James Dorfman [20719824]
Brad Huang [20715503]
Jeffer Peng [20720513]
Robbie Zhuang [20710951]

CS348 Project Report

Summary

It is currently unfathomably difficult for students at the University of Waterloo's Faculty of Mathematics to plan out the remainder of their degrees and determine the remaining courses that they need to graduate. This process has so much friction because information about degree requirements, course prerequisites, course descriptions, class offerings, and course professors is scattered around different areas of the internet. Our application will remove this friction and enable students to effortlessly find all of the information they need for their degree planning. You can check out our demo on YouTube.

Who are the Users and How to Interact with the Application?

Student Interface: Students can provide their target degree and courses that they have already taken, and then they will be presented with all the remaining courses they need to graduate. Our app will give students key data about each course, such as prerequisites, term offerings, and common professors for the course. It will also allow students to search for a professor and see what courses they have taught in the last.

Administrator Interface: Our application will also provide an interface for administrators to add courses and term offerings.

Key Features

1. **My path to desired degree + What courses do I need to take to get there?**

   Students can provide their desired degree and a list of courses they have taken and the app will generate the list of courses that they need to take to complete their degree. They're shown a list of prerequisite groups where each will display a list of courses and the amount needed.

   Clicking on a prerequisite will pop up a flowchart of the prerequisites for that course. All the ones that have been taken will be in green, and ones that can be taken next are in blue. It's simple to see the OR options of different equivalent prerequisites. One can also expand and collapse courses to their personal preference.

Query Description:

- For the path to desired degree, we retrieve course codes & groups from the natural join between the course group member table and the degree requirement table. The endpoint returns a map of courses to their groupID.
- To generate the prerequisite graph for a course, we use a recursive SQL query. The base case of our recursive query selects the course we want a graph for, and our recursive step joins the prerequisite table onto this recursive query and selects our course's prerequisites. In the end, we retrieve a table of courses and their parents. Each course's parent represents a course it is a prerequisite for. To convert this table into a graph, we use a Python script to convert the table into a recursive data structure. Then, we send it to the front end.

Implementation: the backend is implemented in main.py, and the front-end is done in degreeReq.component.[ts/html/css].

2. **How do I take that course? Let's see some information about that course.**

   Students can provide a desired course and the app will return the general description of the course, course title and course types. In addition, a list of prerequisites that need to be taken before enrolling in the course and the prerequisite graph described in feature 1 can be shown as well.

   Query Description:

   The sql query for course general information is a select query from the Course table. The prerequisites are obtained by finding all the courses in the CoursegroupMember table by using the coursegroupId in the Prerequisite table.

   Implementation: the backend is done in main.py, where the front-end is done in find-course.component.[ts/html/css].

3. **Let's make sure my course won't be full!**

   Students can see the historical data of a given course. The term info tab displays the offering information of the current as well as the next academic term. It also shows a table for the historical availability of the course. Each row represents a year and there are 3 columns, Winter, Summer, and Fall. In each box, we display the number of sections, and the professors that are teaching them, and based on how full the capacity is, we will use the color red, yellow, or green.

   Query Description:

   The current term and next term course information data is obtained by selecting information in courseoffering table based on the current and next term's code.

To calculate historical course data, we used 3 separate transactions to generate and join multiple temp tables. We used advanced PostgreSQL features such as cases and array aggregations.

Implementation: the backend is done in main.py, where the front-end is done in find-course.component.[ts/html/css].

4. **I wonder which professors have taught this course before?**

   In the prof info tab, we show a list of all the professors that have taught this course and order them by the total number of times they taught. The total number of terms that each prof has taught the course is also shown.

   Query Description:

   We perform a group by on courseoffering by the professor's first name, last name, and coursecode. We then count the aggregated information to get the number of terms and sections they have taught using the postgres array_agg() function.

   Implementation: the backend is in main.py; the front-end is in find-course.component.[ts/html/css].

5. **Professor XYZ is my favorite prof, I would like to find out what courses s/he is teaching!**

   In addition to searching for courses to see who is teaching them, users can also search for specific professors and see what courses they have taught. Users can enter all or part of a professor's first and last name, and a list of professors that match the search is returned. The search supports partial matching to names (it works even if the user has typos).

   When the user clicks on a professor's name, a popup will show to display the RateMyprof ratings of this professor (should they have one), the courses that she/he has been teaching, and the specific semesters when the professor taught these courses in the past.

   Query Description:

   This feature involves complex setups including views and indexes. We created a "professor" view to join the course offering table with the rate my prof record table. We then use a pattern-matching index on the prof names for text search.

   When we click on the name of a prof, we use a different endpoint, and the following query gathers what courses they have been teaching, again, with group by and the array_agg function.

   Implementation: the backend is done in main.py, the front-end is done in searchprof.component.[ts/html/css].

6. **I'm an administrative user who saw a new course update, I want to add it!**

Say a new course is created at Waterloo, the “Add New Course” feature allows Professors and Admins to add new courses. They can specify course metadata such as name, description, type, course code and the course’s prerequisites. When users search for courses, they will be able to see the newly added courses if they are relevant.

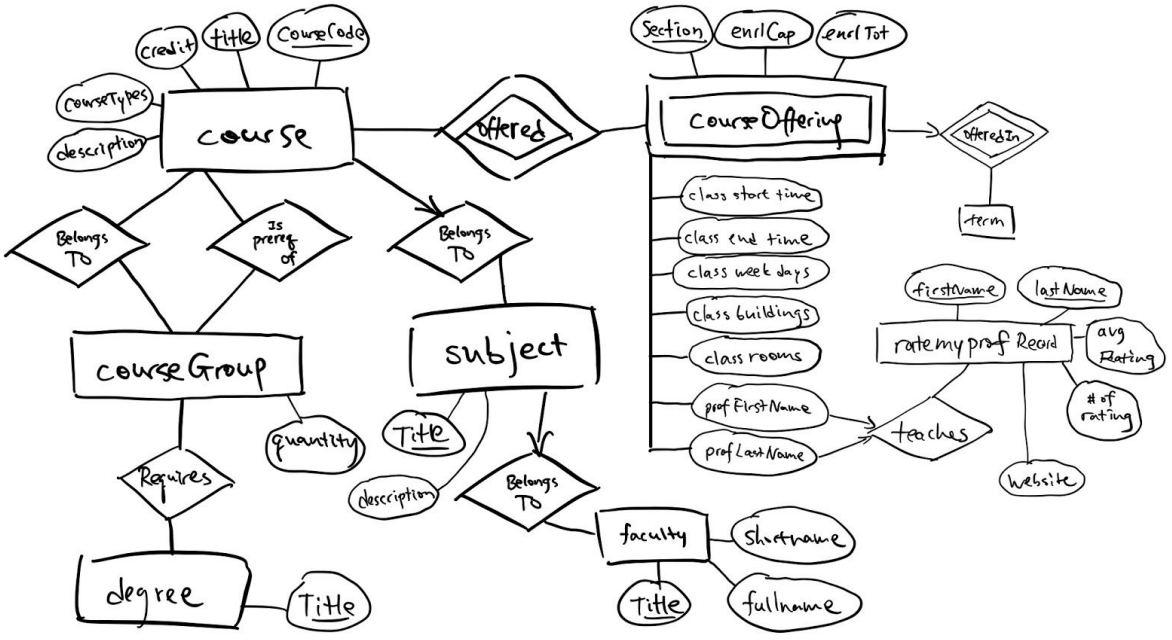
Query Description:

We used 6 INSERT INTO statements to insert into various tables and maintain foreign key constraints and data integrity.

Implementation: the backend is done in main.py, the front-end is done in add-course.component.ts/html/css.

Database Design Schema

E/R Diagram



Database Tables

faculty(<u>title</u>, shortName, fullName)

degree(<u>title</u>)

subject(<u>title</u>, facultyTitle, description)

course(<u>courseCode</u>, title, credit, courseTypes, description, subjectTitle)

courseGroup(<u>groupId</u>, quantity)

courseGroupMember(<u>courseCode, courseGroupID</u>)

prerequisite(<u>courseCode, prereqCourseGroupID</u>)

degreeRequirement(<u>degreeTitle, courseGroupID</u>)

term(<u>code</u>)

courseOffering(<u>courseCode, termCode, section</u>, profFirstName, profLastName, relatedComponent1, relatedComponent, enrlCap, enrlTot, classStartTime, classEndTime, classWeekdays, classBuilding, classRoom)

rateMyProfRecords(<u>firstName, lastName</u>, averageRating, numbersOfRatings, website)

Assumptions

We shall specify the following assumption on our data:

- Faculty titles are unique.
- Degree titles are unique.
- Subject titles are unique.
- Professor names are unique.
- Course codes are unique and course information is unchanged throughout the years.
- Term codes (1185, 1199, 1201, etc.) are unique
- For each term for each course, each course offering has a unique section number.

Plan for obtaining dataset

It will be difficult to gather data for **degree requirements**. Those are scattered in various places around the University of Waterloo's website. There does not exist a centralized repository with all of them. In fact, this disorganized nature of the data is what will make our application so useful and so likely to benefit our classmates.

Getting course degree requirements

1. Manually scrape data from https://ugradcalendar.uwaterloo.ca/group/MATH-Academic-Plans-and-Requirements

    Examples for Actuarial Science, Computer Science, Applied Mathematics

    https://ugradcalendar.uwaterloo.ca/page/MATH-Actuarial-Science1

    https://ugradcalendar.uwaterloo.ca/page/MATH-Bachelor-of-Computer-Science-1

    https://ugradcalendar.uwaterloo.ca/page/MATH-AM-Degree-Requirements-Applied-Mathematics

2. Manually preprocess data into csv containing course groups, required quantities and list of courses

    - Also generate lists of courses where the website only provided department names (ex. CHEM -> ['CHEM 120', 'CHEM 123', 'CHEM 209', 'CHEM '212', ...])

3. Generate SQL dump scripts and put them into db

We can get most of our **other required course metadata** from the uWaterloo openData API https://github.com/uWaterloo/api-documentation

Getting faculties, subjects, courses + info + prereqs, offerings

1. Use python and the requests library to download data from the uWaterloo openData API
    - Examples of relevant API endpoints:
        - Subjects: /codes/subjects
        - Courses + info: /courses
        - Prerequisites: /courses/{subject}/catalog_number/prerequisites
        - Offerings: /codes/terms + /terms/{term}/courses
2. Preprocess this data and format it
    - This step is different for every data type. But as an example, for prerequisites we need to translate each coure's prerequisites into a set of coursegroups
3. Check validity of data with Python and clean invalid data (the API sometimes has data errors)
4. After fetching all of this data, we use a Python script to generate INSERT queries from the data (a SQL dump script) so that we can move it into our database.

Here is a sample of some of the computer science courses (we got this data from the API linked above)

Subject	Course #	ID	Title	Description	Prereq
CS	100	4360	Introduction to Computing through Applications	Using personal computers as effective problem solving tools for the present and the future...	
CS	105	15054	Introduction to Computer Programming 1	An introduction to the fundamentals of computer programming through media computation...	
CS	106	15055	Introduction to Computer Programming 2	A continuation of the introduction to computer programming begun in CS 105...	CS 105

Hosting Platform and System Support

Database: Our database version of choice is PostgreSQL. Production version of the database is hosted on Google Cloud SQL Engine. Production data is procured off-platform and SQL dump files are generated and are ready to deploy through Google Cloud Storage.

Backend: The backend server is built with the Flask library on Python. We built REST endpoints with psycopg2 to access the SQL database. This backend is ready to be hosted on Google App Engine.

Frontend: Website built in HTML, CSS, and Javascript. We will use the web framework “Angular”, which is built in Typescript (a superset of Javascript).