Triggers in SQL

- A Trigger is a set of SQL statements that reside in system memory with unique names.
- It is a specialized category of stored procedure that is called automatically when a database server event occurs.
- Each trigger is always associated with a table.

Uses for triggers:

- Enforce business rules
- Validate input data
- Generate a unique value for a newly-inserted row in a different file.
- Write to other files for audit trail purposes
- Query from other files for cross-referencing purposes
- Access system functions
- Replicate data to different files to achieve data consistency

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.

What is a Trigger in MySQL?

A trigger is a named MySQL object that activates when an event occurs in a table. Triggers are a particular type of stored procedure associated with a specific table.

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 you use:

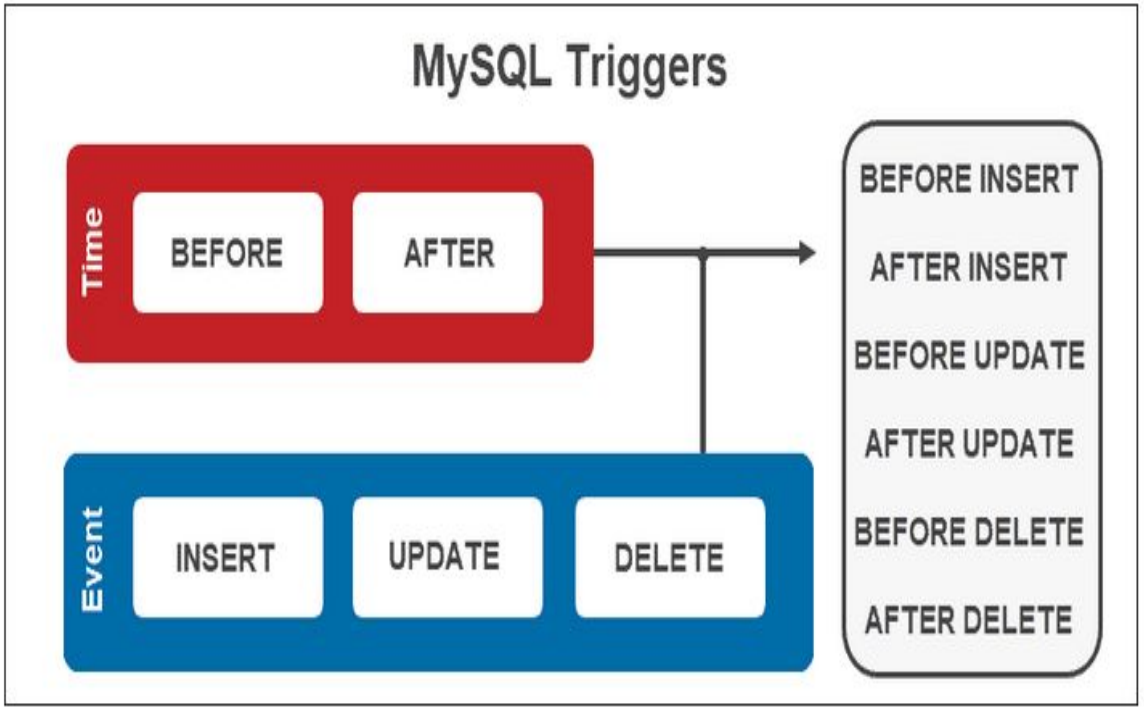
Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Using MySQL Triggers:

Every trigger associated with a table has a unique name and function based on two factors:

1. **Time**. **BEFORE** or **AFTER** a specific row event.

2. **Event**. **INSERT**, **UPDATE** or **DELETE**.

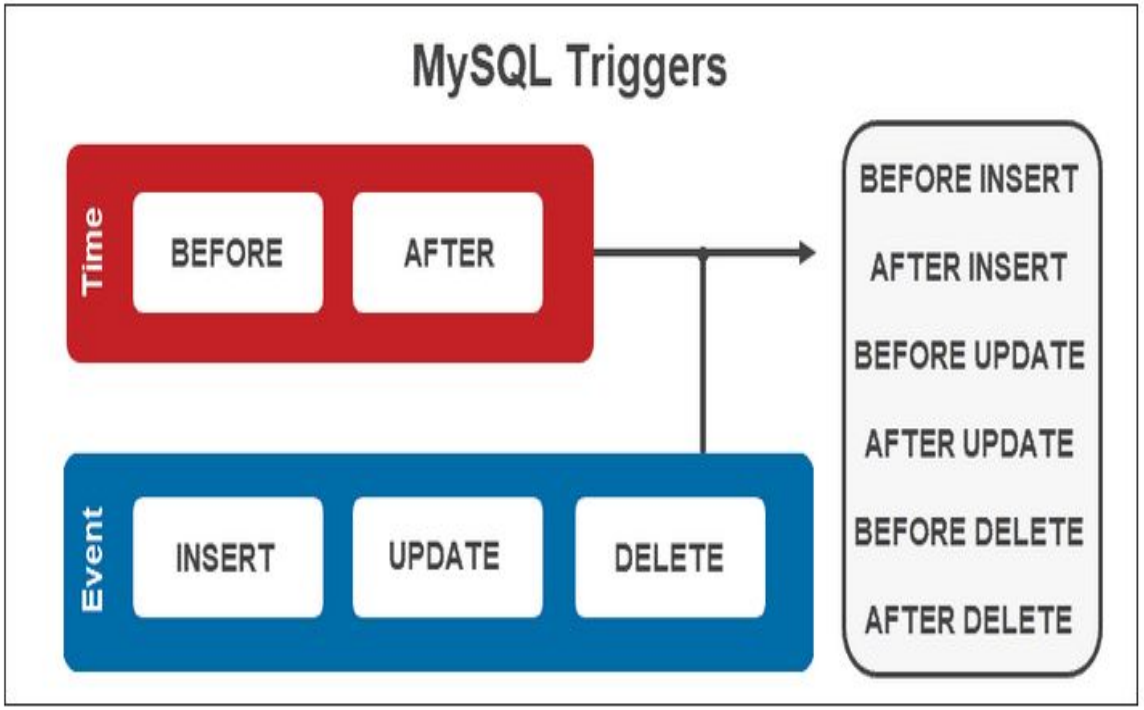


Using MySQL Triggers:

Every trigger associated with a table has a unique name and function based on two factors:

1. **Time**. **BEFORE** or **AFTER** a specific row event.

2. **Event**. **INSERT**, **UPDATE** or **DELETE**.



MySQL triggers fire depending on the activation time and the event for a total of six unique trigger combinations. The before statements help to check data and make changes before making commitments, whereas the after statements commit the data first and then execute statements.

The execution of a set of actions happens automatically, affecting all inserted, deleted, or updated rows in the statement.

Creating Triggers

Use the **CREATE TRIGGER** statement syntax to create a new trigger:

CREATE TRIGGER <trigger name> <trigger time > <trigger event>
ON <table name>
FOR EACH ROW
<trigger body>;

The best practice is to name the trigger with the following information:

<trigger time>_<table name>_<trigger event>

For example, if a trigger fires **before insert** on a table named **employee**, the best convention is to call the trigger:

before_employee_insert

Alternatively, a common practice is to use the following format:

<table name>_<first letter of trigger time><first letter of trigger name>

The before insert trigger name for the table employee looks like this:

employee_bi

Dropping a Triggers

Delete Triggers

To delete a trigger, use the **DROP TRIGGER** statement:

```
DROP TRIGGER <trigger name>;
```

```
mysql> DROP TRIGGER person_bi;
Query OK, 0 rows affected (0.41 sec)
```

Alternatively, use:

```
DROP TRIGGER IF EXISTS <trigger name>;
```

```
mysql> DROP TRIGGER IF EXISTS person_bi;
Query OK, 0 rows affected, 1 warning (0.09 sec)

mysql> DROP TRIGGER person_bi;
ERROR 1360 (HY000): Trigger does not exist
```

1.Example of Trigger

```
SQL> create table try1
  2  (col1 number(10),
  3  col2 varchar2(10));
Table created.
```

```
SQL> create table try2
  2  (col1 number(10),
  3  col2 varchar2(10));
Table created.
```

```
SQL>  create or replace  trigger try_trigger
  2  after insert on try1
  3  for each row
  4  begin
  5  insert into try2(col1,col2) values(:new.col1,:new.col2);
  6  end;
  7  /
Trigger created.
```

```
SQL> insert into try1 values(1,'sana');
1 row created.

SQL> select * from try1;
      COL1       COL2
----------  ----------
         1       sana
```

(Since we have created Trigger, after inserting a record into try1 table automatically same record will inserted into try2 table)

```
SQL> select * from try2;
      COL1       COL2
----------  ----------
         1       sana
```