**ASSIGNMENT 5**

**AIM:-**

PL/SQL Stored Procedure and Stored Function : Write a PL/SQL procedure to find

the number of students ranging from 100-80%, 79-70%, 69-60%,59-50 & below 49%

in each course from the Student\_course table given by the procedure as parameter.

Student\_course(Roll\_no,Course,Course\_code,Semester,Total\_Marks,Percentage).

**OBJECTIVE:-**

To implement PL/SQL Stored Procedure.

**THEORY:-**

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 −

1.At the schema level

2.Inside a package

3.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.

**STORED PROCEDURE**

A stored procedure or in simple a proc is a named PL/SQL block which performs one or more specific task.

This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure.

The body consists of declaration section, execution section and exception section similar to a general PL/SQL Block.

**Creating a Procedure**

A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Syntax:-

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

Where,

**procedure-name** specifies the name of the procedure.

**[OR REPLACE]** option allows the modification of an existing procedure.

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.

**procedure-body** contains the executable part.

**IS** keyword is used, when the procedure is nested into some other blocks.

The **AS** keyword is used instead of the IS keyword for creating a standalone procedure.

**STORED FUNCTION:-**

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

General Syntax to create a function is

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.

**PROGRAM:-**

Select \*from student\_course;

Delimiter /

CREATE PROCEDURE GETRANK(IN crc varchar(255))

begin

declare distinction int default 0;

declare firstclass int default 0;

declare highersec int default 0;

declare sec int default 0;

declare pass int default 0;

declare per int default 0;

DECLARE finished int default 0;

declare r

cursor for

select Percentage from student\_course

where Course=crc;

DECLARE CONTINUE HANDLER

FOR NOT FOUND SET finished=1;

open r;

lp:loop

FETCH r INTO per;

IF finished =1 THEN LEAVE lp;

elseif per<100 and per>=80 then set distinction :=distinction+1;

elseif per<80 and per>=70 then set firstclass:=firstclass+1;

elseif per<70 and per>=60 then set highersec:=highersec+1;

elseif per<60 and per>=50 then set sec:=sec+1;

elseif per<50 then set pass:=pass+1;

end if;

END LOOP lp;

CLOSE r;

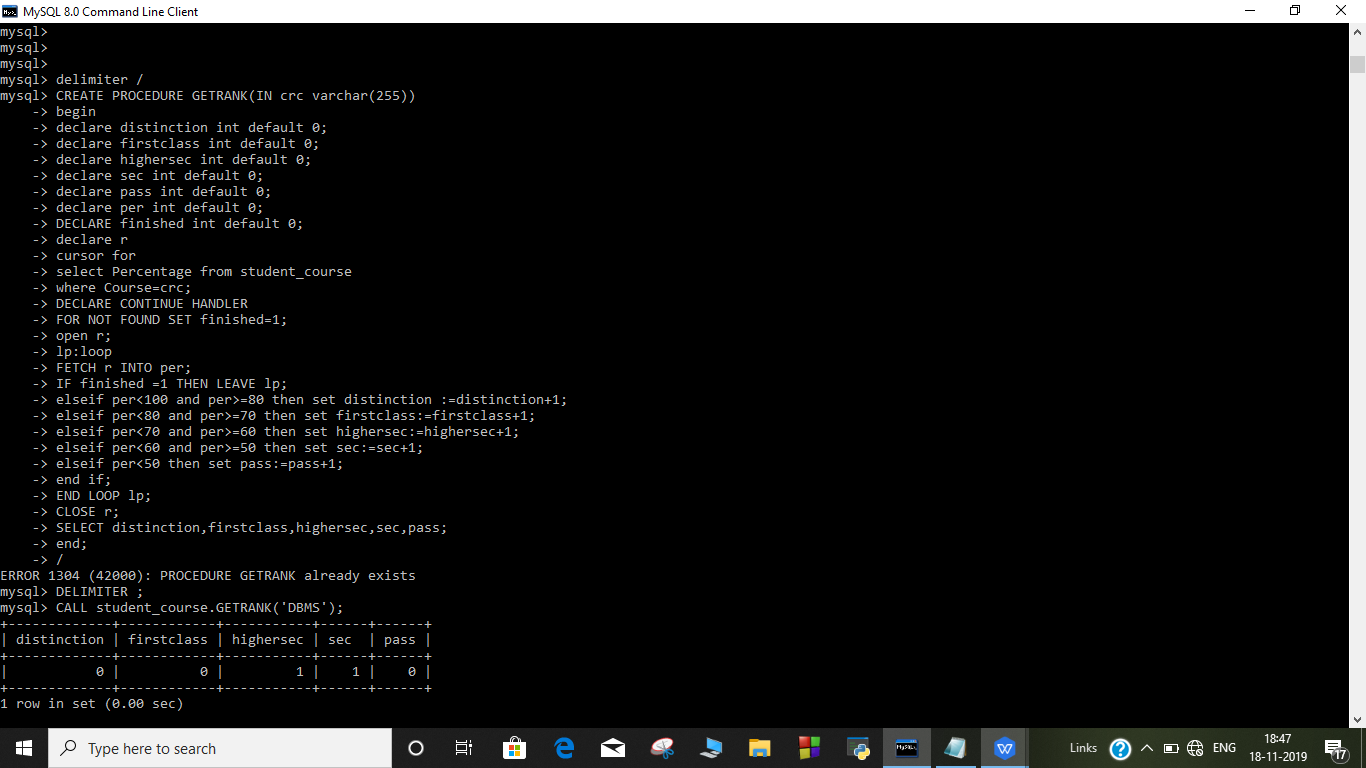
SELECT distinction,firstclass,highersec,sec,pass;

end;

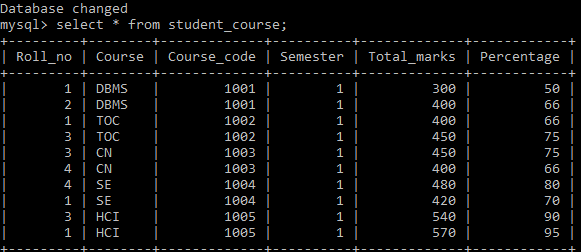
/

DELIMITER ;

CALL student\_course.GETRANK('DBMS');



**OUTPUT:-**

****

