# Laboratory 4 trigger

**Development and Editing:** Z. Garofalaki, A. Tsolakidis, P. Andritsos

## Target

Familiarity with the use of triggers.

## Tools

### A. Trigger

A trigger is a routine associated with an array. It will fire when an event occurs on the given table such as INSERT, UPDATE and DELETE
and can be called before or after the event.

Common Reasons for Use

- Calculation of generated values.
- Entering values into other associated tables
- Check restrictions before changing records.
- Record automatically the users and the dates that changes are made to a table.

**1. Trigger creation** is done with the following general command format.

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

**2.** Each call of the trigger concerns a record (row) of the table, which is created, updated or deleted. Each record can have two states in relation to the moment you execute the trigger. We use the global variables OLD and NEW to refer to the initial and final value of the record respectively.

a. OLD.column_name. We refer to the value of the column that the record had before the event.

b. NEW.column_name You were referring to the value of the column that the record will have after the event.

**3.** When do we use OLD and NEW:

a. In a trigger for INSERT

i. There is no OLD record.

ii. NEW points to the record to be inserted, or to the record that it just happened.

b. In a trigger for UPDATE

i. OLD refers to the values of the record before the update.

ii. NEW refers to either the registration values that will apply after the update.

c. In a trigger for DELETE

i. OLD refers to the record to be deleted.

ii. There is no NEW record.

**4.** Here is an example of a trigger that takes care of entering the names of the departments with capital letters in the dept table of the database. The dept_insert_update trigger is stored and is executed before the department table elements are inserted and transcribes these elements with capital letters.

| | |
|---|---|
| delimiter // | I defined as delimiter the characters // |
| CREATE TRIGGER dept_insert<br>BEFORE INSERT ON<br>department<br>FOR EACH ROW<br>BEGIN<br>SET NEW.dname =<br>UPPER(NEW.dname);<br>END? | I define the dept_insert object |
| // | The definition is complete and the server creates the trigger |
| delimiter ; | Do I change the delimiter so that the sever waits for the character? so that he understands that every command I give him next has been completed. |

**Explanation**

I. Given the statement **INSERT INTO department VALUES(70, 'Learn');**

II. Due to the fact **"BEFORE INSERT ON department"** that exists in the definition of "wakes up" the trigger **dept_insert .**

III. Because an INSERT statement was given and the trigger is row the pairs of global variables that correspond to the Department table columns have the following values:

a. OLD.deptno < - - NULL, OLD.dname < -- NULL

b. NEW.deptno < - - 70, NEW.dname < -- 'Learn' IV.

Because of the command SET NEW.dname = UPPER(NEW.dname); the trigger transcribes to head- the value of the variable NEW.dname.

V. The trigger is terminated and the statement is executed:

**a. INSERT INTO department VALUES(70, 'LEARN');**

**5.** Deletion **of the trigger** named dept_insert is done with the command:

```
DROP TRIGGER dept_insert;
```

# Activities

Carry out the following activities. The command or commands required for the implementation of each step, as well as the result of its execution should be included in a deliverable file in text or screenshot format. The file or files with your answers should be compressed into a **xx_ÿÿÿÿÿ_EPONYMO.zip,** where: (a) xx is the number of the section you belong to (e.g. for group [02] MONDAY 12:00-13 :00, **xx = 02)** and (b) YYYYYY your Registration Number. This final file will be submitted to the e-class -> DATABASES II -> Assignments.

1. **Connect** to your system's MySQL using any of the above methods you wish.

2. Check if there is a DB with the name **personnel.**
3. Select the NW personnel to **use.**
4. Create the trigger named dept_update that is woken by the event "BEFORE UPDATE ON".

```
delimiter //

CREATE TRIGGER dept_update

BEFORE UPDATE ON department

FOR EACH ROW

BEGIN

SET NEW.dname = UPPER(NEW.dname);

END?

//

delimiter ;
```

**Essay:**

/* testing */

UPDATE department SET dname = 'Operations' WHERE deptno=70;

\

5. In the Department table we will add the column no_of_employees (number of employees all that the department has).

```
ALTER TABLE department ADD (no_of_employees INT);
```

**6.** In the Department table we will update the column no_of_employees (number of employees all that the department has).

```
UPDATE department

SET no_of_employees =

                                    (SELECT COUNT(*)

                                     FROM employee

                                     WHERE employee.deptno = department.deptno);
```

**7.** In the Trigger table that is awakened by the AFTER INSERT ON event

```
delimiter //

CREATE TRIGGER emp_insert

AFTER INSERT ON employee

FOR EACH ROW

BEGIN

UPDATE department

SET no_of_employees = IFNULL(no_of_employees, 0) + 1

WHERE deptno= NEW.deptno;

END?

//

delimiter ;
```

**8.** Testing

```
INSERT INTO employee VALUES(7985, 'CLARKE', 10);

SELECT * FROM employee;

SELECT * FROM department;
```

**9.** Explanation

**1.** The trigger "wakes up" due to the condition AFTER INSERT ON em-
ployee
**2.** OLD.empno<--NULL, OLD.ename<--NULL, OLD.deptno<--NULL **3.**
NEW.empno<-7985, NEW.ename<-'NAVATHE', NEW.deptno<-10

**So the statement**

```
UPDATE department

SET no_of_employees = NVL(no_of_employees, 0) + 1;

WHERE deptno= :NEW.deptno;
```

**It is equivalent to the statement**

```
UPDATE department

SET no_of_employees = NVL(no_of_employees, 0) + 1;

WHERE deptno= 10;
```

**10.** Create, test and explain the trigger that is awakened by the AFTER event
DELETE ON

**11.** Create, test and explain the trigger that is awakened by the AFTER event
UPDATE ON

**12.** Presentation of information about Triggers

```
/* see triggers */ DESCRIBE
Information_schema.TRIGGERS;

SELECT TRIGGER_NAME, EVENT_MANIPULATION, TRIGGER_SCHEMA FROM
INFORMATION_SCHEMA.TRIGGERS WHERE
TRIGGER_SCHEMA = 'my_first_triggers_db'
ORDER BY TRIGGER_NAME;
```

**13.** Deleting Triggers DROP

```
TRIGGER dept_insert;

DROP TRIGGER dept_update;

.....
```

## NW personnel

The tables contained in the personnel database should have the following structure and contents:

| Columns | Data type |
|---|---|
| **DEPT.DEPTNO, EMP.DEPTNO** | numeric(2) |
| **DNAME, JOB_DESCR** | varchar(24) |
| **LOC** | char(23) |
| **JOBCODE, JOBNO** | numeric(3) |
| **SAL, COMM** | numeric(10,2) |
| **EMPNO** | numeric(4) |
| **PROJECT.P_ID** | int |
| **PROJECT.P_NAME** | varchar(255) |

**Table 1.** Data types of tables EMP, JOB, DEPT

**EMP**

| EMPNO | NAME | JOBNO | DEPTNO | COMM |
|---|---|---|---|---|
| 10 | CODD | 100 | 50 | |
| 20 | NAVATHE | 200 | 50 | 450 |
| 30 | ELMASRI | 300 | 60 | |
| 40 | DATE | 100 | 50 | |

**JOB**

| JOBCODE | JOB_DESCR | SAL |
|---|---|---|
| 100 | SALESMAN | 2000 |
| 200 | ANALYST | 2000 |
| 300 | DBA | 3000 |

**DEPT**

| DEPTNO | DNAME | LOC |
|---|---|---|
| 50 | SALES | ATHENS |
| 60 | ACCOUNTING | ATHENS |
| 70 | PAYROL | VOLOS |

**Figure 1.** EMP, JOB, DEPT table data