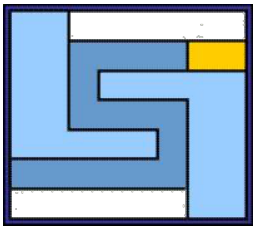


Lab 02: Table creation I

Introduction to Software Engineering and Information Systems I

Course 2021/22

David Ruiz, Inma Hernández, Agustín Borrego, Daniel Ayala

Index

1. Objective

The goal of this lab is to learn the SQL statements for the creation of database tables. The student will learn to:

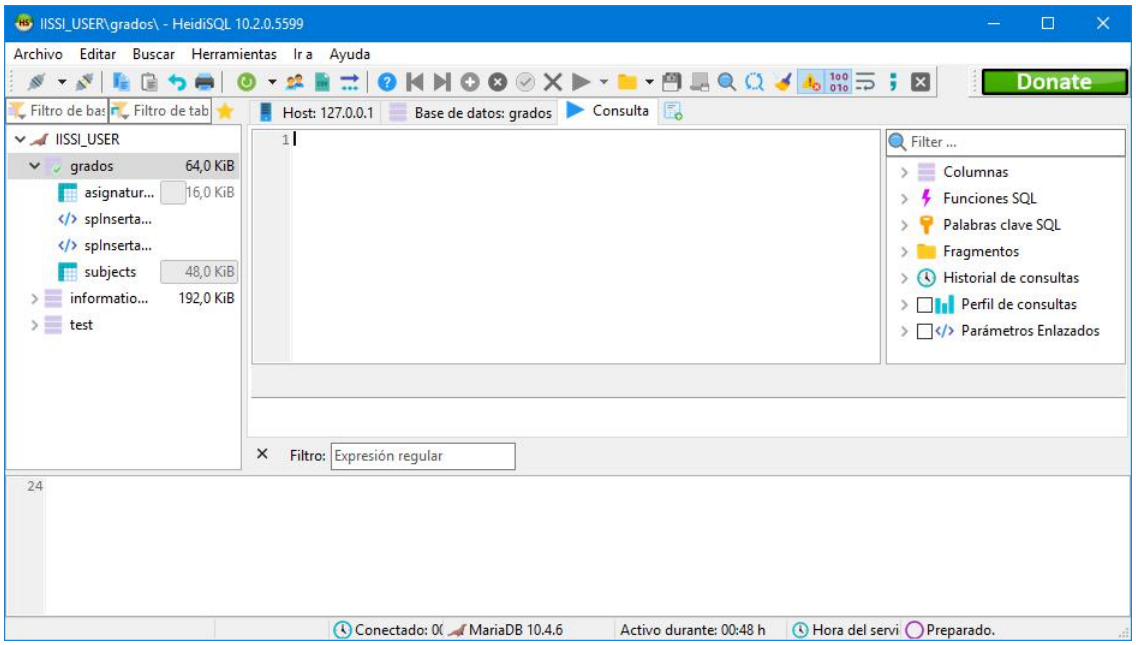
- Create tables using SQL statements.
- Define attributes with different types of data.
- Define primary keys.
- Define foreign keys and implement 1: N and N: M relationships.
- Define fields with auto-increment.
- Define default values for fields.

Only the subject management system will be considered. When implementing relationships, attributes that refer to relationships that are not part of this system should be ignored.

2. Environment setup

If MariaDB was successfully installed by following the procedure shown in the previous lab, the DBMS is now registered as a Windows service, so it should start automatically with the operating system.

We must now start HeidiSQL and connect as the iissi_user created in the previous lab. We select the degrees database and open the Query tab. In that tab we can write SQL code that will be executed on the selected database. The content of the tab can be written manually or be loaded from a .sql file:



3. Style considerations

SQL is not case sensitive, so code from different sources may use different code style. We will follow the following rules to unify our code style:

- The reserved words of the SQL language (that is, everything that is not names defined by us) will be written entirely in uppercase. For example: INSERT, UPDATE, DELETE.
- The table names will be written with the first letter in capital letters, in PascalCase[1] if they are made up of several words, and in plural. For example: Degrees, Subjects, Photos, UserComments.
- The names of attributes, constraints, and other elements that are not tables, will be written completely in lower case, in camelCase if they consist of several words. For example: degreeId, course, photoUrl, text.
- The code should be written in English.

[1] The CamelCase style joins several words without spaces, with the first letter of each capitalized: This Is An Example. The first letter of all can be uppercase or lowercase.

4. Create Degrees table

First we should implement in MariaDB the relation corresponding to the degrees:

Degrees (!degreeId, #name, years)

To do this we should write the following code in the Query tab (some words are colored differently by HeidiSQL because they are normally used in different contexts, ignore the coloring):

```
1 DROP TABLE IF EXISTS Degrees;
2
3 CREATE TABLE Degrees(
4    degreeId INT AUTO_INCREMENT,
5    name VARCHAR(60),
6    years INT DEFAULT(4),
7    PRIMARY KEY (degreeId)
8 );
```

Observe the following:

- The first line, DROP TABLE IF EXISTS Degrees;, is responsible for dropping the table if it already existed. Without this line, an error would be raised if we try to create the table when it already exists (for example, because by testing we want to run the script multiple times). It is convenient that, in a table creation script, the first thing done is to delete all the related tables if they exist.
- The content within CREATE TABLE Degrees contains the table definition. T definition of attributes or constraints are separated by commas.
- INT and VARCHAR are data types. VARCHAR corresponds to String, and the number in parentheses is the maximum number of characters.
- In the last line we indicate which is the primary key. The attributes that are part of it are written in parentheses. If a primary key consists of several attributes, they would all be included separated by commas.
- We make sure that the names of the IDs are always different (for example: degreeId, subjectId). Giving different names to the IDs of all the tables avoids problems in case of confusion.
- Attributes marked AUTO_INCREMENT will be automatically given value following a sequence. The attributes marked with AUTO_INCREMENT must be a key, either primary or alternate (to be seen in the next lab).
- With DEFAULT () we indicate the default value of an attribute.

5. Create Subjects table

We implement the relationship corresponding to the subjects:

Subjects (!subjectId, @degreeId, #name, #acronym, credits, year, type)

```
1 DROP TABLE IF EXISTS Subjects;
2
3 CREATE TABLE Subjects (
4    subjectId INT AUTO_INCREMENT,
5    degreeId INT NOT NULL,
6    name VARCHAR(100),
7    acronym VARCHAR(8),
8    credits INT,
9    year INT,
10   type VARCHAR(20),
11   PRIMARY KEY (subjectId),
12   FOREIGN KEY (degreeId) REFERENCES DEGREES (degreeId)
13 );
```

Add table drop if exists before creating it and note the following:

- The maximum length of VARCHAR fields is modified accordingly.
- NOT NULL is used to force a field to be filled, that is, it must take a value.
- For now, there is nothing to force the type field to take a value from the set of defined enumeration values.
- To set a foreign key, we should include an attribute of the same type as the key that we want to reference, and we use the FOREIGN KEY statement. The statement first receives the attributes belonging to the foreign key, the referenced table, and the attributes of the referenced table. The latter must be a primary or alternate key.
- The name of the foreign key does not have to be the same as the referenced primary key, but this is a common practice.
- If a foreign key consists of multiple attributes, they would be enclosed in parentheses and separated by commas.
- In this case we have implemented a 1: N relationship. For these relationships, the foreign key is added on the N side of the relationship.
- The existence of foreign keys can create issues, for example, if you try to delete or modify a row that is referenced by another through a foreign key. By default, it is not allowed to delete referenced rows in other tables.

6. Create the Groups table

We must now implement the relation corresponding to the groups:

Groups (!groupId, @subjectId, name, activity, year)

```
26 CREATE TABLE Groups(
27    groupId INT AUTO_INCREMENT,
28    name VARCHAR(30),
29    activity VARCHAR(20),
30    year INT,
31    subjectId INT,
32    PRIMARY KEY (groupId),
33    FOREIGN KEY (subjectId) REFERENCES Subjects (subjectId)
34 );
```

Add table drop if it exists before creating it. Notice how composition is implemented the same as a normal relationship (a subject is made up of multiple groups). However, it might be set that, when deleting a subject, the groups that reference that group should be deleted instead of raising an error.

7. Create the Students table

We implement the relation corresponding to the students:

Students (!studentId, accessMethod, #dni, firstName, surname, birthDate, #email)

```
39 CREATE TABLE Students(
40    studentId INT AUTO_INCREMENT,
41    accessMethod VARCHAR(30),
42    dni CHAR(9),
43    firstName VARCHAR(100),
44    surname VARCHAR(100),
45    birthDate DATE,
46    email VARCHAR(250),
47    PRIMARY KEY (studentId)
48 );
```

Add table drop if it exists before creating it and note the following:

- When the CHAR type is used, the maximum length of the attribute is not indicated, but the exact length, in this case 9 characters. Unlike VARCHAR, in this case it would not be allowed to enter data with a length other than 9 characters.

- The DATE type allows you to save a date without a time. To save a time, the type TIME would be used, and for a date with time, the type DATETIME.
- The N: M relationship between students and groups has not yet been implemented. With a foreign key in one or both tables, the relationship could not be implemented well. An additional table is required.

To implement the N: M relationship we must create an additional table containing related pairs of students and groups. We create it with the following code:

```
50 CREATE TABLE GroupsStudents(
51    groupStudentId INT AUTO_INCREMENT,
52    groupId INT,
53    studentId INT,
54    PRIMARY KEY (groupStudentId),
55    FOREIGN KEY (groupId) REFERENCES Groups (groupId),
56    FOREIGN KEY (studentId) REFERENCES Students (studentId)
57 );
```

Add table drop if it exists before creating it and note the following:

- The table only has the attributes necessary to relate groups to students.
- The name of a table used to represent an N: M relationship is the union of the names of the two tables that it relates.
- Right now, it would be possible to repeatedly introduce the same student associated with the same group. This is incorrect, as a student can only belong to a group once. In the next newsletter we will see how to fix this problem using a restriction.

8. Create the Grades table

We implement the relation corresponding to the notes:

Grades (!gradeId, @studentId, @groupId, value, gradeCall, withHonours)

```
59 CREATE TABLE Grades(
60    gradeId INT AUTO_INCREMENT,
61    value DECIMAL(4,2),
62    gradeCall INT,
63    withHonours BOOLEAN,
64    studentId INT,
65    groupId INT,
66    PRIMARY KEY (gradeId),
67    FOREIGN KEY (studentId) REFERENCES Students (studentId),
68    FOREIGN KEY (groupId) REFERENCES Groups (groupId)
69 );
```

Add table drop if it exists before creating it and note the following:

- The attribute name gradeCall has been used instead of just call (call) because call is a reserved keyword in MariaDB, and cannot be used as a name, otherwise an error would be raised. The reserved words in MariaDB can be consulted at the official list.
- The DECIMAL type is used to define numbers that can have decimals. Decimal attributes definition requires two parameters: 1) the total number of digits, 2) how many of them are decimal. As the maximum note is 10.00, and we want there to be a precision of two decimal places, we give as parameters (4,2).
- The BOOLEAN type is used to define Booleans. Internally they are stored as numeric values 0 or 1.
- Several of the words used to define types in MariaDB are synonymous and represent various ways of writing the same type. For example, INT is a synonym for INTEGER; DECIMAL stands for DEC, NUMERIC for FIXED; and BOOLEAN is a synonym for BOOL or TINYINT (1).
- Apart from DECIMAL, there are the FLOAT and DOUBLE numeric data type. These types also allow you to store decimal numbers, but using floating point numbers, a way to represent decimal numbers roughly (due to binary systems used by computers) with a fixed number of bits. The DECIMAL type stores numbers exactly, albeit at the cost of using more memory and being less efficient.

9. Table deletion restrictions

By default, SQL databases do not allow table deletion while there are others that reference it through foreign keys. Therefore, at the beginning of our table creation scripts, we will delete them if they already exist in the opposite order to their creation. This guarantees that, at the time of deleting each table, there is no other one that references it. In the case of the tables created in this lab, it would result in the following:

```sql
DROP TABLE IF EXISTS Grades;
DROP TABLE IF EXISTS GroupsStudents;
DROP TABLE IF EXISTS Students;
DROP TABLE IF EXISTS Groups;
DROP TABLE IF EXISTS Subjects;
DROP TABLE IF EXISTS Degrees;
```

Performing these operations at the beginning of the table creation script allows you to run them as many times as you want, because if you try to create a table again when it already exists, an error will be raised, as aforementioned.

10. Task

Make corrections if necessary in the conceptual model of the course project. Make the relational model from the transformation scheme seen in class. Implement in SQL the creation of the rest of the tables of the course project.

To perform the task, create a copy of the repository for this session with your GitHub user. Make any necessary modifications to the supplied files, creating new files if necessary, and upload those changes to your copy on GitHub. Remember to set the privacy of your repository as "Private" and give the dfernandez6 user access as a collaborator.