Triggers in MySQL

- Automatic execution of SQL in response to table events

Instructor : Sriya Ivaturi

Introduction to Triggers

- - A trigger is a named set of SQL statements
- - Executes automatically in response to INSERT, UPDATE, DELETE
- - Can run BEFORE or AFTER the event

When to Use Triggers

- - Automatically enforce rules
- - Track changes in audit tables
- - Maintain summary/derived data
- - Validate or transform input data

Trigger Syntax

- CREATE TRIGGER trigger_name
- {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
- ON table_name
- FOR EACH ROW
- BEGIN
- -- trigger body
- END;

Types of Triggers

- BEFORE INSERT → modify/validate data before inserting
- AFTER INSERT → log insertions
- BEFORE UPDATE → check/alter new values
- AFTER UPDATE → log changes, sync
- BEFORE DELETE → prevent deletion
- AFTER DELETE → archive/log old data

AFTER INSERT Trigger Example

- CREATE TRIGGER after_student_insert
- AFTER INSERT ON students
- FOR EACH ROW
- BEGIN
- INSERT INTO audit_log(action, name)
- VALUES ('New student added', NEW.name);
- END;

BEFORE UPDATE Trigger Example

- CREATE TRIGGER before_name_update
- BEFORE UPDATE ON employees
- FOR EACH ROW
- BEGIN
- SET NEW.name = UPPER(NEW.name);
- END;

Understanding NEW and OLD

- - `NEW`: new row value (INSERT or UPDATE)
- - `OLD`: existing row value (UPDATE or DELETE)
- - Use to compare, validate, or log data changes

When to Use NEW and OLD

- INSERT: NEW ✓, OLD ✗
- UPDATE: NEW ✓, OLD ✓
- DELETE: NEW ✗, OLD ✓
- - Only use NEW in BEFORE to modify values
- - OLD is always read-only

Operation	NEW	OLD
BEFORE INSERT	✓ Yes (can modify)	✗ No
AFTER INSERT	✓ Yes (read-only)	✗ No
BEFORE UPDATE	✓ Yes (can modify)	✓ Yes (read-only)
AFTER UPDATE	✓ Yes (read-only)	✓ Yes (read-only)
BEFORE DELETE	✗ No	✓ Yes (read-only)
AFTER DELETE	✗ No	✓ Yes (read-only)

Examples with NEW and OLD

- 1. BEFORE INSERT:
- SET NEW.name = UPPER(NEW.name);

- 2. BEFORE UPDATE:
- IF NEW.salary < OLD.salary THEN error;

- 3. AFTER DELETE:
- Log OLD.id and OLD.name into archive table

Rules for NEW and OLD

- - ❌ Cannot assign to OLD
- - ✅ Can assign to NEW only in BEFORE triggers
- - ❌ NEW not available in DELETE
- - ❌ OLD not available in INSERT
- - Triggers execute per row → use row-wise logic

BEFORE INSERT – Modify NEW

- CREATE TRIGGER before_insert_user
- BEFORE INSERT ON users
- FOR EACH ROW
- BEGIN
- SET NEW.name = UPPER(NEW.name); -- Capitalize name
- END;

BEFORE UPDATE – Compare OLD and NEW

- CREATE TRIGGER before_salary_change
- 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;

AFTER DELETE – Log OLD Data

- CREATE TRIGGER after_delete_product
- AFTER DELETE ON products
- FOR EACH ROW
- BEGIN
- INSERT INTO deleted_products_log (product_id, name)
- VALUES (OLD.id, OLD.name);
- END;

Things to Remember about NEW and OLD

- ⃠ You cannot assign values to OLD in any trigger.
- ✓ You can assign to NEW only in BEFORE triggers.
- ⃠ You cannot use NEW in DELETE triggers.
- ⃠ You cannot use OLD in INSERT triggers.
- Triggers are row-level, so NEW/OLD refer to one row at a time.

Common Use Cases

- - Enforce business rules
- - Audit trails
- - Maintain consistency
- - Auto-formatting data
- - Cascading changes

Questions

- Q1: Can triggers call stored procedures?
- Q2: Difference between BEFORE and AFTER?
- Q3: Can you change OLD values in a trigger?

Answers

- A1: Yes, but only indirectly with SQL logic
- A2: BEFORE → before data is committed
- AFTER → after change happens
- A3: No, OLD is read-only

Summary

- - Triggers run automatically on table events
- - Use NEW/OLD to access row values
- - Ideal for automation, validation, logging
- - Must be used with care to avoid recursion