**MySQL Triggers**

A trigger in MySQL is a database object that automatically executes (or "fires") when a specified event occurs on a particular table. Triggers can be used for tasks like validating data, enforcing business rules, logging, or automatic updates.

**Types of Triggers in MySQL:**

1. Before Triggers: Executed before the event that activates them (INSERT, UPDATE, or DELETE).

2. After Triggers: Executed after the event that activates them.

**Trigger Events:**

- `INSERT`: Fired when a new record is added to a table.

- `UPDATE`: Fired when a record is modified.

- `DELETE`: Fired when a record is removed from a table.

**Trigger Syntax:**

```sql

CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {INSERT | UPDATE | DELETE}

ON table\_name FOR EACH ROW

trigger\_body;

```

- `trigger\_name`: The name of the trigger.

- `BEFORE | AFTER`: Defines whether the trigger should be executed before or after the event.

- `INSERT | UPDATE | DELETE`: The event that causes the trigger to fire.

- `table\_name`: The table on which the trigger is defined.

- `FOR EACH ROW`: Specifies that the trigger is executed once for each row affected by the triggering event.

- `trigger\_body`: The SQL statements that define what the trigger does.

**Example :**

**Assume we have a student table.**

CREATE TABLE student\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

message VARCHAR(255),

log\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER after\_student\_insert

AFTER INSERT ON student

FOR EACH ROW

BEGIN

-- SELECT \* from student;

-- Insert a message into the log table

INSERT INTO student\_log (message)

VALUES (CONCAT('A new student with ID ', NEW.roll\_no, ' has been added.'));

END $$

DELIMITER ;

insert into student (roll\_no,name,age,address) values (155,"rohit",24,"varanshi");

select \* from student\_log;

**Example 1: BEFORE INSERT Trigger**

This trigger validates data before it’s inserted into the table. For example, if you want to ensure that the salary is not negative in the `employees` table.

```sql

CREATE TRIGGER check\_salary\_before\_insert

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF NEW.salary < 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salary cannot be negative';

END IF;

END;

```

- `NEW`: A keyword used in triggers to refer to the new data that is being inserted or updated.

- If a negative salary is attempted, the trigger will raise an error and prevent the insert.

Example 2: AFTER INSERT Trigger

Suppose you want to automatically insert a log into a `log\_table` every time a new record is added to the `employees` table.

```sql

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO log\_table (action, employee\_id, timestamp)

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

END;

```

In this case:

- After a new employee is inserted, a record is added to the `log\_table` indicating the action, the employee's ID, and the current timestamp.

**Example 3: BEFORE UPDATE Trigger**

If you want to log changes to an employee’s salary before it's updated:

```sql

CREATE TRIGGER log\_salary\_before\_update

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

IF NEW.salary <> OLD.salary THEN

INSERT INTO salary\_log (employee\_id, old\_salary, new\_salary, change\_date)

VALUES (OLD.id, OLD.salary, NEW.salary, NOW());

END IF;

END;

```

- `OLD`: Refers to the current (existing) values in the database before the update occurs.

- `NEW`: Refers to the new values that are being updated.

- This trigger logs salary changes to a separate `salary\_log` table.

**Example 4: AFTER DELETE Trigger**

To log when a record is deleted from the `employees` table:

```sql

CREATE TRIGGER after\_employee\_delete

AFTER DELETE ON employees

FOR EACH ROW

BEGIN

INSERT INTO log\_table (action, employee\_id, timestamp)

VALUES ('DELETE', OLD.id, NOW());

END;

```

- After an employee is deleted, a log entry is created showing that the action was a `DELETE` and recording the employee's ID and timestamp.

**Important Notes:**

1. Limitations:

- A table can only have one trigger for each combination of event and timing (e.g., one `BEFORE INSERT` trigger per table).

- Triggers cannot call `COMMIT`, `ROLLBACK`, or transaction control statements directly.

- Triggers cannot change data in the table they are triggered on.

2. Triggers and Performance:

- Overuse of triggers can lead to performance issues, especially if a trigger is doing a lot of work for each row processed by an insert, update, or delete operation.

- Be mindful of the potential overhead.

3. Debugging:

- It can be difficult to debug triggers because they fire automatically. It’s useful to include logging within triggers for easier troubleshooting.

Viewing and Dropping Triggers:

- To view existing triggers:

```sql

SHOW TRIGGERS;

```

- To drop a trigger:

```sql

DROP TRIGGER IF EXISTS trigger\_name;

```

Real-World Use Cases:

- Auditing: Automatically log changes or deletions in a separate table.

- Data Validation: Ensure that certain business rules are enforced, like making sure no negative values are inserted.

- Automatic Calculations: Automatically update a summary table when related data changes.

Triggers are a powerful feature in MySQL, providing a way to automate certain actions and enforce rules at the database level. However, careful planning is needed to avoid performance pitfalls and maintainable code.