Assignment No.5

Title: Write and execute suitable database triggers. Consider row level and statement level triggers.

Aim: Study and implementation of database MYSQL Triggers

Objectives: To understand the concept of database MYSQL Trigger

Theory:

1) Introduction to MYSQL Trigger

What is a Trigger?

A trigger is a MYSQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

2) Types of triggers

I) Types of Triggers

There are two types of triggers based on the which level it is triggered.

1) Row level trigger - An event is triggered for each row upated, inserted or deleted.
2) Statement level trigger - An event is triggered for each sql statement executed.

Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

1) BEFORE statement trigger fires first.
2) Next BEFORE row level trigger fires, once for each row affected.
3) Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.
4) Finally the AFTER statement level trigger fires.

Syntax of Triggers

The Syntax for creating a trigger is:

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

[FOR EACH ROW]
WHEN (condition)
BEGIN
--- sql statements
END;

- *CREATE TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *{BEFORE | AFTER | INSTEAD OF }* - This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
- *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- *[OF col_name]* - This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
- *CREATE [OR REPLACE] TRIGGER trigger_name* - This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
- *[ON table_name]* - This clause identifies the name of the table or view to which the trigger is associated.
- *[REFERENCING OLD AS o NEW AS n]* - This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
- *[FOR EACH ROW]* - This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
- *WHEN (condition)* - This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified

Trigger Examples

Example 1)

This example is based on the following two tables:

```
CREATE  TABLE  T4 ( a  INTEGER ,  b  CHAR(10));

CREATE  TABLE  T5 ( c  CHAR(10) ,  d  INTEGER);
```

-- create a trigger that may insert a tuple into T5 when a tuple is inserted into T4. inserts the reverse tuple into T5:

1) Create trigger as follows:

```
CREATE TRIGGER trig1 AFTER INSERT ON T4

 FOR EACH ROW BEGIN

INSERT INTO t5 SET c = NEW.b,d = NEW.a;

    END;
```

2) Insert values in T4.

3) Check the values in T5.

Example2)

1)The price of a product changes constantly. It is important to maintain the history of the prices of the products. Create a trigger to update the 'product_price_history' table when the price of the product is updated in the 'product' table.

Create the 'product' table and 'product_price_history' table

```
CREATE TABLE product_price_history
(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2) );

CREATE TABLE product
```

(product_id number(5),
product_name varchar2(32),
supplier_name varchar2(32),
unit_price number(7,2));
drop trigger if exists price_history_trigger;

CREATE TRIGGER price_history_trigger

BEFORE UPDATE on product1

FOR EACH ROW BEGIN

INSERT INTO product_price_history

set product_id=old.product_id,

product_name=old.product_name,

supplier_name=old.supplier_name,

unit_price=old.unit_price;

END

3) Lets update the price of a product.

UPDATE PRODUCT SET unit_price = 800 WHERE product_id = 100

Once the above update query is executed, the trigger fires and updates the 'product_price_history' table.

Example 3
create table account(accno int,amount int)

Create a trigger on account table before update in new inserted amount is less than “0” then set amount “0” else if amount is greater than 100 then set amount 100

CREATE TRIGGER upd_check BEFORE UPDATE ON account

FOR EACH ROW

BEGIN

IF NEW.amount < 0 THEN

SET NEW.amount = 0;

```
 ELSEIF NEW.amount > 100 THEN

   SET NEW.amount = 100;

    END IF;

END
```

```
 update account set amount= -12 where accno=101
```

Deleting a trigger
DROP TRIGGER

Name

DROP TRIGGER -- Removes a trigger definition from a database.

Synopsis

DROP TRIGGER *name* ON *table*

Parameters

name

The name of the trigger you wish to remove.

table
The name of the table the trigger is on.

Results

Conclusion: Studied and implemented MYSQL Trigger

FAQs:-

1) What is trigger and cursor in PL/SQL?
2) What are the types of trigger and cursor?
3) How to delete a trigger?
4) Why we write a cerate or replace in PL/SQL Block?
5) What is row level and statement level trigger?

Department of Infromation Technology

Software Laborotory-I

Assignment No:-5 (PART-B)

Problem Statement

Consider **Account table** containing information of **acc_no, name, amount** of minimum 05 customers.

Write a Trigger that maintains a log of a Account table.

Whenever there is any update operation perform on account table amount, account_log table maintains acc_no, name, amount (that is deposited or withdrawn), Type of transaction, and Timestamp which includes Date and Time.

Write and execute suitable database triggers .

Consider Account table containing information of five customers.

STEP 1: Create account table, It consist of information of acc_no, name, amount of customers.

```
mysql> create table account
   -> (
   -> acc_no int(20),
   -> name varchar(20),
   -> amount int(20)
   -> );
Query OK, 0 rows affected (0.30 sec)
```

STEP 2: In step 1 table created succesfully, now insert five customer information into account table.

```
mysql> insert into account
    -> values
    -> (1,'RAJ',2000),
    -> (2,'AJAY',3000),
    -> (3,'RAHUL',6000);
    -> (4,'AMIT',7000);
    -> (5,'SATISH',8000);
Query OK, 5 rows affected (0.06 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

View the all customer information present in account table

```
mysql> select * from account//
+--------+--------+--------+
| acc_no | name   | amount |
+--------+--------+--------+
|      1 | RAJ    |   2000 |
|      2 | AJAY   |   3000 |
|      3 | RAHUL  |   6000 |
|      4 | AMIT   |   7000 |
|      5 | SATISH |   8000 |
+--------+--------+--------+
5 rows in set (0.00 sec)
```

STEP 3: Create a new table named account_log table to maintain the log(changes) of the account table.

```
mysql> create table account_log
    -> (
    -> acc_no int(20),
    -> name varchar(20),
    -> amount int(20),
    -> tran_type varchar(20),
    -> tran_time timestamp
    -> );
Query OK, 0 rows affected (0.26 sec)
```

STEP 4: Describe the both tables account and account_log tables.

```
mysql> desc account;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| acc_no | int(20)     | YES  |     | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| amount | int(20)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> desc account_log;
+-----------+-------------+------+-----+-------------------+-----------------------------+
| Field     | Type        | Null | Key | Default           | Extra                       |
+-----------+-------------+------+-----+-------------------+-----------------------------+
| acc_no    | int(20)     | YES  |     | NULL              |                             |
| name      | varchar(20) | YES  |     | NULL              |                             |
| amount    | int(20)     | YES  |     | NULL              |                             |
| tran_type | varchar(20) | YES  |     | NULL              |                             |
| tran_time | timestamp   | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-----------+-------------+------+-----+-------------------+-----------------------------+
5 rows in set (0.01 sec)
```

STEP 5: Create a AFTER UPDATE trigger that is invoked after a upade is made to the amount in account table.

```
mysql> create trigger acc_log
    -> after update
    -> on account
    -> for each row
    -> begin
    -> declare tran_type varchar(20);
    -> declare amt varchar(20);
    -> if old.amount > new.amount then
    -> SET tran_type='Withdraw';
    -> SET amt=old.amount-new.amount;
    -> else
    -> SET tran_type='Deposited';
    -> SET amt=new.amount-old.amount;
    -> end if;
    -> insert into account_log values(old.acc_no,old.name,amt,tran_type,now());
    -> end
    -> //
Query OK, 0 rows affected (0.07 sec)
```

STEP 6: After that, update customer amount in account table to check whether the trigger is invoked.

Before trigger invoked customer information of account table is shown in table.

```
mysql> select * from account//
+--------+--------+--------+
| acc_no | name   | amount |
+--------+--------+--------+
|      1 | RAJ    |   2000 |
|      2 | AJAY   |   3000 |
|      3 | RAHUL  |   6000 |
|      4 | AMIT   |   7000 |
|      5 | SATISH |   8000 |
+--------+--------+--------+
5 rows in set (0.00 sec)
```

Case I- Update amount in account table of customer acc_no is 1

```
mysql> update account set amount=5000 where acc_no=1//
        Query OK, 1 row affected (0.10 sec)
        Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account//
+--------+--------+--------+
| acc_no | name   | amount |
+--------+--------+--------+
|      1 | RAJ    |   5000 |
|      2 | AJAY   |   3000 |
|      3 | RAHUL  |   6000 |
|      4 | AMIT   |   7000 |
|      5 | SATISH |   8000 |
+--------+--------+--------+
5 rows in set (0.00 sec)
```

(In this case, amount is deposited because old amount (2000) is less than new update amount (5000) in account table. i.e. amt(3000)=new.amount(5000)-old.amount(2000))

STEP 7: Finally, to check if the trigger was invoked by the UPDATE statement, you can query the account_log table using the following query:

CASE I:- Deposited amount

```
mysql> select * from account_log//
+--------+------+--------+-----------+--------------------+
| acc_no | name | amount | tran_type | tran_time          |
+--------+------+--------+-----------+--------------------+
|      1 | RAJ  |   3000 | Deposited | 2020-10-24 16:14:43 |
+--------+------+--------+-----------+--------------------+
1 row in set (0.00 sec)
```

CASE II:- Update amount in account table of customer acc_no is 1.

Before trigger invoked customer information of account table is shown in table.

```
mysql> select * from account//
+--------+--------+--------+
| acc_no | name   | amount |
+--------+--------+--------+
|      1 | RAJ    |   5000 |
|      2 | AJAY   |   3000 |
|      3 | RAHUL  |   6000 |
|      4 | AMIT   |   7000 |
|      5 | SATISH |   8000 |
+--------+--------+--------+
5 rows in set (0.00 sec)
```

```
mysql> update account set amount=4000 where acc_no=1//
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from account//
+--------+--------+--------+
| acc_no | name   | amount |
+--------+--------+--------+
|      1 | RAJ    |  4000 |
|      2 | AJAY   |   3000 |
|      3 | RAHUL  |   6000 |
|      4 | AMIT   |   7000 |
|      5 | SATISH |   8000 |
+--------+--------+--------+
5 rows in set (0.00 sec)
```

(In this case, amount is withdraw because old amount (5000) is gretter than new update amount (4000) in account table. i.e. amt(1000)=old.amount(5000)-new.amount(4000))

STEP 6: Finally, to check if the trigger was invoked by the UPDATE statement, you can query the account_log table using the following query:

CASE II:- Withdraw amount

```
mysql> select * from account_log//
+--------+------+--------+-----------+---------------------+
| acc_no | name | amount | tran_type | tran_time           |
+--------+------+--------+-----------+---------------------+
|      1 | RAJ  |   3000 | Deposited | 2020-10-24 16:14:43 |
|      1 | RAJ  |   1000 | Withdraw  | 2020-10-24 16:15:39 |
+--------+------+--------+-----------+---------------------+
2 rows in set (0.00 sec)
```

Assignment No. 6

Title: Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use.

Aim: Write and execute PL/SQL stored procedure and function to perform a suitable task on the database. Demonstrate its use

Objective: 1) To understand the differences between procedure and function

2) To understand commands related to procedure and function

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:

- At schema level
- Inside a package
- Inside a MYSQL block

Parts of a MYSQL Subprog ram

Each MYSQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:
1. Declarative Part
2. Executable part
3. Exception-handling

What is procedure? How to create it?

Procedures: these subprogram rams do not return a value directly, mainly used to perform an action.

Creating a Procedure
A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]

BEGIN
< procedure_body >
END ;
```

Where,
procedure-name specifies the name of the procedure.

[OR REPLACE] option allows modifying an existing procedure.

The optional parameter list contains name, mode and types of the parameters. IN represents, that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

Procedure-body contains the executable part.
The IS keyword is used for creating a standalone procedure.
T he following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
Delimiter //
CREATE OR REPLACE PROCEDURE greeting
Select concat('Hello World!');/
```

When above code is executed using SQL prompt, it will produce the following result:

```
Query ok
```

(2) How to execute procedure?

Executing a Standalone Procedure

- Calling the name of the procedure from a PL/SQL block

```
Call greeting( )
```

```
Output:
Hello world
```

Deleting a Standalone Procedure
A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is:

```
DROP PROCEDURE procedure-name;
```

So you can drop *greetings* procedure by using the following statement:

```
DROP PROCEDURE greetings;
```

Parameter modes in PL/SQL subprograms:
1. IN:
An IN parameter lets you pass a value to the subprogram.
It is a read-only parameter.

It is the default mode of parameter passing.
Parameters are passed by reference.

2. OUT:
An OUT parameter returns a value to the calling program.
The actual parameter must be variable and it is passed by value.

3. IN-OUT:
An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller.
Actual parameter is passed by value.
IN & OUT Mode Example 1
T his program finds the minimum of two values, here procedure takes two numbers using IN mode and returns their minimum using OUT parameters.

```
delimiter $
create procedure addp()
begin
declare a,b,c int;
set a=2;
set b=3;
set c=a+b;
select concat('value',c);
end;
$
```

```
delimiter ;
call addp();
Result: value 5
```

```
mysql> delimiter //
mysql> create procedure difference (in a int,in b int, out c int)
    -> begin
    -> if a>b then
    -> set c=1;
    -> else if a=b then
    -> set c=2;
    -> else
    -> set c=3;
    -> end if;
```

```
mysql> call difference(5,9,@x);
    -> select @x;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create table student
    -> ( sid int(5) not null,
    -> student_name varchar(9),
    -> DOB date,
    -> primary key(sid));
Query OK, 0 rows affected (0.06 sec)

mysql> insert into student values(5,'Harry',20130412);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(6,'Jhon',20100215);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(7,'Mary',20140516);
Query OK, 1 row affected (0.03 sec)

mysql> insert into student values(8,'Kay',20131116);
Query OK, 1 row affected (0.01 sec)

mysql> select * from student;
+-----+--------------+------------+
| sid | student_name | DOB        |
+-----+--------------+------------+
|   5 | Harry        | 2013-04-12 |
|   6 | Jhon         | 2010-02-15 |
|   7 | Mary         | 2014-05-16 |
|   8 | Kay          | 2013-11-16 |
+-----+--------------+------------+
```

Q] Write a Procedure to display SID & Student.

```
mysql> delimiter //
mysql> create procedure myprocedure()
    -> select sid,student_name from student
    -> //
Query OK, 0 rows affected (0.55 sec)

mysql> call myprocedure()//
+-----+--------------+
| sid | student_name |
+-----+--------------+
|   5 | Harry        |
|   6 | Jhon         |
|   7 | Mary         |
|   8 | Kay          |
```

Q] Write a procedure which gets the name of the student when the student id is passed ?

```
mysql> create procedure stud(IN id INT(5),OUT name varchar(9))
    -> begin
    -> select student_name into name
    -> from student
    -> where sid=id;
    -> end//
Query OK, 0 rows affected (0.01 sec)

mysql> call stud(5,@x)//
Query OK, 0 rows affected (0.00 sec)

mysql> select @x//
+--------+
| @x     |
+--------+
| Harry |
+--------+
1 row in set (0.00 sec)

mysql> call stud(7,@x)//
Query OK, 0 rows affected (0.00 sec)

mysql> select @x//
+------+
| @x   |
+------+
| Mary |
+------+
1 row in set (0.00 sec)

mysql> call stud(5,@x);
    -> select @x;
    -> //
Query OK, 0 rows affected (0.00 sec)

+--------+
| @x     |
+--------+
| Harry |
+--------+
```

Q] Write a procedure cleanup() to delete all the students records from student table.

```
mysql> create procedure cleanup()
   -> delete from student;
   -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call cleanup()//
Query OK, 4 rows affected (0.03 sec)

mysql> select * from student;//
Empty set (0.00 sec)
```

2.*FUNCTIONS*

Functions: these subprograms return a single value, mainly used to compute and return a value.
Creating a Function:

A standalone function is created using the CREATE FUNCT ION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE  FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
BEGIN
< function_body >
RETURN variable
END [function_name];
```

Where,

function-name specifies the name of the function.
[OR REPLACE] option allows modifying an existing function.
The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
T he function must contain a **return** statement.
RETURN clause specifies that data type you are going to return from the function.

function-body contains the executable part.

Example:

Following example illustrates creating and calling a standalone function. The function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table.

```
delimiter &
mysql> create function hello(s char(20))
   -> returns char(50)
   -> return concat('hello,s,!');
   ->  &
```

When above code is executed using MYSQL prompt, it will produce the following result:

```
Query OK, 0 rows affected (0.01 sec)
```

Calling a Function

While creating a function, you give a definition of what the function has to do. T o use a function, you will have to call that function to perform the defined task. When a program calls a function, program control is transferred to the called function.

A called function performs defined task and when its return statement is executed or when it last end statement is reached, it returns program control back to the main program.

T o call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function from an anonymous block:

```
->select hello('world');
```

When the above code is executed at SQL prompt, it produces the following result:

```
-> Hell  world
```

```
mysql> delimiter *
mysql> create function add1(a int, b int) returns int
   -> return (a+b);
   -> select add1(10,20);
   -> *
Query OK, 0 rows affected (0.00 sec)

+-------------+
| add1(10,20) |
+-------------+
```

```
|         30 |
+-------------+
1 row in set (0.02 sec)
```

Example:

The following is one more example which demonstrates Declaring , Defining , and Invoking a Simple MYSQL Function that computes and returns the maximum of two values.

```
mysql> delimiter //
mysql> CREATE FUNCTION grt(a INT,b INT,c INT) RETURNS INT
    -> BEGIN
    -> if a>b AND a>c then
    -> RETURN a;
    -> end if;
    -> if b>c AND b>a then
    -> RETURN b;
    -> end if;
    -> RETURN c;
    -> end;
    -> //
Query OK, 0 rows affected (0.12 sec)

mysql> select grt(23,78,98);
    -> //
+---------------+
| grt(23,78,98) |
+---------------+
|           98 |
+---------------+
1 row in set (0.05 sec)

mysql> select grt(23,98,72);
    -> //
+---------------+
| grt(23,98,72) |
+---------------+
|           98 |
+---------------+
1 row in set (0.01 sec)

mysql> select grt(45,2,3); //
+-------------+
| grt(45,2,3) |
+-------------+
|         45 |
```

```
+-------------+
1 row in set (0.00 sec)

mysql> delimiter //
mysql> CREATE FUNCTION odd_even(a INT) RETURNS varchar(20)
    -> BEGIN
    -> if a%2=0 then
    -> RETURN 'even';
    -> end if;
    -> RETURN 'odd';
    -> end;
    -> //
Query OK, 0 rows affected (0.06 sec)

mysql> select odd_even(54);
    -> //
+--------------+
| odd_even(54) |
+--------------+
| even         |
+--------------+
1 row in set (0.03 sec)

mysql> select odd_even(51); //
+--------------+
| odd_even(51) |
+--------------+
| odd          |
+--------------+
1 row in set (0.00 sec)
```

Conclusion: performed implementation of procedures and functions in MYSQL successfully.

Department of Infromation Technology

Software Laborotory-I

Assignment No:-6 (PART-B)

Problem Statement

1) Write a PL/SQL block or Stored Procedure to calculate the total and percentage of minimum 10 students.

```
mysql> use PVGTE;
Database changed
mysql> show tables;
+-----------------+
| Tables_in_PVGTE |
+-----------------+
| student         |
+-----------------+
1 rows in set (0.00 sec)
mysql> select * from student;
+------+--------+------+------+------+-------+------------+
| sno  | name   | sub1 | sub2 | sub3 | total | percentage |
+------+--------+------+------+------+-------+------------+
|    1 | AMIT   |   55 |   66 |   77 |  NULL |       NULL |
|    2 | AJIT   |   66 |   44 |   77 |  NULL |       NULL |
|    3 | SUJIT  |   50 |   60 |   70 |  NULL |       NULL |
|    4 | JIT    |   56 |   74 |   70 |  NULL |       NULL |
|    5 | RAHUL  |   50 |   50 |   48 |  NULL |       NULL |
|    6 | KAPIL  |   65 |   45 |   75 |  NULL |       NULL |
|    7 | RAJ    |   52 |   60 |   50 |  NULL |       NULL |
|    8 | SUSHIL |   40 |   30 |   35 |  NULL |       NULL |
|    9 | RAMESH |   50 |   50 |   50 |  NULL |       NULL |
|   10 | SUNIL  |   60 |   40 |   70 |  NULL |       NULL |
+------+--------+------+------+------+-------+------------+
```

```
mysql> DELIMITER //
mysql> create procedure CalPercentage()
    ->     begin
    ->     declare s1,s2,s3,tot int(3);
    ->     declare id int(10);
    ->     declare per float(5,2);
    ->     declare i,cnt int(3);
    ->     SET i=0;
    ->     select count(*) into cnt from student;
    ->     WHILE i <= cnt DO
    ->         select sno,sub1,sub2,sub3 into id,s1,s2,s3 from student limit i,1;
    ->         SET tot=(s1+s2+s3);
    ->         SET per=(tot/3);
    ->         update student set total=tot,percentage=per where sno=id;
    ->         SET i=i+1;
    ->     END WHILE;
    ->     end
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> call CalPercentage()//
Query OK, 0 rows affected (0.62 sec)

mysql> select * from student//
+------+--------+------+------+------+-------+------------+
| sno  | name   | sub1 | sub2 | sub3 | total | percentage |
+------+--------+------+------+------+-------+------------+
|    1 | AMIT   |   55 |   66 |   77 |   198 |      66.00 |
|    2 | AJIT   |   66 |   44 |   77 |   187 |      62.33 |
|    3 | SUJIT  |   50 |   60 |   70 |   180 |      60.00 |
|    4 | JIT    |   56 |   74 |   70 |   200 |      66.67 |
|    5 | RAHUL  |   50 |   50 |   48 |   148 |      49.33 |
|    6 | KAPIL  |   65 |   45 |   75 |   185 |      61.67 |
|    7 | RAJ    |   52 |   60 |   50 |   162 |      54.00 |
```

```
|    8 | SUSHIL |   40 |   30 |   35 |   105 |      35.00 |
|    9 | RAMESH |   50 |   50 |   50 |   150 |      50.00 |
|   10 | SUNIL  |   60 |   40 |   70 |   170 |      56.67 |
+------+--------+------+------+------+-------+------------+
10 rows in set (0.00 sec)

mysql> DELIMITER ;
```

2) Write a PL/SQL function which accepts basic salary of an employee and returns a Gross salary of an employee.

Gross salary = Basic + HRA + DA + TA

Where HRA = 50% OF BASIC

DA = 100% OF BASIC

TA = 20% OF (BASIC + HRA)

```
create function GrossSalary(eid int(10))
-> returns int
-> deterministic
-> begin
-> declare bs int;
-> declare HRA,DA,TA,GS float;
-> select sal into bs from emp where id = eid;
-> set HRA = 0.5*bs;
-> set DA =1.0*bs;
-> set TA =0.2*(bs+hra);
-> set GS =bs+HRA+DA+TA;
-> return GS;
-> end;
-> //
Query OK, 0 rows affected (0.00 sec)
```

```
select GrossSalary(1) //
+--------+
| GrossSalary(1) |
+--------+
| 2800 |
+--------+
1 row in set (0.00 sec)
```

Assignment No. 7

Title: Write a PL/SQL block to implement all types of cursor.

Aim: Write a PL/SQL block to implement cursors.

Objective: 1] to understand the basic concept of cursors used in PL/SQL

Theory:

1] Cursor its use:

- 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.

2] Types of 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 , UPDAT E and DELET E) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDAT E and

DELET E operations, the cursor identifies the rows that would be affected.

Attribute	Desc ription
%FOUND	Returns T RUE if an INSERT , UPDAT E, or DELET E statement affected one or more rows or a SELECT INT O statement returned one or more rows. Otherwise, it returns FALSE.
%NOT FOUND	T he logical opposite of %FOUND. It returns T RUE if an INSERT , UPDAT E, or DELET E statement affected no rows, or a SELECT INT O 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 , UPDAT E, or DELET E statement, or returned by a SELECT INT O 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.
T he syntax for creating an explicit cursor is :
CURSOR cursor_name IS select_statement;

Working with an explicit cursor involves four steps:
Declaring the cursor for initializing in the memory
Opening the cursor for allocating memory
Fetching the cursor for retrieving data
Closing the cursor to release 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 memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open 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 aboveopened
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 above-opened cursor as
follows:

```
CLOSE c_customers;
```

Cursor Example

Example 1

Create table emp_tbl as follows

Emp_tbl(first_name,last_name,salary);

Write a procedure with cursor to display employees first name and last name whose salary greater than 1000;

```
drop procedure if exists pcursor;

create procedure pcursor()

begin

DECLARE fn varchar(30);

declare ln varchar(30);

DECLARE cur1 CURSOR FOR SELECT first_name,last_name from emp_tbl where salary>1000;

OPEN  cur1;

read_loop: LOOP

   FETCH cur1 INTO fn,ln;

select concat(fn,'   ',ln) as name;

end loop;

CLOSE  cur1;

END
```

Example 2

```
create table t1(id int,data int);( id char(16),data int)

create table t2(i int);

create table t3(i1 int,12 int);     //t3 table blank (i1 char(16),i2 int)

CREATE PROCEDURE curdemo()
BEGIN
```

```
DECLARE done INT DEFAULT FALSE;
DECLARE a CHAR(16);
DECLARE b, c INT;
DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
OPEN cur1;
OPEN cur2;
read_loop: LOOP
   FETCH cur1 INTO a, b;
   FETCH cur2 INTO c;
  IF b < c THEN
 INSERT INTO test.t3 VALUES (a,b);
   ELSE
     INSERT INTO test.t3 VALUES (a,c);
  END IF;
 END LOOP;
 CLOSE cur1;
 CLOSE cur2;
END;
```

FAQs:

1) What is cursor? State its type.
2) Explain difference between implicit cursor and explicit cursors.

Conclusion: Thoroughly understood the basic concept of cursors used in PL/SQL.

Department of Infromation Technology

Software Laborotory-I

Assignment No:-6 (PART-B)

Problem Statement

Consider Student table containing information of Exam_no, Name, Sub1, Sub2,Sub3 marks of minimum 10 students.

Write a cursor in which percentages for each student are calculated.

If it is greater than or equal to 60, make that entry into Stud_First (Exam_no, Name, Sub1, Sub2,Sub3, Percent) table.

If less then 60, make that entry into Stud_Pass (Exam_no, Name, Sub1, Sub2, Sub3, Percent) table.

If student is fail in any subject(less than 40 in any subject), make that entry into Stud_fail(Exam_no,Name,sub1,sub2,sub3) table.

```
mysql> create table Student_Info(Exam_No int, Stu_Name varchar(20), Sub1 int, Sub2 int, Sub3 int);
Query OK, 0 rows affected (0.29 sec)

mysql> create table Stud_First(Exam_No int, Stu_Name varchar(20), Sub1 int, Sub2 int, Sub3 int, Percent float );
Query OK, 0 rows affected (0.26 sec)

mysql> create table Stud_Pass(Exam_No int, Stu_Name varchar(20), Sub1 int, Sub2 int, Sub3 int,Percent float );
Query OK, 0 rows affected (0.26 sec)
```

```
mysql> create table Stud_Fail(Exam_No int, Stu_Name varchar(20), Sub1 int,
Sub2 int, Sub3 int,Percent float );
Query OK, 0 rows affected (0.27 sec)


mysql> select * from Student_Info;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|     101 | AAA      |   70 |   80 |   65 |
|     102 | BBB      |   50 |   50 |   50 |
|     103 | CCC      |   35 |   40 |   20 |
|     104 | DDD      |   60 |   60 |   70 |
|     105 | EEE      |   55 |   65 |   75 |
|     106 | FFF      |   15 |   15 |   15 |
|     107 | GGG      |   80 |   75 |   85 |
|     108 | HHH      |   30 |   45 |   45 |
|     109 | III      |   30 |   50 |   60 |
|     110 | JJJ      |   60 |   50 |   80 |
+---------+----------+------+------+------+
10 rows in set (0.00 sec)


mysql>
mysql> CREATE PROCEDURE InsAccPerc()
    -> BEGIN
    -> Declare examno int;
    -> Declare name varchar(25);
    -> Declare sub1,sub2,sub3 int;
    -> Declare perc float;
    -> Declare count int;
    -> Declare C1 CURSOR for SELECT * from Student_Info;
    -> select count(*) into count from Student_Info;
```

```
   -> OPEN C1;
   -> WHILE(count > 0) DO
   -> FETCH C1 into examno,name,sub1,sub2,sub3;
   -> SET perc=0;
   -> SET perc=(sub1+sub2+sub3)/3;
     -> if (sub1<40 || sub2 < 40 || sub3 < 40) then insert into Stud_Fail
values(examno,name,sub1,sub2,sub3,perc);
          -> elseif (perc >= 60.00) then insert into Stud_First
values(examno,name,sub1,sub2,sub3,perc);
   -> else insert into Stud_Pass values(examno,name,sub1,sub2,sub3,perc);
   -> end if;
   -> SET count=count-1;
   -> END WHILE;
   -> CLOSE C1;
   -> END
   -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call InsAccPerc()//
Query OK, 1 row affected (0.59 sec)

mysql> select * from Stud_First;
   -> //
+---------+----------+------+------+------+---------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 | Percent |
+---------+----------+------+------+------+---------+
|     101 | AAA      |   70 |   80 |   65 | 71.6667 |
|     101 | AAA      |   70 |   80 |   65 | 71.6667 |
|     104 | DDD      |   60 |   60 |   70 | 63.3333 |
|     105 | EEE      |   55 |   65 |   75 |      65 |
|     107 | GGG      |   80 |   75 |   85 |      80 |
|     110 | JJJ      |   60 |   50 |   80 | 63.3333 |
+---------+----------+------+------+------+---------+
6 rows in set (0.00 sec)
```

```
mysql> select * from Stud_Pass//
+---------+----------+------+------+------+---------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 | Percent |
+---------+----------+------+------+------+---------+
|     102 | BBB      |   50 |   50 |   50 |      50 |
+---------+----------+------+------+------+---------+
2 rows in set (0.00 sec)

mysql> select * from Stud_Fail//
+---------+----------+------+------+------+---------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 | Percent |
+---------+----------+------+------+------+---------+
|     103 | CCC      |   35 |   40 |   20 | 31.6667 |
|     106 | FFF      |   15 |   15 |   15 |      15 |
|     108 | HHH      |   30 |   45 |   45 |      40 |
|     109 | III      |   30 |   50 |   60 | 46.6667 |
+---------+----------+------+------+------+---------+
4 rows in set (0.00 sec)

mysql>
```

Assignment No. 8

Title: Execute DDL statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.

Aim: Execute DDL statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.

Objective: Understand the concept of view and perform various operations on view

Theory:

What is View?

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;
```

Then, we can query the view as follows:

```
SELECT * FROM [Current Product List];
```

MySQL Create View with JOIN

CREATE VIEW command can be used along with a JOIN statement.

Example :
Sample table : category
Sample table : purchase

```
CREATE VIEW view_purchase
AS SELECT a.cate_id,a.cate_descrip, b.invoice_no,
b.invoice_dt,b.book_name
FROM category a,purchase b
WHERE a.cate_id=b.cate_id;
```

The above MySQL statement will create a view 'view_purchase' along with a JOIN statement.

The JOIN statement here retrieves cate_id, cate_descrip from category table and invoice_no, invoice_dt and book_name from purchase table if cate_id of category table and that of purchase are same.

MySQL Create View with LIKE

CREATE VIEW command can be used with LIKE operator.

Example :
Sample table : author
Code :

```
1. CREATE VIEW view_author
2. AS SELECT *
3. FROM author
4. WHERE aut_name
5. NOT LIKE 'T%' AND aut_name NOT LIKE 'W%';
```

The above MySQL statement will create a view 'view_author' taking all the records of author table, if (A)name of the author (aut_name) does not start with 'T' and (B) name of the author (aut_name) does not start with 'W'.

MySQL Create View using Subquery

CREATE VIEW command can be used with subqueries.

Example :

Sample table : purchase
Sample table : book_mast
Code :

```
1. CREATE VIEW view_purchase
2. AS SELECT invoice_no,book_name,cate_id
3. FROM purchase
4. WHERE cate_id=  (SELECT cate_id FROM book_mast WHERE no_page=201)
5.
```

Create table Employee(ID,First_Name,Last_Name,Start_Date,End_Date,Salary,City).

1. Create a simple view to display First_Name,Last_Name from employee.
2. Create a view to display First_Name,Last_Name of those employee whose salary is greater than 2000 from employee table.
3. Create a view to display first_name starting with “S” and last_name end with “t”.

Conclusion: Implemented Views and performed operation on view

Department of Infromation Technology

Software Laborotory-I

Assignment No:-8 (PART-B)

Problem Statement

Consider Student table containing information of

Student(Exam_no,Name,Sub1,Sub2,Sub3) marks of minimum 10 students.

Create view on Student table and Try to update the base table(Student) using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables.

```
mysql> create table Student_Info(Exam_No int, Stu_Name varchar(20), Sub1 int, Sub2 int, Sub3 int);
Query OK, 0 rows affected (0.29 sec)

mysql> insert into Student_Info values(101,'AAA',70,80,65);
Query OK, 1 row affected (0.09 sec)

mysql> insert into Student_Info values(102,'BBB',50,50,50);
Query OK, 1 row affected (0.08 sec)

mysql> insert into Student_Info values(103,'CCC',35,40,20);
Query OK, 1 row affected (0.06 sec)

mysql> insert into Student_Info values(104,'DDD',60,60,70);
Query OK, 1 row affected (0.06 sec)
```

```
mysql> insert into Student_Info values(105,'EEE',55,65,75);
Query OK, 1 row affected (0.05 sec)

mysql> insert into Student_Info values(106,'FFF',15,15,15);
Query OK, 1 row affected (0.06 sec)

mysql> insert into Student_Info values(107,'GGG',80,75,85);
Query OK, 1 row affected (0.05 sec)

mysql> insert into Student_Info values(108,'HHH',30,45,45);
Query OK, 1 row affected (0.05 sec)

mysql> insert into Student_Info values(109,'III',30,50,60);
Query OK, 1 row affected (0.05 sec)

mysql> insert into Student_Info values(110,'JJJ',60,50,80);
Query OK, 1 row affected (0.06 sec)

mysql> select * from Student_Info;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|     101 | AAA      |   70 |   80 |   65 |
|     102 | BBB      |   50 |   50 |   50 |
|     103 | CCC      |   35 |   40 |   20 |
|     104 | DDD      |   60 |   60 |   70 |
|     105 | EEE      |   55 |   65 |   75 |
|     106 | FFF      |   15 |   15 |   15 |
|     107 | GGG      |   80 |   75 |   85 |
|     108 | HHH      |   30 |   45 |   45 |
|     109 | III      |   30 |   50 |   60 |
|     110 | JJJ      |   60 |   50 |   80 |
+---------+----------+------+------+------+
```

```
10 rows in set (0.00 sec)

mysql> create view StudentView as select * from Student_Info;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from StudentView;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   15 |   15 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
|    109 | III      |   30 |   50 |   60 |
|    110 | JJJ      |   60 |   50 |   80 |
+---------+----------+------+------+------+
10 rows in set (0.00 sec)

mysql>

mysql> update Student_Info set Sub1=35,Sub2=45 where Exam_No=106;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Student_Info;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
```

```
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
|    109 | III      |   30 |   50 |   60 |
|    110 | JJJ      |   60 |   50 |   80 |
+---------+----------+------+------+------+
10 rows in set (0.00 sec)

mysql> select * from StudentView;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
|    109 | III      |   30 |   50 |   60 |
|    110 | JJJ      |   60 |   50 |   80 |
+---------+----------+------+------+------+
10 rows in set (0.00 sec)

mysql> delete from StudentView where Exam_No=110;
Query OK, 1 row affected (0.06 sec)

mysql> select * from StudentView;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
```

```
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
|    109 | III      |   30 |   50 |   60 |
+---------+----------+------+------+------+
9 rows in set (0.00 sec)

mysql> select * from Student_Info;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
|    109 | III      |   30 |   50 |   60 |
+---------+----------+------+------+------+
9 rows in set (0.01 sec)

mysql> delete from Student_Info where Exam_No=109;
Query OK, 1 row affected (0.06 sec)

mysql> select * from Student_Info;
+---------+----------+------+------+------+
```

```
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
+---------+----------+------+------+------+
8 rows in set (0.00 sec)

mysql> select * from StudentView;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   35 |   40 |   20 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
+---------+----------+------+------+------+
8 rows in set (0.00 sec)

mysql>
mysql> update StudentView set Sub1=50,Sub2=50,Sub3=50 where
Exam_No=103
    -> ;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from StudentView;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   50 |   50 |   50 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
+---------+----------+------+------+------+
8 rows in set (0.00 sec)

mysql> select * from Student_Info;
+---------+----------+------+------+------+
| Exam_No | Stu_Name | Sub1 | Sub2 | Sub3 |
+---------+----------+------+------+------+
|    101 | AAA      |   70 |   80 |   65 |
|    102 | BBB      |   50 |   50 |   50 |
|    103 | CCC      |   50 |   50 |   50 |
|    104 | DDD      |   60 |   60 |   70 |
|    105 | EEE      |   55 |   65 |   75 |
|    106 | FFF      |   35 |   45 |   15 |
|    107 | GGG      |   80 |   75 |   85 |
|    108 | HHH      |   30 |   45 |   45 |
+---------+----------+------+------+------+
8 rows in set (0.00 sec)

mysql>
```