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

CS348 Project Report

Note: please see the highlighted texts for changes made since Milestone 2.

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.

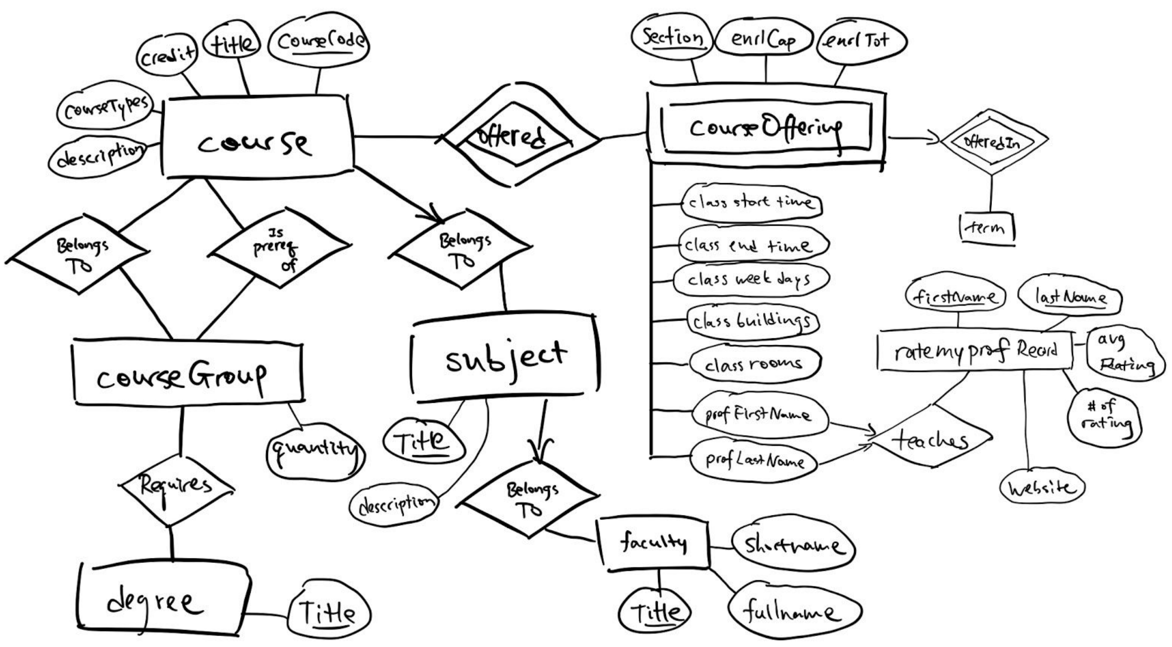
   Query Description:

   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).

SUBMIT EVERYTHING BEFORE THIS PAGE

Script

Presentation contents & clarity [18 points in total]

[James]

Application overview [2 point]

It is unfathomably difficult for current students at Waterloo to plan out their degrees and determine the courses they still need to graduate. This process has so much friction because information about degree requirements, and courses is scattered around different areas of the internet. Our application removes this friction and enables students to effortlessly find all of the information they need for their degree planning.

Feature Demo

How to use your application? (demonstrating functionalities/features) [6 points]

Application quality: [30 points in total]

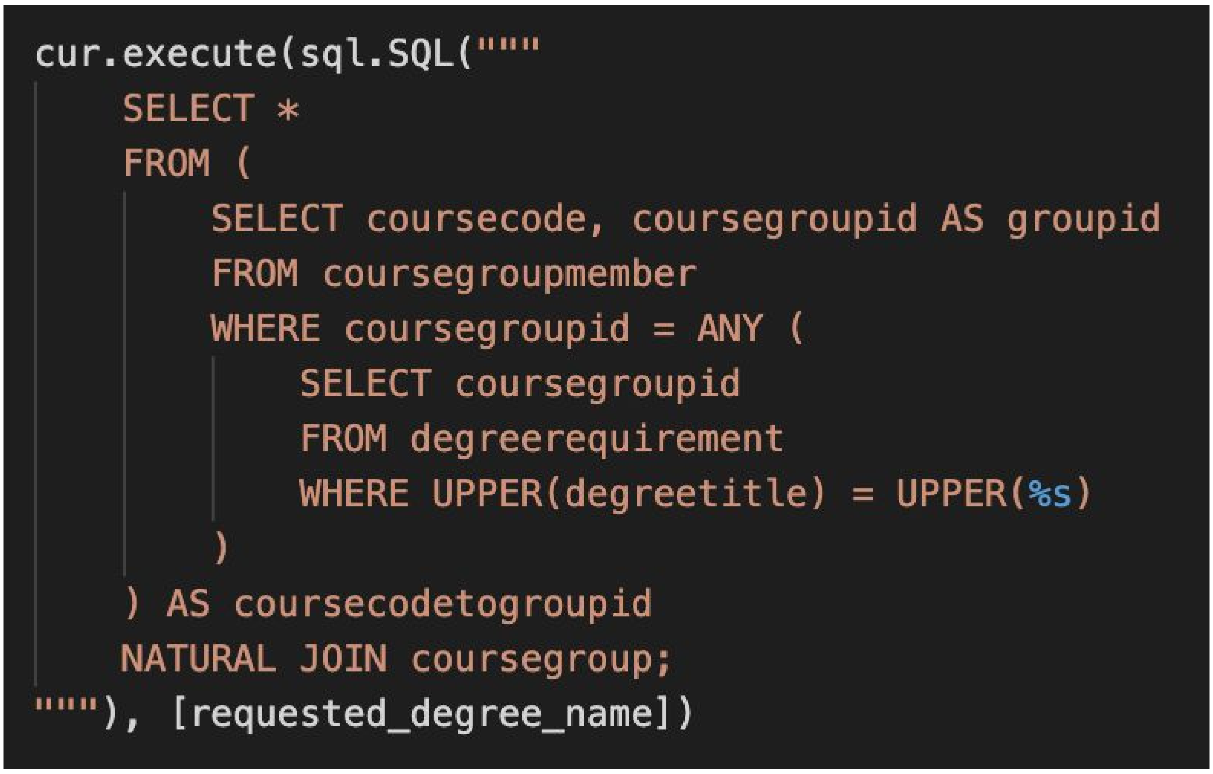
- *6 promised features [4 points for each completed functionality, capped by 24 points]*
- *If any of the features are impressive or innovative (e.g. well designed/user-friendly interface; improved query performance with indexes; a feature built based on complicated queries.) [2 point for each fancy feature, capped by 6 points.]*

1. **Path to Desired Degree**

- Robbie
- Actions
    - `Degree Requirement` select Computer Science, add CS 240, ECE 222, CS 246, take a look at the courses required (MATH 235 is on there).
    - Add MATH 235, click GO again and see it disappear!
- 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.

  The best part about this is that all the courses that the user has taken is filtered out.

Backend Query: Get course codes & groups from the natural join between the course group member table and the degree requirement table. Gives us a map of courses to it's group id.



```
cur.execute(sql.SQL("""
    SELECT *
    FROM (
        SELECT coursecode, coursegroupid AS groupid
        FROM coursegroupmember
        WHERE coursegroupid = ANY (
            SELECT coursegroupid
            FROM degreerequirement
            WHERE UPPER(degreetitle) = UPPER(%s)
        )
    ) AS coursecodetogroupid
    NATURAL JOIN coursegroup;
"""), [requested_degree_name])
```

What have I taken and what do I still need to take?

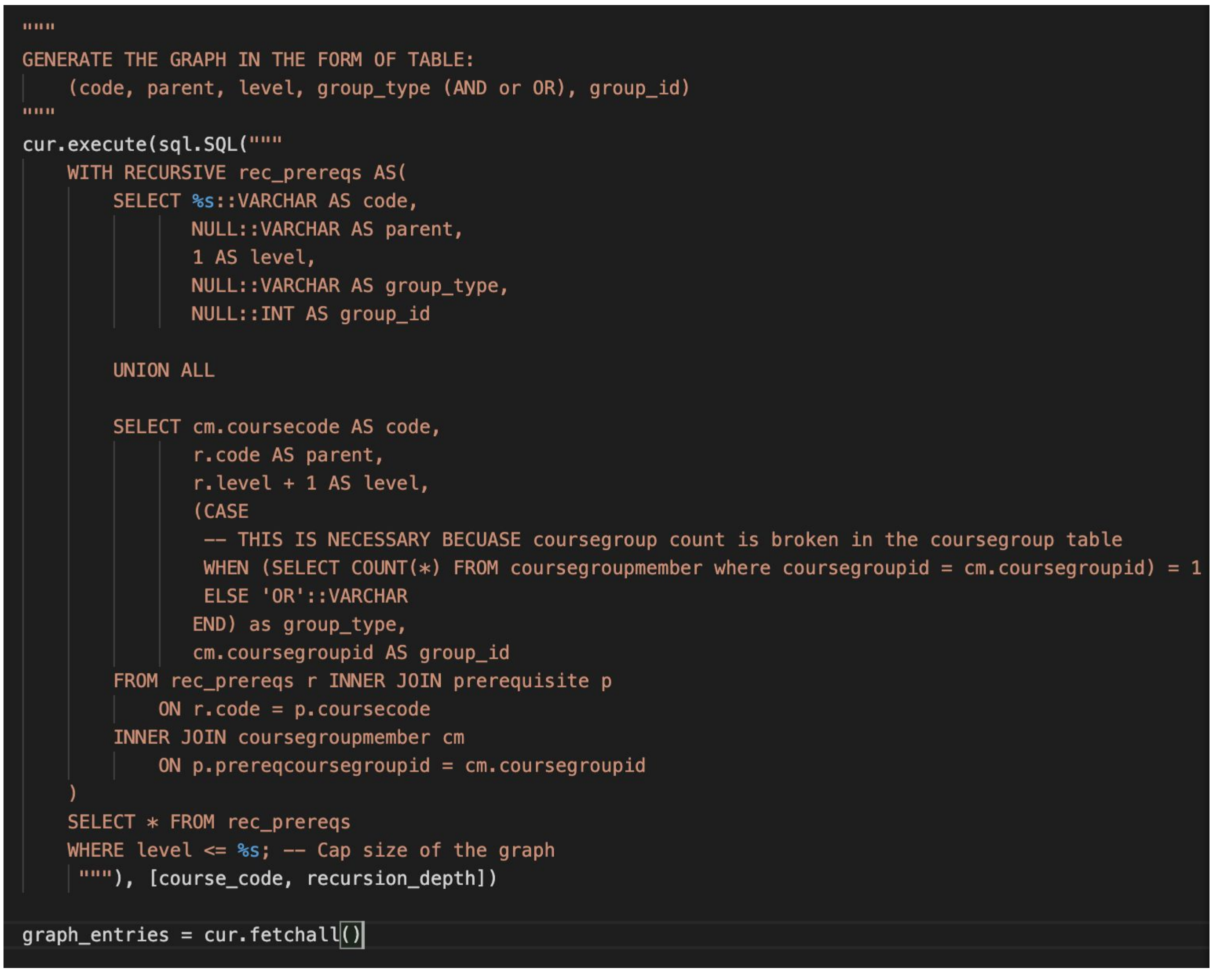
- Robbie/James?
- Actions:
    - `Degree Requirement` select Computer Science, add CS 240, ECE 222, CS 247, click CS 444 to see graph with coloured nodes
    - Collapse SE 350 in the popup to demonstrate I can hide paths I'm not interested in
- Clicking on a prerequisite will pop up a flowchart of the 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. Complex flowchart frontend. This also appears in the course search. This is our Fancy Feature #1.

Backend Query (fancy)**:** Complex recursive SQL query to generate the prerequisite graph This is our Fancy Feature #2.

- The base case of our recursive query just selects the course we want a graph for.
- Our recursive step joins the prerequisite table onto this recursive query and selects our course's prerequisites
- In the end, we get a table of courses and their parents. The course with no parents is the initial course
- To convert this table into a graph, we put the table into a recursive data structure and send it to the front end
- This is one of our fancy features

[James] Actions:

- Show the query. Flip to a terminal window and demonstrate the (preloaded) table that it fetches

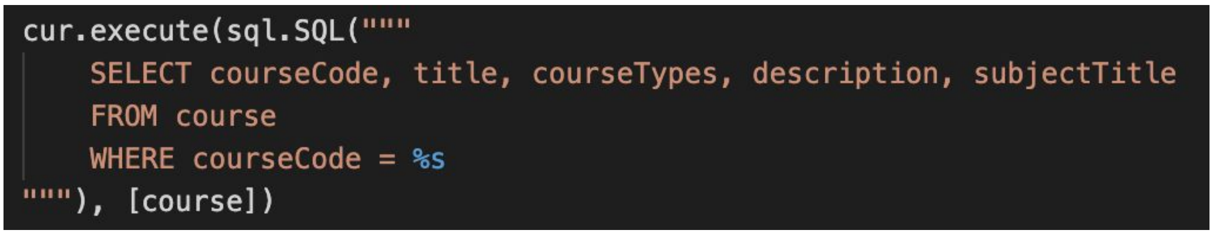


```
"""
GENERATE THE GRAPH IN THE FORM OF TABLE:
    (code, parent, level, group_type (AND or OR), group_id)
"""
cur.execute(sql.SQL("""
    WITH RECURSIVE rec_prereqs AS(
        SELECT %s::VARCHAR AS code,
               NULL::VARCHAR AS parent,
               1 AS level,
               NULL::VARCHAR AS group_type,
               NULL::INT AS group_id

        UNION ALL

        SELECT cm.coursecode AS code,
               r.code AS parent,
               r.level + 1 AS level,
               (CASE
                -- THIS IS NECESSARY BECUASE coursegroup count is broken in the coursegroup table
                WHEN (SELECT COUNT(*) FROM coursegroupmember where coursegroupid = cm.coursegroupid) = 1
                ELSE 'OR'::VARCHAR
               END) as group_type,
               cm.coursegroupid AS group_id
        FROM rec_prereqs r INNER JOIN prerequisite p
            ON r.code = p.coursecode
        INNER JOIN coursegroupmember cm
            ON p.prereqcoursegroupid = cm.coursegroupid
    )
    SELECT * FROM rec_prereqs
    WHERE level <= %s; -- Cap size of the graph
    """), [course_code, recursion_depth])

graph_entries = cur.fetchall()
```

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

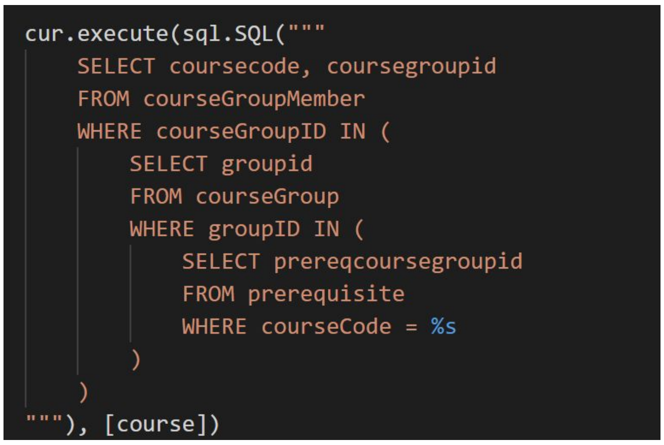
- [Jeff] Actions
    - Input cs348, shows the information about the course, the title, code, description, prerequisites. Also click open the prerequisite graph.
- Students can provide a desired course and the app will return the general description of the course, course title and course types. In addition, a list of prerequisites that need to be taken before enrolling in the course and the prerequisite graph can be shown as well

  Backend Query: The sql query for course general information is a select query from course table.



```
cur.execute(sql.SQL("""
    SELECT courseCode, title, courseTypes, description, subjectTitle
    FROM course
    WHERE courseCode = %s
"""), [course])
```

The prerequisites are obtained by finding all the courses in the coursegroupMember table by using the coursegroupId in the Prerequisite table.



```
cur.execute(sql.SQL("""
    SELECT coursecode, coursegroupid
    FROM courseGroupMember
    WHERE courseGroupID IN (
        SELECT groupid
        FROM courseGroup
        WHERE groupID IN (
            SELECT prereqcoursegroupid
            FROM prerequisite
            WHERE courseCode = %s
        )
    )
"""), [course])
```

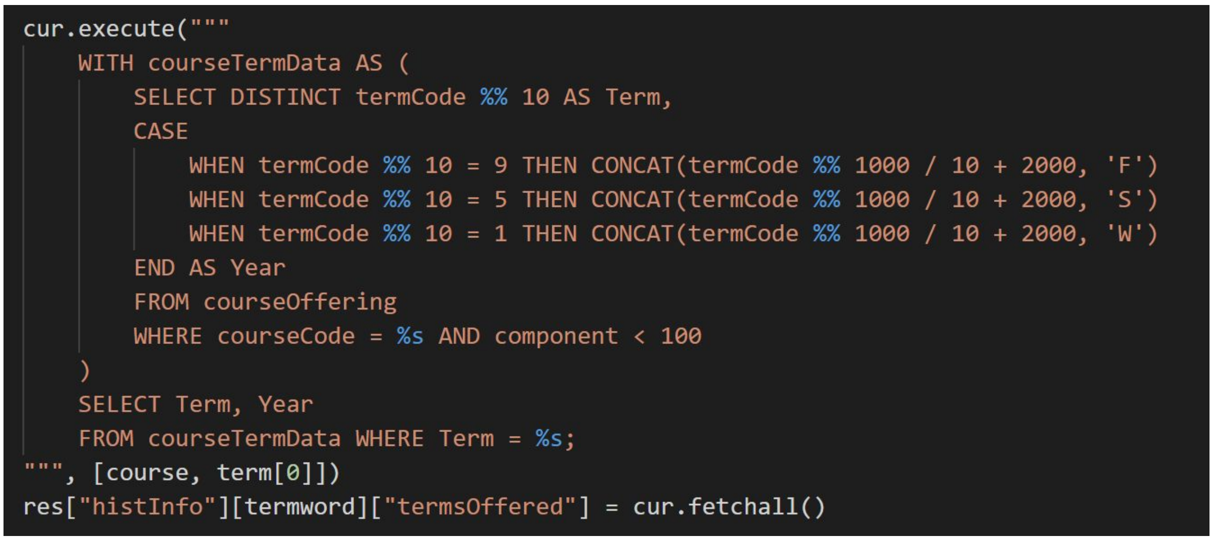
3. **Let's make sure my course won't be full!**

   [Jeff] Actions

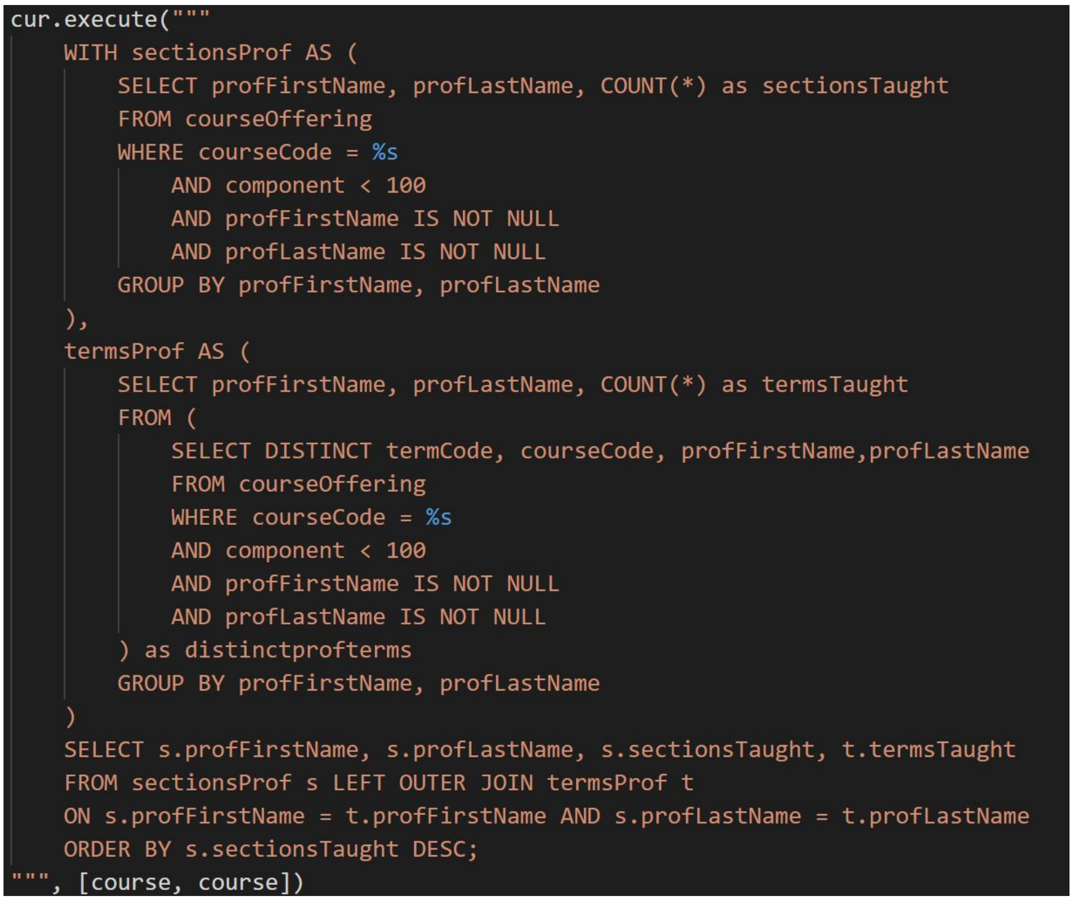
   - Click on historical data. This shows some historic information about this course.
   - Click on the term info tab.

- Students can see the historical data of a given course. The term tab displays the current term information and next information, and 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.

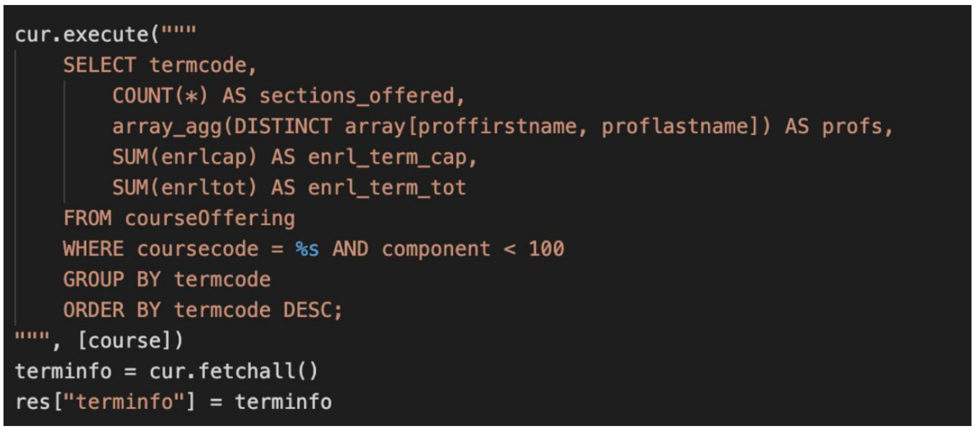
Backend Query (fancy)**:**



```
cur.execute("""
    WITH courseTermData AS (
        SELECT DISTINCT termCode %% 10 AS Term,
        CASE
            WHEN termCode %% 10 = 9 THEN CONCAT(termCode %% 1000 / 10 + 2000, 'F')
            WHEN termCode %% 10 = 5 THEN CONCAT(termCode %% 1000 / 10 + 2000, 'S')
            WHEN termCode %% 10 = 1 THEN CONCAT(termCode %% 1000 / 10 + 2000, 'W')
        END AS Year
        FROM courseOffering
        WHERE courseCode = %s AND component < 100
    )
    SELECT Term, Year
    FROM courseTermData WHERE Term = %s;
""", [course, term[0]])
res["histInfo"][termword]["termsOffered"] = cur.fetchall()
```

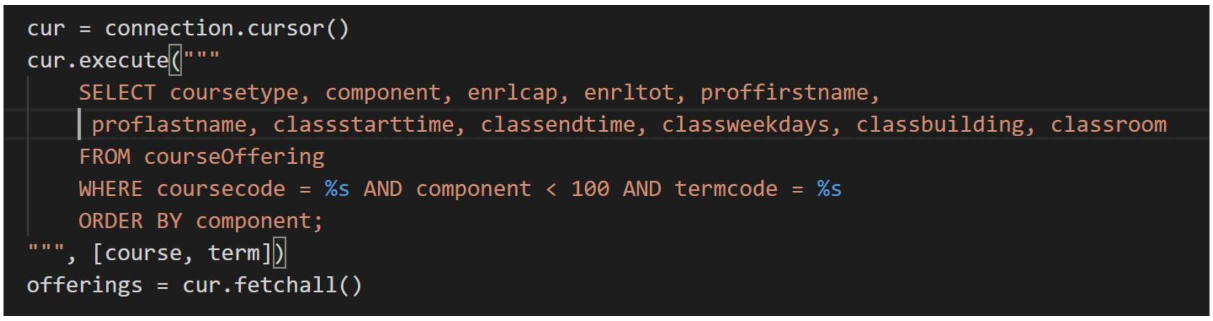


```
cur.execute("""
    WITH sectionsProf AS (
        SELECT profFirstName, profLastName, COUNT(*) as sectionsTaught
        FROM courseOffering
        WHERE courseCode = %s
            AND component < 100
            AND profFirstName IS NOT NULL
            AND profLastName IS NOT NULL
        GROUP BY profFirstName, profLastName
    ),
    termsProf AS (
        SELECT profFirstName, profLastName, COUNT(*) as termsTaught
        FROM (
            SELECT DISTINCT termCode, courseCode, profFirstName,profLastName
            FROM courseOffering
            WHERE courseCode = %s
            AND component < 100
            AND profFirstName IS NOT NULL
            AND profLastName IS NOT NULL
        ) as distinctprofterms
        GROUP BY profFirstName, profLastName
    )
    SELECT s.profFirstName, s.profLastName, s.sectionsTaught, t.termsTaught
    FROM sectionsProf s LEFT OUTER JOIN termsProf t
    ON s.profFirstName = t.profFirstName AND s.profLastName = t.profLastName
    ORDER BY s.sectionsTaught DESC;
""", [course, course])
```



```
cur.execute("""
    SELECT termcode,
        COUNT(*) AS sections_offered,
        array_agg(DISTINCT array[proffirstname, proflastname]) AS profs,
        SUM(enrlcap) AS enrl_term_cap,
        SUM(enrltot) AS enrl_term_tot
    FROM courseOffering
    WHERE coursecode = %s AND component < 100
    GROUP BY termcode
    ORDER BY termcode DESC;
""", [course])
terminfo = cur.fetchall()
res["terminfo"] = terminfo
```

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



```
cur = connection.cursor()
cur.execute("""
    SELECT coursetype, component, enrlcap, enrltot, proffirstname,
     proflastname, classstarttime, classendtime, classweekdays, classbuilding, classroom
    FROM courseOffering
    WHERE coursecode = %s AND component < 100 AND termcode = %s
    ORDER BY component;
""", [course, term])
offerings = cur.fetchall()
```

As for the SQL query, 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, array aggregations, and group by.

All four transactions are parts of this feature, and this is one of our fancy features.

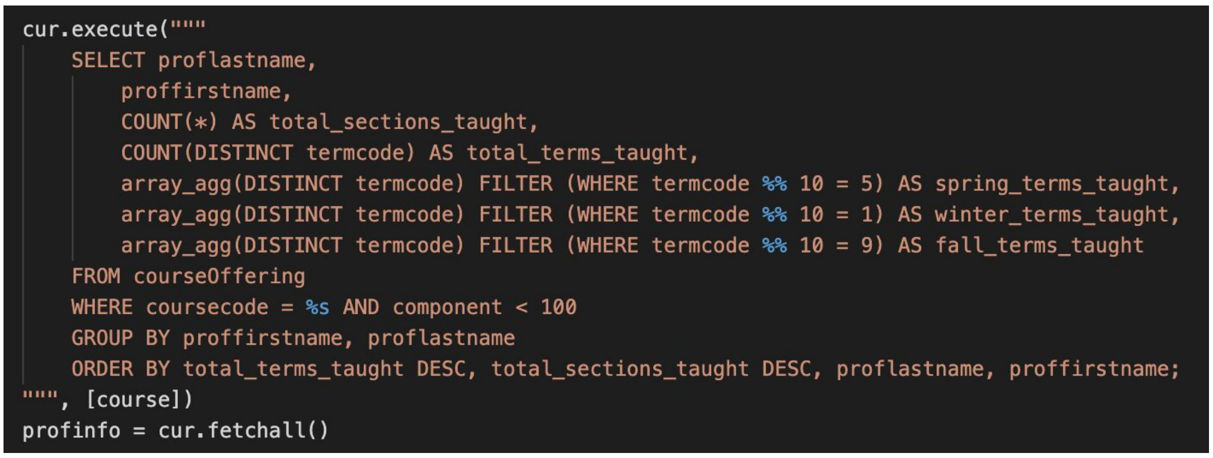
4. I wonder who are the professors that have been teaching this course?

- Brad
- Actions:

    - In the course search page, search for "CS 350". Go to the prof info tab. Click on a prof to show details about that professor.
- 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.

Backend Query:

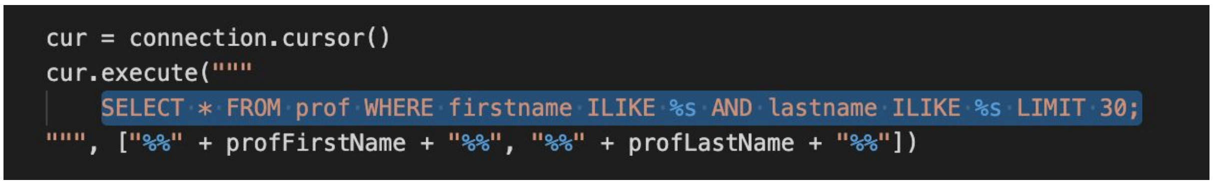
We are using a group by query on courseoffering by the professor's first and last name based on the coursecode. We then count the aggregated information to get the number of terms and sections they have taught using the postgres array_agg() function.



```
cur.execute("""
    SELECT proflastname,
        proffirstname,
        COUNT(*) AS total_sections_taught,
        COUNT(DISTINCT termcode) AS total_terms_taught,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 5) AS spring_terms_taught,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 1) AS winter_terms_taught,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 9) AS fall_terms_taught
    FROM courseOffering
    WHERE coursecode = %s AND component < 100
    GROUP BY proffirstname, proflastname
    ORDER BY total_terms_taught DESC, total_sections_taught DESC, proflastname, proffirstname;
""", [course])
profinfo = cur.fetchall()
```

5. **Professor XYZ is my favorite prof, I would like to find out what courses s/he is teaching!**
    - Brad
    - Actions:
        - In the prof search page, search for "lesl" for first name, and "i" for last name. In the list of professors, click "Lesley Ann Istead" and show the popup.
    - 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 been teaching. They search professors by a partial match in first name and last name, a list of professors that match this name is returned.
    - When we click on a professor’s name, a popup will show to display the RateMyprof ratings of this professor (should they have one) and the courses that she/he have been teaching and during what year and term the professor taught these courses.

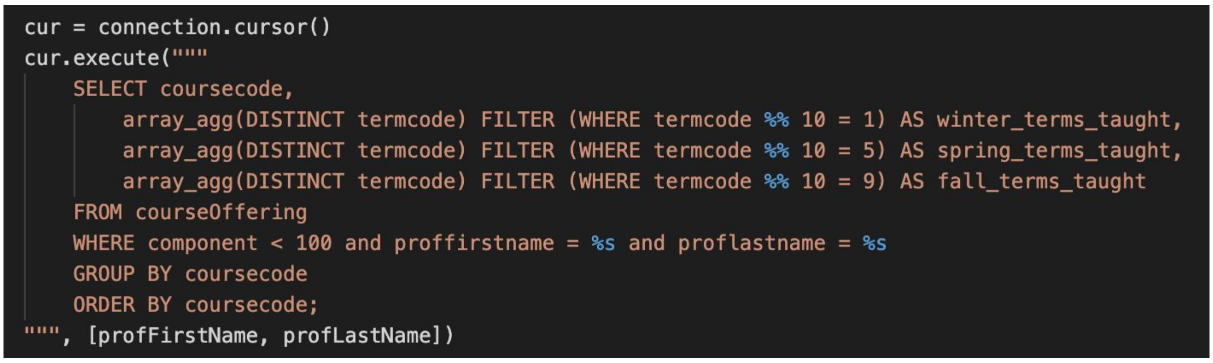
Backend Query (Fancy)**:**



```
cur = connection.cursor()
cur.execute("""
    SELECT * FROM prof WHERE firstname ILIKE %s AND lastname ILIKE %s LIMIT 30;
""", ["%%" + profFirstName + "%%", "%%" + profLastName + "%%"])
```

This is one of our fancy features that includes complex setups including views and indexes. We created the following 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.



```
cur = connection.cursor()
cur.execute("""
    SELECT coursecode,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 1) AS winter_terms_taught,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 5) AS spring_terms_taught,
        array_agg(DISTINCT termcode) FILTER (WHERE termcode %% 10 = 9) AS fall_terms_taught
    FROM courseOffering
    WHERE component < 100 and proffirstname = %s and proflastname = %s
    GROUP BY coursecode
    ORDER BY coursecode;
""", [profFirstName, profLastName])
```

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

Robbie

- Actions:
    - Add a course in `Add New Course` then go to `Search Course` to search 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 as well as prerequisites. When students search for courses, they will be able to see the newly added courses if they are relevant.

Backend Query: 6 INSERT INTO statements (inserting into various tables and maintaining foreign key constraints and data integrity)

System Support

System support for your application and briefly explain the back-end queries for each feature [6 points]

Brad

In terms of system support, this application uses Angular web framework for frontend, Flask library in Python with REST endpoints for backend, and PostgreSQL database server. We use psycopg2 to access the SQL database. The server and the database are hosted on GCP, and the frontend is hosted on Netlify.

The contribution of each member [2 points]

Robbie

Brad - infrastructure, backends, SQL scripts, data scraping for school info, course/offering data, RateMyProf, UW OpenAPI

Jeffer - frontend for many features, backend & SQL queries

James - course info scraping, ETL on UW OpenAPI data, backend & sql

Robbie - data scraping, frontend & backend for various features

Summary [2 points]

James

Overall, this was a great learning experience for us. We were able to apply new techniques we learned in this course, while creating something that will be useful to us and our fellow classmates in the future. We will be able to plan out our degrees, quickly understand course prerequisite paths, and compare different course offerings in greater detail than we ever could before.

This project was also extremely educational. Not only did we improve our existing skills in team-work, SQL, and web development, but we also got to develop new skills we learned in CS 348, such as normalizing data schemas, recursive SQL, and creating indexes and constraints to enforce data integrity.

Overall, we learned a lot and we hope you enjoyed watching this demo as much as we enjoyed making it.

Submit your finalized members.txt, report.pdf and code.zip under “Project Final”