	Dr. D. Y. Patil Unitech Society's **Dr. D. Y. Patil Institute of Technology,** **Pimpri, Pune-411018**

ASSIGNMENT NO. 5

Aim: Study of Pl SQL Control Structures and Exception Handling.

Input: Student roll no and attendance is input to Procedure**.**

Theory:

- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database. Following are certain notable facts about PL/SQL −
- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Features of PL/SQL

- PL/SQL has the following features −
- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

- PL/SQL has the following advantages −

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

DECLARE :- if you want to decalre a variable in plsql program then it takes place in declare section

BEGIN:- is used to start the working of program and end is used to terminate the begin.

Delimiter is used to run (/)

SET SERVEROUTPUT ON ; is run before every time when you compiled a program in a session.

SET ECHO ON : is optional

DBMS_OUTPUT.PUT_LINE command for e.g. if sal=10 and you want to print it Then it looks like dbms_output.put_line(‘the salary is ‘ ||sal);

IF STATEMEN

Common syntax

```
IF condition THEN
        statement 1;
ELSE
        statement 2;
 END IF;
```

INTO command: is used to catch a value in variable from table under some while condition

Only one value must be returned For e.g. in the above example if there are two people who's name is john then it shows error

Exception Handling:

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using *WHEN others THEN* −

```
DECLARE
  <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling goes here >
   WHEN exception1 THEN
      exception1-handling-statements
   WHEN exception2  THEN
      exception2-handling-statements
   WHEN exception3 THEN
      exception3-handling-statements
   ........
   WHEN others THEN
      exception3-handling-statements
END;
```

Problem Statement:

Use of Control structure and Exception handling is mandatory. 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)

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

Solution in Mysql:

Steps are:
1) Create borrower and fine table with primary and foreign keys
2) insert records in borrower table
3) create procedure to insert entries in fine table with exception handling
4) call procedure to calculate fine and display fine table.

```
mysql> create table borrower(rollin int primary key,name varchar(20),dateofissue date,nameofbook varchar(20),status varchar(20));
Query OK, 0 rows affected (0.30 sec)
```

```
mysql> desc borrower;
+-------------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| rollin | int(11) | NO | PRI | NULL | |
| name | varchar(20) | YES | | NULL | |
| dateofissue | date | YES | | NULL | |
| nameofbook | varchar(20) | YES | | NULL | |
| status | varchar(20) | YES | | NULL | |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> create table fine(rollno int,foreign key(rollno) references borrower(rollin),returndate date,amount int);
Query OK, 0 rows affected (0.38 sec)
```

```
mysql> desc fine;
+------------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| roll_no | int(11) | YES | MUL | NULL | |
| returndate | date | YES | | NULL | |
| amnt | int(11) | YES | | NULL | |
+------------+---------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

```
mysql> insert into borrower values(1,'abc','2017-08-01','SEPM','PEN')$
Query OK, 1 row affected (0.16 sec)

mysql> insert into borrower values(2,'xyz','2017-07-01','DBMS','PEN')$
Query OK, 1 row affected (0.08 sec)

mysql> insert into borrower values(3,'pqr','2017-08-15','DBMS','PEN')$
Query OK, 1 row affected (0.03 sec)


mysql> delimiter $
mysql> create procedure calc_fine_lib6(in roll int)
begin
declare fine1 int;
declare noofdays int;
declare issuedate date;
declare exit handler for SQLEXCEPTION select'create table definition';
select dateofissue into issuedate from borrower where rollin=roll;
select datediff(curdate(),issuedate) into noofdays;
if noofdays>15 and noofdays<=30 then
set fine1=noofdays*5;
insert into fine values(roll,curdate(),fine1);
elseif noofdays>30 then
set fine1=((noofdays-30)*50) + 15*5;
insert into fine values(roll,curdate(),fine1);
else
insert into fine values(roll,curdate(),0);
end if;
update borrower set status='return' where rollin=roll;
end $

mysql> call calc_fine_lib6(1)$
Query OK, 0 rows affected (0.09 sec)
mysql> call calc_fine_lib6(2)$
Query OK, 0 rows affected (0.09 sec)
mysql> call calc_fine_lib6(3)$
Query OK, 0 rows affected (0.09 sec)

mysql> select * from fine;
-> $
+---------+------------+------+
| roll_no | returndate | amnt |
+---------+------------+------+
| 1 | 2017-08-22 | 105 |
| 2 | 2017-08-22 | 780 |
| 3 | 2017-08-22 | 0 |
+---------+------------+------+
3 rows in set (0.00 sec)

mysql>drop table fine$
Query OK, 0 rows affected (0.21 sec)

mysql> call calc_fine_lib6(1)$
+-------------------------+
| create table definition |
+-------------------------+
| create table definition |
+-------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> create table fine(rollno int,foreign key(rollno) references borrower(rollin),returndate date,amount int)$
Query OK, 0 rows affected (0.34 sec)

mysql> call calc_fine_lib6(1)$
Query OK, 0 rows affected (0.09 sec)

mysql> select * from fine$
+--------+------------+--------+
| rollno | returndate | amount |
+--------+------------+--------+
| 1 | 2017-08-22 | 105 |
+--------+------------+--------+
1 row in set (0.00 sec)
```

Problem Statement: Consider table Stud(Roll, Att,Status)

Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “D”. Otherwise display message “Term granted” and set the status in stud table as “ND”

Solution in Oracle:

```
SQL> create table stud1(roll_no number(5),attendance number(5),status varchar(7));

Table created.

SQL> select * from stud1;

        ROLL_NO     ATTENDANCE     STATUS

        ----------  ----------     -------
```

```
                  101                   80
                  102                   65
                  103                   92
                  104                   55
                  105                   68

SQL> set serveroutput on;

SQL>

declare

roll number(10);

  att number(10);

   begin

   roll:=&roll;

   select attendance into att from stud1 where
  roll_no=roll;  if att<75 then

   dbms_output.put_line(roll||'is detained');

   update stud1 set status='D' where roll_no=roll;

  else

  dbms_output.put_line(roll||'is not detained');

  update stud1 set status='ND' where roll_no=roll;

  end if;

 exception

 when no_data_found then

 dbms_output.put_line(roll||'not found');

 end;

 /

Enter value for roll: 102

old  5: roll:=&roll;

new  5: roll:=102;

102is detained

PL/SQL procedure successfully completed.

SQL> /
```

```
Enter value for roll: 101
old  5: roll:=&roll;
new  5: roll:=101;
101is not detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 103
old  5: roll:=&roll;
new  5: roll:=103;
103is not detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 104
old  5: roll:=&roll;
new  5: roll:=104;
104is detained
PL/SQL procedure successfully completed.
SQL> /
Enter value for roll: 105
old  5: roll:=&roll;
new  5: roll:=105;
105is detained
PL/SQL procedure successfully completed.
SQL> select * from stud1;

  ROLL_NO            ATTENDANCE      STATUS
  -------------      ------------ -  -------
  101                80              ND
  102                65              D
```

```
103                     92                  ND
104                     55                  D

------------------------------------------------------------------------------------------------------------------
```

Conclusion:
Thus we successfully implemented procedures.

ASSIGNMENT NO. 6

Aim: To Study of all types of Cursor (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

Input: New roll calls and old roll calls

Theory:

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.
You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors
Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.
Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.
In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement.

%FOUND
Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

%NOTFOUND
The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.

%ISOPEN
Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

%ROWCOUNT
Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.
The syntax for creating an explicit cursor is −
CURSOR cursor_name IS select_statement;
Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor
Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor
Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor
Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor
Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Problem Statement :

Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Solution in MySQL:

Steps are:
1) Create new_roll call and old_roll call tables
2) Insert records in both tables with few records duplication
3) create procedure and use cursor to merge above two tables to finalize roll list without duplication.
Assignment code:

```
mysql>create table new_roll(roll int,name varchar(10));
Query OK, 0 rows affected (0.29 sec)
mysql> create table old_roll(roll int,name varchar(10));
Query OK, 0 rows affected (0.28 sec)
mysql> insert into new_roll values(2,'b')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into old_roll values(4,'d')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into old_roll values(3,'bcd')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into old_roll values(1,'bc')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into old_roll values(5,'bch')$
Query OK, 1 row affected (0.04 sec)
mysql> insert into new_roll values(5,'bch')$
Query OK, 1 row affected (0.05 sec)
mysql> insert into new_roll values(1,'bc')$
Query OK, 1 row affected (0.04 sec)
mysql> select * from new_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 5 | bch |
| 1 | bc |
+------+------+
4 rows in set (0.00 sec)

mysql> select * from old_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 3 | bcd |
| 1 | bc |
| 5 | bch |
+------+------+
5 rows in set (0.00 sec)

create procedure roll_list()
begin
declare a int;
declare a1 varchar(10);
declare b int;
declare b1 varchar(10);
declare done int default false;
declare c1 cursor for select roll,name from old_roll;
declare c2 cursor for select roll,name from new_roll;
declare continue handler for not found set done=true;
open c1;
open c2;
loop1:loop
fetch c1 into a,a1;
if done then
leave loop1;
end if;
loop2:loop
fetch c2 into b,b1;
if done then
insert into new_roll values(a,a1);
leave loop2;
end if;
if a=b then
leave loop2;
end if;
end loop;
end loop;
close c1;
close c2;
end $

mysql> call roll_list()$
Query OK, 1 row affected (0.04 sec)

mysql> select * from new_roll$
+------+------+
| roll | name |
+------+------+
| 2 | b |
| 4 | d |
| 5 | bch |
| 1 | bc |
| 3 | bcd |
+------+------+
5 rows in set (0.01 sec)
```

Problem Statement 2: The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

Solution in Oracle:

```
Declare
Rows_affe number(10);
Begin
update bankcursor set status='active'where
status='inactive'; Rows_affe:=(SQL%rowcount);
dbms_output.put_line(Rows_affe||' rows are
affected...'); END;
```

Solution :

```
SQL> create table bankcursor(acc_no number(10),status varchar(10));
Table created.

SQL> select * from bankcursor;

      ACC_NO     STATUS
      ----------  ----------
      101            active
      102            inactive
      103            inactive
      104            active
      105            inactive

SQL>
Declare
Rows_affe number(10);
  Begin
  update bankcursor set status='active'where status='inactive';
  Rows_affe:=(SQL%rowcount);
  dbms_output.put_line(Rows_affe||' rows are affected...');
  END;
  /

3 rows are affected...
PL/SQL procedure successfully completed.

SQL> select * from bankcursor;

    ACC_NO       STATUS
      ----------  ----------
      101            active
      102            active
      103            active
      104            active
      105            active
```

Problem Statement 3: Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

EMP (E_no , Salary)
increment_salary(E_no ,
Salary) code:

Solution in Oracle:

```
Declare
Cursor crsr_sal is select e_no,salary from emp2 where salary<(select avg(salary) from emp2);
me_no emp2.e_no%type;
msalary emp2.salary%type;

Begin
open crsr_sal;
if crsr_sal%isopen then
loop
fetch crsr_sal into me_no,msalary;
exit when crsr_sal%notfound;
if crsr_sal%found then
update emp2 set salary=salary+(salary*0.1) where
e_no=me_no; select salary into msalary from emp2 where
e_no=me_no; insert into increament_t values(me_no,msalary);
end if;
end loop;
end if;
end;
```

```
SQL> create table emp2(e_no number(10),salary number(10));

Table created.

SQL> select * from emp2;

      E_NO      SALARY
      ----------  ----------
      101          1000
      102          2000
      103          113
      104          4000
```

```
SQL> create table increament_t(eno number(10),sal number(10));

Table created.

SQL>
Declare
Cursor crsr_sal is select e_no,salary from emp2 where salary<(select
avg(salary) from emp2);
  me_no emp2.e_no%type;
  msalary emp2.salary%type;

  Begin
  open crsr_sal;
  if crsr_sal%isopen then
  loop
 fetch crsr_sal into me_no,msalary;
 exit when crsr_sal%notfound;
 if crsr_sal%found then
 update emp2 set salary=salary+(salary*0.1) where
 e_no=me_no; 14 select salary into msalary from emp2 where
 e_no=me_no;
 insert into increament_t values(me_no,msalary);
 end if;
 end loop;
 end if;
 end;
 /

PL/SQL procedure successfully completed.

SQL> select * from emp2;

          E_NO          SALARY
          ----------     ----------
          101              1100
          102              2000
          103              113
          104              4000

SQL> select * from increament_t;

          ENO        SAL
          ---------- ----------
          •          1100
          103        113
```

Conclusion:

Thus we successfully implemented procedures.

Dr. D. Y. Patil Unitech Society's

Dr. D. Y. Patil Institute of Technology,

Pimpri, Pune-411018

ASSIGNMENT NO. 7

Aim: To Study of PL/SQL Stored Procedure and Stored Function.

Input: Students details and marks

Theory:

Procedure:

- A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.
- A subprogram can be created −
- At the schema level
- Inside a package
- Inside a PL/SQL block
- At the schema level, 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.
- A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter **'PL/SQL - Packages'**.
- 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.
- This chapter is going to cover important aspects of a **PL/SQL procedure**. We will discuss **PL/SQL function** in the next chapter.

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.

Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.
A called function performs the defined task and when its return statement is executed or when the **last end statement** is reached, it returns the program control back to the main program.
To call a function, you simply need to pass the required parameters along with the function name and if the function returns a value, then you can store the returned value.

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

Problem Statement 1: PL/SQL Stored Procedure and Stored Function.

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class
Write a PL/SQL block for using procedure created with above requirement.
Stud_Marks(name, total_marks) Result(Roll,Name, Class)

Solution in MySQL:

Steps:

1) create stud_marks and result table with primary and foreign keys

2)insert values in stud_marks

3)write and execute PL/SQL procedure for inserting grades in result table

Assignment is as follows:

```
mysql> desc stud_marks;

+-------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+-------------+------+-----+---------+-------+

| name | varchar(20) | NO | PRI | NULL | |

| total_marks | int(11) | YES | | NULL | |

+-------------+-------------+------+-----+---------+-------+

2 rows in set (0.01 sec)


mysql> desc result;

+-------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+-------+

| roll | int(11) | YES | | NULL | |

| class | varchar(10) | YES | | NULL | |

| name | varchar(20) | YES | MUL | NULL | |

+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into stud_marks values('abhijit',1020)$
Query OK, 1 row affected (0.15 sec)
mysql> insert into stud_marks values('anand',979)$
Query OK, 1 row affected (0.04 sec)
mysql> insert into stud_marks values('vijay',864)$
Query OK, 1 row affected (0.04 sec)
mysql> insert into stud_marks values('vikas',755)$
Query OK, 1 row affected (0.03 sec)

Create procedure proc_grade()
begin
declare done int default false;
declare roll int;
declare totmarks int;
declare class varchar(10);
declare name1 varchar(20);
declare c1 cursor for select name,total_marks from stud_marks;
declare continue handler for not found set done=true;
open c1;
set roll=1;
readloop:loop
fetch c1 into name1,totmarks;
if done then
leave readloop;
end if;
if totmarks<=1500 and totmarks>=990 then
insert into result values(roll,'dist',name1);
elseif totmarks<=989 and totmarks>=900 then
```

```
insert into result values(roll,'first',name1);
elseif totmarks<=899 and totmarks>=825 then
insert into result values(roll,'HSC',name1);
else
insert into result values(roll,'poor',name1);
end if;
set roll=roll+1;
end loop;
end $
mysql> call proc_grade()$
Query OK, 0 rows affected (0.38 sec)
mysql> select * from result$
+------+-------+---------+
| roll | class | name |
+------+-------+---------+
| 1 | dist | abhijit |
| 2 | first | anand |
| 3 | HSC | vijay |
| 4 | poor | vikas |
+------+-------+---------+
4 rows in set (0.00 sec)
```

Problem Statement 2. Write a PL/SQL stored Procedure for following requirements and call the procedure in appropriate PL/SQL block.

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 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

Solution :

```
SQL>create or replace function cal_fine(diffdate number) return number is
begin
if diffdate<15 then
return 0;
elsif diffdate<30 then
return (5*(diffdate-15));
else
return (50*(diffdate-30)+5*(15));end if;
end;
/
------------------------------------------------------------------------------------------
SQL> Declare
troll_no varchar(5);
tdays number(5);
tdate date;
diffdate number(5);
begin
troll_no := '&troll_no';
select to_date(sysdate,'DD-MM-YY')"Now" into tdate from dual;
select ((select to_date(sysdate,'DD-MM-YY')"Now" from dual)-dateofissue) into diffdate
from Borrower
where roll_no=troll_no;
insert into Fine values(troll_no,tdate,cal_fine(diffdate));
update borrower set status = 'R' where roll_no=troll_no;
End;
/
```

Conclusion:

Thus we have successfully implemented PL/SQL stored procedures.

	Dr. D. Y. Patil Unitech Society's **Dr. D. Y. Patil Institute of Technology,** **Pimpri, Pune-411018**

ASSIGNMENT NO. 8

Aim: To Study all types of Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

Input: Student library books information

Theory:

- Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −
- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

- Triggers can be written for the following purposes −
- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

- The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
```

```
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Problem Statement:

Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Solution in MySQL:

Steps are:
1) Create lib_audit and lib_audit2 tables
2) Insert records in lib_audit
3) create trigger for before update and before delete on lib_audit.

//Trigger for delete on lib_audit

```
mysql> create table lib_audit(bookid int,bookname varchar(20),price int)$
Query OK, 0 rows affected (0.58 sec)

mysql> create table lib_audit2(bookid int,bookname varchar(20),price int)$
Query OK, 0 rows affected (0.36 sec)

mysql> Create trigger before_delete_lib_audit before delete on lib_audit for each row
begin
insert into lib_audit2 values(old.bookid,old.bookname,old.price);
end$
Query OK, 0 rows affected (0.13 sec)

mysql> insert into lib_audit values(1,'ab',100)$
Query OK, 1 row affected (0.05 sec)
mysql> insert into lib_audit values(2,'cd',10)$
Query OK, 1 row affected (0.05 sec)
mysql> insert into lib_audit values(3,'dg',101)$
Query OK, 1 row affected (0.05 sec)

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
| 2 | cd | 10 |
| 3 | dg | 101 |
+--------+----------+-------+
3 rows in set (0.00 sec)

mysql> select * from lib_audit2$
Empty set (0.00 sec)

mysql> delete from lib_audit where bookid=1$
Query OK, 1 row affected (0.14 sec)

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 2 | cd | 10 |
| 3 | dg | 101 |
+--------+----------+-------+
2 rows in set (0.00 sec)

mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
+--------+----------+-------+
1 row in set (0.00 sec)

mysql> delete from lib_audit where bookid=3$
Query OK, 1 row affected (0.04 sec)

mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
| 3 | dg | 101 |
+--------+----------+-------+
2 rows in set (0.00 sec)

//Trigger for update on lib_audit
mysql> Create trigger before_update_lib_audit before update on lib_audit for each row
begin
insert into lib_audit2 values(old.bookid,old.bookname,old.price);
end$

mysql> update lib_audit set bookname='xy' where bookid=2$
Query OK, 1 row affected (0.07 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from lib_audit$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 2 | xy | 10 |
+--------+----------+-------+
1 row in set (0.00 sec)
mysql> select * from lib_audit2$
+--------+----------+-------+
| bookid | bookname | price |
+--------+----------+-------+
| 1 | ab | 100 |
| 3 | dg | 101 |
| 2 | cd | 10 |
+--------+----------+-------+
3 rows in set (0.00 sec)
```

Problem Statement : Write a update, delete trigger on client mstr table. The System should keep track of the records that ARE BEING updated or deleted. The old value of updated or deleted records should be added in audit trade table. (separate implementation using both row and statement triggers).

Solution in Oracle:

Row trigger:

```
SQL> create or replace trigger t1 after update or delete on client_master 2
for each row
declare
op varchar(10);
begin
if updating then
op:='update';
end if;
if deleting then
op:='Delete';   end if;
into stat values(:old.id,op);
insert into audit_trade values(:old.id,:old.cname);
dbms_output.put_line('Details updated to stat and audit_trade table');
end;
/ Trigger created.
```

Statement Trigger:

```
SQL> create or replace trigger t1 after update or delete on client_master 2
for each row
declare
op varchar(10);
begin
if updating then
op:='update';
end if;
if deleting then
op:='Delete';   end if;
into stat values('',op);
insert into audit_trade values(:old.id,:old.cname);
dbms_output.put_line('Details updated to stat and audit_trade table');   end;
/ Trigger created.
```

Conclusion: Thus we have successfully implemented trigger.