# SD202 “Databases” class: final project

The goal of this lab session is to to design a Web application to manage students and classes in a university. The code for the Web application is already written, but it is missing everything that relates to the database, i.e., the schema definition and queries. Your goal is to design the database schema and complete the queries.

## Step 0: Retrieving the files

Retrieve the lab session skeleton from Moodle and decompress it. It consists of several files:

- `README`: readme file for step 1
- `forms.py`: form definitions (do not edit)
- `model.py`: definition of the model (to edit)
- `requirements.txt`: list of required Python modules for step 1
- `rooms.json`: list of Télécom rooms for step 6
- `run.py`: definition of the Web interface (run, but do not edit)
- `static/style.css`: CSS style of the Web interface (do not edit)
- `templates/*.html`: HTML templates (do not edit)

## Step 1: installing dependencies

You should already have set up the environment on your computer. If you have not, please check the `README` file and do it now.

## Step 2: getting the Web application to work

Open a terminal, navigate to the folder of the Web application, and run:

```
python3 run.py
```

You should get a message of the form:

```
$ ./run.py
 * Serving Flask app "run" (lazy loading)
 * Environment: production
   WARNING: This is a development server.
   Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with watchdog (inotify)
 * Debugger is active!
```

Leave the terminal open, open a Web browser on your machine, and go to: `http://127.0.0.1:5000/`. Check that the Web interface loads. Of course, at this point, only the main page will work, and other Web pages will give an error.

## Step 3: designing the basic schema

In the first step of this project, we want to represent people, rooms, curriculums (sets of courses), and courses. Here are the constraints (you may need to add identifiers in addition to the following):

- People have a first name and last name, an address, and a phone number.
- Rooms have a name and a capacity.
- Curriculums have a name, a secretary, and a director (who are people).
- Courses have a name and a teacher (who is a person).
- People can be registered in a curriculum (or several curriculums, or no curriculums).
- Courses can be part of a curriculum, with a number of ECTS indicating their value. The same course can be part of multiple curriculums, possibly with different ECTS values.
- Each course can have an unlimited number of room reservations, with a starting and ending date and time. (In this lab, we will not try to guarantee that room reservations do not overlap.)

Make an entity-relationship diagram and create a database schema to represent this information. (For simplicity, do not use inheritance, and only use one single People entity, not multiple entities to distinguish between students and professors.) Make sure that people, rooms, curriculums, and courses have an `id` column serving as a primary key to refer to them. Feel free to make any choices you wish for constraints not specified here.

Your end schema should have 4 tables representing entity types, and 3 tables representing relationships.

## Step 4: creating the basic schema

SQLite is a self-contained database engine which stores a database in an individual file. Write an file `schema.sql` with the SQL commands to create the database schema designed in the previous step.

Hint: if you wish to use foreign keys, you have to enable them by running `PRAGMA foreign_keys = ON` before running any other command. Warning: the `id` columns should auto-increment; in SQLite this is achieved by declaring them as `INTEGER PRIMARY KEY`.

Once the file `schema.sql` is ready, create these tables in a database called `univ.db`. To do this, there are two possibilities. The first possibility is to use or adapt the `init_db.py` Python script provided on Moodle; you can then go directly to Step 5. The second possibility is to do it on the command line, which we explain in the rest of Step 4. For the second possibility, you should navigate to the project folder, and run the command:

```
sqlite3 univ.db
```

Then, issue the following in the SQLite prompt to execute the commands that you wrote:

```
.read file.sql
```

You can also run SQL commands directly in the SQLite prompt, i.e., creating tables, creating test data, or trying out one problematic command. However, it is recommended to write all your code in a separate file (and not directly in the SQLite prompt), so that you can more easily edit it if you change your mind about something.

Once you are done with SQLite3, run `.exit`. If necessary, you can destroy everything and reinitialize the database by removing the `univ.db` file, and recreate it as explained above.

## Step 5: completing the queries

Open the file `model.py` and complete the queries marked `TODO`, from `TODO4` to `TODO15`.

Note that the queries are *prepared queries*, i.e., they take as parameters the values that are provided as a second argument to the `self.cursor.execute` call. These parameters should be used in the query, in the order in which they are given, with the character `?`. The first queries (Q1, Q2, Q3) can be used as an example, but you will need to adjust them to the schema that you created.

When deleting objects, make sure that no broken references to them remain, i.e., removing a person should automatically delete any enrolment information for this person and any class or curriculum for which they teacher, secretary, or director. This can be achieved with foreign keys and `ON DELETE CASCADE`.

Whenever you modify the file `model.py`, the Web application should automatically reload, and you can then refresh pages in your Web browser to test.

Once you have completed questions 2 to 15, you should be able to open the tabs "Persons", "Curriculums", "Courses", and "Rooms", you should be able to create and delete every type of objects, and you should be able to go to the details of a room and add room reservations. It is normal for now that going to the detail of other objects will cause an error, as not all queries are completed yet.

## Step 6: importing rooms

The list of all classrooms in Télécom is provided as the file `rooms.json`. Import this dataset into your database, so that the tab "Rooms" lists the actual rooms available at Télécom.

To do this, one first option is to write a Python program that imports the JSON (using `import json` and `json.loads`) and runs database queries using SQLite (using `import sqlite3` as in `model.py`).

Alternatively, you can transform the JSON file into SQL commands or a CSV file, e.g., using the `jq` command-line utility.

## Step 7: advanced schema

We now consider a more advanced version of the database schema:

- Each course has one or several *examinations*, which have names (e.g., "final exam", "mid-term exam"), a date when they take place, and a coefficient
- Every student following a class (i.e., registered to a curriculum containing the class) may have a *grade* for each examination in the class (if the grade is not known in the database, then by default it is 0).

Extend the database schema with tables to contain this information.

Complete the `models.py` file to write the corresponding queries and finish the Web application.

## Acknowledgements

This lab is adapted from material by Louis Jachiet.