Triggers:

In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

Key characteristics and components of MySQL Triggers:

Association with a Table: A trigger is always linked to a single table.

Triggering Event: The event that activates the trigger can be INSERT, UPDATE, or DELETE.

Trigger Action Time:

BEFORE: The trigger executes before the triggering event occurs. This is often used for data validation,

modification, or cleansing before the data is actually inserted, updated, or deleted.

AFTER: The trigger executes after the triggering event has completed. This is commonly used for auditing,

logging changes, or performing actions on other tables based on the changes in the triggered table.

Trigger Body: This is the block of SQL statements that are executed when the trigger is activated. It can

contain one or more SQL statements.

FOR EACH ROW: This clause specifies that the trigger should activate for every row affected by the

triggering event.

OLD and NEW Keywords: Within the trigger body:

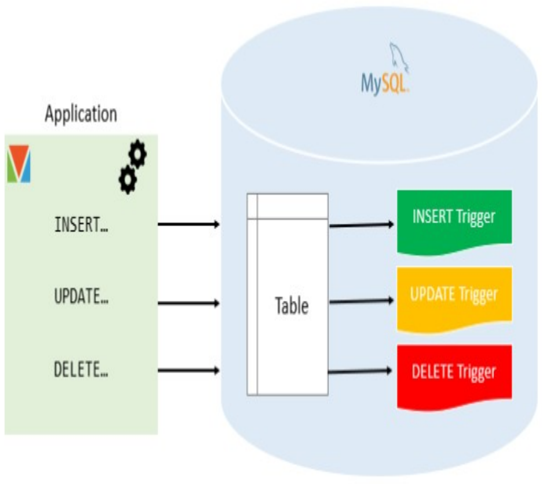
NEW.column_name: Refers to the value of a column after an INSERT or UPDATE operation.

OLD.column_name: Refers to the value of a column before an UPDATE or DELETE operation.

The SQL standard defines two types of triggers: row-level triggers and statement-level triggers.

A **row-level trigger** is activated for each row that is inserted, updated, or deleted. For example, if a table has 100 rows inserted, updated, or deleted, the trigger is automatically invoked 100 times for the 100 rows affected.

A **statement-level trigger** is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.



MySQL supports only **row-level triggers.** It doesn't support statement-level triggers.

Advantages of triggers

Triggers provide another way to check the integrity of data.

Triggers handle errors from the database layer.

Triggers give an alternative way to run **scheduled tasks**. By using triggers, you don't have to wait for the scheduled events to run because the triggers are invoked automatically before or after a change is made to the data in a table.

Triggers can be useful for auditing the data changes in tables.

The **SHOW TRIGGERS** statement list triggers defined for tables in the current database

```
SHOW TRIGGERS FROM database_name;
```

BEFORE INSERT TRIGGER:

```
DELIMITER $$


CREATE TRIGGER trigger_name

   BEFORE INSERT

   ON table_name FOR EACH ROW

BEGIN

   -- statements

END$$


DELIMITER ;
```

Before Insert Example:

First, create a new table called WorkCenters:

```
DROP TABLE IF EXISTS WorkCenters;
```

```
CREATE TABLE WorkCenters (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL,

   capacity INT NOT NULL

);
```

Second, create another table called WorkCenterStats that stores the summary of the capacity of the work centers:

```
DROP TABLE IF EXISTS WorkCenterStats;

CREATE TABLE WorkCenterStats(

   totalCapacity INT NOT NULL

);
```

Create BEFORE INSERT TRIGGER:

```
DELIMITER $$


CREATE TRIGGER before_workcenters_insert

BEFORE INSERT

ON WorkCenters FOR EACH ROW

BEGIN

   DECLARE rowcount INT;


   SELECT COUNT(*)

   INTO rowcount

   FROM WorkCenterStats;


   IF rowcount > 0 THEN

      UPDATE WorkCenterStats

      SET totalCapacity = totalCapacity + new.capacity;
   ELSE

      INSERT INTO WorkCenterStats(totalCapacity)

      VALUES(new.capacity);

   END IF;


END $$


DELIMITER ;
```

Testing the MySQL BEFORE INSERT trigger

First, insert a new row into the WorkCenter table:

```
INSERT INTO WorkCenters(name, capacity)

VALUES('Mold Machine',100);
```

Second, query data from the WorkCenterStats table:

```
SELECT * FROM WorkCenterStats;
```

AFTER INSERT Example:

```
DELIMITER //

CREATE TRIGGER after_employee_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee_audit (employee_id, action, timestamp)

VALUES (NEW.employee_id, 'INSERT', NOW());

END; //

DELIMITER ;
```