Exercise: Triggers

a) Test the usage of triggers in MySQL with the following scenario:

Add a column to the table student with the name „gradeAverage". The meaning of this value is the average of all grades that this student has achieved, and which are registered in the table Takes.

Then define the necessary triggers to keep this derived attribute consistent with the data in the table Takes, even when new grades are entered, grades are deleted, or grades are updated.

Hint:
After creating the new column „gradeAverage", the values of this columns are all NULL. If you want to update them to the averages that follow from the existing tuples in the Takes table before executing the trigger for the first time, you can do it this way:

```
UPDATE Student s
SET gradeAverage =
(SELECT AVG(grade)FROM Takes WHERE s.matNr = Takes.matNr);
```

b) Just for fun:

Is it possible in MySQL to implement a trigger that removes a data record that has just been inserted into a given table? Give this trigger the name „virus".

c) Set up an example where the firing of a trigger causes an event that fires a second trigger. Test it in MySQL. What would happen if there is a cycle, i.e. the firing of the first trigger causes a second trigger to fire which causes the first trigger to fire again, i.e. an endless loop?

Solution Example

a)

```
ALTER TABLE Student
ADD gpa double(3,1);

DROP TRIGGER IF EXISTS gpa_insert;
DROP TRIGGER IF EXISTS gpa_update;
DROP TRIGGER IF EXISTS gpa_delete;

CREATE TRIGGER gpa_insert
   AFTER INSERT ON Takes
   FOR EACH ROW
   UPDATE Student
   SET gpa =
   (SELECT AVG(grade) FROM Takes WHERE Takes.matNr = NEW.matNr)
   WHERE Student.matNr = NEW.matNr;

CREATE TRIGGER gpa_update
   AFTER UPDATE ON Takes
   FOR EACH ROW
   UPDATE Student
   SET gpa =
   (SELECT AVG(grade) FROM Takes WHERE Takes.matNr = NEW.matNr)
   WHERE Student.matNr = NEW.matNr;

CREATE TRIGGER gpa_delete
   AFTER DELETE ON Takes
   FOR EACH ROW
   UPDATE Student
   SET gpa =
   (SELECT AVG(grade) FROM Takes WHERE Takes.matNr = OLD.matNr)
   WHERE Student.matNr = OLD.matNr;
```

b)
The trigger could be defined like this:

```
CREATE TRIGGER virus
   AFTER INSERT ON Student
   FOR EACH ROW
   DELETE FROM Student where Student.matNr = NEW.matNr;
```

MySQL accepts it, but it delivers a runtime error when executed because the trigger tries to update the table that is being updated by the statement invoking the trigger.

c) Testing an infinite loop:

Take the example from the classnotes:

```
CREATE TABLE TACandidates
(matNr INT(11),
classNr VARCHAR(9),
pName VARCHAR(15),
PRIMARY KEY(matNr, classNr));

CREATE TRIGGER findTACandidates
AFTER INSERT ON Takes
FOR EACH ROW
INSERT INTO TACandidates
(SELECT NEW.matNr, NEW.classNr, Class.pName
FROM Class
WHERE Class.classNr = NEW.classNr
AND NEW.grade <= 3);

DELIMITER //

CREATE TRIGGER looping
AFTER INSERT ON TACandidates
FOR EACH ROW
BEGIN
  INSERT INTO Student values (NEW.matNr +1, 'Koch');
  INSERT INTO Takes values (NEW.matNr + 1, 'DTB-SS93', 1.0);
END//

DELIMITER ;

INSERT INTO Student values (6000, 'Koch');
INSERT INTO Takes VALUES (6000, 'DTB-SS93', 1.0);
```

This delivers the error message:

```
mysql> INSERT INTO Takes VALUES (6000, 'DTB-SS93', 1.0) //
ERROR 1442 (HY000): Can't update table 'Takes' in stored
function/trigger because it is already used by statement which
invoked this stored function/trigger.
```

Hence, an infinite loop can be created, but it cannot be executed because it produces a runtime error.