**Triggers and Events in MySQL**

**Triggers** and **events** in MySQL are used to automate database operations. They enable actions to be performed when certain conditions are met.

**1. Triggers**

A **trigger** is a set of instructions automatically executed in response to certain database events, such as INSERT, UPDATE, or DELETE.

**Features of Triggers**

* Defined on a specific table.
* Automatically executed before or after a specified event.
* Commonly used for auditing, validation, or enforcing business rules.

**Syntax**

sql

Copy code

CREATE TRIGGER trigger\_name

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

ON table\_name

FOR EACH ROW

BEGIN

-- SQL statements

END;

**Example: Auditing with Triggers**

Create an audit table:

sql

Copy code

CREATE TABLE audit\_log (

id INT AUTO\_INCREMENT PRIMARY KEY,

action\_type VARCHAR(10),

table\_name VARCHAR(50),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

old\_value TEXT,

new\_value TEXT

);

Create a trigger to log updates:

sql

Copy code

DELIMITER //

CREATE TRIGGER after\_employee\_update

AFTER UPDATE

ON employees

FOR EACH ROW

BEGIN

INSERT INTO audit\_log (action\_type, table\_name, old\_value, new\_value)

VALUES (

'UPDATE',

'employees',

CONCAT('Old Salary: ', OLD.salary),

CONCAT('New Salary: ', NEW.salary)

);

END //

DELIMITER ;

**Explanation:**

* The trigger logs updates to the employees table.
* OLD refers to the pre-update values, and NEW refers to the post-update values.

**Example: Preventing Negative Salaries**

sql

Copy code

DELIMITER //

CREATE TRIGGER before\_salary\_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 //

DELIMITER ;

**Viewing Triggers**

sql

Copy code

SHOW TRIGGERS;

**Drop a Trigger**

sql

Copy code

DROP TRIGGER trigger\_name;

**2. Events**

An **event** in MySQL is a task that runs at a scheduled time or interval. It’s similar to a cron job in Unix/Linux systems.

**Features of Events**

* Scheduled tasks at specific intervals or one-time execution.
* Commonly used for automated backups, cleaning data, or periodic updates.
* Requires the **event scheduler** to be enabled.

**Enabling the Event Scheduler**

sql

Copy code

SET GLOBAL event\_scheduler = ON;

**Syntax**

sql

Copy code

CREATE EVENT event\_name

ON SCHEDULE {AT timestamp | EVERY interval}

DO

BEGIN

-- SQL statements

END;

**Example: Archiving Old Data**

Create an archive table:

sql

Copy code

CREATE TABLE employee\_archive AS SELECT \* FROM employees WHERE 1 = 0; -- Empty table structure

Create an event to move old employee records:

sql

Copy code

DELIMITER //

CREATE EVENT archive\_old\_employees

ON SCHEDULE EVERY 1 DAY

DO

BEGIN

INSERT INTO employee\_archive

SELECT \* FROM employees WHERE hire\_date < CURDATE() - INTERVAL 1 YEAR;

DELETE FROM employees WHERE hire\_date < CURDATE() - INTERVAL 1 YEAR;

END //

DELIMITER ;

**Explanation:**

* This event runs daily to move employees hired over a year ago to an archive table and deletes them from the main table.

**Example: One-Time Event**

Send a reminder email 1 hour from now:

sql

Copy code

DELIMITER //

CREATE EVENT send\_reminder

ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 1 HOUR

DO

BEGIN

-- Placeholder for email logic

INSERT INTO reminders (message, created\_at)

VALUES ('Reminder: Complete your profile', NOW());

END //

DELIMITER ;

**Viewing Events**

sql

Copy code

SHOW EVENTS;

**Altering an Event**

sql

Copy code

ALTER EVENT event\_name

ON SCHEDULE EVERY 1 WEEK;

**Dropping an Event**

sql

Copy code

DROP EVENT event\_name;

**Comparison of Triggers and Events**

| **Feature** | **Triggers** | **Events** |
| --- | --- | --- |
| **Execution** | Automatically triggered by DML actions (INSERT, UPDATE, DELETE). | Executes at a scheduled time. |
| **Use Case** | Enforcing rules, logging, validations. | Scheduled tasks like backups or data cleanup. |
| **Trigger Timing** | Runs BEFORE or AFTER the event. | Independent of other database actions. |

**Best Practices**

1. **Use When Necessary:**
   * Avoid overusing triggers as they can complicate debugging.
   * Use events for periodic tasks instead of manual intervention.
2. **Test Thoroughly:**
   * Test triggers and events to ensure they work as expected without affecting performance.
3. **Monitor Performance:**
   * Triggers and events can add overhead; monitor their impact on database performance.
4. **Keep Code Simple:**
   * Write concise and maintainable logic within triggers and events.