Lab. 03: Table creation II

Introduction to Software Engineering and Information Systems I

Course 2021/22

David Ruiz, Inma Hernández, Agustín Borrego, Daniel Ayala

Index

1. Objective

The objective of this practice is to implement restrictions in the creation of tables using SQL scripts. The student will learn to:

- Add restrictions to attributes when defining tables.
- Define alternative keys.
- Implement enumerates.
- Define the behaviors of a foreign key in case of deletion/update of a related table.

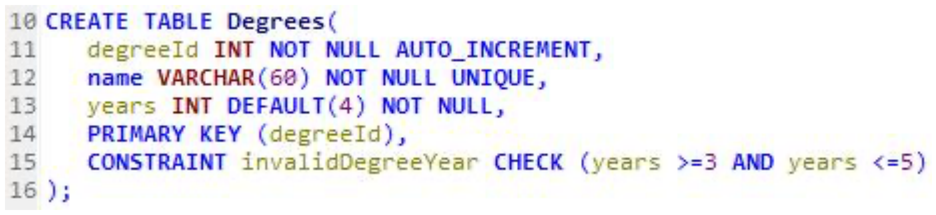
In the previous lab session, an SQL script was run to create tables with their attributes and relationships. However, as they were created, the tables allowed the introduction of invalid values for the attributes, such as negative numbers in some of them or values that are not part of an enumerate. In this lab we will refine the tables to include data constraints with which to implement alternative keys, enumerates, and simple business rules.

2. Environment setup

Connect to the database and open the tables.sql file used in the previous lab.

3. Constraints on the Degrees table

The most common and simple constraints may be defined along with the definition of the related attribute (inline definition), while those that require a more complicated boolean expression are defined separately (extra constraints). We add constraints to the Degrees table as follows:



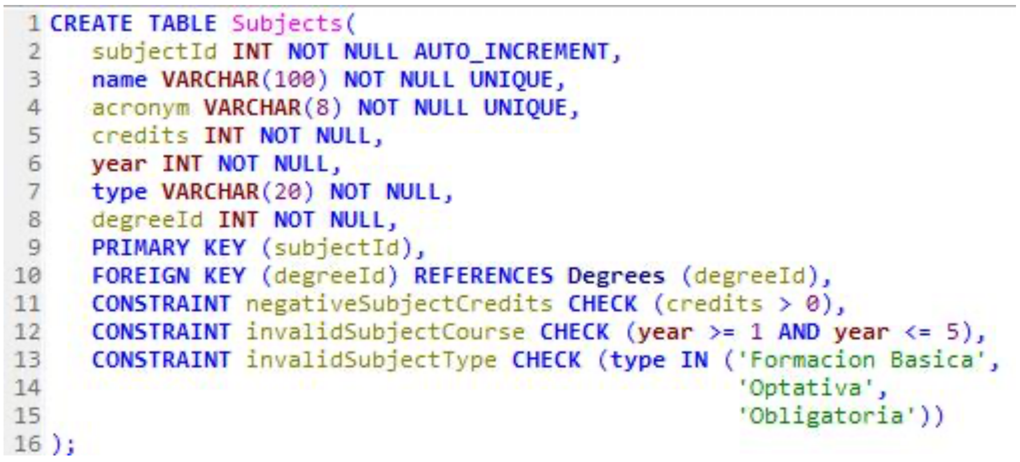
```sql
CREATE TABLE Degrees(
   degreeId INT NOT NULL AUTO_INCREMENT,
   name VARCHAR(60) NOT NULL UNIQUE,
   years INT DEFAULT(4) NOT NULL,
   PRIMARY KEY (degreeId),
   CONSTRAINT invalidDegreeYear CHECK (years >=3 AND years <=5)
);
```

Observe the following:

- NOT NULL constraint denotes that an attribute cannot be null, i.e, a value must be present. Attributes marked as primary key are NOT NULL by default. However, it may be explicitly written as a consistency reminder.
- UNIQUE constraint prevents the related attribute from accepting repeated values. This enables the attribute to work as an alternative key.
- Extra CONSTRAINT constraints are added to include expressions that must be satisfied. We write the name of the constraint (which will appear in the error message if it is not met), CHECK, and the Boolean expression that must be met. In this case, the constraint checks that the number of years of the grade is between 3 and 5.

4. Constraints on the Subjects table

We add constraints to the Subjects table as follows:



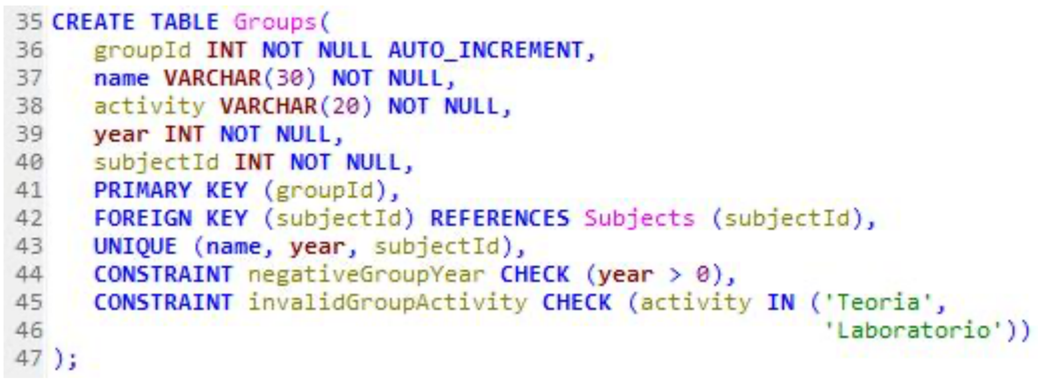
```
CREATE TABLE Subjects(
   subjectId INT NOT NULL AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL UNIQUE,
   acronym VARCHAR(8) NOT NULL UNIQUE,
   credits INT NOT NULL,
   year INT NOT NULL,
   type VARCHAR(20) NOT NULL,
   degreeId INT NOT NULL,
   PRIMARY KEY (subjectId),
   FOREIGN KEY (degreeId) REFERENCES Degrees (degreeId),
   CONSTRAINT negativeSubjectCredits CHECK (credits > 0),
   CONSTRAINT invalidSubjectCourse CHECK (year >= 1 AND year <= 5),
   CONSTRAINT invalidSubjectType CHECK (type IN ('Formacion Basica',
                                                  'Optativa',
                                                  'Obligatoria'))
);
```

Observe the following:

- The NOT NULL constraint is very common, since so far no attribute is optional.
- A foreign key can also be NOT NULL, indicating that the relationship is not optional.
  It is the difference between multiplicity 0..1 and multiplicity 1.
- In the last constraint we have implemented the "type" attribute as an enumerate by checking that its value is in a set of possible values.

5. Constraints on the Groups table

We add restrictions to the Groups table as follows:



```sql
CREATE TABLE Groups(
   groupId INT NOT NULL AUTO_INCREMENT,
   name VARCHAR(30) NOT NULL,
   activity VARCHAR(20) NOT NULL,
   year INT NOT NULL,
   subjectId INT NOT NULL,
   PRIMARY KEY (groupId),
   FOREIGN KEY (subjectId) REFERENCES Subjects (subjectId),
   UNIQUE (name, year, subjectId),
   CONSTRAINT negativeGroupYear CHECK (year > 0),
   CONSTRAINT invalidGroupActivity CHECK (activity IN ('Teoria',
                                                        'Laboratorio'))
);
```

Notice how this time we wrote the UNIQUE clause on a separate line instead of the inline version used previously. The difference is that this time the UNIQUE constraint involves three columns that have been indicated in parentheses: name, year, and subjectId, indicating that there cannot be two rows in which the combination of these three attributes is the same. In this way, we achieve that there cannot be two groups with the same name in the same subject and the same year.

6. Constraints on the Students and GroupsStudents tables

We add constraints to the Students table as follows:

```sql
CREATE TABLE Students(
   studentId INT NOT NULL AUTO_INCREMENT,
   accessMethod VARCHAR(30) NOT NULL,
   dni CHAR(9) NOT NULL UNIQUE,
   firstName VARCHAR(100) NOT NULL,
   surname VARCHAR(100) NOT NULL,
   birthDate DATE NOT NULL,
   email VARCHAR(250) NOT NULL UNIQUE,
   PRIMARY KEY (studentId),
   CONSTRAINT invalidStudentAccessMethod CHECK (accessMethod IN ('Selectividad',
                                                                 'Ciclo',
                                                                 'Mayor',
                                                                 'Titulado Extranjero'))
);

CREATE TABLE GroupsStudents(
   groupStudentId INT NOT NULL AUTO_INCREMENT,
   groupId INT NOT NULL,
   studentId INT NOT NULL,
   PRIMARY KEY (groupStudentId),
   FOREIGN KEY (groupId) REFERENCES Groups (groupId),
   FOREIGN KEY (studentId) REFERENCES Students (studentId),
   UNIQUE (groupId, studentId)
);
```

Notice the UNIQUE constraint on the GroupsStudents table. By making the foreign key pair unique, we make each student can only be associated with a group once.

7. Constraints on the Grades table

We add constraints to the Grades table as follows:

```sql
CREATE TABLE Grades(
   gradeId INT NOT NULL AUTO_INCREMENT,
   value DECIMAL(4,2) NOT NULL,
   gradeCall INT NOT NULL,
   withHonours BOOLEAN NOT NULL,
   studentId INT NOT NULL,
   groupId INT NOT NULL,
   PRIMARY KEY (gradeId),
   FOREIGN KEY (studentId) REFERENCES Students (studentId),
   FOREIGN KEY (groupId) REFERENCES Groups (groupId),
   CONSTRAINT invalidGradeValue CHECK (value >= 0 AND value <= 10),
   CONSTRAINT invalidGradeCall CHECK (gradeCall >= 1 AND gradeCall <= 3),
   CONSTRAINT duplicatedCallGrade UNIQUE (gradeCall, studentId, groupId)
);
```

Notice the following:

- This time, the UNIQUE constraint has been included as part of a constraint. This way the constraint is given a name that will be displayed when not met. In this case, the CHECK reserved keyword is unnecessary.
- The UNIQUE constraint ensures that there are not several grades for the same student, in the same subject, in the same session. However, several grades may be present for different calls.

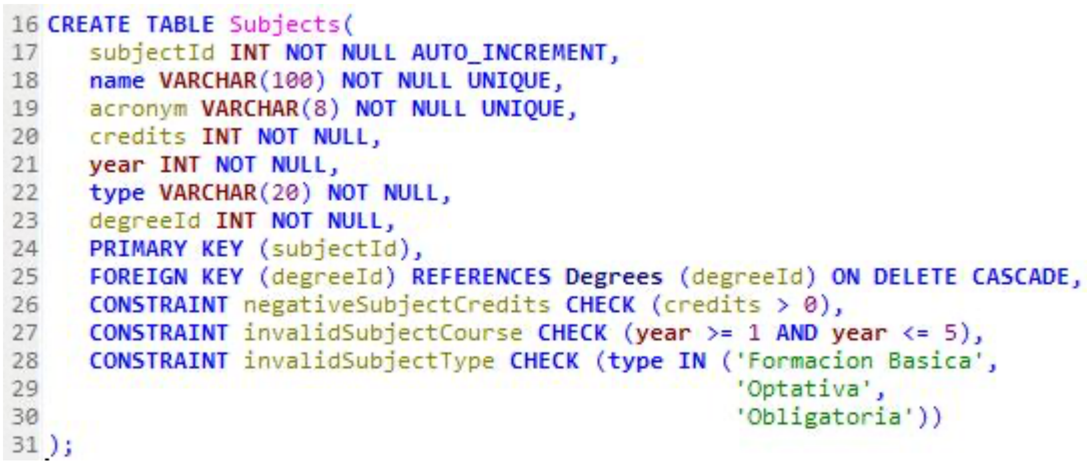
8. Deletion/Update strategies for foreign keys

As explained in the previous lab session, foreign keys impose by default a restriction on the deletion of related rows: for example, consider the Degrees and Subjects tables, where degreeId is the primary key of Degrees and the foreign key of Subjects, relating the subjects with the degree.

In this case, a degree could not be deleted if there is at least one subject that refers to such degree through the foreign key degreeId defined in the Subjects table. The same would happen if we try to modify the value of the column degreeId in the table Degrees if there is, at least, one related subject. However, this foreign key behavior can be changed using SQL statements:

- ON DELETE/UPDATE RESTRICT prevents deletion/update in the referenced row. Is the behavior default.
- ON DELETE/UPDATE CASCADE deletes/updates the foreign key row when the referenced row is deleted/updated (cascading erase/update, hence its name).
- ON DELETE SET NULL sets the foreign key to Null when the referenced row is removed. Only possible if the foreign key does not have the constraint NOT NULL, that is, for multiplicity 0..1.
- ON DELETE/UPDATE SET DEFAULT sets the foreign key to its default value when the row referenced is cleared. Only possible if a default value has been defined for the foreign key using DEFAULT ().

For example, suppose we want to delete a subject when the related degree is deleted from the database. In that case, we should modify the declaration of the Subjects table:



```sql
CREATE TABLE Subjects(
   subjectId INT NOT NULL AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL UNIQUE,
   acronym VARCHAR(8) NOT NULL UNIQUE,
   credits INT NOT NULL,
   year INT NOT NULL,
   type VARCHAR(20) NOT NULL,
   degreeId INT NOT NULL,
   PRIMARY KEY (subjectId),
   FOREIGN KEY (degreeId) REFERENCES Degrees (degreeId) ON DELETE CASCADE,
   CONSTRAINT negativeSubjectCredits CHECK (credits > 0),
   CONSTRAINT invalidSubjectCourse CHECK (year >= 1 AND year <= 5),
   CONSTRAINT invalidSubjectType CHECK (type IN ('Formacion Basica',
                                                  'Optativa',
                                                  'Obligatoria'))
);
```

Notice how the ON DELETE CASCADE statement has been added to the foreign key declaration of degreeId, which will cause the subject to be deleted if the referenced degree is removes.

9. Tasks

Implement as many business rules and model constraints as possible using NOT NULL, UNIQUE, CONSTRAINT, and ON DELETE/UPDATE.

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 dfernandez6 access as a collaborator.