RDBMS

**Unit-4: Trigger**

**Note:** -**.** The basic objective of this material is to supplement teaching and discussion in the classroom. Student is required to go for extra reading in the subject through library work.

**Triggers**

* ***Definition:*** *A trigger is a set of actions that will be executed when defined event like insert, update and delete occurs.*
* The trigger event can be following statement:
  + Insert
  + Update
  + Delete
* Trigger is defined for specific table.
* Once trigger is defined, it will automatically active.
* A table have multiple triggers defined on it. If multiple triggers defined for a given table, the order of given trigger activation is based on the trigger creation timestamp.
* ***Timestamp****: order in which trigger were created.*

**Trigger can use following statements inside code:**

* Compound statements (BEGIN / END)
* Variable declaration (DECLARE) and assignment (SET)
* Flow control statements (IF, CASE, WHILE, LOOP, WHILE, REPEAT, LEAVE, ITERATE)
* Condition declarations
* Handler declarations

**MySQL trigger limitations (Statement Limitation)**

1. Use SHOW, LOAD DATA, LOAD TABLE, BACKUP, RESTORE, FLUSH and RETURN statements.
2. Use statements that commit or rollback implicitly or explicitly such as [COMMIT, ROLLBACK,](http://www.mysqltutorial.org/mysql-transaction.aspx) START TRANSACTIO[N](http://www.mysqltutorial.org/mysql-transaction.aspx) ,LOCK/UNLOCK TABLE[S](http://www.mysqltutorial.org/mysql-table-locking/) , ALTER, CREATE, DROP, [RENAME](http://www.mysqltutorial.org/mysql-rename-table/) , etc.
3. Use [prepared statements](http://www.mysqltutorial.org/mysql-prepared-statement.aspx) such as PREPARE, EXECUTE, etc.
4. Use dynamic SQL statements. **Types of trigger**

* Trigger can be defined to fire (be activate) in two way:  **Before Trigger** 
  + Activated before integrity constraints are checked.
  + **After Trigger**
  + Occur after the trigger event executes, and after the database manager checks all constraints

**Syntax of Trigger**

*CREATE OR REPLACE TRIGGER trig\_name*

*<Trigger Time > < Trigger Event> ON table\_name*

*REFERENCING NEW | OLD AS var\_name*

*FOR EACH ROW*

*BEGIN*

*….. SQL CODE…..*

*END*

**Key Point:**

Trigger Time: BEFORE or AFTER

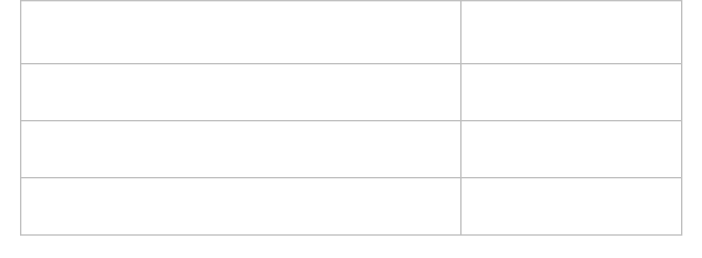
Trigger Event: INSERT | UPDATE | DELETE



BEFORE INSERT

–

activated before data is inserted into the table.



**Trigger**

**Row**

**trigger**

Before or After Insert

New

Before or After Update

Old, New

Before or After Delete

Old

* AFTER INSERT – activated after data is inserted into the table.
* BEFORE UPDATE – activated before data in the table is updated.
* AFTER UPDATE – activated after data in the table is updated.
* BEFORE DELETE – activated before data is removed from the table.
* AFTER DELETE – activated after data is removed from the table.

**Before Trigger**

* A before trigger will fire for each row in the set of affected rows before the triggering statement executes.
* Therefore, the trigger body is seeing the new data values prior to their being inserted or updated into the table.
* A BEFORE trigger is activated before integrity constraints are checked and may be violated by the trigger event.

**Example: Write a trigger to calculate and insert profit or loss whenever new record is inserted into prod\_master**.

Consider table: Prod\_master(pId, pname, quantity, cost\_price, sale\_price, profit, loss)

**Solution:**

CREATE TRIGGER pro\_profit\_loss

BEFORE INSERT ON prod\_master

FOR EACH ROW

BEGIN

IF new. cost\_price <new.sale\_price THEN

SET new.profit=new.sale\_price-new. cost\_price;

ELSEIF new.cost\_price>new.sale\_price THEN

SET new.loss=new.cost\_price-new.sale\_price;

ELSE

SET new.profit=0;

SET new.loss=0;

END IF;

END;

**Example: Write a trigger to calculate result whenever new record is inserted into test table**.

Consider table: Exam (test\_id, name, tdate, tmarks, pass\_marks, score, result)

**Solution:**

CREATE TRIGGER PASSFAIL

BEFORE INSERT ON exam

FOR EACH ROW

BEGIN

IF (new.score>= new.pass\_marks) THEN

SET new. result ='Pass';

ELSE

SET new. result ='Fail';

END IF;

End;

**Signal Statement**

* SIGNAL is the way to “return” an error.
* SIGNAL provides error information to a handler, to an outer portion of the application, or to the client.
* it provides control over the error's characteristics (error number, SQLSTATE value, message).

**Syntax**:

SIGNAL SQLSTATE VALUE SET MESSAGE\_TEXT = 'Error Message';

**Example:**

if (mark > 100 ) then signal sqlstate '10000' set MESSAGE\_TEXT='obtained Marks cannot greater than 100';

End if;

**Example: Don’t allow insertion on emp table on Sunday**

CREATE TRIGGER rest

BEFORE INSERT ON emp

FOR EACH ROW

BEGIN

IF (DAYNAME(NOW()) > 'sunday') THEN

SIGNAL SQLSTATE '10000' SET MESSAGE\_TEXT='SUNDAY INSERTION IS NOT ALLOWED'; END IF;

END;;

**Example: write a trigger to check salary of employee does not less than 0;**

CREATE TRIGGER CheckSal

BEFORE INSERT ON EMP FOR EACH ROW

BEGIN

IF (NEW.SALARY < 0 ) THEN

SIGNAL SQLSTATE '10000' SET MESSAGE\_TEXT='SALARY MUST BE GREATER THAN 0';

END IF;

END;;

**After Trigger**

 An AFTER trigger occur after the trigger event executes, and after the database manager checks all constraints that the trigger event may affect, including actions of referential constraints.

**Examples:**

1. **Write a trigger that insert record into audit\_log table whenever any record deleted from prod\_master.**

Consider table: Prod\_master(pId, pname, quantity, cost\_price, sale\_price, profit, loss) Audit\_log(pid, pname, cdate,user, action) **Solution:**

CREATE TRIGGER del\_trig

AFTER DELETE ON prod\_master

FOR EACH ROW

BEGIN

INSERT INTO Audit\_log VALUES (old.pid, old.pname, CURRENT DATE, USER, ‘Record

Deleted’);

END;

1. **Write a trigger that insert record in audit\_work whenever any changes occurs in work table.**

Consider Table: Work (empNo, ename, job, sal, comm)

Audit\_work(empno, ename,old\_sal, new\_sal,cdate) **Solution:**

CREATE TRIGGER update\_Work

AFTER UPDATE ON work

FOR EACH ROW

BEGIN

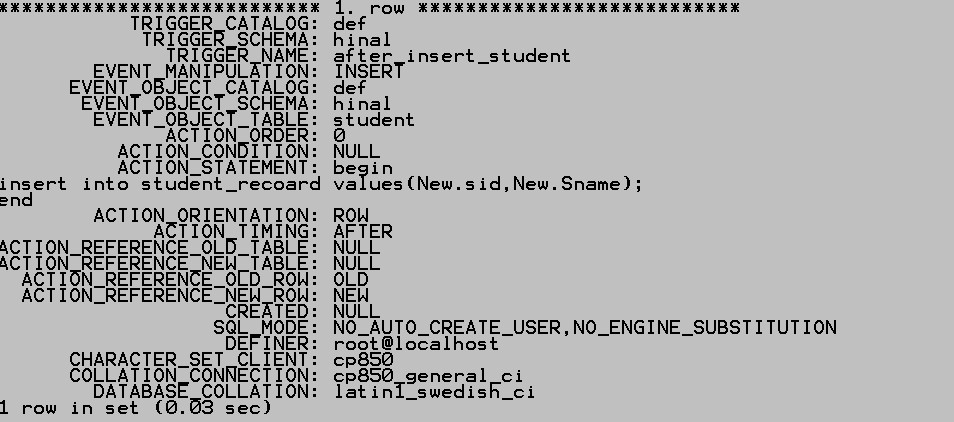
INSERT INTO Audit\_work VALUES (new. empNo, new.ename, old.sal, new.sal,

CURRENT DATE); END;

**Trigger Storage**

1. SELECT \*FROM information\_schema.triggers WHERE trigger\_schema = 'database\_name';
2. SELECT \* FROM INFORMATION\_SCHEMA.TRIGGERS where trigger\_name

='after\_insert\_student'



**Exercise**

1. Write a trigger which takes restrict the insertion on Thursday and every day after 4:30 in product table.
2. Write a trigger that maintain log of product in audit\_prod table whenever any updation on price take place in product table.

 Consider tables: Product(prodid, pname, category, unitprice)

Audit\_prod(proid, pname,category,newprice,oldprice,date)

2. Write a trigger that maintain log of the student into stud\_Remove table whenever student cancel their admission.

* Consider tables: Student(sid, course, passingpercent, percentscored,result)

Stud\_Remove(sid, course, passper,perscored,result ,Cdate)

**Cascading Trigger**

* **Definition:** Trigger can fire other trigger or same trigger or other constraint are known as cascading triggers.
* No Cascade is used to avoid cascading effects.

 No cascade is used after the trigger name.

* By default effect is cascading.
* Example: Create or replace trigger check\_id no cascade

**Example:**

CREATE OR REPLACE TRIGGER update\_Work no cascade

AFTER UPDATE ON work

REFERENCING OLD AS a NEW AS n

FOR EACH ROW

BEGIN

INSERT INTO Audit\_work VALUES (n. empNo, n.ename, a.sal, n.sal, CURRENT DATE); END;  **Result:**

* If before or after insert trigger is define on Audit\_work then it will not fire after above trigger because no cascade properties is set.

**Trigger Usage- Advantage**

1. **Data Validation**: Ensure that a new data value is within the proper range.
2. **Data Conditioning:** Implemented using triggers that fires before data record modification.
3. **Data Integrity:** Can be used to ensure cross-table dependencies are maintained.
4. **View Handling**: Instead-of triggers allows the user to control how modifications applied to view.
5. **Reduce cost:** Reducedamount of application development cost and make development faster.
6. Provide a **global environment** for your business rule: Defines once and stored in database, so available to all application.
7. **Reduce maintenance** of your application.

**Disadvantages**

1. It is called and executed invisible from the client applications,  It is difficult to figure out what happen in the database layer.
2. It increases the overhead of the database server.