Prof. DI Dr. Erich Gams

Stored Procedures

informationssysteme htl-wels

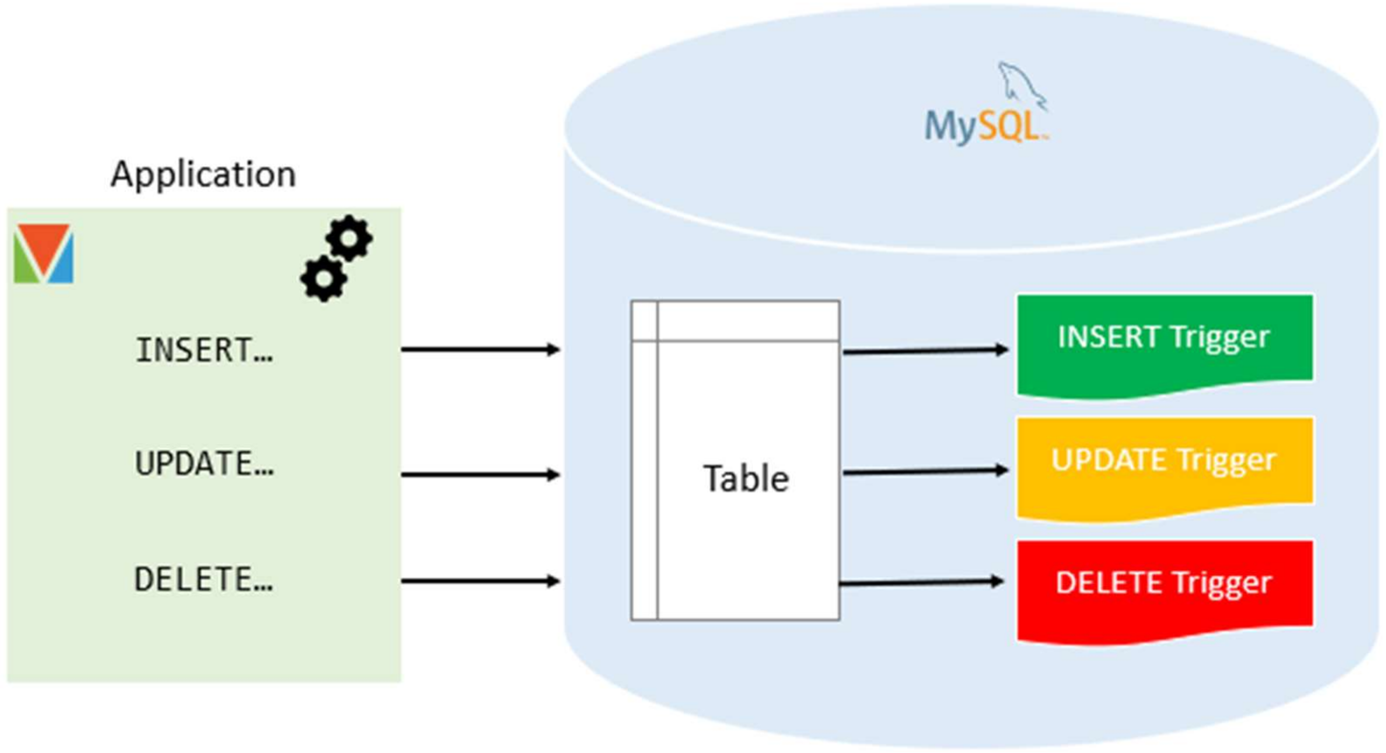
Übersicht ☛ Was lernen wir?

- What is a trigger?
- Syntax and Examples
- Hands-on

Trigger definition

- a trigger is a stored program invoked automatically in response to an event
  - insert,
  - update, or
  - delete that occurs in the associated table.
- For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

Trigger


MySQL
Application
INSERT...
UPDATE...
DELETE...
Table
INSERT Trigger
UPDATE Trigger
DELETE Trigger

SQL standard

- row-level trigger
  - activated for each row that is inserted, updated, or deleted.
- statement-level trigger
  - executed once for each transaction regardless of how many rows are inserted, updated, or deleted.
- MySQL supports only row-level triggers!

Triggers Pros

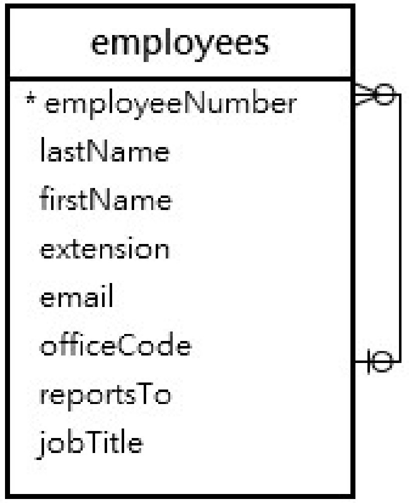
- provide another way to check the integrity of data.
- handle errors from the database layer.
- give an alternative way to run scheduled tasks.
- are invoked automatically before or after a change is made to the data in a table.
- can be useful for auditing the data changes in tables.

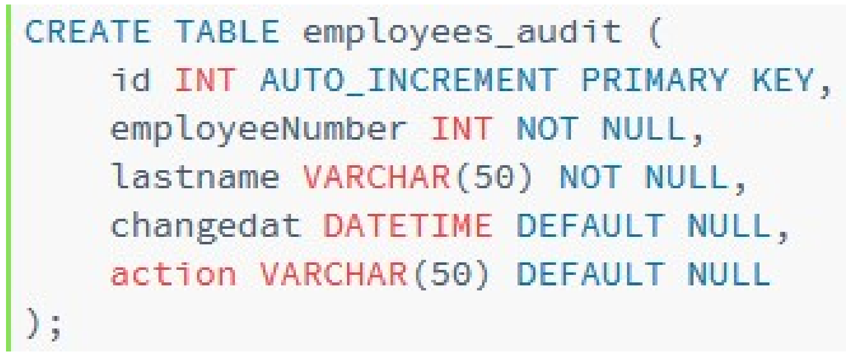
Triggers Cons

- execute automatically in the database, which may not invisible to the client applications.
- may increase the overhead of the MySQL Server.

Create a trigger

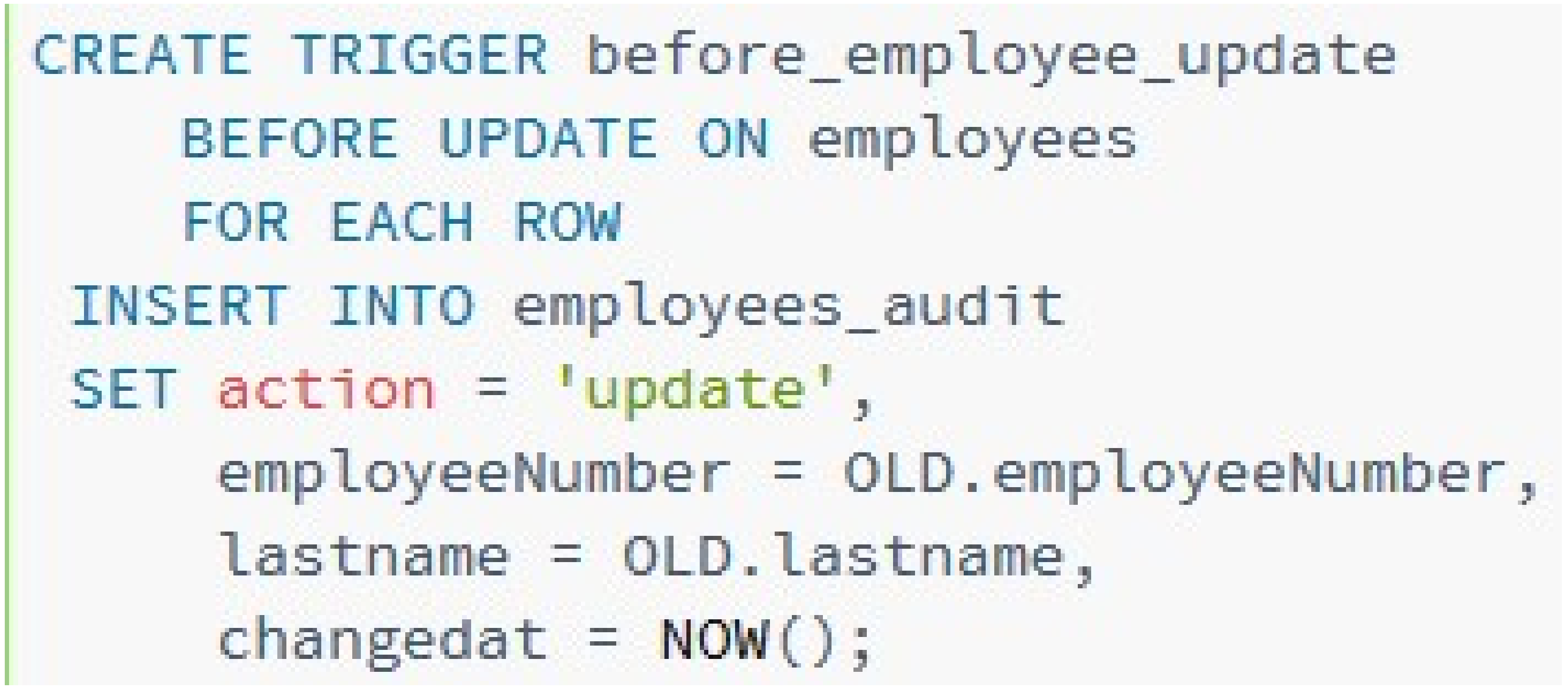
- To distinguish between the value of the columns BEFORE and AFTER the DML has fired -> use NEW or OLD
- Example:





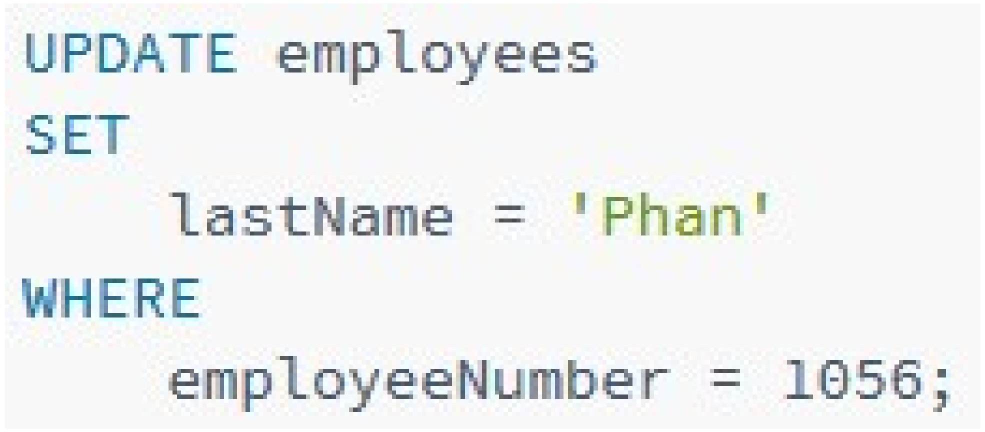
```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

Trigger invoked before a change is made

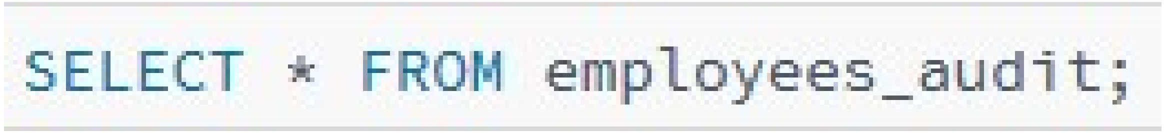


```
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
 INSERT INTO employees_audit
 SET action = 'update',
     employeeNumber = OLD.employeeNumber,
     lastname = OLD.lastname,
     changedat = NOW();
```

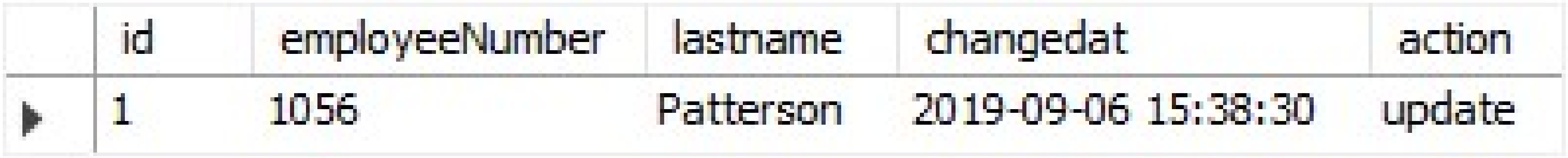
Fire the trigger



```sql
UPDATE employees
SET
    lastName = 'Phan'
WHERE
    employeeNumber = 1056;
```



```sql
SELECT * FROM employees_audit;
```



	id	employeeNumber	lastname	changedat	action
▶	1	1056	Patterson	2019-09-06 15:38:30	update

Before Insert Trigger Example

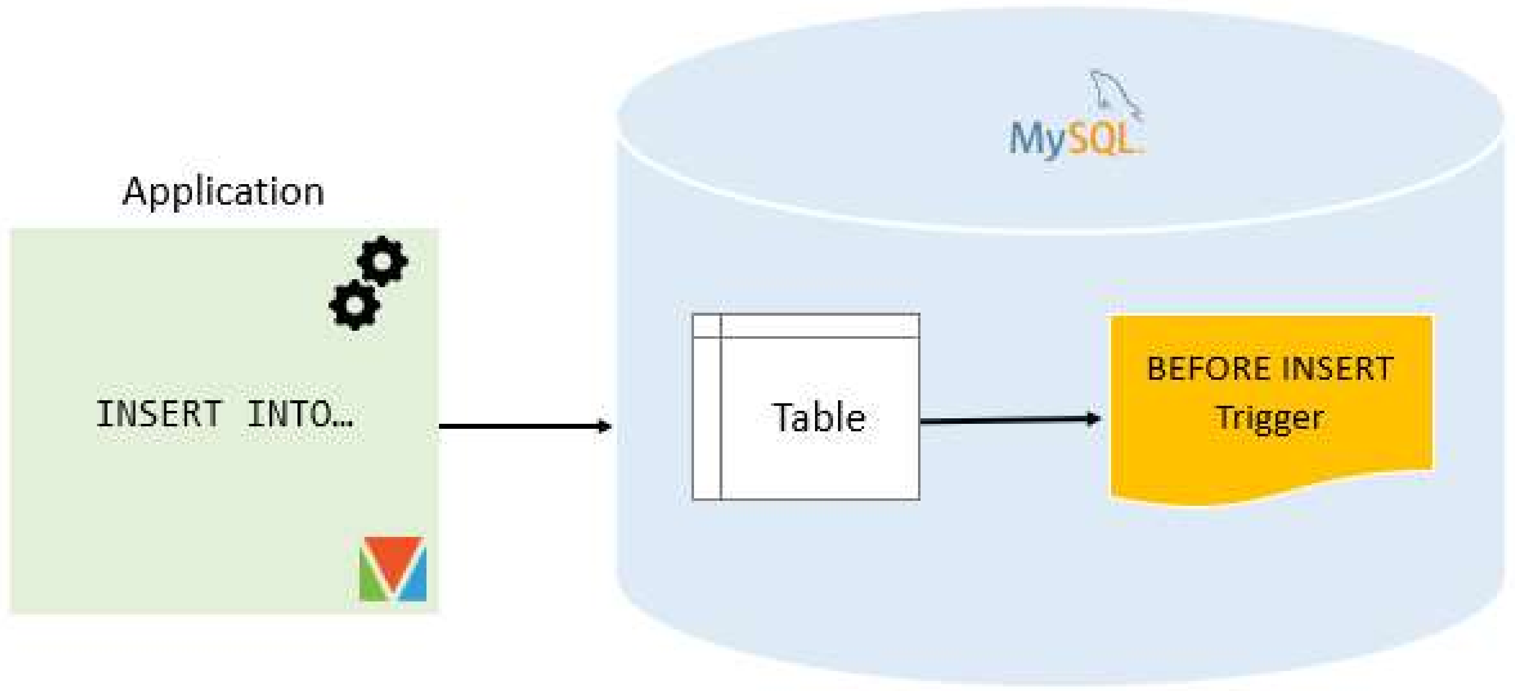
› First 2 tables are created

```
CREATE TABLE WorkCenters (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    capacity INT NOT NULL
);
```

```
CREATE TABLE WorkCenterStats(
    totalCapacity INT NOT NULL
);
```

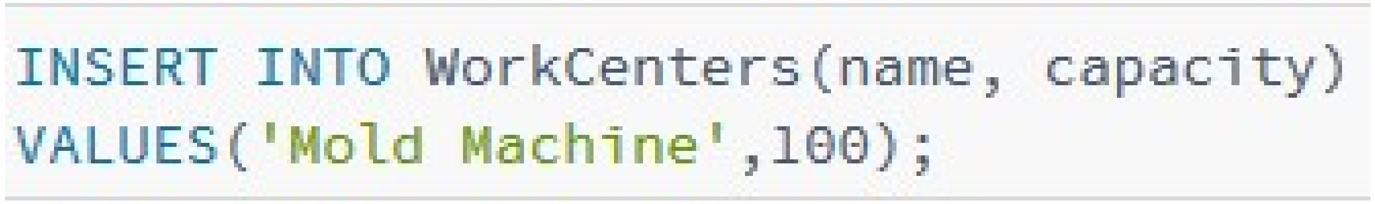
Before Insert Trigger

- Note that in a BEFORE INSERT trigger, you can access and change the NEW values.

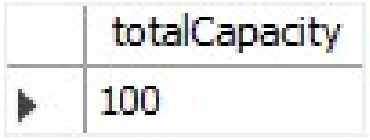


```
DELIMITER $$

CREATE TRIGGER before_workcenters_insert
BEFORE INSERT
ON WorkCenters FOR EACH ROW
BEGIN
    DECLARE rowcount INT;

    SELECT COUNT(*)
    INTO rowcount
    FROM WorkCenterStats;

    IF rowcount > 0 THEN
        UPDATE WorkCenterStats
        SET totalCapacity = totalCapacity + new.capacity;
    ELSE
        INSERT INTO WorkCenterStats(totalCapacity)
        VALUES(new.capacity);
    END IF;

END $$

DELIMITER ;
```

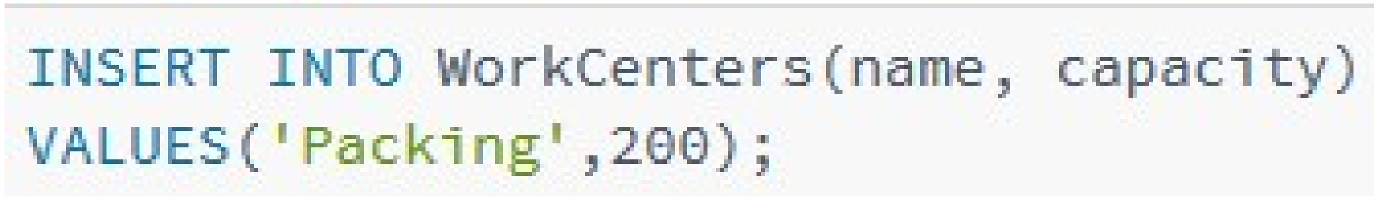
Test the trigger



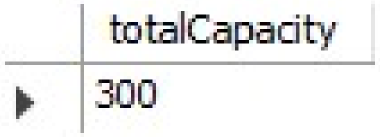
```sql
INSERT INTO WorkCenters(name, capacity)
VALUES('Mold Machine',100);
```



```
  totalCapacity
▶ 100
```



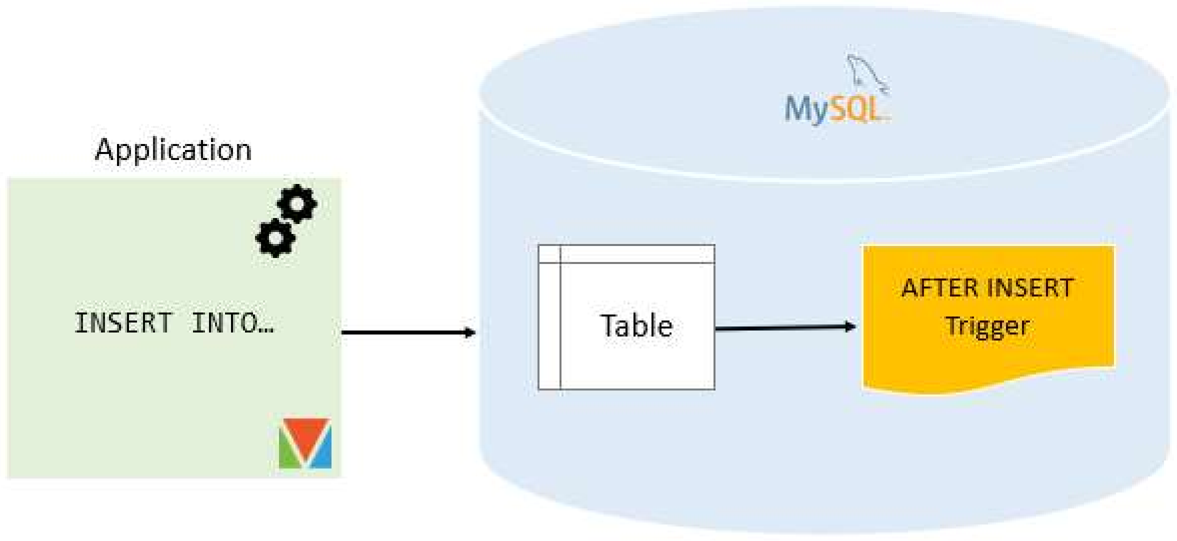
```sql
INSERT INTO WorkCenters(name, capacity)
VALUES('Packing',200);
```



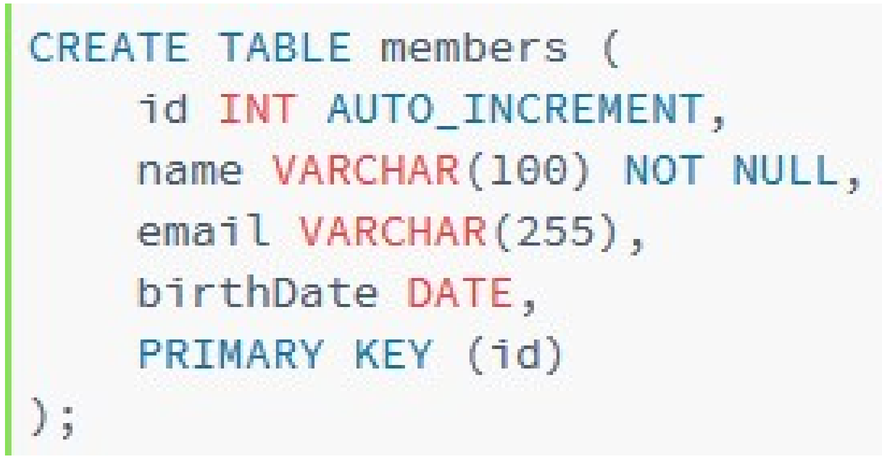
```
  totalCapacity
▶ 300
```

AFTER INSERT Trigger

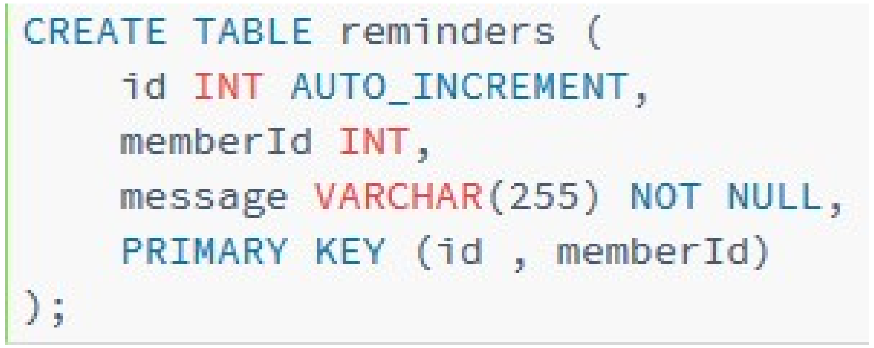
› In an AFTER INSERT trigger, you can access the NEW values but you cannot change them. Also, you cannot access the OLD values because there is no OLD on INSERT triggers.



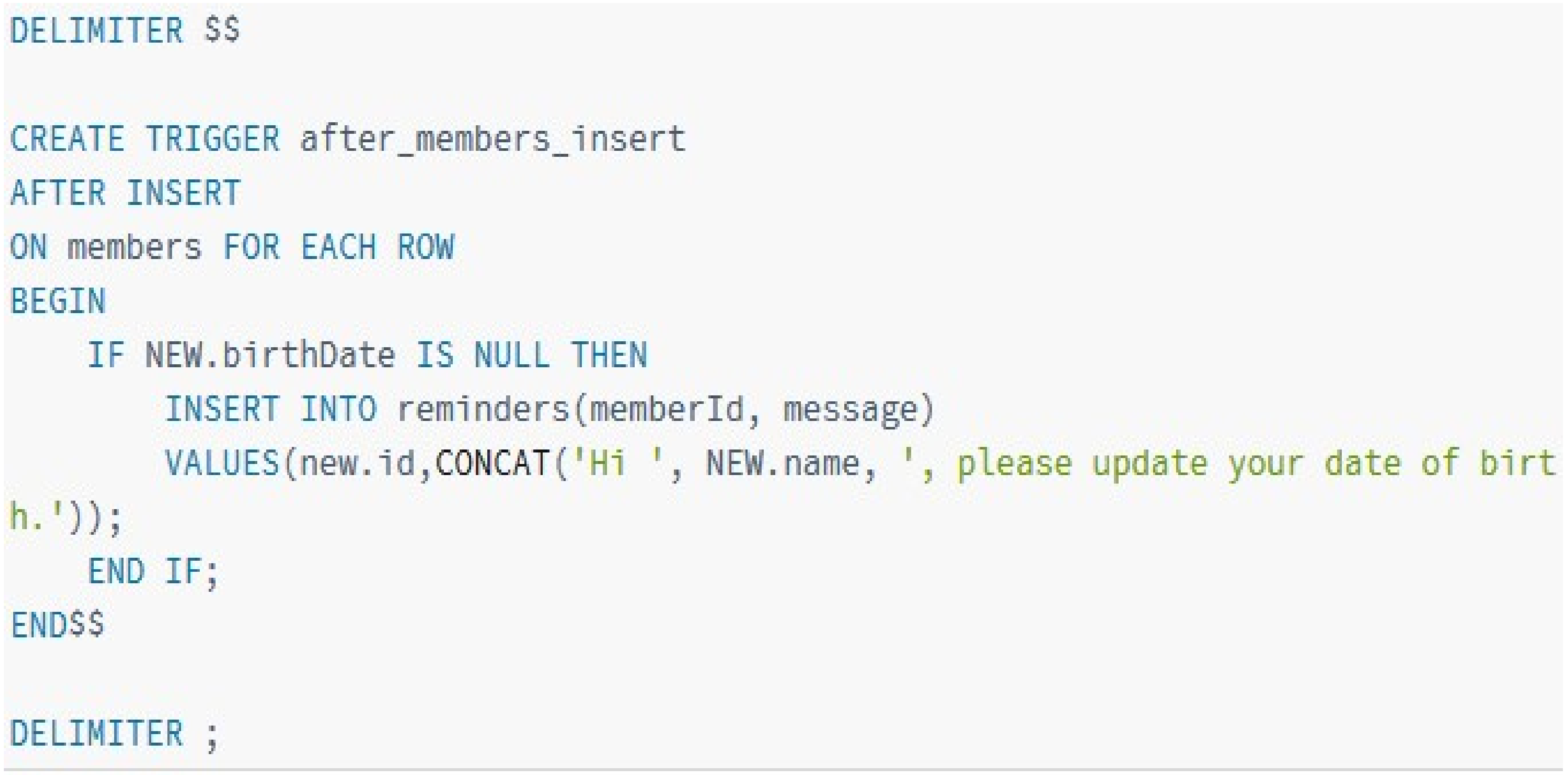
After Insert Example



```sql
CREATE TABLE members (
    id INT AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255),
    birthDate DATE,
    PRIMARY KEY (id)
);
```



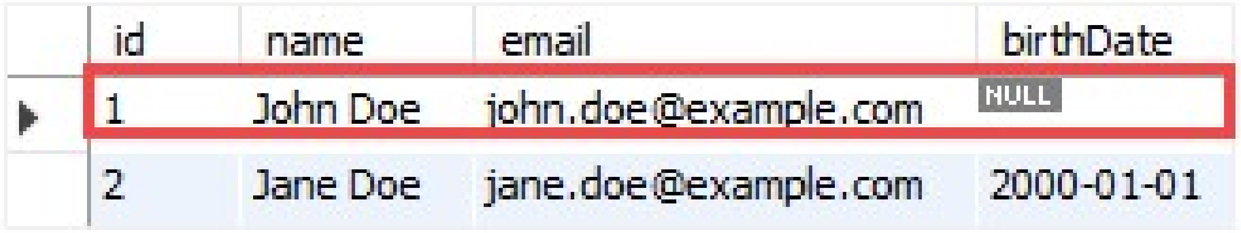
```sql
CREATE TABLE reminders (
    id INT AUTO_INCREMENT,
    memberId INT,
    message VARCHAR(255) NOT NULL,
    PRIMARY KEY (id , memberId)
);
```



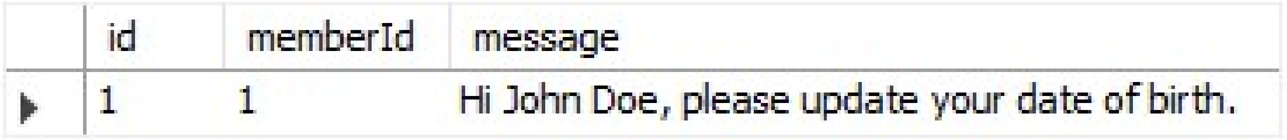
```
DELIMITER $$

CREATE TRIGGER after_members_insert
AFTER INSERT
ON members FOR EACH ROW
BEGIN
    IF NEW.birthDate IS NULL THEN
        INSERT INTO reminders(memberId, message)
        VALUES(new.id,CONCAT('Hi ', NEW.name, ', please update your date of birt
h.'));
    END IF;
END$$

DELIMITER ;
```

Trigger execution

```sql
INSERT INTO members(name, email, birthDate)
VALUES
    ('John Doe', 'john.doe@example.com', NULL),
    ('Jane Doe', 'jane.doe@example.com','2000-01-01');
```

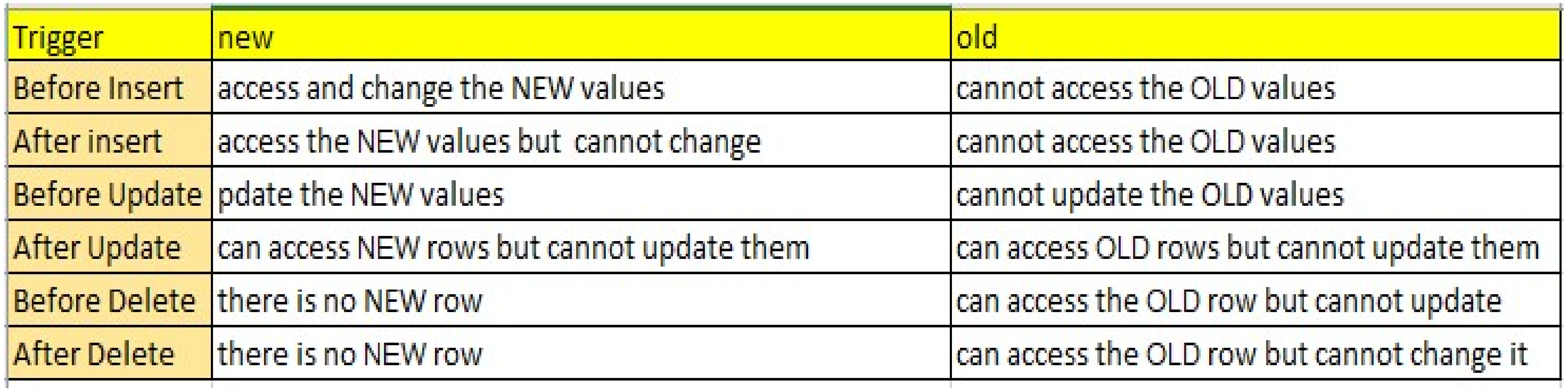


id	name	email	birthDate
1	John Doe	john.doe@example.com	NULL
2	Jane Doe	jane.doe@example.com	2000-01-01



id	memberId	message
1	1	Hi John Doe, please update your date of birth.

Overview Triggers



Trigger	new	old
Before Insert	access and change the NEW values	cannot access the OLD values
After insert	access the NEW values but cannot change	cannot access the OLD values
Before Update	pdate the NEW values	cannot update the OLD values
After Update	can access NEW rows but cannot update them	can access OLD rows but cannot update them
Before Delete	there is no NEW row	can access the OLD row but cannot update
After Delete	there is no NEW row	can access the OLD row but cannot change it

Exercises

- Siehe Moodle!

Auf los geht‘s los ;-)

Quellen

- http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx
- https://www.plsqltutorial.com/what-is-plsql/
- https://www.w3schools.com/sql/sql_stored_procedures.asp