

Student-course-grade example in SQL

Fall 2019, Term 2

In this short tutorial we will create a small database for students, courses and grades (similar to the example we have used a couple of times in class) and show how different integrity constraints work. Instead of just reading the tutorial, try these things out on your own computer. You can use SQLite3¹, a simple RDBMS that does not require a database server. Run

```
sqlite3 scg.db
```

in a terminal (or cmd in Windows). The parameter (`scg.db`) is the filename where the database will be stored (SQLite3 stores the whole database in a single file). For historical reasons, the foreign key constraints are disabled by default. To enable them, enter

```
PRAGMA foreign_keys = ON;
```

every time you start sqlite3.

Let's first create the tables for students, courses and grades:

```
CREATE TABLE student(  
  id int NOT NULL PRIMARY KEY,  
  first_name varchar(255) NOT NULL,  
  last_name varchar(255) NOT NULL  
);
```

```
CREATE TABLE course(  
  id int NOT NULL PRIMARY KEY,  
  code varchar(30) NOT NULL UNIQUE,  
  title varchar(255)  
);
```

```
CREATE TABLE grade(  
  student_id int NOT NULL,  
  course_id int NOT NULL,  
  grade char(1),  
  PRIMARY KEY(student_id, course_id),  
  FOREIGN KEY(student_id) REFERENCES student(id),  
  FOREIGN KEY(course_id) REFERENCES course(id)  
);
```

We have used NOT NULL constraints for all attributes except for the course title and the grade. Note also how the PRIMARY KEY, UNIQUE and FOREIGN KEY constraints are specified.

Now, let's add some students:

```
INSERT INTO student(id, first_name, last_name) VALUES (1, 'Homer', 'Simpson');  
INSERT INTO student(id, first_name, last_name) VALUES (2, 'Marge', 'Bouvier');
```

We can look at the contents of the table *student* by running the following SQL query:

```
SELECT * FROM student;
```

If we now try to insert another tuple into the table using ID 2 (there is already a student with that ID):

```
INSERT INTO student(id, first_name, last_name) VALUES (2, 'Bart', 'Simpson');
```

¹If you use Mac or Linux, or you have installed Anaconda on your computer, you should be able to use SQLite3 without installing any additional packages.

we will get an error. (In SQLite we will get “Error: UNIQUE constraint failed: student.id”).

Let’s now add two courses in our database: 1DL301 Database Design I and 1DL321 for which we do not know the name (recall that we have allowed NULLs for the course title):

```
INSERT INTO course(id, code, title) VALUES (1, '1DL301', 'Database Design I');
INSERT INTO course(id, code) VALUES (2, '1DL321');
```

We could also use

```
INSERT INTO course(id, code, title) VALUES (2, '1DL321', NULL);
```

instead of the second statement.

The title for 1DL321 is “Compiler Design I” so let’s update the course:

```
UPDATE course SET title='Compiler Design I' WHERE id=2;
```

To see the list of all courses, we can run the following query:

```
SELECT * FROM course;
```

If we at this point try to insert a grade (say, U) for Homer Simpson (i.e., the student with ID 1) and the course with ID 3 by running

```
INSERT INTO grade(student_id, course_id, grade) VALUES (1, 3, 'U');
```

we will get an error since we are trying to refer to a non-existing course. (In SQLite: “Error: FOREIGN KEY constraint failed”).

Alright, the grade is actually for the Database Design I course (which has ID 1). Marge took that course too but with the grade of 5:

```
INSERT INTO grade(student_id, course_id, grade) VALUES (1, 1, 'U');
INSERT INTO grade(student_id, course_id, grade) VALUES (2, 1, '5');
```

Homer was not doing great at UU (he has not finished any other course) and returned to Springfield. We want to remove him from our database:

```
DELETE FROM student WHERE id=1;
```

This does not work since there are rows in other tables that are referencing Homer (there is one grade for him in the grade table). (In SQLite we get “Error: FOREIGN KEY constraint failed”). We could remove these first and then remove Homer.

But we can also instruct the database to do that automatically: Let’s drop the grade table (the content will be removed too, but we don’t care about that in this tutorial) and re-create it again with ON DELETE CASCADE options for both foreign keys:

```
DROP TABLE grade;
```

```
CREATE TABLE grade(
  student_id int NOT NULL,
  course_id int NOT NULL,
  grade char(1),
  PRIMARY KEY(student_id, course_id),
  FOREIGN KEY(student_id) REFERENCES student(id) ON DELETE CASCADE,
  FOREIGN KEY(course_id) REFERENCES course(id) ON DELETE CASCADE
);
```

```
INSERT INTO grade(student_id, course_id, grade) VALUES (1, 1, 'U');
INSERT INTO grade(student_id, course_id, grade) VALUES (2, 1, '5');
```

Let’s try to delete Homer from the student table again:

```
DELETE FROM student WHERE id=1;
```

We did not get any error. Check the contents of the student and grade tables: Homer was indeed removed from the student table, so were all the tuples from the grade table referring to him.

The first ON DELETE CASCADE specifies that all rows (in the *grade* table) referencing a particular student will be removed when that student is removed. (And similarly, all rows referencing a particular course will be removed when that course is removed because of the second ON DELETE CASCADE .)

Very similar situation occurs if we try to change the primary key for a student. If we want to change Marge Simpson's ID to, say, 100 (no other student has this ID so we should be able to do so):

```
UPDATE student SET id=100 WHERE id=2;
```

we will get an error again (Error: FOREIGN KEY constraint failed). The reason for this is that the grade table has a row saying "the student with ID 2" got a grade of 5 in "the course with ID 1", and after changing the ID there would be no student with ID 2. We can instruct the database to update student IDs (and course IDs) in other tables by using ON UPDATE CASCADE options:

```
DROP TABLE grade;
```

```
CREATE TABLE grade(  
    student_id int NOT NULL,  
    course_id int NOT NULL,  
    grade char(1),  
    PRIMARY KEY(student_id, course_id),  
    FOREIGN KEY(student_id) REFERENCES student(id) ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY(course_id) REFERENCES course(id) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
INSERT INTO grade(student_id, course_id, grade) VALUES (1, 1, 'U');  
INSERT INTO grade(student_id, course_id, grade) VALUES (2, 1, '5');
```

Now let's try again:

```
UPDATE student SET id=100 WHERE id=2;
```

If you now check the contents of the student and grade tables, you will see that the ID of Marge has been updated to 100 in both tables.

Some final notes:

- The keywords in SQL are case-insensitive (i.e., `SELECT * FROM grade` is the same as `select * from grade`).
- You can write SQL statements and queries on a single line or multiple lines (as we did in this tutorial).
- The names of tables and columns might be case-sensitive or case-insensitive depending on the RDBMS you use.
- Several SQLs are separated by semicolons.