Triggers (MySQL)

Triggers are the SQL codes that are automatically executed in response to certain events on a particular table. These are used to maintain the integrity of the data.
It is name PL (procedural language) / SQL block.

To understand this let's consider a hypothetical situation.
John is the marketing officer in a company. When a new customer data is entered into the company's database he has to send the welcome message to each new customer. If it is one or two customers John can do it manually, but what if the count is more than a thousand? Well in such scenario triggers come in handy.
Thus, now John can easily create a trigger which will automatically send a welcome email to the new customers once their data is entered into the database.
Important points:

- To maintain Business data in Uniform case.
- To maintain audit information of any table data.

Triggers can be made to insert, update and delete statements in SQL. We have two types of triggers:

1. **Row Level Triggers**

   In a row-level trigger, the changes are put on all the rows on which the insert,delete, or update transaction is performed.
   If we have 1000 rows in a database and a delete operation is being run on them,then the trigger would also run 1000 times automatically.
   Executed once for each row affected.

2. **Statement Level Triggers**

   In the statement-level triggers, the operation is under execution only once nomatter how many rows are involved in the operation.
   Executed only once for the statement.

Parts of Triggers:
Trigger Event: Insert / Update / Delete
Trigger action: Before / after
Trigger body: Logic / functionality (What Trigger is doing)
Triggers allow access to values from the table for comparison purposes using **NEW** and **OLD**. The availability of the modifiers depends on the trigger event youuse:
New: get new value from column (Insert & after update). Old: get old value from the column (Delete & before update).

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Syntax:
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-statementsBEGIN
Executable-statements EXCEPTION
Exception-handling-statementsEND;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DMLoperation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with thetrigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

BEFORE INSERT

AFTER INSERT:

- DROP TABLE student;
- CREATE TABLE student(id int,name char(30),gmail char(20),pwdchar(20));
- INSERT INTO student values(101,"Deepak","gujjul@gmail.com","123456");
- SELECT * FROM student;
- CREATE TABLE student_copy(gmailcopy char(20),pwdcopy char(20));

DELIMITER //
CREATE TRIGGER ainsert AFTER INSERT ON studentFOR EACH ROW

```
INSERT INTO student_copy(gmailcopy,pwdcopy)VALUES(new.gmail,new.pwd);
INSERT INTO student VALUES(102,"Sagar","sagar@gmail.com","996655");
```

- SELECT * FROM student;
- SELECT * FROM student_copy;

BEFORE UPDATE:

- DROP TABLE emp;
- CREATE TABLE emp(empno int, name char(30));
- INSERT INTO emp VALUES(10, "Ravi")

```
Delimiter //
CREATE TRIGGER uexample
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
IF NEW.empno <= 0 THEN   SET NEW.empno= NULL; END IF;
END
//
```

- INSERT INTO emp VALUES(40,"Raju");
- SELECT * FROM emp;
- update emp set empno=0 where empno=40;
- SELECT * FROM emp;

AFTER UPDATE:

- CREATE TABLE log(user char(30),status char(40));

```
CREATE TRIGGER auexample
AFTER UPDATE ON emp
FOR EACH ROW
INSERT INTO LOG VALUES(current_user(),concat('Updated by',old.name," ",now()));
```

- SELECT * FROM emp;
- update emp set empno=50 where empno=10;
- SELECT * FROM emp;
- Select * from log;

BEFORE DELETE

- CREATE TABLE salaries(empno int,name varchar(20),salary int);
- INSERT INTO salaries VALUES
  (501,"Raju",12000),
  (502,"Rajesh",13000),
  (503,"Raghu",14000),

(504,"Ranjith",15000);

- SELECT * FROM salaries;
- CREATE TABLE salary_deleted(empno int,name varchar(20),salary int,deleted_time timestamp default now());
- DELIMITER //

```
CREATE TRIGGER bdelete BEFORE DELETE ON salariesFOR EACH ROW
BEGIN
INSERT INTO salary_deleted(empno,name,salary)VALUES
(old.empno,old.name,old.salary);END //
```

- SELECT * FROM salary_deleted;//Empty
- DELETE FROM salaries WHERE empno=501;
- SELECT * FROM salary_deleted;

AFTER DELETE:

- drop table salaries;
- CREATE TABLE salaries(empno int,salary int);
- INSERT INTO salaries values (501,3000),(502,2000),(503,1000);
- CREATE table salary_avg(sal int);
- INSERT INTO salary_avg (sal) SELECT SUM(salary) FROM salaries;
- SELECT * from salary_avg;

```
DELIMITER //
CREATE TRIGGER adeleteAFTER DELETE
ON salaries FOR EACH ROWBEGIN
UPDATE salary_avg SET sal=sal-old.salary;END //
```

- SELECT * FROM salaries;
- SELECT * FROM salary_avg;
- DELETE FROM salaries where empno=501;
- SELECT * FROM salaries;
- SELECT * FROM salary_avg;