PRACTICAL NO. 9

To create triggers for various events such as insertion, updation, etc.,

PROCEDURE

a) PL/SQL Syntax:

TRIGGER

A Trigger is a stored procedure that defines an action that the database automatically take when some database-related event such as Insert, Update or Delete occur.

TRIGGER VS. PROCEDURE VS CURSOR

TRIGGER	PROCEDURES	CURSORS
These are named PL/SQL blocks.	These are named PL/SQL blocks.	These are named PL/SQL blocks.
These are invoked automatically.	User as per need invokes these.	These can be created both explicitly and implicitly.
These can"t take parameters.	These can take parameters.	These can take parameters.
These are stored in database.	These are stored in database.	These are not stored in database.

TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before**: It fires the trigger before executing the trigger statement.
- **After**: It fires the trigger after executing the trigger statement.
- **For each row**: It specifies that the trigger fires once per row.
- **For each statement**: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

Row Level Trigger vs. Statement Level Trigger:

Row Level Trigger	**Statement Level Trigger**
These are fired for each row affected by the DML statement.	These are fired once for the statement instead of the no of rows modified by it.
These are used for generating/checking the values begin inserted or updated.	These are used for generated the summary information.

Before trigger vs. after trigger

Before Triggers	After Triggers
Before triggers are fired before the DML statement is actually executed.	After triggers are fired after the DML statement has finished execution.

Sytax:

Create or replace trigger <trg_name> Before /After Insert/Update/Delete
[of column_name, column_name….]
on
<table_name>
[for each row]
[when
condition]
begin
---statement
end;

Consider the following Tables:

EMPLOYEE(Emp_id, EMP_name,Job_name,Manager_id,Hire_date,Salary,Deptno)

DEPARTMENT(Deptno, Dname, MGRSSN)

PROJECT(Pname,Pno,Plocation,Deptno)

```
 emp_id | emp_name | job_name  | manager_id | hire_date  | salary  | E_Bonus | dep_no

  68319 | KAYLING  | PRESIDENT |            | 1991-11-18 | 6000.00 |   300.00   1001
  66928 | BLAZE    | MANAGER   |      68319 | 1991-05-01 | 2750.00 |   200.00   3001
  67832 | CLARE    | MANAGER   |      68319 | 1991-06-09 | 2550.00 |   200.00   1001
  65646 | JONAS    | MANAGER   |      68319 | 1991-04-02 | 2957.00 |   200.00   2001
  67858 | SCARLET  | ANALYST   |      65646 | 1997-04-19 | 3100.00 |   250.00   2001
  69062 | FRANK    | ANALYST   |      65646 | 1991-12-03 | 3100.00 |   250.00   2001
  63679 | SANDRINE | CLERK     |      69062 | 1990-12-18 |  900.00 |   150.00   2001
  64989 | ADELYN   | SALESMAN  |      66928 | 1991-02-20 | 1700.00 |   180.00   3001
  65271 | WADE     | SALESMAN  |      66928 | 1991-02-22 | 1350.00 |   180.00   3001
  66564 | MADDEN   | SALESMAN  |      66928 | 1991-09-28 | 1350.00 |   180.00   3001
  68454 | TUCKER   | SALESMAN  |      66928 | 1991-09-08 | 1600.00 |   180.00   3001
  68736 | ADNRES   | CLERK     |      67858 | 1997-05-23 | 1200.00 |   150.00   2001
  69000 | JULIUS   | CLERK     |      66928 | 1991-12-03 | 1050.00 |   150.00   3001
  69324 | MARKER   | CLERK     |      67832 | 1992-01-23 | 1400.00 |   150.00   1001
```

Department Table

deptno	dname	Citylocation	dCountry
1001	Accounting	New York	United States of America,
2001	Research	Dallas	United States
3001	Sales	Chicago	United States of America
4001	Marketing	Los Angeles	United States

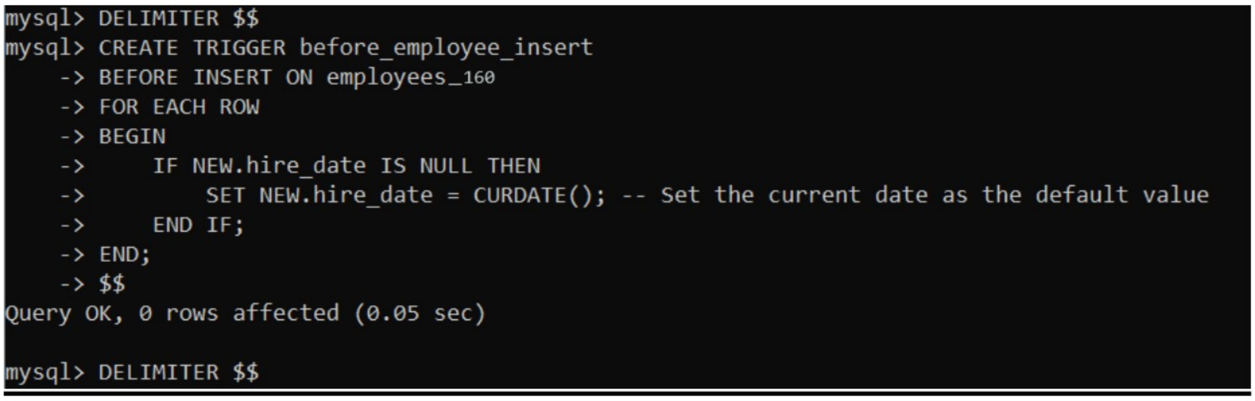
Project Table

Pno	Pname	PCitylocation	PCountry
111	P_1	New York	United States of America,
112	P_2	Dallas	United States
113	P_3	Chicago	United States of America
114	P_4	Denmark	northern Europe
115	P_5	Paris	France
116	P_6	Chicago	United States of America

Write a query for the following:-

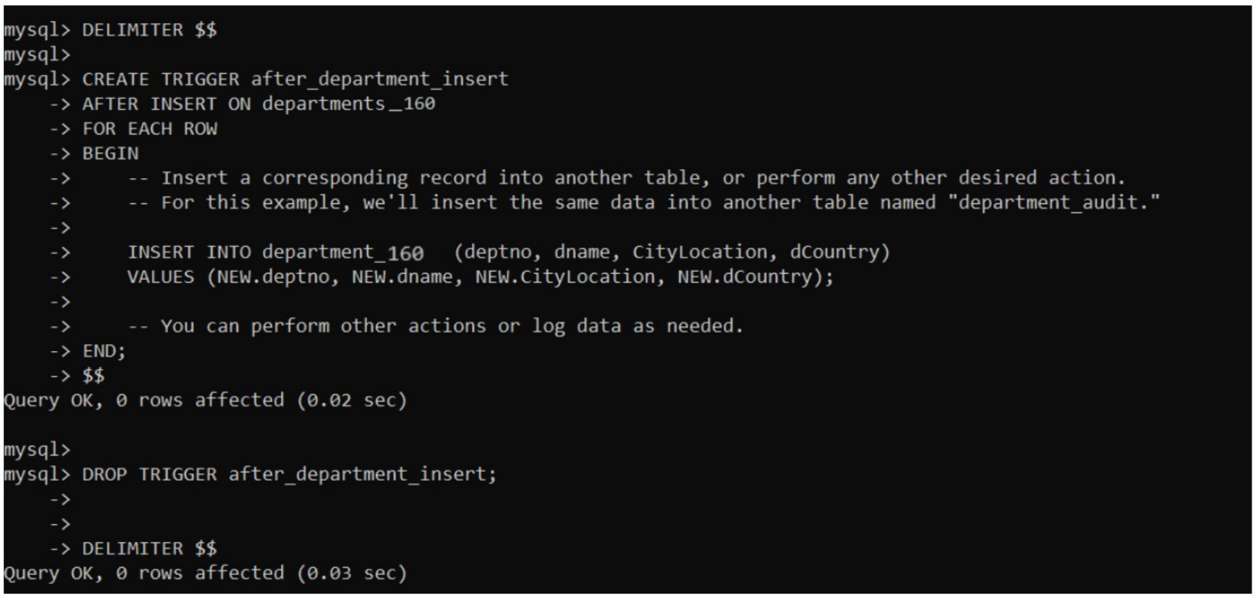
Practical – 09

Q1. Create a trigger before insert on employee table on each row.



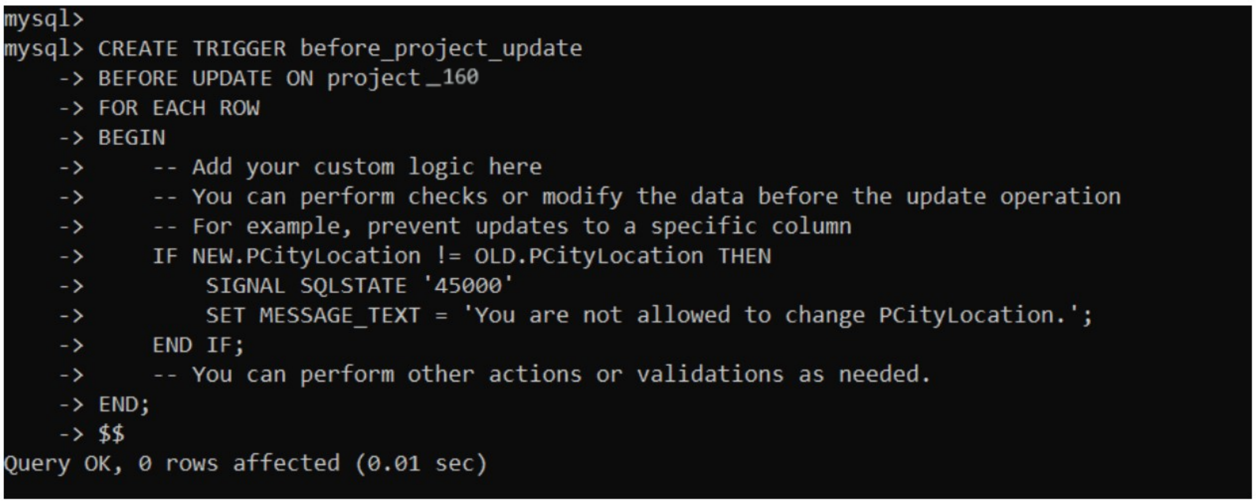
```
mysql> DELIMITER $$
mysql> CREATE TRIGGER before_employee_insert
    -> BEFORE INSERT ON employees_160
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.hire_date IS NULL THEN
    ->         SET NEW.hire_date = CURDATE(); -- Set the current date as the default value
    ->     END IF;
    -> END;
    -> $$
Query OK, 0 rows affected (0.05 sec)

mysql> DELIMITER $$
```

Q2. Create a trigger after into department table and Drop a trigger on department table.



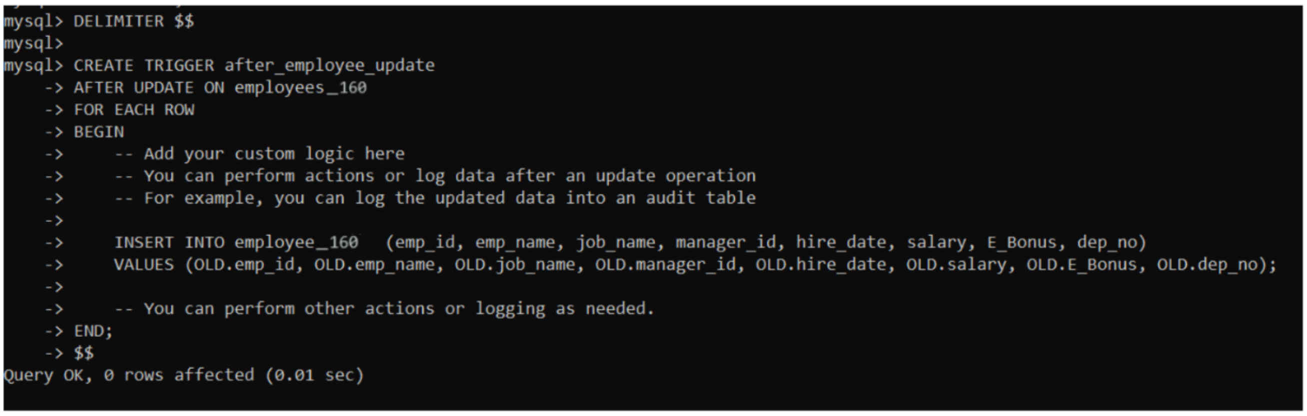
```
mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER after_department_insert
    -> AFTER INSERT ON departments_160
    -> FOR EACH ROW
    -> BEGIN
    ->     -- Insert a corresponding record into another table, or perform any other desired action.
    ->     -- For this example, we'll insert the same data into another table named "department_audit."
    ->
    ->     INSERT INTO department_160   (deptno, dname, CityLocation, dCountry)
    ->     VALUES (NEW.deptno, NEW.dname, NEW.CityLocation, NEW.dCountry);
    ->
    ->     -- You can perform other actions or log data as needed.
    -> END;
    -> $$
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DROP TRIGGER after_department_insert;
    ->
    ->
    -> DELIMITER $$
Query OK, 0 rows affected (0.03 sec)
```

Q3. Create a trigger on project table before update.



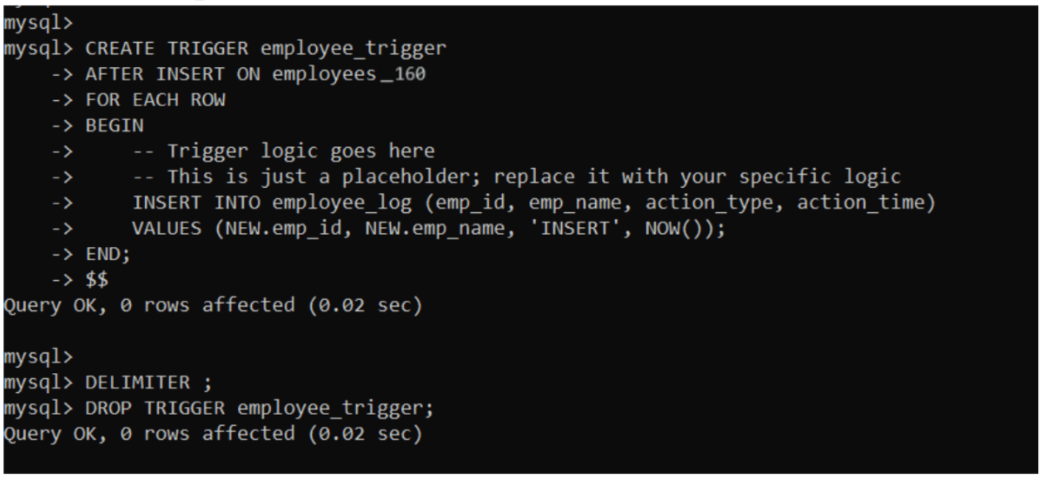
```
mysql>
mysql> CREATE TRIGGER before_project_update
    -> BEFORE UPDATE ON project_160
    -> FOR EACH ROW
    -> BEGIN
    ->     -- Add your custom logic here
    ->     -- You can perform checks or modify the data before the update operation
    ->     -- For example, prevent updates to a specific column
    ->     IF NEW.PCityLocation != OLD.PCityLocation THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'You are not allowed to change PCityLocation.';
    ->     END IF;
    ->     -- You can perform other actions or validations as needed.
    -> END;
    -> $$
Query OK, 0 rows affected (0.01 sec)
```

Q4. Create a trigger on employee table for update.



```
mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER after_employee_update
    -> AFTER UPDATE ON employees_160
    -> FOR EACH ROW
    -> BEGIN
    ->     -- Add your custom logic here
    ->     -- You can perform actions or log data after an update operation
    ->     -- For example, you can log the updated data into an audit table
    ->
    ->     INSERT INTO employee_160   (emp_id, emp_name, job_name, manager_id, hire_date, salary, E_Bonus, dep_no)
    ->     VALUES (OLD.emp_id, OLD.emp_name, OLD.job_name, OLD.manager_id, OLD.hire_date, OLD.salary, OLD.E_Bonus, OLD.dep_no);
    ->
    ->     -- You can perform other actions or logging as needed.
    -> END;
    -> $$
Query OK, 0 rows affected (0.01 sec)
```

Q6. Create a trigger on employee
table and drop it



```
mysql>
mysql> CREATE TRIGGER employee_trigger
    -> AFTER INSERT ON employees_160
    -> FOR EACH ROW
    -> BEGIN
    ->     -- Trigger logic goes here
    ->     -- This is just a placeholder; replace it with your specific logic
    ->     INSERT INTO employee_log (emp_id, emp_name, action_type, action_time)
    ->     VALUES (NEW.emp_id, NEW.emp_name, 'INSERT', NOW());
    -> END;
    -> $$
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> DROP TRIGGER employee_trigger;
Query OK, 0 rows affected (0.02 sec)
```