(c)	Demonstrate Create Trigger, Table Level triggers.
13(a)	Write a PL/SQL program to find the total and average of 6 subjects and display the grade.
(b)	Create a stored procedure, Alter and Drop a procedure, IN, OUT, IN & OUT parameters
(c)	Demonstrate Create Trigger, Before Insert.
14(a)	Write a PL/ SQL program to check whether the given number is

7. Write a PL/SQL program to find the total and average of 6 subjects and display the grade

Solution:

```
set serveroutput on
declare
        sno number(3);
        sna varchar(20);
        s1 number(3);
        s2 number(3);
        s3 number(3);
        tot number(4);
        avg1 number(5,2);
        resu varchar(5);
        grade varchar(15);
begin
        sno := &sno;
        sna := '&sna';
        s1 := &s1;
        s2 := &s2;
        s3 := &s3;
tot := s1 + s2 + s3;
avg1 := tot/3;
if (s1 >= 35 and s2 >= 35 and s3 >= 35) then
        resu := 'pass';
else
        resu := 'fail';
end if;
if resu = 'fail' then
```

```
        avg1 := 0;
        grade := 'No Division';
end if;
if avg1 >= 35 and avg1 < 50 then
        grade := 'Third';
elsif avg1 >= 50 and avg1 <=60 then
        grade := 'Second';
elsif avg1 >= 60 and avg1 < 70 then
        grade := 'First';
elsif avg1 >= 70 then
        grade := 'Distinction';
end if;
dbms_output.put_line('TOTAL MARKS….' || tot);
dbms_output.put_line('AVERAGE…..' || avg1);
dbms_output.put_line('RESULT…..' || resu);
dbms_output.put_line('GRADE….' || grade);
end;
```

To save a program:

Alt + f

And save as program name.sql

To run a program:

SQL>@program name.sql

/ (press enter key)

How to Create a Simple Stored Procedure in SQL?

Creating a stored procedure in SQL is as easy as it can get. The syntax of SQL stored procedure is:

```
CREATE or REPLACE PROCEDURE name(parameters)

    AS

    variabl es;

BEGIN;

    //statements;

END;
```

In the syntax mentioned above, the only thing to note here are the parameters, which can be the following three types:

- IN: It is the default parameter that will receive input value from the program
- OUT: It will send output value to the program
- IN OUT: It is the combination of both IN and OUT. Thus, it receives from, as well as sends a value to the program

Note: You will work with and look at examples for different parameters in this article.

- IN: It is the default parameter that will receive input value from the program
- to the program
- of both IN and OUT. Thus, it receives from, as well as sends a value to the

Note: You will work with and look at examples for different parameters in this article.

You will use the syntax to create a simple stored procedure in SQL. But before that, create two tables using the CREATE TABLE command that you will use throughout the article. You will also insert some values in them using the INSERT INTO command.

```
CREATE TABLE Car(

CarID INT,

CarName VARCHAR(100)

);

INSERT INTO Car VALUES (101,'Mercedes-Benz');

INSERT INTO Car VALUES (201,'BMW');

INSERT INTO Car VALUES (301,'Ferrari');
```



```
INSERT INTO Car VALUES (401,'Lamborghini');

INSERT INTO Car VALUES (501,'Porsche');

SELECT * FROM Car;
```

Output:

Results Messages

	CarID	CarDescription
1	101	Luxury vehicle from Germany
2	201	Luxury motorcycle from Germany
3	301	Luxury sports car from Italy
4	401	Luxury SUV from Italy
5	501	High-performance sports car from Germany

Now create the second table named CarDescription.

```
CREATE TABLE CarDescription(

CarID INT,

CarDescription VARCHAR(800)

);

INSERT INTO CarDescription VALUES (101,'Luxury vehicle from the German automotive');

INSERT INTO CarDescription VALUES (201,'Luxury motorcycle from the German automotive');

INSERT INTO CarDescription VALUES (301,'Luxury sports car from the Italian manufacturer');
```

INSERT INTO CarDescription VALUES (101,'Luxury vehicle from the German automotive');

INSERT INTO CarDescription VALUES (201,'Luxury motorcycle from the German automotive');

INSERT INTO CarDescription VALUES (301,'Luxury sports car from the Italian manufacturer');

INSERT INTO CarDescription VALUES (401,'Luxury SUV from the Italian automotive');

INSERT INTO CarDescription VALUES (501,'High-performance sports car from the German manufacturer');

SELECT * FROM CarDescription;

Output:

	CarID	CarDescription
1	101	Luxury vehicle from Germany
2	201	Luxury motorcycle from Germany
3	301	Luxury sports car from Italy
4	401	Luxury SUV from Italy
5	501	High-performance sports car from Germany

Now that you have created both the tables, start creating the stored procedure in SQL with the syntax mentioned earlier. For the simple procedure, you will have to use the JOIN keyword to join both the tables, and output a new one with CarID, CarName, and CarDescription.

```
CREATE PROCEDURE GetCarDesc

AS

BEGIN

SET NOCOUNT ON

SELECT C.CarID,C.CarName,CD.CarDescription  FROM

Car C

INNER JOIN CarDescription CD ON C.CarID=CD.CarID

END
```

This will create the stored procedure, and you will see the "command(s) executed successfully" message in Microsoft SQL Server Management Studio. Now, since you have created the procedure, it's time to execute it. The syntax to execute the procedure is:

```
EXEC procedure_name
```

Let's execute the procedure we have created.

EXEC GetCarDesc;

Output:

	CarID	CarName	CarDescription
1	101	Mercedes-Benz	Luxury vehicle from Germany
2	201	BMW	Luxury motorcycle from Germany
3	301	Ferrari	Luxury sports car from Italy
4	401	Lamborghini	Luxury SUV from Italy
5	501	Porsche	High-performance sports car from Germany

As you can see in the output, the stored procedure executed the Join statement and gave the desired result.

{insert | update | delete}: This specifies the DML operation.

on [table_name]: This specifies the name of the table associated with the trigger.

[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

[trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run. AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and percentage of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

.

Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)
```

.


```
mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```