MySQL Triggers

A **trigger** is a stored program that is automatically executed in response to certain events on a table (such as `INSERT`, `UPDATE`, or `DELETE`).

Trigger Basics

General Syntax:

```
CREATE TRIGGER trigger_name
trigger_time trigger_event
ON table_name
FOR EACH ROW
BEGIN
    -- trigger logic here
END;
```

- **trigger_time** → `BEFORE` or `AFTER`
- **trigger_event** → `INSERT`, `UPDATE`, `DELETE`
- **FOR EACH ROW** → runs for each affected row, not just once per statement.

---

1. BEFORE INSERT Trigger

Example: Validate data before insertion.

```
Use mydb;
DELIMITER //
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary < 30000 THEN
        SET NEW.salary = 30000; -- minimum salary rule
    END IF;
END //
DELIMITER ;

-- Test
INSERT INTO employees (emp_id, first_name, last_name, department, salary)
VALUES (101, 'John', 'Smith', 'HR', 25000);


CREATE A PRODUCT TABLE WITH COLUMNS : PROD_ID , PROn D_NAME AND QUANTITY.
WRITE A TRIGGER TO VALIDATE THE QUANTITY. THE QUANTITY SHOULD BE BETWEEN 10
AND 50. IF IT IS NOT BETWEEN 10 AND 50 , SET THE VALUE OF QUANTITY TO 25.
```

```
Create table product (
P_id INT ,
p_name VARCHAR(10) ,
qtY INT
);



Delimiter //
create trigger validate_qty
Before insert
On product
FOR EACH ROW
Begin
        If new.qty NOT BETWEEN 10 AND 50 then
                Set new.qty = 25 ;
        End if ;
End //
Delimiter //

Insert into product (p_id , p_name ,qty)
values(1 , ‘keyboard’,10);
```

---

2. AFTER INSERT Trigger

Example: Log new hires.

```
CREATE TABLE employee_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    action VARCHAR(50),
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, action)
    VALUES (NEW.emp_id, 'New Employee Added');
END //
DELIMITER ;

-- Test
INSERT INTO employees (emp_id, first_name, last_name, department, salary)
VALUES (102, 'Mary', 'Lee', 'IT', 55000);
Select * from employee_log;
```

---

3. BEFORE UPDATE Trigger

Example: Prevent salary decrease.

```
DELIMITER //
CREATE TRIGGER before_salary_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary < OLD.salary THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Salary decrease not allowed';
    END IF;
END //
DELIMITER ;

-- Test
UPDATE employees SET salary = 40000 WHERE emp_id = 102;
```

---

4. AFTER UPDATE Trigger

Example: Log salary changes.

```
DELIMITER //
CREATE TRIGGER after_salary_update
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary <> OLD.salary THEN
        INSERT INTO employee_log (emp_id, action)
        VALUES (NEW.emp_id, CONCAT('Salary updated from ', OLD.salary, ' to
', NEW.salary));
    END IF;
END //
DELIMITER ;

-- Test the trigger
UPDATE employees SET salary = 40000 WHERE emp_id = 102;

--Check the log table
Select * from employee_log;
```

---

5. BEFORE DELETE Trigger

Example: Archive deleted employees.

```
CREATE TABLE archived_employees AS SELECT * FROM employees WHERE 1=0;

DELIMITER //
CREATE TRIGGER before_employee_delete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO archived_employees
    VALUES (OLD.emp_id, OLD.first_name, OLD.last_name, OLD.department,
OLD.salary);
END //
DELIMITER ;

-- Test
DELETE FROM employees WHERE emp_id = 101;
```

---

6. AFTER DELETE Trigger

Example: Log employee deletions.

```
DELIMITER //
CREATE TRIGGER after_employee_delete
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_log (emp_id, action)
    VALUES (OLD.emp_id, 'Employee Deleted');
END //
DELIMITER ;
--Test
Delete from employees where emp_id = 102;
Select * from employees;
```

---

7. Show Existing Triggers

```
SHOW TRIGGERS;
```