount
1	2021-09-13	13	0
2	2021-09-13	24	120
3	2021-09-13	44	2200
4	2021-09-13	9	0
5	2021-09-13	33	1650

```
/* B]
  Problem statement :-
 Write a PL/SQL code block to calculate the area of a circle for a value of radius
    varying from 5 to 9. Store the radius and the corresponding values of calculated
    area in an empty table named areas, consisting of two columns, radius and area.
 Note: Instructor will frame the problem statement for writing PL/SQL block in line     with above
statement.
*/


CREATE TABLE Area_C(
     Radius INT PRIMARY KEY,
     AreaOfCircle FLOAT
 );


delimiter $$
CREATE PROCEDURE Circle_Area(IN r1 INT, IN r2 INT)
     BEGIN
          DECLARE not_null INT DEFAULT 0;
          BEGIN
               DECLARE rad INT;
               DECLARE EXIT HANDLER FOR 1048 SET not_null = 1;
               SET rad = r1;
                st : LOOP
                     IF rad = 0 then
                          INSERT INTO Area_C
                          VALUES(NULL, 3.14159*rad*rad);
                ELSE
                          INSERT INTO Area_C
                               VALUES(rad, 3.14159*rad*rad);
               END IF;

               IF rad = r2 THEN
                    LEAVE st;
               END IF;

     SET rad = rad+1;
     END LOOP st;
     SELECT * FROM Area_C;
END;

          IF not_null = 1 THEN
               SELECT 'Unsuccessful Insert - NULL value' AS ERROR;
          END IF;
END $$

  delimiter ;
   CALL Circle_Area(5,9);
```

Radius	AreaOfCircle
5	78.5397
6	113.097
7	153.938
8	201.062
9	254.469