Database Systems

Lecture

Triggers

Contents

- Triggers overview
- Creating a trigger – syntax
- Creating a trigger – example
- Some notes on using triggers
- Benefits and uses of triggers
- Cautions in using triggers

Triggers overview

- In the context of databases, a trigger is a database object that is associated with a table
  - It is activated when a certain event occurs on the table
  - Upon activation it performs certain actions
- Triggers are used to monitor database state for any violations of business rules, and take appropriate actions if any such violations are observed

Triggers overview

- MySQL Triggers consist of two components
  - The event(s)
    - Usually the database operations such as insertion, update or deletion of rows
  - The action(s)
    - The tasks to be performed if an event occurred

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
<action>;
```

Specifies the time of firing the trigger. Possible values:

BEFORE

AFTER

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
<action>;
```

Specifies the event which fires the event. Possible values:

INSERT
UPDATE
DELETE

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
<action>;
```

The table for which the trigger is being created

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
<action>;
```

Specifies that the action will be carried out on each row that is affected by the triggering event

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
    <action>;
```

The SQL command to be executed

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
<action>;
```

Create Trigger syntax

```
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
BEGIN
<action>
END;
```

If there are multiple SQL commands in the action, then these MUST be put in a BEGIN…END block

Create Trigger syntax

```
delimiter //
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
BEGIN
<action>
END;
//
delimiter ;
```

Now this is something not to be worried about much. Just use it! Basically we're telling MySQL ***not*** to use the semicolon as end of command, but use // instead

This is only required if we're using BEGIN…END

Create Trigger syntax

```
delimiter //
CREATE TRIGGER trigger_name
<trigger activation time>
<trigger event>
ON table_name
FOR EACH ROW
BEGIN
<action>
END;
//
delimiter ;
```

We have finished defining the trigger – now we can switch back to the semicolon as the delimiter

Trigger example

```
delimiter |
CREATE TRIGGER      check_emp_salary
BEFORE INSERT ON    Employee
FOR EACH ROW
BEGIN
     IF   NEW.Salary < 0
     THEN SET NEW.Salary = 0;
     END IF;
END;
|
Delimiter ;
```

The NEW keyword is used for INSERT triggers – it represents the row to be added

Trigger example

- Let's assume we have a table 'Employees' containing attributes such as E_SSN., Name, Department, S_SSN, and Salary
- Suppose the business rule states that if the salary of a employee is more than his/her supervisor's salary, then the supervisor's salary will be increased to match the employee's salary
- What could be possible events for this trigger?
  - Addition of a new employee
  - Update of an employee's salary
  - Change of an employee's supervisor
- The code in the action component will check if the salary of supervisor is less than the employee. If yes, then the supervisor's salary will be made equal to the employee's salary!

Trigger example

```
delimiter |
CREATE TRIGGER      check_emp_salary
BEFORE INSERT ON    Employee
FOR EACH ROW
BEGIN
      IF      NEW.Salary > (SELECT Salary FROM Employee
                            WHERE E_SSN = NEW.S_SSN)
      THEN    UPDATE Employee SET Salary = NEW.Salary
              WHERE E_SSN = NEW.S_SSN;
      END IF;
END;
|
Delimiter ;
```

Some notes on Triggers

- We just saw that for `INSERT` triggers, we can access the data to be inserted using `NEW` keyword which represents the new row to be added
- For `DELETE` triggers, the row that has been deleted can be accessed using `OLD` keyword. There is no `OLD` in `INSERT` triggers, and there is no `NEW` in `DELETE` triggers!
- You must be wondering about the `UPDATE` triggers, right?
- In an `UPDATE` trigger, you can access both the `OLD` and `NEW` rows, representing the data before and after update

Benefits of using Triggers

- Fast and consistent application development
  - The triggers are stored in the database, so the application developer do not need to code the trigger actions into each database application
- Global enforcement of business rules
  - Define a trigger once and then reuse it for any application that uses the database
- Easier maintenance
  - If a business rule changes, we only need to change the corresponding trigger action and it’ll be applied to all applications using the database

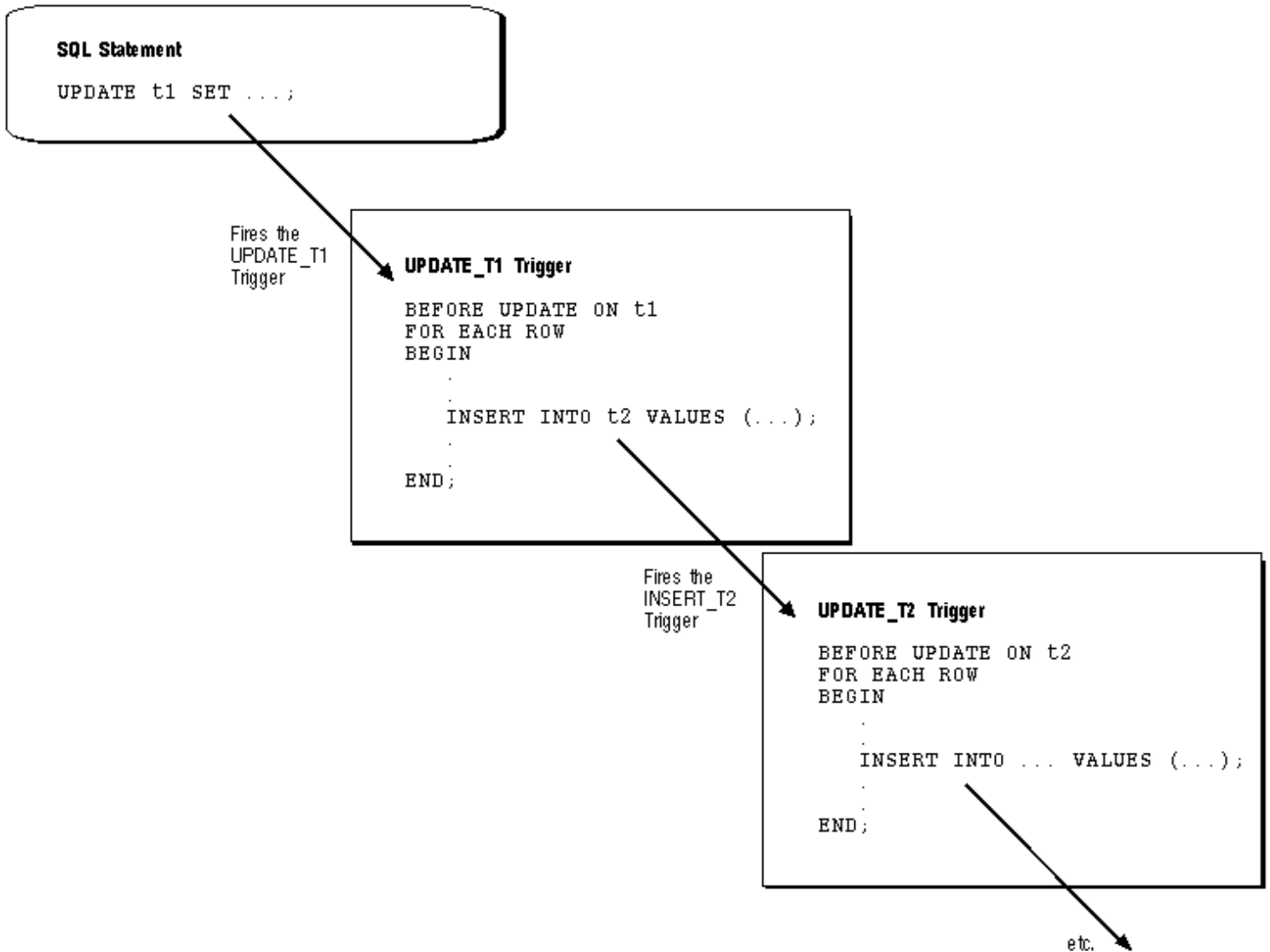
Uses of Triggers

- Maintaining database consistency
- Monitoring database updates
- Updating derived attributes automatically
- Preventing invalid transactions
- Enforcing complex business rules
- Providing event logging

Caution in use of Triggers

- If triggers are not used carefully, these can result in unintended modifications in the data
- This is especially true in the case of cascaded Triggers, where the action performed in one trigger activates another trigger, which may activate another…

Caution in use of Triggers



Thanks a lot