IT Department

Triggers in MySQL (Log of Records)

Soma Soleiman Zadeh

Database Systems II (IT 216)

Spring 2024 - 2025

Week 15

May 13, 2025

Outline

- **Triggers** in MySQL
- **Row_level** Triggers vs. **Statement_level** Triggers
- **BEFORE** Row_Level Trigger and **AFTER** Row_Level Trigger
- Syntax of Creating Row_Level **Trigger** in MySQL
- **Triggers** for Log of Record

What is Trigger?

- A **trigger** is a special type of **stored procedure** that is *invoked automatically in response to an event*.
  - Each trigger is associated with a table,
  - which is activated on any **DML statement** such as **INSERT**, **UPDATE**, or **DELETE**.

Why We Need Triggers in Database?

- Triggers are useful in many situations. Some of the main reasons for using triggers are:
  - **Validating** Input Data
  - Keeping a **Log of Records**
  - **Enforce Business Rules**

Trigger vs. Procedure

- **Trigger** is a special procedure, but
  - **Procedure** must be called by the user.
  - **Trigger** is activated (called) automatically when a data modification event is made on a table.

Row-Level Trigger vs. Statement-Level Trigger

- **Row-Level Trigger :**
  - This type of trigger is executed once for each row affected by a data modification operation, such as **INSERT**, **UPDATE**, or **DELETE**.
- **Statement-Level Trigger :**
  - This type of trigger fires once for each SQL statement executed, **regardless of the number of rows that have been changed**.

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 a Row-Level Trigger in MySQL

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

Log of Records Scenario

- We are going to create a trigger that is activated after **inserting** a row into the **product** table.
- The trigger is activated whenever a user inserted data into the **product** table and fills the **Product_Log** table with a log of the insertion event.
- The log contains information on the action name, which user did the insertion and at which date this insertion happened.

Product Table

PID	Pname	Price
1	Tablet	150
..	..	..
..	..	..

```
insert into Product values (1, 'Tablet', 150);
```

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	Null	150	root	10 May 2025

Log of Records Scenario

```
insert into Product values (1, 'Tablet', 150);
insert into Product values (2, 'Laptop', 1200);
```

Product Table

PID	Pname	Price
1	Tablet	150
2	Laptop	1200
..	..	..

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	Null	150	root	10 May 2025
2	Insert	Null	1200	ali	12 May 2025

Product and Product_Log Tables

- Suppose Product table is already created.

```
create table Product
(PID int auto_increment primary key,
PName varchar(100),
Price int);
```

Product Table

PID	Pname	Price

- We also need another table (**Product_Log**) to keep log of records.

```
create table Product_log
(id int auto_increment primary key,
action_name varchar(10),
oldPrice int,
newPrice int,
by_User varchar(20),
action_date date);
```

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date

Triggers for Log of Records

- Now, we will create three Triggers for the **Product** table and save the log of data into the **Product_Log** table.
  - **First Trigger** is activated after executing **INSERT** statement on **Product** table.
  - **Second Trigger** is activated after executing **DELETE** statement on **Product** table.
  - **Third Trigger** is activated after executing **UPDATE** statement on **Product** table.

Product Table

PID	Pname	Price
1	Tablet	150

→

PID	Pname	Price
1	Tablet	150
2	Laptop	1200

→

PID	Pname	Price
1	Tablet	150
2	Laptop	1000

→

PID	Pname	Price
~~1~~	~~Tablet~~	~~150~~
2	Laptop	1000

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	*Null*	150	root	10 May 2025
2	Insert	*Null*	1200	user1	12 May 2025
3	Update	1200	1000	user1	13 May 2025
4	Delete	150	*Null*	user2	15 May 2025

Creating Insertion Trigger

```
delimiter &&
create trigger product_insert
after insert on product
for each row
begin
    insert into product_log
    set
        action_name = 'Insert',
        newPrice = new.price,
        by_User = user(),
        action_date = current_date();
end&&
delimiter ;
```

The Effect of Insertion Trigger

```
insert into product(PName, Price) values ('Tablet', 150);
insert into product(PName, Price) values ('Laptop', 1200);
```

Product Table

PID	Pname	Price
1	Tablet	150
2	Laptop	1200

The Effect of Insertion Trigger

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	*Null*	150	root	10 May 2025
2	Insert	*Null*	1200	user1	12 May 2025

Creating Updating Trigger

```
delimiter &&
create trigger product_update
after update on product
for each row
begin
    insert into product_log
    set
        action_name = 'Update',
        oldPrice = old.price,
        newPrice = new.price,
        by_User = user(),
        action_date = current_date();
end&&
delimiter ;
```

The Effect of Updating Trigger

```
insert into product(PName, Price) values ('Tablet', 150);
insert into product(PName, Price) values ('Laptop', 1200);
```

```
update Product set Price=1000 where PName='Laptop';
```

Product Table

PID	Pname	Price
1	Tablet	150
2	Laptop	**1000**

The Effect of Updating Trigger

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	*Null*	150	root	10 May 2025
2	Insert	*Null*	1200	user1	12 May 2025
3	Update	1200	1000	user1	13 May 2025

Creating Deletion Trigger

```
delimiter &&
create trigger product_delete
after delete on product
for each row
begin
    insert into product_log
    set
        action_name = 'Delete',
        oldPrice = old.price,
        by_User = user(),
        action_date = current_date();
end&&
delimiter ;
```

The Effect of Deletion Trigger

```
insert into product(PName, Price) values ('Tablet', 150);
insert into product(PName, Price) values ('Laptop', 1200);
```

```
update Product set Price=1000 where PName='Laptop';
```

```
delete from product where PName = 'Tablet';
```

Product Table

PID	Pname	Price
~~1~~	~~Tablet~~	~~150~~
2	Laptop	1200

The Effect of Deletion Trigger

Product_Log Table

id	Action_name	Old Price	New Price	By_User	Action_Date
1	Insert	*Null*	150	root	10 May 2025
2	Insert	*Null*	1200	user1	12 May 2025
3	Update	1200	1000	user1	13 May 2025
4	Delete	150	*Null*	user2	15 May 2025