**A trigger** in MySQL is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, triggers are of two types according to the SQL standard: row-level triggers and statement-level triggers.

**Row-Level Trigger:** It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

**Why we need/use triggers in MySQL?**

**We need/use triggers in MySQL due to the following features:**

Triggers help us to enforce business rules.

Triggers help us to validate data even before they are inserted or updated.

Triggers help us to keep a log of records like maintaining audit trails in tables.

SQL triggers provide an alternative way to check the integrity of data.

Triggers provide an alternative way to run the scheduled task.

Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.

Triggers reduce the client-side code that saves time and effort.

Triggers help us to scale our application across different platforms.

Triggers are easy to maintain.

**Limitations of Using Triggers in MySQL**

MySQL triggers do not allow to use of all validations; they only provide extended validations. For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.

Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.

Triggers may increase the overhead of the database server.

**Types of Triggers in MySQL?**

We can define the maximum six types of actions or events in the form of triggers:

**Before Insert:** It is activated before the insertion of data into the table.

**After Insert:** It is activated after the insertion of data into the table.

**Before Update:** It is activated before the update of data in the table.

**After Update:** It is activated after the update of the data in the table.

**Before Delete:** It is activated before the data is removed from the table.

**After Delete:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

**Trigger Activation Time: BEFORE | AFTER**

**Trigger Event: INSERT | UPDATE | DELETE**

**Syntax**

CREATE TRIGGER trigger\_name

(AFTER | BEFORE) (INSERT | UPDATE | DELETE)

ON table\_name FOR EACH ROW

BEGIN

--variable declarations

--trigger code

END;

CREATE TRIGGER trigger\_name trigger\_time trigger\_event

ON table\_name FOR EACH ROW

BEGIN

--variable declarations

--trigger code

END;

**Parameter Explanation**

The create trigger syntax contains the following parameters:

**trigger\_name:** It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement. It is to make sure that the trigger name should be unique within the schema.

**trigger\_time:** It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

**trigger\_event:** It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation. The trigger can invoke only one event at one time. If we want to define a trigger which is invoked by multiple events, it is required to define multiple triggers, and one for each event.

**table\_name:** It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

**BEGIN END Block:** Finally, we will specify the statement for execution when the trigger is activated. If we want to execute multiple statements, we will use the BEGIN END block that contains a set of queries to define the logic for the trigger.

The trigger body can access the column's values, which are affected by the DML statement. The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement. We can use the column name with NEW and OLD modifiers as OLD.col\_name and NEW.col\_name. The OLD.column\_name indicates the column of an existing row before the updation or deletion occurs. NEW.col\_name indicates the column of a new row that will be inserted or an existing row after it is updated.

For example, suppose we want to update the column name message\_info using the trigger. In the trigger body, we can access the column value before the update as OLD.message\_info and the new value NEW.message\_info.

Example:

CREATE TABLE employee(

name varchar(45) NOT NULL,

occupation varchar(35) NOT NULL,

working\_date date,

working\_hours varchar(10)

);

INSERT INTO employee VALUES

('Robin', 'Scientist', '2020-10-04', 12),

('Warner', 'Engineer', '2020-10-04', 10),

('Peter', 'Actor', '2020-10-04', 13),

('Marco', 'Doctor', '2020-10-04', 14),

('Brayden', 'Teacher', '2020-10-04', 12),

('Antonio', 'Business', '2020-10-04', 11);

mysql> DELIMITER //

mysql> Create Trigger before\_insert\_empworkinghours

BEFORE INSERT ON employee FOR EACH ROW

BEGIN

IF NEW.working\_hours < 0 THEN SET NEW.working\_hours = 0;

END IF;

END //

insert the nehative values

INSERT INTO employee VALUES

('Alexander', 'Actor', '2020-10-012', -13);

INSERT INTO employee VALUES

('Alexander', 'Actor', '2020-10-012', -1);