IT Department

Triggers in MySQL (Validating Input Data)

Soma Soleiman Zadeh

Database Systems II (IT 216)

Spring 2025 - 2026

Week 14

May 04, 2026

Outline

- **BEFORE** Row_level Triggers
- **Triggers** for Validating Input Data
- **Triggers** for Enforcing Business Rules

Why We Need Triggers in Database?

- We already mentioned following reasons as some of the main reasons for using triggers:
  - **Validating Input Data**
  - Keeping a Log of Records
  - **Enforce Business Rules**
- In the previous lecture, we explained **After** Row-Level triggers for ***keeping a log of records***.
- This lecture's focus is on **Before** Row-Level triggers that are used for ***validating input data*** *or* ***enforcing business rules***.

Types of Triggers

1. **Before Insert** : It is activated **before the insertion** of data into the table.
2. **After Insert** : It is activated **after the insertion** of data into the table.
3. **Before Update** : It is activated **before the update** of data in the table.
4. **After Update** : It is activated **after the update** of the data in the table.
5. **Before Delete** : It is activated **before the deletion** of data from the table.
6. **After Delete** : It is activated **after the deletion** of data from the table.

Syntax of Creating Trigger in MySQL

```
DELIMITER //
CREATE TRIGGER trigger_name
(BEFORE | AFTER) (INSERT | UPDATE | DELETE) ON table_name
FOR EACH ROW
BEGIN
        <Trigger Statements>
END//
DELIMITER ;
```

Validating Input Data Scenario

- We are going to create a trigger that is activated **before** **inserting** a row into the **product** table.
- The trigger is activated **before** a user inserted data into the **product** table and avoids entering negative value for **price** column.
- If a user wanted to insert data of a new product to the table while the **price** value is negative, the trigger doesn't allow this insertion and shows an error message to the user.

ERROR: Price Can not be Negative!

- In case **price** value of the new product is not negative, the trigger lets the insertion happens.

Product Table

PID	Pname	Price

```
insert into Product values (1, 'Tablet', -200);
```

How to Return Error Message to User?

- By using **SIGNAL** Statement:
  - **SIGNAL** statement is a way to "return" an error.
  - **SIGNAL** provides error information to the client.

- Syntax of **SIGNAL** Statement:

  SIGNAL SQLSTATE sqlstate_value **SET** Error_Message;

- To signal a generic **SQLSTATE** value, use **'45000'**, which means "unhandled user-defined exception."

Creating Validate_Price Trigger on Product Table

```
delimiter //
CREATE TRIGGER validate_price
BEFORE INSERT ON product
FOR EACH ROW
BEGIN
    IF NEW.price < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Price cannot be negative';
    END IF;
END//
delimiter ;
```

Enforcing Business Rules Scenario

- Suppose that a business has a limitation on employees' salaries as follows:

Limitation for Salary: **0 <= Salary <= 5000**

- Define a trigger that is activated before any update is going to happen on the **Employee** table.
- The trigger considers the above limitation for salary. So, if the updated salary becomes more than $5000, it automatically sets to the maximum amount of salary ($5000), and if the updated salary becomes less than $0, it sets to $0.

Enforcing Business Rules Scenario

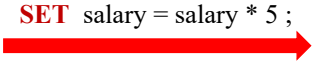
Limitation for Salary: **0 <= Salary <= 5000**

- For example, when the following update statement is executed, before it affects the **Employee** table, the trigger is executed to check the rule of salary amount (0 <= Salary <= 5000).
- The current salary for "Peter" is $900, and after updating his salary, it becomes $900*5=$4500, which is in allowed range of salary.
- The current salary for "Sandy" is $1300, and after updating his salary, it becomes $1300*5=$6500, which is NOT in allowed range of salary. So, the trigger automatically sets his salary to $5000, instead of $6500.

Employee Table

ID	name	salary
1	Peter	900
2	Sandy	1300

```
UPDATE Employee
SET salary = salary * 5 ;
```



ID	name	salary
1	Peter	**4500**
2	Sandy	**5000**

Creating Validate_Salary Trigger

```
delimiter //
CREATE TRIGGER validate_salary
BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
    IF NEW.salary > 5000 Then SET NEW.salary=5000;
    ELSEIF NEW.salary < 0 Then SET NEW.salary = 0;
    END IF;
END//
delimiter ;
```

Enforcing Business Rules Scenario

- For the previous scenario, define a trigger that is activated before any update is going to happen on the **Employee** table.

Limitation for Salary: **0 <= Salary <= 5000**

- This trigger considers the above limitation for salary, but if the updated salary becomes more than $5000 or less than $0, the trigger sends an error message to the user and doesn't allow the updating statement to be executed.

Employee Table

ID	name	salary
1	Peter	900
2	Sandy	1300

UPDATE Employee
SET salary = salary * 5 ;

ERROR: Update is against the rules.

ID	name	salary
1	Peter	4500
2	Sandy	1300

Creating Validate_Salary_V2 Trigger

```
delimiter //
CREATE TRIGGER validate_salary_v2
BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
    IF NEW.salary > 5000 OR New.salary < 0 THEN
        SIGNAL SQLSTATE '45000' SET message_text='Update is against the rules!';
    END IF;
END//
delimiter ;
```