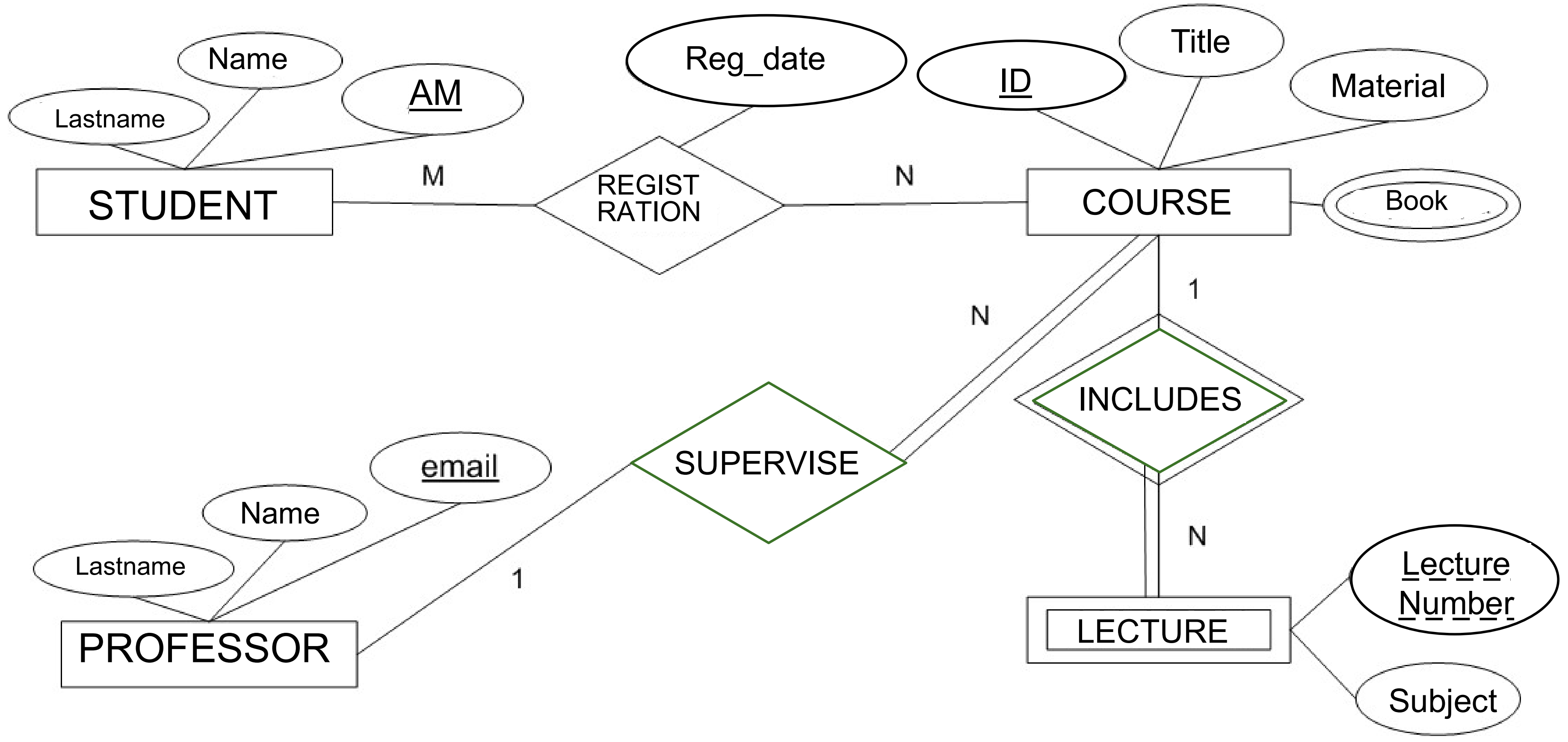
Database Systems Lab

Triggers

ER-diagram example



AM is student’s registration number

Relational-model example

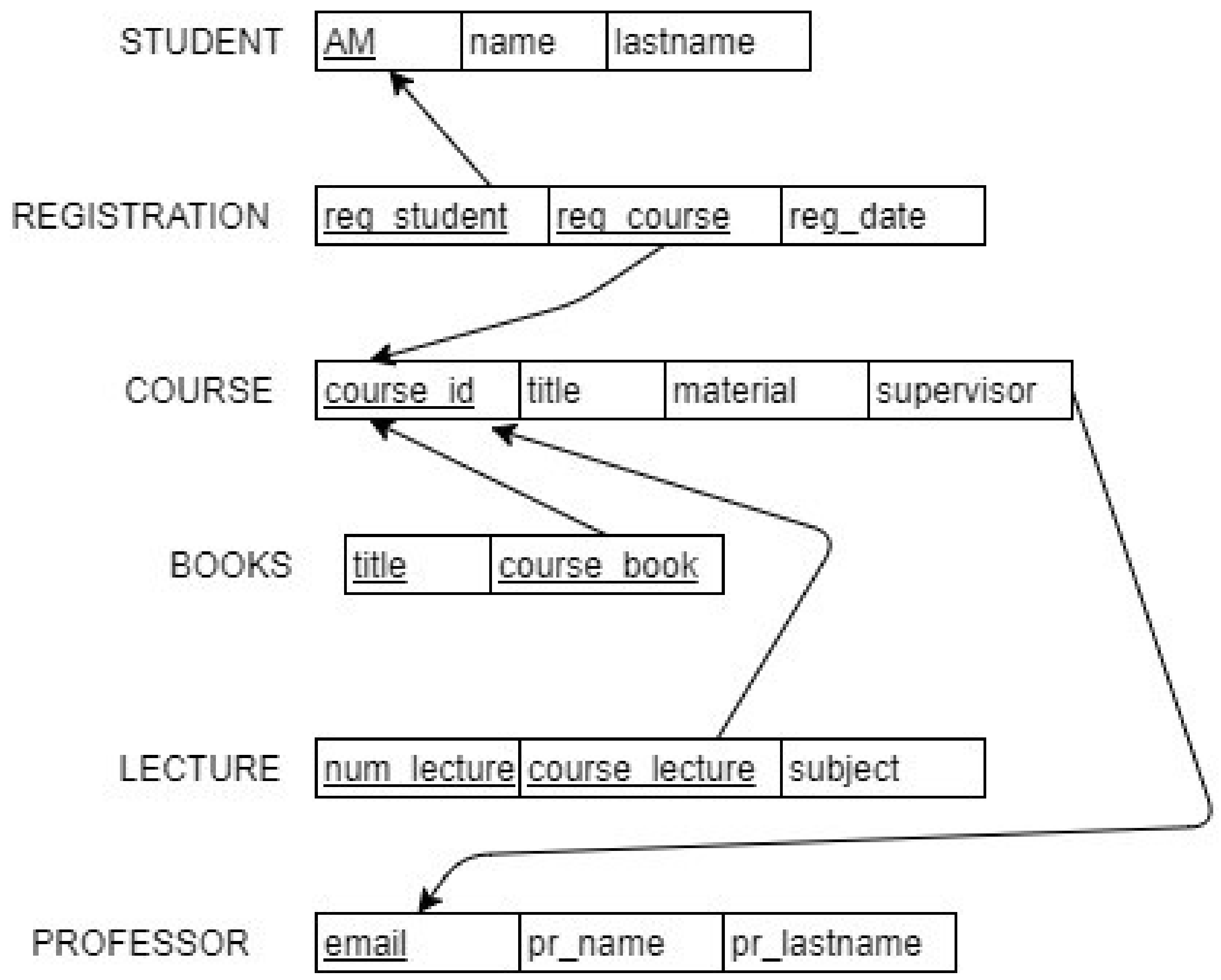


Table-creation statements

CREATE TABLE student(

name **VARCHAR**(25) **DEFAULT** 'unknown' **NOT NULL**,

lastname **VARCHAR**(25) **DEFAULT** 'unknown' **NOT NULL**,

AM **INT**(5) **NOT NULL AUTO_INCREMENT**,

PRIMARY KEY(AM)

);

CREATE TABLE professor(

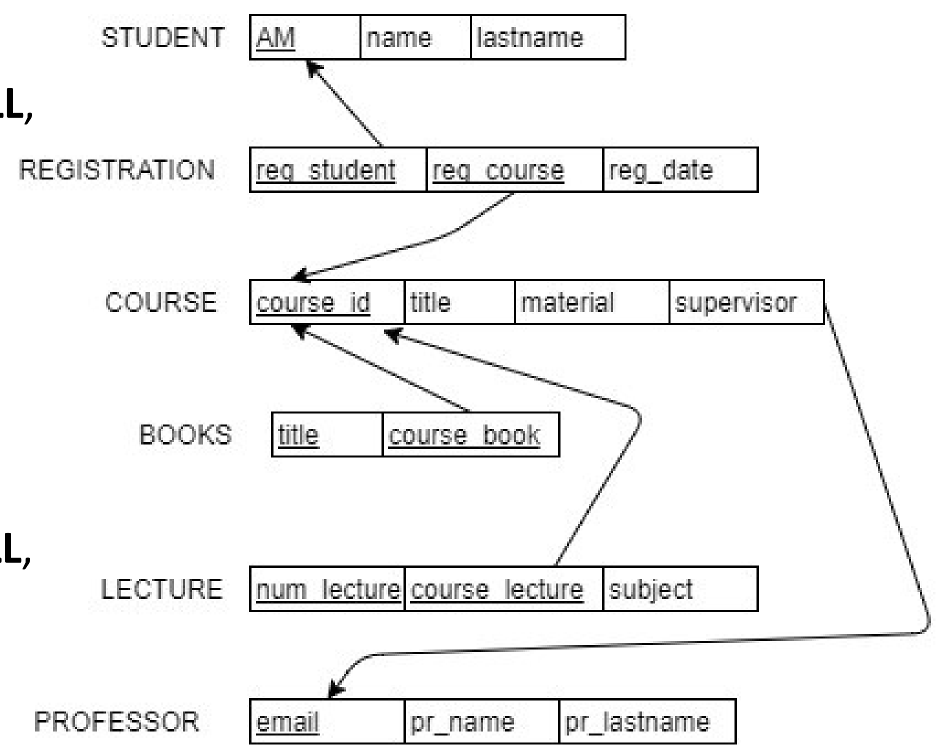
pr_name **VARCHAR**(25) **DEFAULT** 'unknown' **NOT NULL**,

pr_lastname **VARCHAR**(25) **DEFAULT** 'unknown' **NOT NULL**,

email **VARCHAR**(255) **NOT NULL**,

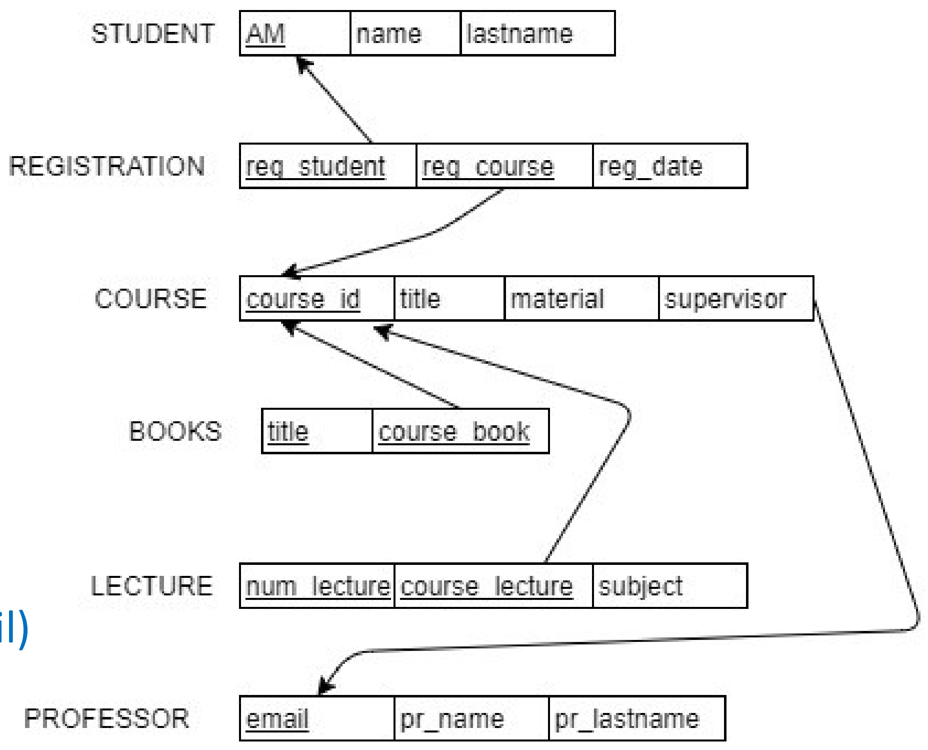
PRIMARY KEY(email)

);

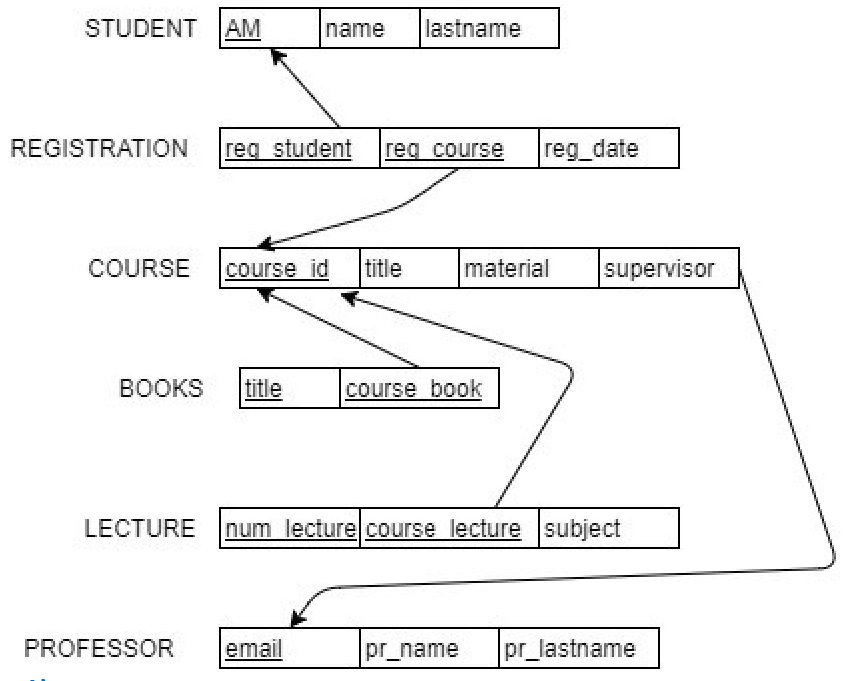


```
CREATE TABLE course (
    title VARCHAR(255) DEFAULT 'unknown' NOT NULL,
    material  TEXT,
    course_id  INT(4) NOT NULL  AUTO_INCREMENT,
    supervisor VARCHAR(255) NOT  NULL,
    PRIMARY KEY(course_id),
    UNIQUE(title),
    CONSTRAINT SUPERVISED
    FOREIGN KEY (supervisor) REFERENCES professor(email)
    ON DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE books (
    title VARCHAR(128) DEFAULT 'Title' NOT NULL,
    course_book INT(4) NOT NULL,
    PRIMARY KEY(title,course_book),
    CONSTRAINT CRSBOOK
    FOREIGN KEY (course_book) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```



```
CREATE TABLE lecture (
    subject VARCHAR(128),
    num_lecture INT(2) NOT NULL,
    course_lecture INT(4) NOT NULL,
    PRIMARY KEY(num_lecture,course_lecture),
    CONSTRAINT CRSLECTURE
    FOREIGN KEY (course_lecture) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```



```
CREATE TABLE registration (
    reg_date DATE NOT NULL,
    reg_student INT(5) NOT NULL,
    reg_course INT(4) NOT NULL,
    PRIMARY KEY(reg_student,reg_course),
    CONSTRAINT CRSREGISTRATION
    FOREIGN KEY (reg_course) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT STDNTREGISTRATION
    FOREIGN KEY (reg_student) REFERENCES student(AM)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

Triggers

- A trigger is a special stored procedure.
- It is invoked automatically in response to a specific event that occurs in an associated table.
- A trigger is used for:
  - Validating data before they are inserted in a table.
  - Enforcing rules regarding business logic.
  - Calculating values for derived fields.
  - Keeping access and table-modification logs.
- Triggers slow down the access to associated tables; therefore, their use adds to database overhead.
  - Triggers are supported in MySQL from version 5.0.2.
  - Before version 5.1.6 the SUPER privilege was needed for declaring triggers.

Managing Triggers

- Trigger creation statement

```
CREATE TRIGGER <trigger name>
```

- Trigger deletion statement

```
DROP TRIGGER <trigger name>
```

- Display trigger creation code

```
SHOW CREATE TRIGGER <trigger name>
```

- List triggers in a table or database

```
SHOW TRIGGERS
```

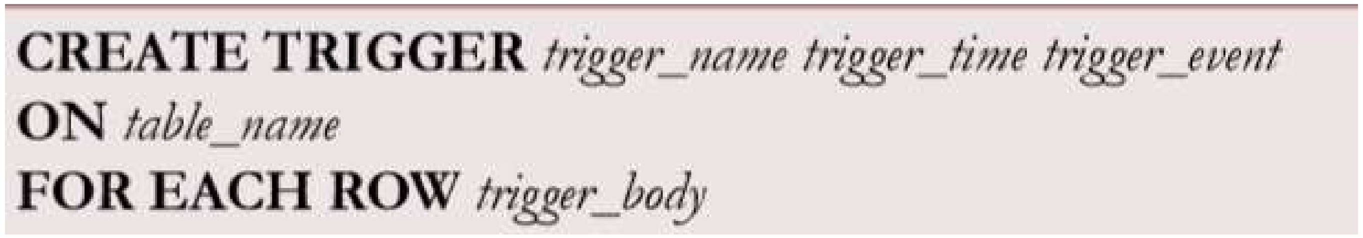
- Trigger call

  A trigger cannot be directly called. It is *automatically invoked* when a specific data-modification event occurs on the associated table.

Creating a Trigger

- When creating a trigger, the following are specified
  - The **table** which the trigger is associated with:
    - This can be any table of the database the trigger is created in.
  - The **event** that invokes the trigger:
    - INSERT
    - UPDATE
    - DELETE
  - The trigger action time:
    - BEFORE the event.
    - AFTER the event is completed.
  - The **operation** of the trigger:
    - In the body of the trigger, we specify the statements to be executed when the trigger is invoked.

Trigger Creation Statement Syntax



CREATE TRIGGER *trigger_name trigger_time trigger_event*
ON *table_name*
FOR EACH ROW *trigger_body*

- trigger_name Trigger’s name. All triggers must have unique names within a schema
- trigger_time The trigger action time. It can be BEFORE or AFTER.
- trigger_event The type of modification (event) that activates the trigger
- ON table_name The name of the table the triggers is associated with
- FOR EACH ROW Defines that the code in the trigger’s body will be executed for each row involved in the activating event
- trigger_body The statement(s) to execute when the trigger activates

Block Statement

- The body of the trigger can include more than one statements
  - To execute multiple statements, use the BEGIN END compound statement

`BEGIN …`*`block statement`*`… END`

- Redefine the default delimiter to distinguish the (end of the) statements in the body of the trigger from the (end of the) `CREATE TRIGGER` statement.
  - To redefine the default delimiter, use the `DELIMITER` command.

MySQL User-Defined Variables

- Users can create their own variables in MySQL.
  - User-defined variables are session-specific.
  - To separate from system variables, user-defined variables begin with @
  - To assign a value to a user-defined variable, use the SET command

```
mysql> SET @x=4;
mysql> SET @y=7;
mysql> SET @z=@x-@y;
```

  - The SELECT statement can be used for printing the value of @z

```
mysql>SELECT @z;
+-------+
| @z    |
+-------+
|    -3 |
+-------+
1 row in set (0.00 sec)
```

Course table creation statement

```
CREATE TABLE course (
    title VARCHAR(255) DEFAULT 'unknown' NOT NULL,
    material  TEXT,
    course_id  INT(4) NOT NULL  AUTO_INCREMENT,
    supervisor VARCHAR(255) NOT  NULL,
    PRIMARY KEY(course_id),
    UNIQUE(title),
    CONSTRAINT SUPERVISED
    FOREIGN KEY (supervisor) REFERENCES professor(email)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

course	title	material	course_id	supervisor
	Database Systems	Introduction to relational databases	2	pap@ceid.upatras.gr
	Database Systems II	Advanced Database Systems	3	alex@ceid.upatras.gr

Trigger Creation – Example

Name keep_count

Operation When inserting a new *course*, the value of a variable that holds the number of courses increases by 1

Why AFTER?

```
mysql>CREATE TRIGGER keep_count
->AFTER INSERT ON course
->FOR EACH ROW
->SET @courseCount=@courseCount+1;
```

```
mysql>SET @courseCount=(SELECT COUNT(*) FROM course);
mysql>SELECT @courseCount;
+--------------+
| @courseCount |
+--------------+
|            2 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql>INSERT INTO course (title,course_id,supervisor)
->VALUES
->('t1',NULL,'alex@upatras.gr'),
->('t2',NULL,'alex@upatras.gr');
```

```
mysql>SELECT @courseCount;
+--------------+
| @courseCount |
+--------------+
|            4 |
+--------------+
1 row in set (0.00 sec)
```

Trigger Events

- Event is the type of operation that activates the trigger.
- Types of Events:
  - INSERT activates the trigger when a row is inserted in the associated table. The INSERT, LOAD DATA and REPLACE invoke an INSERT Event.
  - UPDATE activates the trigger when a row is modified in the associated table using the UPDATE statement.
  - DELETE activates the trigger when a row is deleted in the associated table. The DELETE and REPLACE statements invoke an UPDATE Event.
    - The DROP TABLE and TRUNCATE *do not activate DELETE-Event triggers*
- Deleting or updating rows because of referential actions caused by foreign key constraints *do not activate* triggers!

Triggers and Associated Tables

- A table can be associated with multiple triggers after MySQL version 5.7
- Before MySQl version 5.7, a table could be associated with only one trigger for a specific trigger_event and trigger_time.
- Thus, before version 5.7 a table could be associated with only six triggers at most.
  - BEFORE INSERT
  - AFTER INSERT
  - BEFORE UPDATE
  - AFTER UPDATE
  - BEFORE DELETE
  - AFTER DELETE

```
mysql> CREATE TRIGGER count_students
    -> AFTER INSERT ON student
    -> FOR EACH ROW
    -> SET @stCount=@stCount+1;
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER count_students_alt
    -> AFTER INSERT ON student
    -> FOR EACH ROW
    -> SET @studentsCount=@studentsCount+1;
ERROR 1235 (42000): This version of MySQL
doesn't yet support 'multiple triggers with
the same action time and event for one
table'
```

Statements in the Trigger Body

- The body of a trigger contains the statements to be executed when the trigger is activated
  - To define these statements, the default delimiter must be changed using the DELIMITER command
- The following statements can be used in the body of a trigger
  - Variable declarations
  - Flow-control statements (IF, CASE, WHILE, LOOP, WHILE, REPEAT)
  - SELECT INTO, Handler declarations and Cursor statements
  - Stored-procedures call statements
  - Multiple statements must be included inside a BEGIN...END compound statement
- A trigger cannot
  - Use the SELECT statement for showing data
  - Modify its associated table
- Stored procedures that are called by triggers share the same limitations

Trigger code – Example 1

Name init_book

Function After a new *course* has been inserted, insert a new *book* for this course using the default value for *title*

```
mysql>DELIMITER $
mysql>CREATE TRIGGER init_book
->AFTER INSERT ON course
->FOR EACH ROW
->BEGIN
-> DECLARE cid INT(4);
-> SELECT MAX(course_id) INTO cid FROM course;
-> INSERT INTO books(title, course_book)
-> VALUES (DEFAULT, cid);
->END$
mysql>DELIMITER ;
```

Books table creation statement

```
CREATE TABLE books (
    title VARCHAR(128) DEFAULT 'Title' NOT NULL,
    course_book INT(4) NOT NULL,
    PRIMARY KEY(title,course_book),
    CONSTRAINT CRSBOOK
    FOREIGN KEY (course_book) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

books	title	course_book
	Databases 1	2
	Databases 1 2nd volume	2
	Databases 2	3

Trigger code – Example 1

Check that the trigger init_book works correctly:

```
mysql>INSERT INTO
course(title,course_id,supervisor)
->VALUES
->(‘t3’,NULL,’alex@upatras.gr’),
->(‘t4’,NULL,’alex@upatras.gr’);
```

```
mysql>SELECT books.title as Book,course.title as Course
->FROM books
->INNER JOIN course ON course_book=course_id;
+-----------------------+---------------------+
| Book                  | Course              |
+-----------------------+---------------------+
| Databases 1           | Databases           |
| Databases 1 2nd volume | Databases          |
| Databases 2           | Databases II        |
| Title                 | t3                  |
| Title                 | t4                  |
+-----------------------+---------------------+
5 rows in set (0.00 sec)
```

Trigger code - Limitations

- A trigger cannot use a SELECT statement for showing data

```
mysql> DELIMITER $
mysql> CREATE TRIGGER init_book
    -> AFTER INSERT ON course
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE cid INT(4);
    -> SELECT MAX(course_id) INTO cid FROM course;
    -> INSERT INTO books(title, course_book)
    -> VALUES (DEFAULT,cid);
    -> SELECT title FROM books WHERE course_book=cid;
    -> END$
ERROR 1415 (0A000): Not allowed to return a result set
from a trigger
```

A trigger *cannot modify its associated table* (nor call a stored procedure that modifies this table)

Name overflow_registrations –

```
mysql> DELIMITER $
mysql>CREATE TRIGGER overflow_registrations
->BEFORE INSERT ON registration
->FOR EACH ROW
->BEGIN
-> DECLARE regnum INT;
-> DECLARE oldDate DATE;
-> SELECT COUNT(*) INTO regnum
-> FROM registration;
-> IF regnum>=5 THEN
-> SELECT MIN(reg_date) INTO oldDate
-> FROM registration;
-> DELETE FROM registration
-> WHERE reg_date=oldDate;
-> END IF;
->END$
mysql>DELIMITER ;
mysql> INSERT INTO registration(reg_date, reg_student, reg_course)
-> VALUES ('2012-12-03',2193,2);

ERROR 1442 (HY000): Can't update table 'registration' in stored
function/trigger because it is already used by statement which
invoked this stored function/trigger.
```

```
CREATE TABLE registration (
                reg_dateDATE NOT NULL,
                reg_studentINT(5) NOT NULL,
                reg_courseINT(4) NOT NULL,
                PRIMARY KEY(reg_student,reg_course),
CONSTRAINT CRSREGISTRATION
FOREIGN KEY (reg_course) REFERENCES course(course_id)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT STDNTREGISTRATION
FOREIGN KEY (reg_student) REFERENCES student(AM)
ON DELETE CASCADE ON UPDATE CASCADE);
```

Accessing values of records

- The trigger body can access the values of the record (row) that is affected by the insert, update or delete statement.
- A record has two states with respect to the event that activates the trigger:
  - The state before the event is completed.
  - The state after the event is completed.
- The keywords OLD and NEW are used to access the values of rows before and after the event has finished, respectively:
  - OLD.column_name can access the value of colum_name in the row under modification before the event is completed.
  - NEW.column_name an access the value of colum_name in the row under modification after the event is completed.

OLD and NEW modifiers

- Trigger event: **INSERT**
  - There is not a before state; OLD is not available.
  - NEW modifier can access the value to be inserted before or after the insert event is completed.
- Trigger event: **UPDATE**
  - OLD modifier can access the existing value (the one to be updated) of the record before the update.
  - NEW can access the value that updates the existing value before or after the update.
- Trigger event: **DELETE**
  - OLD can access the existing value (the one to be deleted).
  - There is not an after state, NEW is not available.
- Values accessed with the OLD modifier are read-only
- The NEW modifier can read but also change values.
  - Triggers can change values only in BEFORE time.
  - The value of an AUTO_INCREMENT field is zero BEFORE the event.

Registration table creation statement

```
CREATE TABLE registration (
            reg_date DATE NOT NULL,
            reg_student INT(5) NOT NULL,
            reg_course INT(4) NOT NULL,
            PRIMARY KEY(reg_student,reg_course),
            CONSTRAINT CRSREGISTRATION
FOREIGN KEY (reg_course) REFERENCES
course(course_id)
ON DELETE CASCADE ON UPDATE CASCADE,
            CONSTRAINT STDNTREGISTRATION
FOREIGN KEY (reg_student) REFERENCES student(AM)
ON DELETE CASCADE ON UPDATE CASCADE);
```

OLD and NEW in BEFORE INSERT

If the date to be inserted is in the future, then replace it with the current date.

```
mysql> CREATE TRIGGER checkRegDate
    -> BEFORE INSERT ON registration
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE currDate DATE;
    -> SET currDate=CURDATE();
    -> IF NEW.reg_date>currDate THEN SET NEW.reg_date=currDate;
    -> END IF;
    -> END$
Query OK, 0 rows affected (0.08 sec)
mysql>DELIMITER ;

--Trigger testing:
mysql> INSERT INTO registration (reg_date,reg_student,reg_course)
->VALUES (‘2022-12-30',2194,2);
Query OK, 1 row affected (0.02 sec)

mysql> select * from registration;
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2015-09-15 |        2129 |          2 |
| 2015-10-05 |        2129 |          3 |
| 2015-09-11 |        2193 |          2 |
| 2015-09-25 |        2193 |          3 |
| 2022-12-15 |        2194 |          2 |
| 2015-09-30 |        2194 |          3 |
+------------+-------------+------------+
6 rows in set (0.00 sec)
```

OLD and NEW in AFTER INSERT

```
mysql>DELIMITER $
mysql>CREATE TRIGGER returnId
->AFTER INSERT ON student
->FOR EACH ROW
->BEGIN
-> SET @newStudent=NEW.AM;
->END$
Query OK, 0 rows affected (0.08 sec)
mysql>DELIMITER ;
```

After inserting a new student, return to a user-defined variable the value of this student’s AM (= Record number)

```
--trigger testing:
mysql> INSERT INTO student(AM,name,lastname) VALUES(NULL,’George’,’Bale');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM student WHERE lastname='Αναστασίου';
+---------+-----------+------+
| name    | lastname  | am   |
+---------+-----------+------+
| Georgre | Bale      | 2194 |
+---------+-----------+------+
1 row in set (0.00 sec)

mysql> SELECT @newStudent;
+-------------+
| @newStudent |
+-------------+
| 2194        |
+-------------+
1 row in set (0.00 sec)
```

OLD and NEW in UPDATE

```
mysql>DELIMITER $
mysql>CREATE TRIGGER validateEmail
->BEFORE UPDATE ON professor
->FOR EACH ROW
->BEGIN
-> IF NEW.email NOT LIKE '%@%.%' THEN
-> SET NEW.email=OLD.email;
-> END IF;
->END$
Query OK, 0 rows affected (0.08 sec)
mysql>DELIMITER $
```

Before modifying a professor's email, check the format of the new email. If it is valid then save the new value, otherwise keep the old one.

```
--trigger testing:
mysql> update professor set email='nick@ceid' where email='alex@ceid.upatras.gr';
Query OK, 0 rows affected (0.064 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> update professor set email='nick@ceid.upatras.gr' where email='alex@ceid.upatras.gr';
Query OK, 1 row affected (0.061 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Trigger event cancelation

- Triggers are often used for enforcing data validation and data integrity by checking the values to be modified.
- If a check fails in BEFORE time, then there are two options:
  - Correct the value so it no longer violates any data validation restrictions
  - Cancel the event that activated the trigger.
- There are two methods for canceling an event:
  - Raising an error by executing **an invalid operation** inside a MySQL statement (i.e, inserting a NULL value in field defined to accept only NOT NULL values).
  - Using the **SIGNAL** statement to return an error and terminate the execution (available from MySQL version 5.5)

Course table creation statement

```
CREATE TABLE course (
    title VARCHAR(255) DEFAULT 'unknown' NOT NULL,
    material  TEXT,
    course_id  INT(4) NOT NULL  AUTO_INCREMENT,
    supervisor VARCHAR(255) NOT  NULL,
    PRIMARY KEY(course_id),
    UNIQUE(title),
    CONSTRAINT SUPERVISED
    FOREIGN KEY (supervisor) REFERENCES professor(email)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

Canceling events – invalid operation

```
mysql>DELIMITER $
mysql>CREATE TRIGGER validateSupervisorCourses
->BEFORE INSERT ON course
->FOR EACH ROW
->BEGIN
-> DECLARE numOfCourses INT;
-> SELECT COUNT(*) INTO numOfCourses
-> FROM course
-> WHERE course.supervisor=NEW.supervisor;
-> IF numOfCourses>2 THEN
-> SET NEW.supervisor=NULL;
-> END IF;
->END$
Query OK, 0 rows affected (0.08 sec)
mysql>DELIMITER ;
```

Before assigning a new course to a professor, check the number of courses the professor already supervises. If the professor has the maximum number of assignments (2), then raise an error.

```
--trigger testing:
mysql> SELECT COUNT(*) FROM course WHERE supervisor='pap@ceid.upatras.gr';
+----------+
| COUNT(*) |
+----------+
| 1        |
+----------+
1 row in set (0.00 sec)

mysql> INSERT INTO course(title,supervisor) VALUES ('Lesson2','pap@ceid.upatras.gr');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO course(title,supervisor) VALUES ('Lesson3','pap@ceid.upatras.gr');
ERROR 1048 (23000): Column 'supervisor' cannot be null
```

Canceling events using SIGNAL

Before a student registers to a new course, check the registration date. If the difference between registration and current date is more than a year, then raise an error

```
mysql>DELIMITER $
mysql>CREATE TRIGGER validateRegistration
->BEFORE INSERT ON registration
->FOR EACH ROW
->BEGIN
-> DECLARE currDate DATE;
-> DECLARE diff INT;
-> SET currDate=CURDATE();
-> SET diff=ABS(DATEDIFF(currDate,NEW.reg_date));
-> IF diff>=365 THEN
-> SIGNAL SQLSTATE VALUE '45000'
-> SET MESSAGE_TEXT = 'Invalid registration date! Must be within a year.';
-> END IF;
->END$
mysql>DELIMITER ;
```

```
--trigger testing:
mysql> INSERT INTO registration(reg_date,reg_student,reg_course) VALUES ('2030-04-17',2194,2);
ERROR 1644 (45000): Invalid registration date! Must be within a year.

mysql> INSERT INTO registration(reg_date,reg_student,reg_course) VALUES (‘2021-12-22',2194,2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO registration(reg_date,reg_student,reg_course) VALUES (‘2023-02-12',2193,2);
Query OK, 1 row affected (0.12 sec)
```

Canceling events using SIGNAL

- The **SIGNAL** statement specifies:
  - SQLSTATE CODE – the value returned to the caller.
    - MySQL manual defines specific values and rules for this code.
    - Use value '45000’ which indicates an unhandled user-defined exception.
  - MESSAGE_TEXT – the error message returned to the caller
- MySQL Functions used in previous examples:
  - ABS(): return the absolute value.
  - DATEDIFF(): calculates and returns the number of days between two dates (DATE, DATETIME or TIMESTAMP).

Lecture table creation statement

```
CREATE TABLE lecture (
    subject VARCHAR(128),
    num_lecture INT(2) NOT NULL,
    course_lecture INT(4) NOT NULL,
    PRIMARY KEY(num_lecture,course_lecture),
    CONSTRAINT CRSLECTURE
    FOREIGN KEY (course_lecture) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

AUTO_INCREMENT Implementation Example

```
mysql>DELIMITER $
mysql>CREATE TRIGGER autoIncrementLecture
->BEFORE INSERT ON lecture
->FOR EACH ROW
->BEGIN
-> DECLARE maxNum INT(2);
-> SELECT MAX(num_lecture) INTO maxNum
-> FROM lecture
-> WHERE course_lecture=NEW.course_lecture;
-> SET NEW.num_lecture=maxNum+1;
->END$
mysql>DELIMITER ;
```

Calculate automatically the next number (num_lecture) for a new lecture of a course

```
--trigger testing:
mysql> SELECT * FROM lecture WHERE course_lecture=2;
+---------------------+------------+----------------+
| subject             | num_lecture | course_lecture |
+---------------------+------------+----------------+
| Introduction to DB  | 1          | 2              |
| Requirements Analysis| 2         | 2              |
| ER-Relational model | 3          | 2              |
+---------------------+------------+----------------+
3 rows in set (0.00 sec)
```

```
mysql> INSERT INTO lecture(subject, num_lecture, course_lecture)
-> VALUES ('Introduction to MySQL 1',0,2);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM lecture WHERE course_lecture=2;
+--------------------------+-------------+----------------+
| subject                  | num_lecture | course_lecture |
+--------------------------+-------------+----------------+
| Introduction to DB       | 1           | 2              |
| Requirements Analysis    | 2           | 2              |
| ER-Relational model      | 3           | 2              |
| Introduction to MySQL 1  | 4           | 2              |
+--------------------------+-------------+----------------+
4 rows in set (0.00 sec)


mysql> INSERT INTO lecture(subject, num_lecture, course_lecture)
-> VALUES (' Introduction to MySQL 2',0,2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM lecture WHERE course_lecture=2;
+--------------------------+-------------+----------------+
| subject                  | num_lecture | course_lecture |
+--------------------------+-------------+----------------+
| Introduction to DB       | 1           | 2              |
| Requirements Analysis    | 2           | 2              |
| ER-Relational model      | 3           | 2              |
| Introduction to MySQL 1  | 4           | 2              |
| Introduction to MySQL 2  | 5           | 2              |
+--------------------------+-------------+----------------+
5 rows in set (0.00 sec)
```

AUTO_INCREMENT Implementation Example

Question:

- If the result of the SELECT query is empty?
  - How can number 1 be assigned to the first lecture of a course?

```
mysql>DELIMITER $
mysql>CREATE TRIGGER autoIncrementLecture
->BEFORE INSERT ON lecture
->FOR EACH ROW
->BEGIN
-> DECLARE maxNum INT(2);
-> SELECT MAX(num_lecture) INTO maxNum
-> FROM lecture
-> WHERE course_lecture=NEW.course_lecture;
-> SET NEW.num_lecture=maxNum+1;
->END$
mysql>DELIMITER ;
```

AUTO_INCREMENT Implementation Example

Question:

- If the result of the SELECT query is empty?
  - How can number 1 be assigned to the first lecture of a course?

```
mysql>DELIMITER $
mysql>CREATE TRIGGER autoIncrementLecture
->BEFORE INSERT ON lecture
->FOR EACH ROW
->BEGIN
-> DECLARE maxNum INT(2);
-> SELECT MAX(num_lecture) INTO maxNum
-> FROM lecture
-> WHERE course_lecture=NEW.course_lecture;
-> IF maxNUM is NULL THEN
->    SET maxNum=0;
-> END IF;
-> SET NEW.num_lecture=maxNum+1;
->END$
mysql>DELIMITER ;
```