Katie Rodeghiero

CPSC 408 – Database Management

CoPA and Dodge Student Database

Problem

Despite having nationally ranked programs in dance, theater, and film, Chapman students have no resources to facilitate collaboration across departments. The current state of the art solution is Facebook. If I were looking to collaborate with students in the film department, I would need to scour social media to find a cinematographer to then direct message. If someone was looking for a dance student, they would need to first find the dance department's administrative assistant who would then post in the all-dance majors' Facebook group. From there, students would respond to the inquiry. Due to this convoluted system of finding individuals to work with, there is a lack of cross department collaboration within the Chapman arts community. It's my goal to create a simple contact book-like database application that allows students to easily find collaborators in other disciplines.

Related Work

While there is nothing like this at Chapman, the closest equivalent to an application of this nature is talent management platforms. Talent agencies are the way in which dancers, actors, and other creatives can get hired. While the main role of agencies is to submit clients to jobs, there's also an option to search their clients. This allows creatives searching for other creatives to access reels, headshots, resumes, and contact information. Here is an example of a popular bicoastal talent agency. Despite having a similar application, there is no way for the common user to really query the data. In actuality, the current app is somewhat similar to medical records in the sense that pulling up students is a similar operation to accessing patient records.

Framework

To develop this application, I utilized SQLite, Python, and Streamlit. I set out to utilize Streamlit, a Python based program that establishes a simple to use graphical user interface, to create an interface that's easy and intuitive for users to interact with. Streamlit was created as a dedicated front-end interface for databases. Because of this, the support and integration of SQLite was well thought out and there was enough support for when issues were met. As a result of the ease of use, there was more time allotted to the development of the final project as opposed to just learning about the "simpler" UI elements. While even with more time, the original goals set out to accomplish were not met, however, the additional time proved to be useful in the overall learning experience. Within the application there are four pages users can choose from: Search, Add, Update, and Delete. Within each, there are options for what can be done. I utilized SQLite to interface with my database and query the data.

Database

I created a very minimal database to emulate the information that would be stored for students in Chapman's College of Performing Arts and Dodge College of Film and Media Arts. This includes multiple tables regarding departments, students, classes, and works:

Class: All the classes in the course catalog for programs in both CoPA and Dodge

Department: Dance, Theater, Music, and Film department information

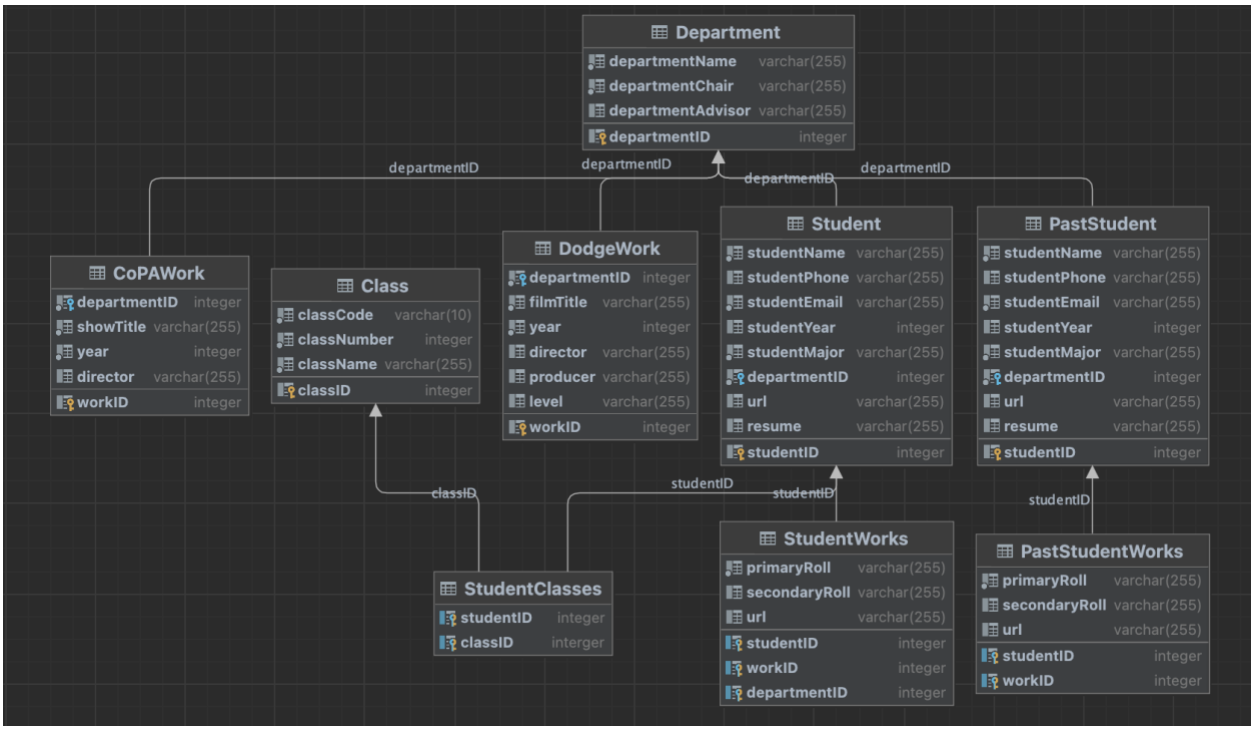
Student: Student identification, contact information, and portfolios for each student

StudentClasses: The classes each student has taken so others can see what classroom experience a possible collaborator has

StudentWorks: The works students participated in and the rolls they fulfilled

DanceWork, TheaterWork, MusicWork, and DodgeWork: Contains information about works specific to each department

I chose to separate work by CoPA and Dodge as they do things a little differently interms of naming conventions and necessary information. While even within CoPA things are a little different, that can be taken care of when querying information by simply including the year. This is because the seven dance shows a year are always called the same thing: Fall Faculty, Masters at Chapman, Works in Progress, Intime, Frameworks, and Mainstage. The same is true for music.



Functionalities

Within the Search functionality, there are a couple of buttons that allow users to quickly search for students of a given major. This returns to the page a list of all the students and their information. I had also intended to make other search option like allowing the user to search all the students of a given department by year. For example, this could return all the sophomores in

the theater department or all the senior in the music department. Another useful functionality I intended to implement was the ability to search by major. It would be beneficial to know who the music composition majors are if you're looking for a composer for your next dance or film piece and it would be beneficial to know who has a cinematography emphasis if you need something filmed.

While these queries have all been to the student table alone thus far, these next few would require a join between the Student, Work (depending on the department), and StudentWorks tables. Users could search by work which would look different to each department. One could search the dance works and input "Masters at Chapman 2018" and get a list of all the dance students involved. This would require a StudentWorks to be left joined with DanceWorks on workID and Student on studentID. One could search the theater works for "Our Town" and get a list of all the students and their respective rolls/character names. Another query done with joins is to output students who have completed certain rolls within a production. For a Dodge student, it's valuable to know who has been a grip or 1st AD to know who can handle of specific responsibilities of the position. For music, it's important to know who first chair has been or who has performed certain sections of a piece before. This would require the same join as before but with MusicWorks instead.

I was also interested in trying to make a "Create Your Own Search" functionality. This would have been implemented by first identifying what the user is interested in seeing. Because there's the possibility needing to join across tables, Python code would then identify the tables we need information from. From there, it would take the first thing that the user is interested in and make the appropriate joins. Lastly, it would output to the screen.

The second functionality of the program is the ability to add. There's a couple of things users could add to the database: Student, StudentClasses, Work (for any department), and StudentWorks. The user doesn't have the ability to add new departments or new classes to the database as those are both preestablished and not changing. When adding a student, there are a couple attributes with NOT NULL constraints. This being places like name, studentID, major, email, and department. This is to ensure the database can serve the purpose it's created for, contacting others across departments. Once you add a student, you would then be prompted to add StudentClasses. Ideally, you'd be able to add multiple at a time. In order to add a work, the user would first be prompted to choose which department they wish to add work to. From there they could input the new work. The user would then be prompted to add the people who worked on the project i.e., StudentWorks. Both StudentWorks and StudentClasses could be added to without first entering a Student or a Work as a student takes more classes each year and there's a possibility that a student wasn't listed in a StudentWork when they should have been.

The next functionality is the ability to update. This functionality can only be used sparingly, and users can't just change any record. There's the ability to update a student and a department. For the student, a user could update name, year, phone, email, URL, major, and resume. The main thing is the inability to update a studentID. Students may be changing a name for personal reasons, might change a major, and will most likely want to update their reels and resumes with the work they've completed at school. The only attributes that could be updated for a department are the department advisor and the department chair.

The final functionality of the web app is the ability to delete. The only attribute that can be deleted is a student. This would occur when a student graduates and would be moved to the PastStudent table. When deleted, there would be an ON DELETE CASCADE that takes out the

student itself as well as the StudentClasses and StudentWorks. It would also remove them from the StudentWorks table as well. On this transfer to a PastStudent table, a transaction would be utilized in order to ensure the data first moved to the PastStudent table and a PastStudentWorks table so no information is lost.

Assessment

I feel that this was a great project and one I enjoyed working on. I feel that the expectations set for the assignment were fair and reasonable. I wish I had more time to set aside and work on this project, I just ran out of steam at the end of the semester. The overall learning experience and technical challenge has been beneficial and overall I enjoyed the project.