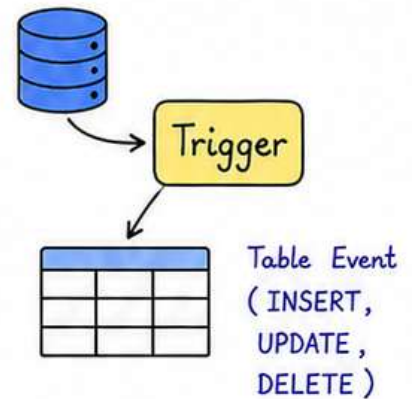


Chapter 21: MySQL Triggers

21.1 Introduction to Triggers

- * A trigger is a special type of stored program (like a function or procedure) that is **automatically executed** in response to certain **events** on a table.
- * Triggers are used to maintain **data integrity**, enforce business rules and perform automatic actions.
- * They are executed when an **INSERT**, **UPDATE** or **DELETE** statement is performed on a table.
- * Triggers cannot be called directly like functions or procedures.



Key Points :

- ✓ Triggers are associated with a specific table.
- ✓ They are fired automatically when events occur.
- ✓ They help in maintaining data consistency.
- ✓ They can be used for auditing, validation, logging, etc.



21.2 CREATE TRIGGER

- * The **CREATE TRIGGER** statement is used to create a new trigger.
- * It specifies the trigger name, timing (**BEFORE** or **AFTER**), event (**INSERT**, **UPDATE** or **DELETE**), table and the SQL statements to be executed.

Syntax :

```
CREATE TRIGGER trigger_name
BEFORE | AFTER INSERT | UPDATE | DELETE
ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```

FOR EACH ROW
means the trigger
is executed once
for each affected
row.

Parameters :

Parameter	Description
trigger_name	Name of the trigger.
BEFORE AFTER	Timing of the trigger.
INSERT UPDATE DELETE	Event that activates the trigger.
table_name	Table on which the trigger is created.
FOR EACH ROW	Specifies the trigger is executed for each row.



Example :

```
DELIMITER $$
CREATE TRIGGER before_insert_emp
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO emp_log (emp_id, action, action_time)
    VALUES (NEW.emp_id, 'Inserted', NOW());
END $$
DELIMITER ;
```

Explanation :

- * Trigger name : before_insert_emp
- * Fires **BEFORE** an INSERT on employees table.
- * For each inserted row, it inserts a record into emp_log table.
- * **NEW** refers to the new row being inserted.



Types of Trigger Events :

- INSERT** - When a new row is inserted.
- UPDATE** - When an existing row is updated.
- DELETE** - When a row is deleted.



21.3 BEFORE INSERT

- * A **BEFORE INSERT** trigger is executed **before** a new row is inserted into the table.
- * It can be used to **validate** or **modify** the data before it is stored.
- * If any condition fails, the insertion can be prevented.
- * It helps in maintaining **data quality** and enforcing business rules.

Syntax :

```
CREATE TRIGGER trigger_name
BEFORE INSERT ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```



BEFORE
INSERT
Trigger

Executes
before a new
row is inserted.

Example :

```
DELIMITER $$
CREATE TRIGGER before_insert_emp
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    -- Set default values before inserting
    IF NEW.join_date IS NULL THEN
        SET NEW.join_date = CURDATE();
    END IF;
END $$
DELIMITER ;
```

Explanation :

- * The trigger before_insert_emp runs before inserting a row into employees.
- * If join_date is not provided, it is set to the current date.
- * **NEW** refers to the new row being inserted.



Use Cases :

- * Set default values.
- * Validate data before insertion.
- * Prevent invalid data from being stored.

21.4 AFTER INSERT

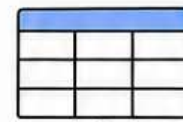
- * An **AFTER INSERT** trigger is executed **after** a new row has been inserted into the table.
- * It is used to perform **additional actions** based on the inserted data.
- * The inserted row is already stored in the table when this trigger runs.
- * It is commonly used for **logging**, **notifications** and **summary updates**.

Syntax :

```
CREATE TRIGGER trigger_name
AFTER INSERT ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```

Example :

```
DELIMITER $$
CREATE TRIGGER after_insert_emp
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    -- Insert log into emp_log table
    INSERT INTO emp_log(emp_id, action, action_time)
    VALUES (NEW.emp_id, 'Inserted', NOW());
END $$
DELIMITER ;
```



AFTER
INSERT
Trigger

Executes
after a new
row is inserted.

Explanation :

- * The trigger after_insert_emp runs after inserting a row into employees.
- * It records the inserted employee id, action and time in emp_log table.
- * This helps in maintaining an **audit trail**.

Use Cases :

- * Keep audit logs.
- * Update summary or related tables.
- * Send notifications or alerts.
- * Maintain history of changes.

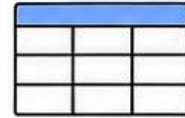


21.5 BEFORE UPDATE

- * A **BEFORE UPDATE** trigger is executed before an existing row is updated.
- * It allows you to **check** or **change** values before the update occurs.
- * It can be used to **validate data** and prevent invalid updates.
- * It helps in maintaining **data integrity** and business rules.

Syntax :

```
CREATE TRIGGER trigger_name
BEFORE UPDATE ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```



BEFORE
UPDATE
Trigger

Executes
before a row
is updated.

Example :

```
DELIMITER $$
CREATE TRIGGER before_update_emp
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Prevent negative salary
    IF NEW.salary < 0 THEN
        SET NEW.salary = OLD.salary;
    END IF;
END $$
DELIMITER ;
```

Explanation :

- * The trigger before_update_emp runs before updating a row in employees.
- * It checks if the new salary is negative.
- * If yes, it restores the old salary value and prevents the update.
- * **NEW** refers to the new row values.
- * **OLD** refers to the existing row values.



Use Cases :

- * Validate data before updating.
- * Prevent invalid or unwanted updates.
- * Enforce business rules on data modifications.
- * Maintain consistency of important data.



21.6 AFTER UPDATE

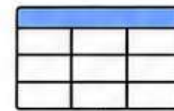
- * An **AFTER UPDATE** trigger is executed **after** a row has been updated.
- * It is used to perform **actions** after the update is complete.
- * The updated data is already stored in the table when this trigger runs.
- * It is commonly used for **logging**, **notifications** and **auditing**.

Syntax :

```
CREATE TRIGGER trigger_name
AFTER UPDATE ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```

Example :

```
DELIMITER $$
CREATE TRIGGER after_update_emp
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Insert update details into audit table
    INSERT INTO emp_audit(emp_id, old_salary, new_salary,
                          action_time)
    VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());
END $$
DELIMITER ;
```



AFTER
UPDATE
Trigger

Executes
after a row
is updated.

Explanation :

- * The trigger after_update_emp runs after updating a row in employees.
- * It records the employee id, old salary, new salary and the time of update into the emp_audit table.
- * It helps in tracking and auditing changes.



Use Cases :

- * Keep audit trail of changes.
- * Track modifications for security purposes.
- * Send notifications or alerts after updates.
- * Update related tables or summary reports.



21.7 BEFORE DELETE

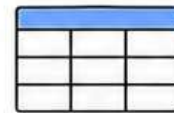
- * A **BEFORE DELETE** trigger is executed before a row is deleted from the table.
- * It allows you to **check data** or perform actions before deletion occurs.
- * It can be used to **prevent deletion** of important data.
- * It helps in maintaining **data integrity** and business rules.

Syntax :

```
CREATE TRIGGER trigger_name
BEFORE DELETE ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```

Example :

```
DELIMITER $$
CREATE TRIGGER before_delete_emp
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    -- Prevent deletion of important employee
    IF OLD.is_manager = 'Y' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete a manager record';
    END IF;
END $$
DELIMITER ;
```



BEFORE
DELETE
Trigger

Executes
before a row
is deleted.

Explanation :

- * The trigger before_delete_emp runs before deleting a row from employees.
- * OLD refers to the row that is about to be deleted.
- * If the employee is a manager, deletion is blocked.
- * This helps in preventing loss of important data.



Use Cases :

- * Prevent deletion of critical records.
- * Validate conditions before deletion.
- * Log data before it is removed.
- * Maintain referential integrity.



21.8 AFTER DELETE

- * An **AFTER DELETE** trigger is executed **after** a row has been deleted.
- * It is used to perform actions after the deletion is complete.
- * The row is already removed from the table when this trigger runs.
- * It is commonly used for **logging**, **cleanup** and **notifications**.

Syntax :

```
CREATE TRIGGER trigger_name
AFTER DELETE ON table_name
FOR EACH ROW
BEGIN
    -- SQL statements
END ;
```

Example :

```
DELIMITER $$
CREATE TRIGGER after_delete_emp
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
    -- Insert deleted record details into log table
    INSERT INTO emp_delete_log(emp_id, name, salary, del_time)
    VALUES(OLD.emp_id, OLD.name, OLD.salary, NOW());
END $$
DELIMITER ;
```



AFTER
DELETE
Trigger

Executes
after a row
is deleted.

Explanation :

- * The trigger after_delete_emp runs after deleting a row from employees.
- * It records the deleted employee details into emp_delete_log table.
- * OLD contains the values of the deleted row.
- * This is useful for audit and tracking.



Use Cases :

- * Keep audit trail of deleted data.
- * Cleanup related data in other tables.
- * Send notifications after deletion.
- * Generate reports on deleted records.



Key Takeaway :

- ✓ **BEFORE** triggers run before the event and can prevent or modify the action.
- ✓ **AFTER** triggers run after the event and are used for logging, auditing and other tasks.
- ✓ **OLD** refers to the existing row. **NEW** refers to the new row (used in **INSERT** and **UPDATE**).



21.9 OLD and NEW

- * MySQL provides **OLD** and **NEW** values in triggers to refer to the data before and after the event.
- * **OLD** refers to the existing row (before change or delete).
- * **NEW** refers to the new row (after insert or update).
- * They are available only in **row-level triggers**.

Syntax :

```
CREATE TRIGGER trigger_name
BEFORE | AFTER INSERT | UPDATE | DELETE
ON table_name
FOR EACH ROW
BEGIN
    -- Use OLD.column_name and NEW.column_name
    -- SQL statements
END ;
```

Example :

```
DELIMITER $$
CREATE TRIGGER before_update_emp
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO emp_audit(emp_id, old_salary, new_salary,
        action_time)
    VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());
END $$
DELIMITER ;
```

OLD (Existing Row)

id	name	salary
1	John	50000

Available in **UPDATE, DELETE**
(before the change)

NEW (New Row)

id	name	salary
1	John	60000

Available in **INSERT, UPDATE**
(after the change)

Availability of OLD and NEW :

Trigger Type	OLD	NEW
BEFORE INSERT	Not Available	Available
AFTER INSERT	Not Available	Available
BEFORE UPDATE	Available	Available
AFTER UPDATE	Available	Available
BEFORE DELETE	Available	Not Available
AFTER DELETE	Available	Not Available

Explanation :

- * **OLD.salary** refers to the salary before update.
- * **NEW.salary** refers to the salary after update.
- * This helps in maintaining an audit trail.



Use Cases :

- * Track changes in data.
- * Maintain history of updates and deletions.
- * Validate data before changes.
- * Prevent invalid operations.
- * Maintain referential integrity.



21.10 Dropping Triggers

- * If a trigger is no longer needed, it can be removed using the **DROP TRIGGER** statement.
- * Once dropped, it will not be executed for any events.
- * It is good practice to **drop triggers** before recreating them to avoid errors.

Example :

```
-- Drop a specific trigger
DROP TRIGGER IF EXISTS before_insert_emp ;
-- Drop another trigger
DROP TRIGGER IF EXISTS after_update_emp ;
-- Without IF EXISTS (if you are sure it exists)
DROP TRIGGER before_delete_emp ;
```

Best Practices :

- ✓ Always use **IF EXISTS** to avoid errors.
- ✓ Drop triggers before modifying the table structure.
- ✓ Document the purpose of each trigger before dropping.
- ✓ Test the application after dropping triggers.

Syntax :

```
DROP TRIGGER [IF EXISTS] trigger_name ;
```



Explanation :

- * **IF EXISTS** prevents an error if the trigger does not exist.
- * Once a trigger is dropped, it is completely removed from the database.
- * You can recreate the trigger anytime using **CREATE TRIGGER**.



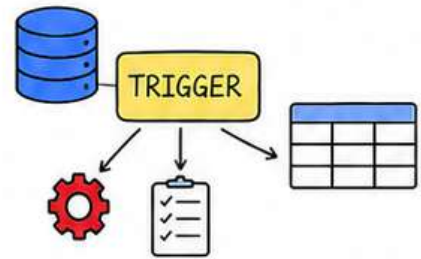
Show Triggers of a Table :

```
SHOW TRIGGERS FROM table_name ;
```



21.11 Trigger Use Cases

- * Triggers are very useful in real-time applications to automate tasks, maintain data integrity and enforce business rules.
- * Below are some common **use cases** of triggers in MySQL.
- * These use cases help **reduce manual work** and **improve consistency**.



Use Case	Trigger Type	Description	Example
1. Audit Logging	AFTER INSERT AFTER UPDATE AFTER DELETE	Log all data changes into an audit table for tracking purpose.	Log employee inserts, updates or deletes into <code>emp_audit</code> table.
2. Data Validation	BEFORE INSERT BEFORE UPDATE	Validate data before it is stored in the table.	Check <code>salary > 0</code> before inserting/updating an employee.
3. Enforce Business Rules	BEFORE INSERT BEFORE UPDATE	Enforce business rules such as unique values, limits, formats, etc.	Prevent inserting duplicate email addresses in <code>users</code> table.
4. Maintain Summary Tables	AFTER INSERT AFTER UPDATE AFTER DELETE	Keep summary or aggregate tables up to date automatically.	Update <code>total_sales</code> in <code>daily_sales</code> table whenever a new order is inserted.
5. Notification System	AFTER INSERT AFTER UPDATE AFTER DELETE	Send notifications or alerts based on data changes.	Send email to <code>admin</code> when a new user is registered.
6. Prevent Invalid Operations	BEFORE DELETE BEFORE UPDATE	Prevent deletion or update of important records.	Prevent deleting a <code>manager</code> record if employees are assigned to that manager.
7. Automatically Update Timestamps	BEFORE UPDATE	Automatically update timestamp columns.	Update <code>updated_at</code> column with current timestamp.
8. Inventory Management	AFTER INSERT AFTER DELETE	Adjust stock levels in inventory based on sales or returns.	Decrease <code>stock</code> when an order is placed, increase on return.

Example 1:

```
-- Audit log trigger after insert
DELIMITER $$
CREATE TRIGGER after_insert_emp
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    -- Insert details into audit table
    INSERT INTO emp_audit(emp_id, action, action_time)
    VALUES (NEW.emp_id, 'INSERT', NOW());
END $$
DELIMITER ;
```

Explanation :

- * This trigger runs after a new employee is inserted.
- * It inserts the employee id, action and time into `emp_audit` table.
- * **NEW** refers to the newly inserted row.
- * Helps in maintaining audit trail.



Use Cases in Real Applications :

- * Banking : Track all transactions or account changes.
- * E-commerce : Update inventory and order summary.
- * HR Systems : Log employee data changes.
- * School Management : Track student record changes.
- * Healthcare : Maintain logs of patient data updates.



Example 2:

```
-- Prevent negative salary
DELIMITER $$
CREATE TRIGGER before_insert_emp
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Salary cannot be negative';
    END IF;
END $$
DELIMITER ;
```

Explanation :

- * This trigger runs before inserting a row.
- * If salary is negative, it raises an error.
- * Prevents invalid data from entering the table.
- * Helps in data validation.



Benefits of Using Triggers :

- * Automates repetitive tasks.
- * Ensures data integrity and consistency.
- * Enforces business rules at database level.
- * Reduces manual errors.
- * Improves performance and reliability.

