# **ASSIGNMENT 5**

**AIM: -**

Database Trigger (Row level and Statement level triggers, Before and After Triggers): Write a database trigger on Employee 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 to a new table when the Employee table is updated. Employee (employee no, employee name, join\_date, designation, salary).**

# **OBJECTIVE**: -

To implement row and statement level trigger.

# **THEORY**: -

In programs sometimes it is required to execute certain code followed by certain events and this requirement can be achieved in PL/SQL through triggers.

Triggers are stored programs that are fired automatically when some event occurs. The code to be fired can be defined as per the requirement.

Oracle has also provided the facility to mention the event upon which the trigger needs to be fire and the timing of the execution.

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 −

1.A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)

2.A database definition (DDL) statement (CREATE, ALTER, or DROP).

3.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.

**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;

# **Syntax Explanation:**

**BEFORE/ AFTER** will specify the event timings. **INSERT/UPDATE/LOGON/CREATE/etc.** will specify the event for which the trigger needs to be fired.

**ON clause** will specify on which object the above mentioned event is valid. For example, this will be the table name on which the DML event may occur in the case of DML Trigger.

**Command "FOR EACH ROW" will specify the ROW level trigger.**

**WHEN clause** will specify the additional condition in which the trigger needs to fire.

The declaration part, execution part, exception handling part is same as that of the other PL/SQL blocks. Declaration part and exception handling part are optional.

# **BENEFITS: -**

1. Triggers can be written for the following purposes

2. Generating some derived column values automatically

3. Enforcing referential integrity

4.Event logging and storing information on table access

5. Auditing

6. Synchronous replication of tables

7. Imposing security authorizations 8. Preventing invalid transactions

**INPUT: -**

# TABLE EMPLOYEE\_

Select \* from employee\_; **TABLE EMPLOYEE\_TABLE**

Create Table employee\_table(employee\_no int(5),employee\_name varchar(20),join\_date varchar(20),designation varchar(20),salary int(10));

Select \* from employee\_table;

**After INSERT** delimiter /

create trigger emp

AFTER INSERT on employee\_ for each row

begin

insert into employee\_table values(NEW.employee\_no,NEW.employee\_name,NEW.join\_date,NE W.designation,NEW.salary);

end;

/

**BEFORE DELETE**

delimiter /

create trigger emp10

BEFORE DELETE on employee\_ for each row

begin

DELETE FROM EMPLOYEE\_TABLE where

OLD.employee\_no=EMPLOYEE\_TABLE.employee\_no; end;

/

**AFTER DELETE**

delimiter /

create trigger emp6

AFTER DELETE on employee\_ for each row

begin

DELETE FROM EMPLOYEE\_TABLE where

OLD.employee\_no=employee\_table.employee\_no; end;

/

**BEFORE UPDATE**

Delimiter /

Create trigger emp2

BEFORE UPDATE on employee\_ for each row

Begin

UPDATE EMPLOYEE\_TABLE SET

employee\_name=old.employee\_name,join\_date=old.join\_date,desi gnation=old.designation,salary=old.salary,employee\_no=old.emplo yee\_no

where

employee\_no=old.employee\_no;

End;

/

**AFTER UPDATE**

delimiter /

create trigger emp11

AFTER UPDATE on employee\_ for each row

begin

UPDATE EMPLOYEE\_TABLE SET

employee\_name=NEW.employee\_name,join\_date=NEW.join\_date,d esignation=NEW.designation,salary=NEW.salary,employee\_no=NE W.employee\_no

where

employee\_no=old.employee\_no;

end;

/

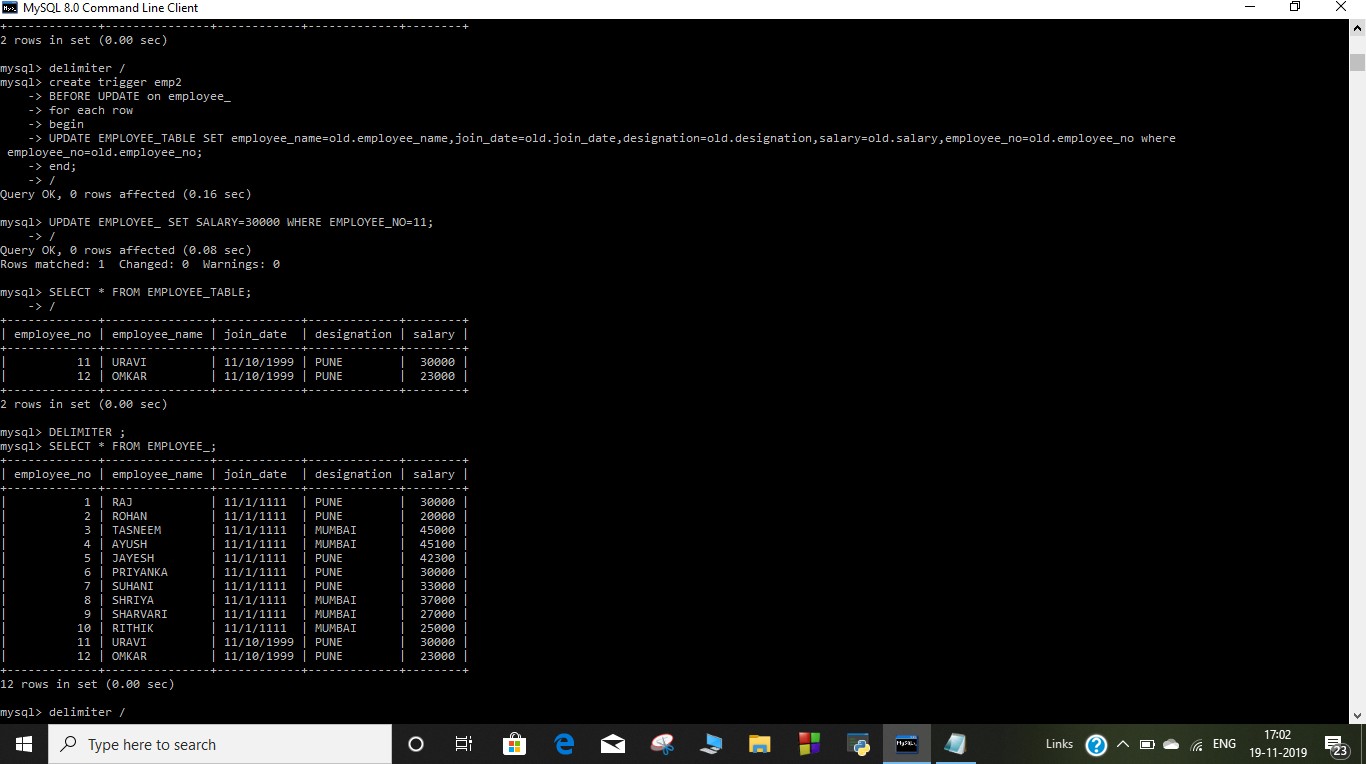
**INSERT, DELETE, UPDATE A ROW**

Insert into employee\_ values(11,'URAVI','11/10/1999','PUNE',40000); UPDATE EMPLOYEE\_ SET SALARY=30000 WHERE EMPLOYEE\_NO=11;

delete from employee\_ where employee\_no=12; delete from employee\_ where employee\_no=13; UPDATE EMPLOYEE\_ SET SALARY=200000 WHERE EMPLOYEE\_NO=8;

**DISPLAY THE TABLE**

Select \* from employee\_; Select \* from employee\_table;

**OUTPUT: -**

