# SQL – Constraints and Triggers

## CS 4750
## Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.5.3]

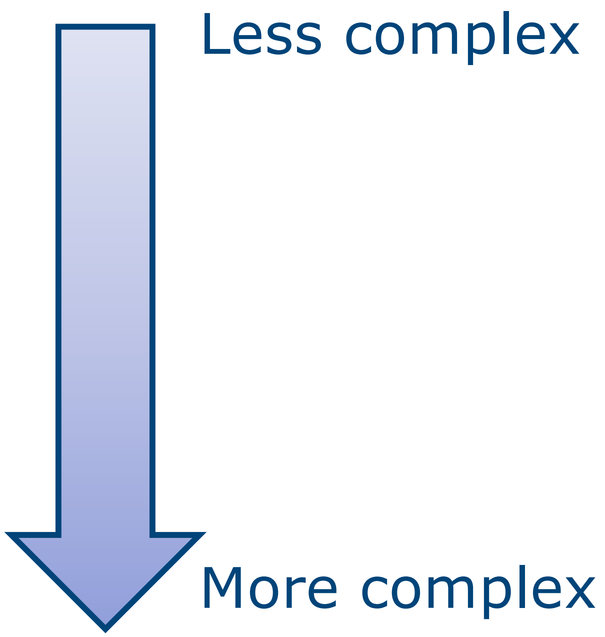
# Advanced SQL Commands

How much of our business logic should reside at the presentation layer, logic layer, or data layer?

Shift some logic from the logic layer to the data layer

Why?

- Data types
- Integrity constraints
- Checks
- Assertions
- Triggers
- Stored procedure



# Why Constraints and Triggers?

- A serious problem with apps that update the database is that the new info could be wrong

- Always verify the info before updating/inserting

- Several ways to verify:
  - Human level
  - App level
  - SQL / Database level

- SQL / database level – expressing integrity constraints as part of the database schema
  - Data types
  - Key constraints: primary key, foreign-key, unique
  - Triggers: event-condition-action rule

# Referential-Integrity Constraints

## Foreign key

- Assertion that values for certain attributes must make sense
- The referenced attribute(s) must be declared `UNIQUE` or the `PRIMARY KEY` of their relation
- Values of the foreign key must also appear in the referenced attributes of some tuple
- Values of the referenced attributes must be non-NULL

```
CREATE TABLE Product(
      pid INT,
      name CHAR(30),
      cid INT,
      PRIMARY KEY (pid),
      FOREIGN KEY (cid) REFERENCES company(cid) );
```

# Add or Drop a Foreign Key

- If the table already exists, alter the table to add the foreign key

```
ALTER TABLE buyer
ADD FOREIGN KEY (bid) REFERENCES cust(id);
```

- Named foreign key

```
ALTER TABLE buyer
ADD CONSTRAINT FK_buyer_cust
FOREIGN KEY (bid) REFERENCES cust(id);
```

- Drop a foreign key

```
ALTER TABLE buyer
DROP FOREIGN KEY FK_buyer_cust;
```

# Maintaining Referential Integrity

What are actions that will be prevented by the DBMS if we have referential integrity?

- Try to insert/update with non-existent foreign key
- Try to delete a referenced attribute

Three options to maintain the integrity

- Default policy: reject
- Cascade policy: also change the referenced attributes
- Set-null policy: set a foreign key value to NULL

## Maintaining Referential Integrity

Default policy: reject

```
ALTER TABLE buyer
ADD CONSTRAINT FK_buyer_cust
FOREIGN KEY (bid) REFERENCES cust(id);
```

Cascade policy: also change the referenced attributes

```
ALTER TABLE buyer
ADD CONSTRAINT FK_buyer_cust
FOREIGN KEY (bid) REFERENCES cust(id)
ON DELETE CASCADE;
```

Set-null policy: set a foreign key value to NULL

```
ALTER TABLE buyer
ADD CONSTRAINT FK_buyer_cust
FOREIGN KEY (bid) REFERENCES cust(id)
ON DELETE SET NULL;  -- allow if bid is not PK of buyer
```

## Attribute-based CHECK Constraints

- Limit the value range for a column

```
CREATE TABLE buyer (
      bid INT NOT NULL,
      bname VARCHAR(20) NOT NULL,
      rating INT NOT NULL,
      age DOUBLE CHECK (age >= 18),
      PRIMARY KEY (bid) );
```

- Attempting to insert a record with age < 18 will be rejected by DBMS

- If the table already exists, alter the table to add the constraint

```
ALTER TABLE buyer
ADD CONSTRAINT checkAge
CHECK (age >= 18);
```

Name is optional

If name is specified, use the given name. Otherwise, use DBMS' default name (e.g., buyer_chk_1)

# Named CHECK Constraint

```
   CREATE TABLE buyer (
         bid INT NOT NULL,
         bname VARCHAR(20) NOT NULL,
         rating INT NOT NULL,
         age DOUBLE,
         CONSTRAINT checkAge CHECK (age >= 18),
         PRIMARY KEY (bid) );
```

```
   ALTER TABLE buyer DROP CONSTRAINT checkAge
```

To see existing check constraints, select `information_schema` database

```
SELECT * FROM CHECK_CONSTRAINTS;
```

# Tuple-based CHECK Constraints

- Limit the value range for multiple columns
- Check the condition every time a tuple is inserted into a table
- Check the condition every time a tuple is updated

```
CREATE TABLE buyer (
      bid INT NOT NULL,
      bname VARCHAR(20) NOT NULL,
      rating INT NOT NULL,
      age DOUBLE,
      CHECK (age >= 18 AND rating > 3),
      PRIMARY KEY (bid) );
```

- Attempting to insert a record with age < 18 or rating < 3 will be rejected by DBMS

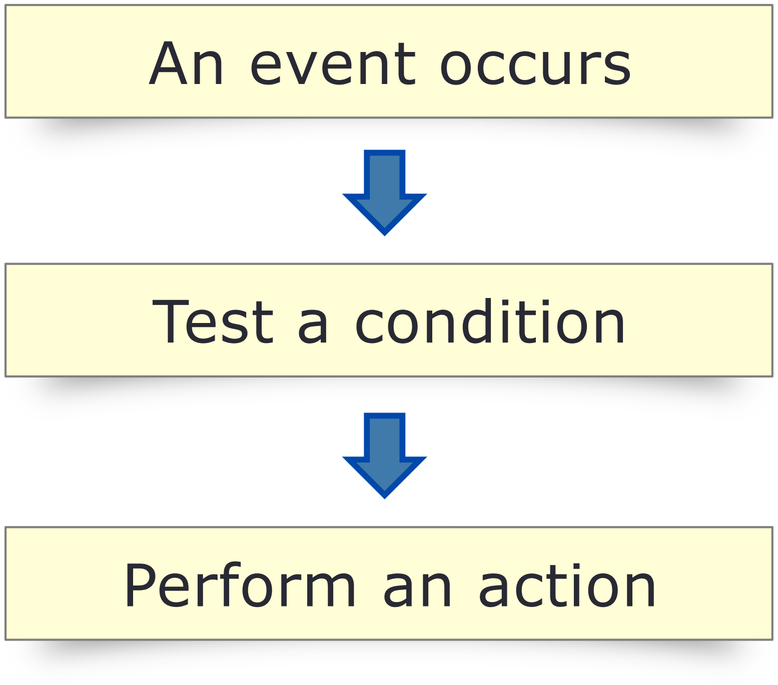
## Named CHECK Constraint

```
CREATE TABLE buyer (
      bid INT NOT NULL,
      bname VARCHAR(20) NOT NULL,
      rating INT NOT NULL,
      age DOUBLE,
      CONSTRAINT checkAgeAndRating
            CHECK (age >= 18 AND rating > 3),
      PRIMARY KEY (bid) );
```

```
ALTER TABLE buyer DROP CONSTRAINT checkAgeAndRating
```

## Triggers

- Ways to enforce business logic – want some action to happen automatically when users insert, update, or delete rows / columns
- Validate input
- Sometimes called "event-condition-action rules" (ECA rules)
- Executed when certain events occur



# Trigger Creation Statement

```
CREATE TRIGGER <trigger_name>

[BEFORE|AFTER] <event> ON <table>

<trigger-type>

BEGIN
    <trigger-body>
END;
```

Determine when the trigger occurs in response to a specific event

The table to which the trigger binds

Could be `INSERT`, `UPDATE`, or `DELETE`

Specify the type of trigger using either `FOR EACH ROW` or `FOR EACH STATEMENT`

The logic of the trigger to be executed

# Example 1 (1)

1. Make sure we have a table that we will have a trigger to manage business logic

```
    CREATE TABLE gradebook(
          sid VARCHAR(10),
          hw1 DOUBLE,
          hw2 DOUBLE,
          hw3 DOUBLE,
          hw4 DOUBLE,
          total DOUBLE,
          PRIMARY KEY (sid) );
```

# Example 1 (2)

2. Create a trigger to ensure that the total score is properly recorded

```
DELIMITER $$
CREATE TRIGGER gradebookTrigger
BEFORE INSERT ON gradebook
FOR EACH ROW
   BEGIN
      SET new.total = new.hw1 + new.hw2 +
            new.hw3 + new.hw4;
   END
$$
DELIMITER ;
```

Execute the trigger before an "INSERT" action happens

Must have a space

Trigger name

On which table

Execute the trigger body

Must have a space

## Example 1 (3)

3. Try inserting some data

Note: This example intentionally inserts incorrect total scores. Retrieve the record and verify if the total scores are stored correctly.

```
INSERT INTO gradebook
      VALUES ('111', 90, 85, 100, 90, 100);
INSERT INTO gradebook
      VALUES ('222', 100, 100, 100, 100, 100);
INSERT INTO gradebook
      VALUES ('333', 100, 95, 100, 90, 100);
```

# Example 1 (4)

To modify a trigger

Create a new trigger if it does not exist; change the trigger if it exists

```
DELIMITER $$
CREATE OR REPLACE TRIGGER gradebookTrigger
BEFORE INSERT ON gradebook
FOR EACH ROW
   BEGIN
      SET new.total = new.hw1 + new.hw2 +
                      new.hw3 + new.hw4 + 0.5;
   END
$$
DELIMITER ;
```

To Drop a trigger

On which table

Which trigger

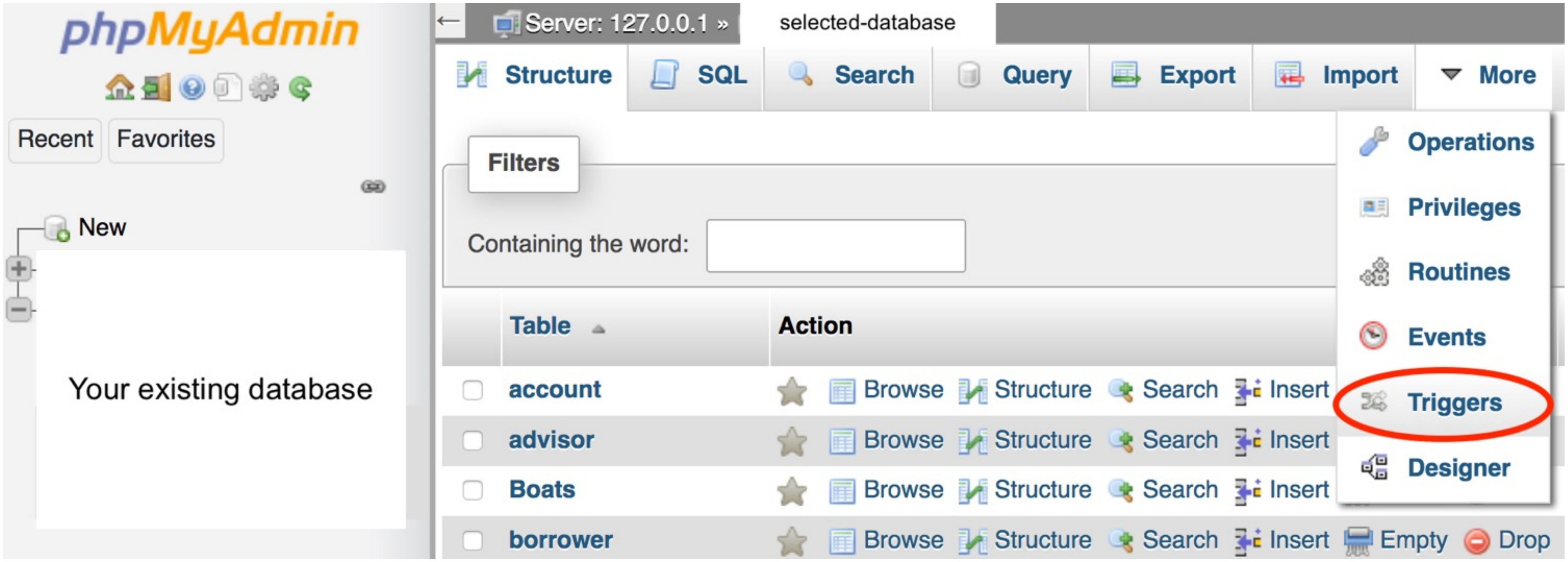
```
DROP TRIGGER IF EXISTS gradebook.gradebookTrigger;
DROP TRIGGER IF EXISTS gradebookTrigger;   -- or skip the table
```

To show existing triggers

```
SHOW TRIGGERS;
```

# Create Triggers (phpMyAdmin)

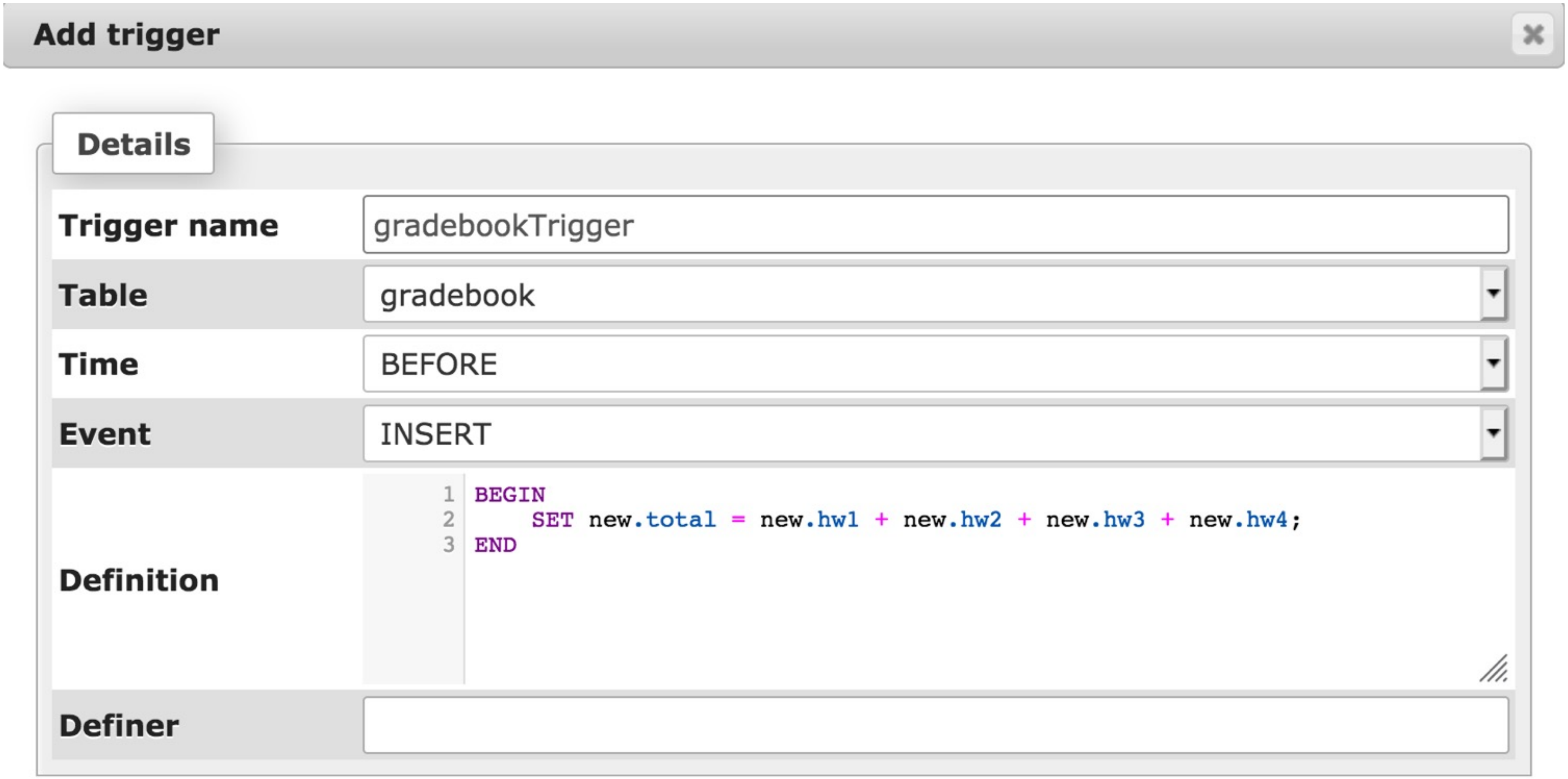
- Select the database you would like to work on
- You may create a trigger using the `Triggers` feature or write SQL manually



| | Table | Action |
|---|---|---|
| | account | Browse Structure Search Insert |
| | advisor | Browse Structure Search Insert |
| | Boats | Browse Structure Search Insert |
| | borrower | Browse Structure Search Insert Empty Drop |

# Example 1 – Use phpMyAdmin

- Enter the trigger name, which table it is associated with, when it will be executed, and what to execute



**Add trigger**

Details

|  |  |
|---|---|
| **Trigger name** | gradebookTrigger |
| **Table** | gradebook |
| **Time** | BEFORE |
| **Event** | INSERT |
| **Definition** | (see below) |
| **Definer** |  |

```
BEGIN
    SET new.total = new.hw1 + new.hw2 + new.hw3 + new.hw4;
END
```

# Example 2 (1)

Must have a space

Trigger name

```
DELIMITER $$
CREATE TRIGGER gradebook_status_update
BEFORE UPDATE ON gradebook
FOR EACH ROW
BEGIN
    IF new.total > 100 THEN
       SET new.total = 100;
    ELSEIF new.total < 60 THEN
       SET new.total = 60;
    END IF;
END
$$
DELIMITER ;
```

Execute the trigger before an "INSERT" action happens

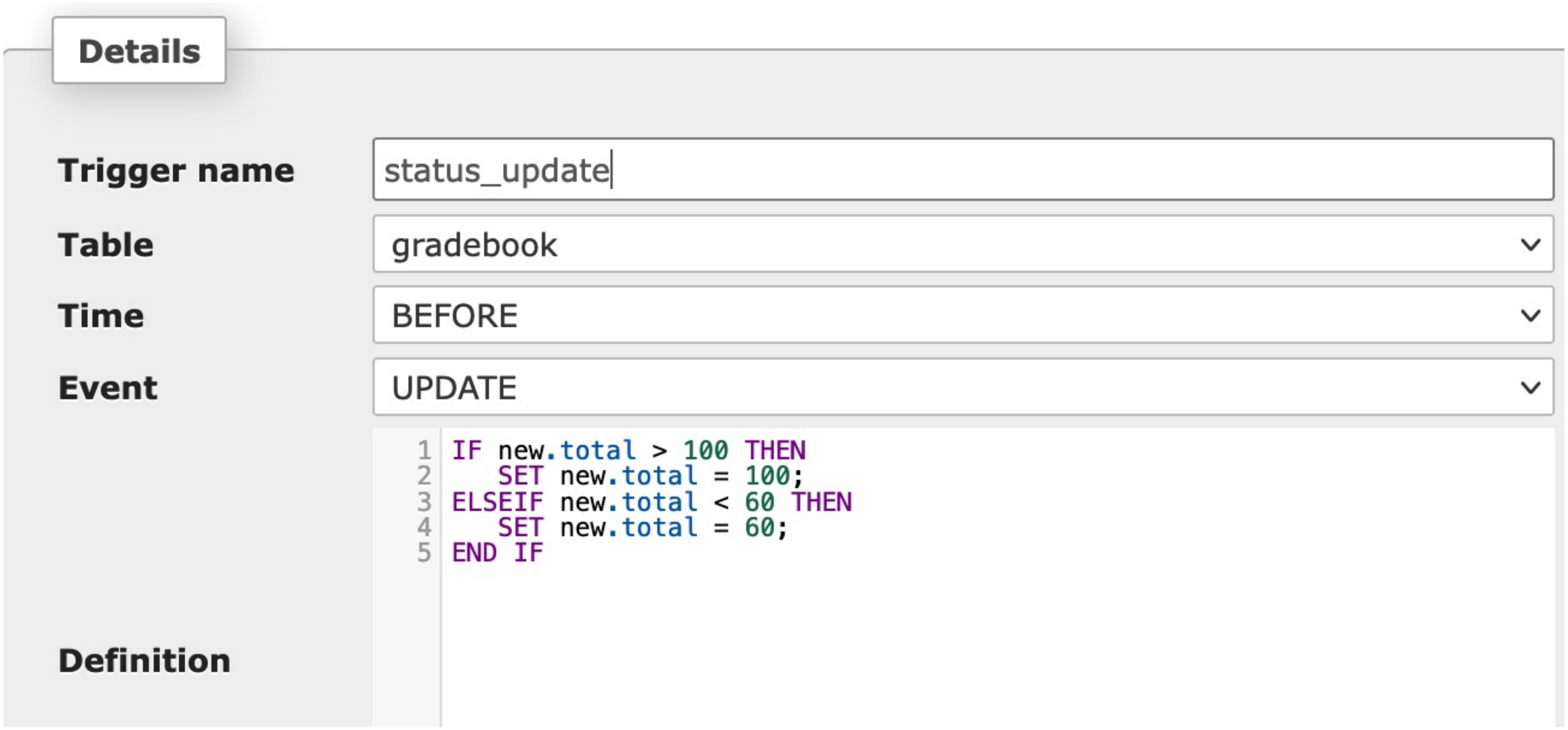
On which table

Execute the trigger body

Must have a space

# Example 2 - Use phpMyAdmin

- Enter the trigger name, which table it is associated with, when it will be executed, and what to execute



| | |
|---|---|
| Trigger name | status_update |
| Table | gradebook |
| Time | BEFORE |
| Event | UPDATE |
| Definition | |

```
IF new.total > 100 THEN
    SET new.total = 100;
ELSEIF new.total < 60 THEN
    SET new.total = 60;
END IF
```

## Example 2 (3)

Try updating some data

```
UPDATE gradebook SET total=120 WHERE sid='111';
UPDATE gradebook SET total=20 WHERE sid='333';
```

Observe the total values of `sid='111'` and `sid='333'`

## Example 3 (1)

```
CREATE TABLE employees (
             emp_id INT,
             firstname VARCHAR(40),
             lastname VARCHAR(40),
             email VARCHAR(255),
             hire_date DATETIME,
             salary DECIMAL(8,2),
             dept_id VARCHAR(40),
             PRIMARY KEY (emp_ID) )
```

Suppose we want to log the changes of values in the `salary` of the `employees` table.

We need a separate table for storing the changes and use a trigger to insert the change into this table.

```
CREATE TABLE salary_changes (
             emp_id INT,
             changed_at DATETIME DEFAULT CURRENT_TIMESTAMP,
             old_salary DECIMAL(8 , 2 ),
             new_salary DECIMAL(8 , 2 ),
             PRIMARY KEY (emp_id , changed_at));
```

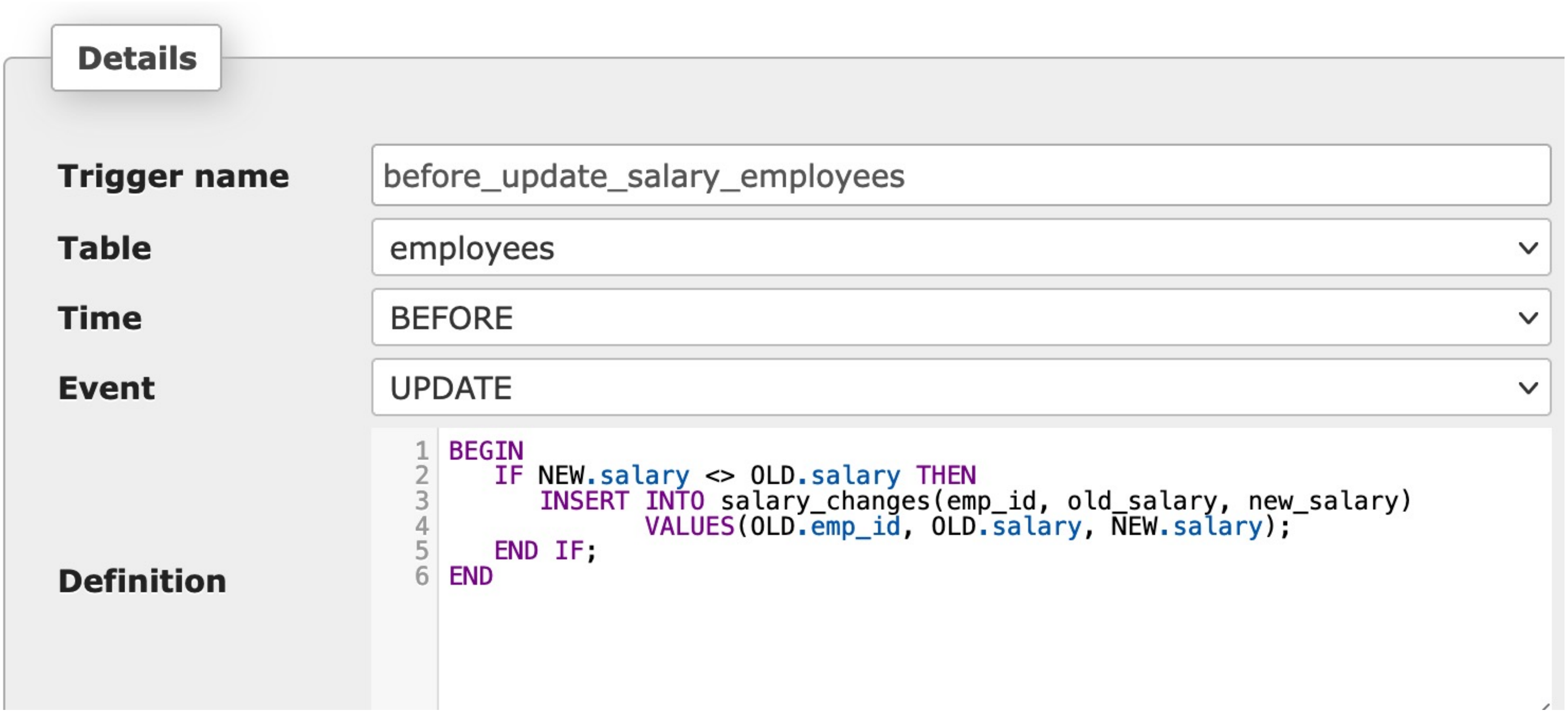
# Example 3 (2)

Then, create a trigger to log any changes of values in the `salary` of the `employees` table.

```
DELIMITER $$
CREATE TRIGGER before_update_salary
BEFORE UPDATE ON employees
FOR EACH ROW
   BEGIN
      IF NEW.salary <> OLD.salary THEN
         INSERT INTO salary_changes(emp_id,
                                    old_salary,
                                    new_salary)
         VALUES(OLD.emp_id,
                OLD.salary,
                NEW.salary);
      END IF;
   END
$$
DELIMITER ;
```

# Example 3 - Use phpMyAdmin

- Enter the trigger name, which table it is associated with, when it will be executed, and what to execute



| | |
|---|---|
| **Trigger name** | before_update_salary_employees |
| **Table** | employees |
| **Time** | BEFORE |
| **Event** | UPDATE |
| **Definition** | (below) |

```
BEGIN
    IF NEW.salary <> OLD.salary THEN
        INSERT INTO salary_changes(emp_id, old_salary, new_salary)
                VALUES(OLD.emp_id, OLD.salary, NEW.salary);
    END IF;
END
```

# Example 3 (3)

Let's insert some data to `employees`

```
INSERT INTO employees
VALUES ('110', 'Humpty', 'Dumpty', 'humpty@uva.edu',
        '2021-09-30', '9000', 'Computer Science');

INSERT INTO employees
VALUES ('111', 'Wacky', 'Tacky', 'wacky@uva.edu',
        '2021-09-30', '10500', 'Computer Science');
```

At this point, there is no record in `salary_changes table.`
Try updating the salary.

```
UPDATE employees SET salary='9800' WHERE emp_id='110';
```

Observe the `salary_changes` table. A log is added.

```
SELECT * FROM salary_changes;
```

## Example 3 (4)

Let’s update a row in `employees with the same salary`

```
UPDATE employees SET salary= '10500' WHERE emp_id='111';
```

Observe the `salary_changes` table. No new log is added.

```
SELECT * FROM salary_changes;
```

## Trigger Usage

- Log table modifications
  - Some table have sensitive data (e.g., customer email, employee salary) that all changes must be logged – need the `UPDATE` trigger to insert the changes into a separate log table

- Enforce complex integrity of data
  - Define triggers to validate the data and reformat the data before inserting or updating using a `BEFORE INSERT` or `BEFORE UPDATE` trigger

## When to Use Triggers

### Read-heavy databases

- Data do not change much
- Triggers will improve performance

### Write-heavy databases

- Data change constantly
- Triggers will negatively impact performance

# Wrap-Up

- Always use data types and do integrity checks
- Use `CHECK` constraints when possible
- Triggers can be quite complicated although very powerful in certain situations

**What's next?**

- Stored procedure