# Trigger

**Trigger** is a statement that a system executes automatically when there is any modification to the database.

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.

**Statement-Level Trigger:** It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

## 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.

## Types of Triggers in MySQL

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

1. **Before Insert:** It is activated before the insertion of data into the table.
2. **After Insert:** It is activated after the insertion of data into the table.
3. **Before Update:** It is activated before the update of data in the table.
4. **After Update:** It is activated after the update of the data in the table.
5. **Before Delete:** It is activated before the data is removed from the table.
6. **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.

### Benefits of using triggers in business:

- Faster application development. Because the database stores triggers, you do not have to code the trigger actions into each database application.
- Global enforcement of business rules. Define a trigger once and then reuse it for any application that uses the database.
- Easier maintenance. If a business policy changes, you need to change only the corresponding trigger program instead of each application program.
- Improve performance in client/server environment. All rules run on the server before the result returns.

## 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.

## Naming Conventions

Naming conventions are the set of rules that we follow to give appropriate unique names. It saves our time to keep the work organize and understandable. Therefore, **we must use a unique name for each trigger associated with a table**. However, it is a good practice to have the same trigger name defined for different tables.

The following naming convention should be used to name the trigger in MySQL:

(BEFOR | **AFTER**) table_name (**INSERT** | **UPDATE** | **DELETE**)

Thus,

**Trigger Activation Time:** BEFORE | AFTER

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

**Syntax : -**

```
Delimiter //
CREATE TRIGGER trigger_name
(AFTER | BEFORE) (INSERT | UPDATE | DELETE)
ON table_name FOR EACH ROW
    BEGIN
      --variable declarations
      --trigger code
      END;  //
```

We assume that you are habituated with "MySQL Stored Procedures". You can use the following statements of MySQL procedure in triggers:

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

| Trigger | Event | Table | Statement | Timing | Created | sql_mode | Definer | character_set_ | collation_conn | Database Collation |
|---|---|---|---|---|---|---|---|---|---|---|
| before_insert_empworkinghou | INSERT | employee | BEGIN<br>IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;<br>END IF;<br>END | BEFORE | 2020-11-13 14:49:05.83 | STRICT_TRANS_TABLES,NC | root@localhost | cp850 | cp850_general_ | utf8mb4_0900 |

The show trigger statement contains several columns in the result set. Let us explain each column in detail.

- **Trigger :-** It is the name of the trigger that we want to create and must be unique within the schema.
- **Event :-** It is the type of operation name that invokes the trigger. It can be either INSERT, UPDATE, or DELETE operation.
- **Table :-** It is the name of the table to which the trigger belongs.
- **Statement :-** It is the body of the trigger that contains logic for the trigger when it activates.
- **Timing :-** It is the activation time of the trigger, either BEFORE or AFTER. It indicates that the trigger will be invoked before or after each row of modifications occurs on the table.
- **Created :-** It represents the time and date when the trigger is created.
- **sql_mode :-** It displays the SQL_MODE when the trigger is executed.
- **Definer :-** It is the name of a user account that created the trigger and should be in the 'user_name'@'host_name' format.
- **Character_set_client :-** It was the session value of the character_set_client system variable when the trigger was created.
- **Collation_connection :-** It was the session value of the character_set_client system variable when the trigger was created.
- **Database Collation :-** It determines the rules that compare and order the character string. It is the collation of the database with which the trigger belongs.

```
create table course(
course_id int,
course_desc varchar(50),
course_mentor varchar(50),
course_price int,
course_discount int,
create_date date
);

delimiter //
create trigger course_before_insert
 before insert
 on course for each row
  begin
  set new.create_date = sysdate();
  end; //

  insert into course (course_id, course_desc, course_mentor, course_price, course_discount)
values
  (101, 'DA', 'MJ', 5000, 1000),
  (104, 'DS', 'HR', 5500, 700),
  (178, 'BD', 'HS', 7000, 2500);

 select * from course;
```

| course_id | course_desc | course_mentor | course_price | course_discount | create_date |
|---|---|---|---|---|---|
| 101 | DA | MJ | 5000 | 1000 | 28-08-2023 |
| 104 | DS | HR | 5500 | 700 | 28-08-2023 |
| 178 | BD | HS | 7000 | 2500 | 28-08-2023 |

```
create table course1(
course_id int,
course_desc varchar(50),
course_mentor varchar(50),
course_price int,
course_discount int,
create_date date,
user_info varchar(50)
);
```

```
delimiter //
create trigger course_before_insert1
 before insert
 on course1 for each row
  begin
  declare user_val varchar(50);
  set new.create_date = sysdate();
  select user() into user_val;
  set new.user_info = user_val;
  end; //

  insert into course1 (course_id, course_desc, course_mentor, course_price, course_discount)
values
  (101, 'DA', 'MJ', 5000, 1000),
  (104, 'DS', 'HR', 5500, 700),
  (178, 'BD', 'HS', 7000, 2500);

  select * from course1;
```

| course_id | course_desc | course_mentor | course_price | course_discount | create_date | user_info |
|---|---|---|---|---|---|---|
| 101 | DA | MJ | 5000 | 1000 | 28-08-2023 | root@localhost |
| 104 | DS | HR | 5500 | 700 | 28-08-2023 | root@localhost |
| 178 | BD | HS | 7000 | 2500 | 28-08-2023 | root@localhost |

```
  create table course2(
  course_id int,
  course_desc varchar(50),
  course_mentor varchar(50),
  course_price int,
  course_discount int,
  create_date date,
  user_info varchar(50)
  );

   create table ref_course(
   record_insert_date date,
   record_insert_user varchar(50)
   );
```

```
delimiter //
create trigger course_before_insert2
 before insert
 on course2 for each row
  begin
  declare user_val varchar(50);
  set new.create_date = sysdate();
  select user() into user_val;
  set new.user_info = user_val;
  insert into ref_course values(sysdate(), user_val);
  end; //

 insert into course2 (course_id, course_desc, course_mentor, course_price, course_discount)
values
  (101, 'DA', 'MJ', 5000, 1000),
  (104, 'DS', 'HR', 5500, 700),
  (178, 'BD', 'HS', 7000, 2500);

 select * from ref_course;
```

| record_insert_date | record_insert_user |
|---|---|
| 28-08-2023 | root@localhost |
| 28-08-2023 | root@localhost |
| 28-08-2023 | root@localhost |

```
create table test1
 (
 c1 varchar(50),
 c2 date,
 c3 int
 );

  create table test2
 (
 c1 varchar(50),
 c2 date,
 c3 int
 );
```

```
 create table test3
(
c1 varchar(50),
c2 date,
c3 int
);

insert into test1 values
('John', sysdate(), 109803);

delimiter //
create trigger to_update_others
before insert
on test1 for each row
begin
    insert into test2 values('MRA', sysdate(), 78786);
    insert into test3 values('MRA', sysdate(), 78786);
end; //
```

```
select * from test1;
```

| c1 | c2 | c3 |
|---|---|---|
| John | 28-08-2023 | 109803 |
| John | 28-08-2023 | 109803 |

```
select * from test2;
```

| c1 | c2 | c3 |
|---|---|---|
| MRA | 28-08-2023 | 78786 |

```
select * from test3;
```

| c1 | c2 | c3 |
|---|---|---|
| MRA | 28-08-2023 | 78786 |

**Update and delete :-**

```
delimiter //
 create trigger to_update_others_table
 before insert
 on test1 for each row
 begin
     update test2 set c1 = 'abc' where c1 = 'MRA';
     delete from test3 where c1 = 'MRA';
 end; //
```

```
insert into test1 values('wood' ,sysdate(), 7728);

 select * from test1;
select * from test2;
select * from test3;
```

## AFTER DELETE

The AFTER DELETE Trigger in MySQL is invoked automatically whenever a delete event is fired on the table.

## Restrictions

- We can access the OLD rows but cannot update them in the AFTER DELETE trigger.
- We cannot access the NEW rows. It is because there are no NEW row exists.
- We cannot create an AFTER DELETE trigger on a VIEW.

```
delimiter //

 create trigger after_delete_trigger

 after delete

 on test1 for each row

 begin

     insert into test3 values('After_delete', sysdate(), 57547);

 end; //

 select * from test1;

 delete from test1 where c1 = 'wood';

 select * from test1;

 select * from test3;
```

| c1 | c2 | c3 |
|---|---|---|
| After_delete | 28-08-2023 | 57547 |

## BEFORE DELETE

BEFORE DELETE Trigger in MySQL is invoked automatically whenever a delete operation is fired on the table.

```
delimiter //
 create trigger before_delete_trigger
 before delete
 on test1 for each row
 begin
      insert into test3 values('After_delete', sysdate(), 57547);
 end; //
  delete from test1 where c1 = 'john';
  select * from test3;
```

## Observation :- Before Delete

```
create table test11
 (
 c1 varchar(50),
 c2 date,
 c3 int
 );

   create table test12
 (
 c1 varchar(50),
 c2 date,
 c3 int
 );

 delimiter //
 create trigger before_delete_trigger_observation1
 before delete
 on test11 for each row
 begin
      insert into test12(c1,c2,c3) values(old.c1,old.c2, old.c3);
 end; //
 insert into test11 values('Aligarh' ,sysdate(), 5346);
 insert into test11 values('Delhi' ,sysdate(), 3528);
 insert into test11 values('Agra' ,sysdate(), 7756);
 insert into test11 values('Noida' ,sysdate(), 8928);
```

select * from test11;

| c1 | c2 | c3 |
|---|---|---|
| Aligarh | 28-08-2023 | 5346 |
| Delhi | 28-08-2023 | 3528 |
| Agra | 28-08-2023 | 7756 |
| Noida | 28-08-2023 | 8928 |

delete from **test11** where c1 = 'Agra';

select * from **test12;**

| c1 | c2 | c3 |
|---|---|---|
| Agra | 28-08-2023 | 7756 |

# Observation :- After Delete

```
delimiter //
 create trigger after_delete_trigger_observation2
 after delete
 on test11 for each row
 begin
     insert into test12(c1,c2,c3) values(old.c1,old.c2, old.c3);
 end; //
```

```
delete from test11 where c1 = 'Agra';
select * from test12;
```

| c1 | c2 | c3 |
|---|---|---|
| Agra | 28-08-2023 | 7756 |

```
delete from test11 where c1 = 'Aligarh';
select * from test12;
```

| c1 | c2 | c3 |
|---|---|---|
| Agra | 28-08-2023 | 7756 |
| Aligarh | 28-08-2023 | 5346 |
| Aligarh | 28-08-2023 | 5346 |

## AFTER UPDATE TRIGGER

The AFTER UPDATE trigger in MySQL is invoked automatically whenever an UPDATE event is fired on the table associated with the triggers.

```
delimiter //
create trigger after_update_trigger_
after update
on test11 for each row
begin
    insert into test12(c1,c2,c3) values(old.c1,old.c2, old.c3);
end; //
```

```
select * from test11;
```

| c1 | c2 | c3 |
|---|---|---|
| Delhi | 28-08-2023 | 3528 |
| Noida | 28-08-2023 | 8928 |

```
update test11 set c1 = 'New York' where c1 = 'Delhi';
```

```
select * from test11;
```

| c1 | c2 | c3 |
|---|---|---|
| New York | 28-08-2023 | 3528 |
| Noida | 28-08-2023 | 8928 |

```
select * from test12;
```

| c1 | c2 | c3 |
|---|---|---|
| Agra | 28-08-2023 | 7756 |
| Aligarh | 28-08-2023 | 5346 |
| Aligarh | 28-08-2023 | 5346 |

## BEFORE UPDATE

BEFORE UPDATE Trigger in MySQL is invoked automatically whenever an update operation is fired on the table associated with the trigger

```
delimiter //
 create trigger before_update_trigger_
 before update
 on test11 for each row
 begin
     insert into test12(c1,c2,c3) values(new.c1,new.c2, new.c3);
 end; //

 select * from test11;

 update test11 set c1 = 'Rome' where c1 = 'Noida';


select * from test11;
```

| c1 | c2 | c3 |
|---|---|---|
| New York | 28-08-2023 | 3528 |
| Rome | 28-08-2023 | 8928 |

```
select * from test12;
```

| c1 | c2 | c3 |
|---|---|---|
| Agra | 28-08-2023 | 7756 |
| Aligarh | 28-08-2023 | 5346 |
| Aligarh | 28-08-2023 | 5346 |
| Rome | 28-08-2023 | 8928 |