**OBJECTIVE:** Get comfortable using one of several SQLAlchemy Many to Many relationships.

**INTRODUCTION:** I have put together another round of sample code for you, this time implementing a relationship from Student to Major. A student **can** declare more than one major (thought it’s not very common) so we need to represent the association between Student and Major as a Many to Many. Since a given Student can never declare a given Major more than once, we will employ a Many to Many without history for this, meaning that the primary key of the junction table between Student and Major will be exactly the migrated foreign keys from Student and Major.

Note that each Major belongs to one Department, but I’m going to assume that we never have more than one major within the university by the same name. That would be terribly confusing. Which means the name of the major is unique, which, in turn, means that the relationship from Department to Major is **non**-identifying which means that I have simple single-column primary keys for both Student (student\_id is the surrogate primary key) and Major. Things will be a tad more challenging in your case.

The sample code for this project is in *Canvas | Modules | Week 7 | Week 7.2 | SQLAlchemy Simple Many to Many*. Download that directly to your SVL by opening the web browser from the SVL, logging into Canvas from there, and doing the download that way. It will save you having to upload the .zip file from your own machine to the SVL.

As before, to make this project work, you will need to install the SQLAlchemy and psycopg2 libraries into your new PyCharm project that you create for this assignment.

Please make sure that your Department class has all of the attributes that I asked you to give it in the first SQLAlchemy assignment. I stripped that class of everything but the primary key to make things simpler in the sample code.

Also note that I made the **name** of the department the primary key in the first assignment, and then changed over to the abbreviation (it’s generally shorter). You are free to use name or abbreviation (but not both) as the primary key of departments.

**Introspection**

From here on out, we are just going to allow the user two options: start from scratch: blow away the tables from the database and recreate them all over again. That option loses all the data that you might have stored. The second and final option is to reuse the existing tables without introspection. This will make your code a little easier to write and easier to grade. I’ve changed the sample code to reflect that in the Introspection menu.

Also, as before, you will work in teams of two.

As before, don’t use the Department.py that you find in the sample code, I just have that there so that I could have a parent for the course table.

**Changes to the sample code**

I’ve changed out db\_connection.py and orm\_base.py so that they no longer prompt you for the connection parameters such as the server, the port number, and so forth. Instead, they both use ConfigParser to read a configuration file. I’ve included a sample of a configuration file called DummyConfig.ini that you can use as a template. Then you’ll have to rename DummyConfig.ini to config.ini for it to work properly.

I’ve also added a SQLAlchemyUtilities.py that has a method called check\_unique in it. If you look at add\_department, you’ll see how to use check\_unique to accept as input an object, and check **all** the uniqueness constraints that apply to the corresponding table. Please study the code that you find in SQLAlchemyUtilities.py. I’m going to ask you do write that same thing in MongoDB in a few weeks. Please use check\_unique in your add\_section and add\_enrollment methods. You can leave the other add methods alone if you want.

**PROCEDURE:** For this homework assignment, you will need to:

1. This assignment will build on everything that you have done so far with SQLAlchemy: both the sample code that I have given you and the code that you have written. This means that if you as an individual student have had several partners so far, you and your latest partner will first have to agree which of your code you will merge into the team’s final product. It’s a thing. These are called “collages” of code where you merge contributions from multiple sources. It gets messy, give yourself time to do a lot of testing.
2. If you look carefully at the starter code that I supply you, I’ve created a method in main.py called boilerplate. The purpose of this is to perform some inserts into the tables so that you don’t have to insert test data. It is entirely up to you whether you use this method, or if you choose to change it. Of course, you could load up your database with sample data and then reuse the tables after that as a time-saver.
3. Use SQLAlchemy to create your enrollments table:
   1. Uniqueness Constraints
      1. This will have, as an OO attribute, a reference to Student **and** Section.
      2. Obviously, as Enrollment is the association class between Student and Section, the primary key of Enrollment must be the combination of the primary key migrating into Enrollment from Student and the primary key migrating into Enrollment from Section.
      3. We want to be sure that no student enrolls in the same **course** more than once in the same semester. For instance, you would not want a student to be able to enroll in section 01 of CECS 323 for Fall of 2023 and section 03 of CECS 323 for Fall of 2023. One way is to create a uniqueness constraint {department\_abbreviation, course\_number, section\_year, semester, student\_id}. But that only works if those columns migrate into enrollments from sections. If you gave sections a surrogate, you will have to find some other way to enforce this constraint.
   2. Student will have a List of instances of Enrollment that it manages so that the Student instance “knows” all Sections that they are enrolled in.
   3. Section will also have a List of instances of Enrollment so that Section “knows” who all the students are who are enrolled in that Section.
   4. In the sample code, you’ll see a method on Student called add\_major that populates the StudentMajor class when a student declares a new Major. A similar function: add\_student appears in Major. You will need to do the same for Student and Section to allow for each of those to maintain the Many to Many between them.
   5. Be careful that the primary key of Enrollment is {student\_id, department\_abbreviation, course\_number, section\_number, section\_year, semester}. I will understand if you decide at this point to add a surrogate to Section.
4. At this point in the process, Enrollment will not have any attributes beyond the migrating foreign keys coming in from its two parents. More on that **next** project.
5. Develop a simple console-based menu application in your main.py that will:
   1. Enroll a Student in a Section
      1. One option in your menu will be to enroll by adding a Section to a Student.
      2. Another option in your menu will be to enroll by adding a Student to a Section.
   2. **Un**enroll a student from a Section.
      1. Do not simply find the corresponding row in enrollments and delete it. Instead,
         1. Allow the user to remove an enrollment by starting with the student, and specifying the section. Your code will then call the student.remove\_enrollment method that you write.
         2. Allow the user to remove an enrollment by starting with the section, and specifying the student that you are dropping. Your code will then call the section.remove\_enrollment method that you write.
      2. Validate that the student is indeed enrolled in that section before attempting to drop that student from that section.
   3. List the enrollments:
      1. Select the student, and list the sections that they are enrolled in.
      2. Select the section and list the students who are enrolled in that section.
      3. Note that this will require that you perform a join in SQLAlchemy. Please consult the sample code for an example of how to do this.
         1. A short tutorial on the subject can be found in Canvas | Modules | Course General Information | Searching in SQLAlchemy | SQLAlchemy SELECT with JOIN.
         2. Also, you’ll find Canvas | Modules | Course General Information | Searching in SQLAlchemy | Overview of SQLAlchemy Query which is an article that treats the add\_columns feature which I found useful when doing joins.
   4. Delete a Section
      1. Be sure to tell the user if they attempt to delete a Section that does not exist.
      2. Be sure to warn the user that they cannot delete a Section that has enrollments.
   5. Delete a Student
      1. Again, do not allow them to delete a student who has registered in one or more sections.
      2. Make sure that they do not attempt to delete a student who does not exist.

**WHAT TO TURN IN:**

* All the Python code that you wrote or updated.
* A screenshot from DataGrip to prove that you were successful inserting into the Enrollments table.
* A movie in which you demonstrate your application. You can do this in Zoom and record it to an .mp4 file that you upload or record to the cloud and give me the URL.
  + Be sure to turn off the logging for the demo.
  + Be sure that there is a verbal narrative telling your audience (yours truly) what you are doing as you go through the demo.
  + Be sure that you prove that each operation succeeded. For instance, if you add a student to a section, then list the students enrolled in that section to illustrate that they are gone.
* Your team’s collaboration file. You will find a template for that at: Canvas | Modules | Course General Information | Collaboration Form.

Your course attribute in [Section.py](http://section.py/) is never initialized. Please look at [Course.py](http://course.py/) set\_department method. In there, I accept the Department instance that is the parent of the course, and migrate the department's abbreviation, and the department attribute of Course is set to point to the parent Department. You need to do the same thing in Section. The difference here is that you need to bring down both the departmentAbbreviation and the courseNumber from the parent Course down into the Section child. There is absolutely no need to include the department abbreviation nor the course number in the arguments for \_\_init\_\_ if you pass in the Course that is the parent of the Section.  
  
You are missing your sections attribute in [Course.py](http://course.py/). Sections will need a back\_populates of 'course' in order to complete the relationship (from the OO perspective) between Course and Section.

You don't check for duplicate rows on any of the three uniqueness constraints for sections (remember, the primary key is just another uniqueness constraint).