CS 4380 – Final Project

Team 8 | May 10, 2018

MedOpps Advising Management System

Tony vazquez, Zandria Johnson, Taylor McNeal, Camille Frazier

Contents

[Introduction 1](#_Toc513758484)

[Entity-Relations Diagram 2](#_Toc513758485)

[Queries 10](#_Toc513758486)

[Analytics 15](#_Toc513758487)

[Normalization 19](#_Toc513758488)

[Indexing Selection 20](#_Toc513758489)

[Optimization & Tuning 21](#_Toc513758490)

[Security Settings 22](#_Toc513758491)

[User’s Manual 23](#_Toc513758492)

[Conclusion & Final Thoughts 30](#_Toc513758493)

This page is intentionally left blank.

Introduction

Our client is the MedOpp advising office, an on-campus resource offering pre-professional advising for pre-health professions students at the University of Missouri. The office adds about 100 students each academic year, and offers support in the form of seminars, workshops, interviews, handling of letters of recommendation, advice, and much more.

Our project laid the fundamentals down for a database driven system to house the office’s student data, tracking their progress and participation in the program, along with a clean and simple UI. Student data ranges from demographics, such as name and ethnicity, to their work and research experience or what schools they have applied to. A more in depth explanation is below.

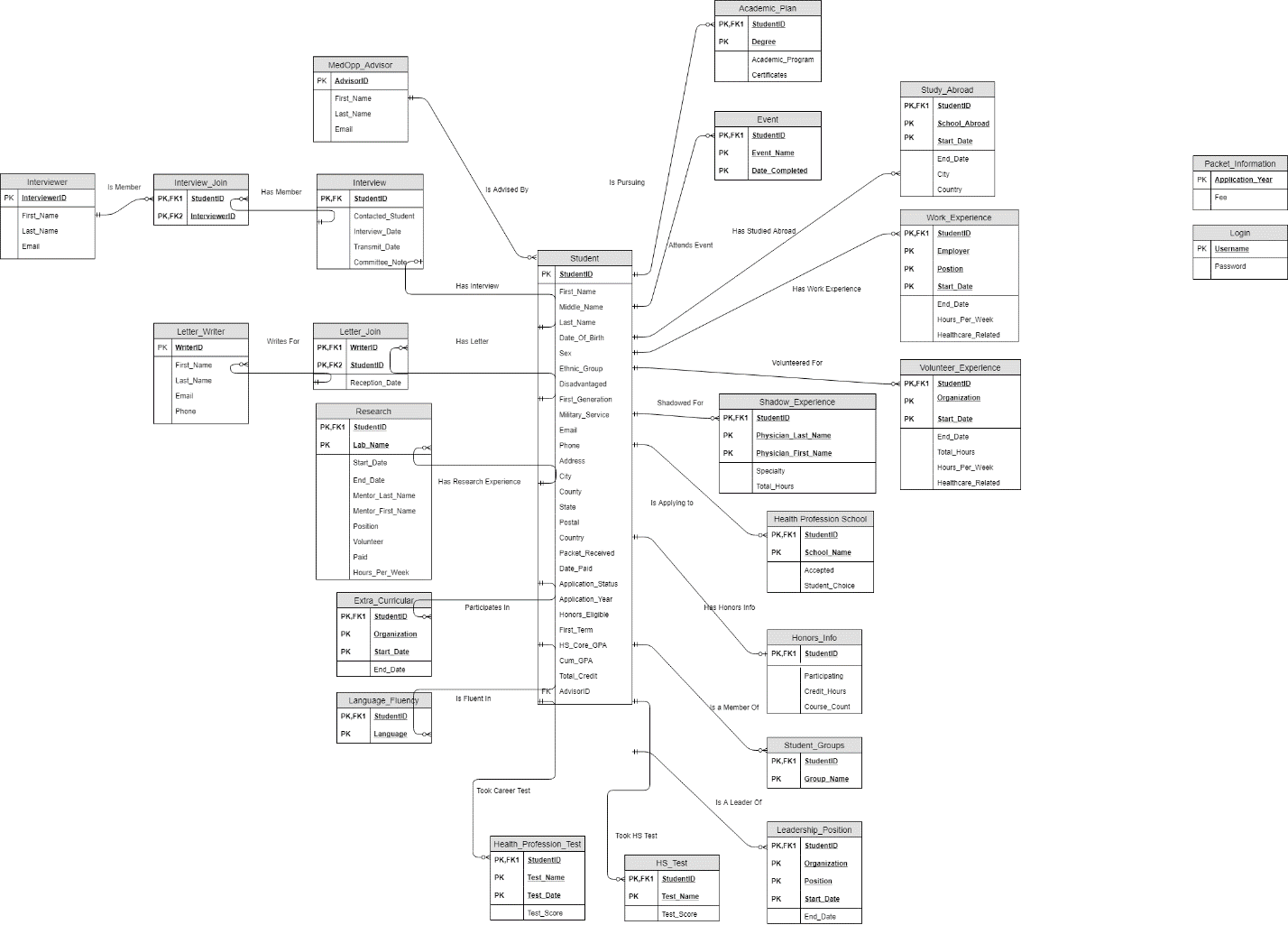
Our client has made it clear that this system would have to be built exclusively for them. They do not want other medical advising programs gleaning details of their advising process by analyzing our system.

<https://web.dsa.missouri.edu/~s18group08/>

We only have one type of user, the advisor themselves.

Their credentials are username = “test” and password = “pass”.

Entity-Relations Diagram



CREATE TABLE MedOpp\_Advisor (

AdvisorID int,

First\_Name varchar(255),

Last\_Name varchar(255),

Email varchar(255),

PRIMARY KEY (AdvisorID)

);

This table holds the MedOpp advisors and their email addresses.

CREATE TABLE Student (

StudentID varchar(10),

First\_Name varchar(255),

Middle\_Name varchar(255),

Last\_Name varchar(255),

AdvisorID int NOT NULL,

Email varchar(255),

Phone varchar(20),

Date\_Of\_Birth DATE,

Sex varchar(25),

Ethnic\_Group varchar(32),

Disadvantaged varchar(3),

First\_Generation varchar(3),

Military\_Service varchar(3),

Address varchar(255),

City varchar(255),

State varchar(255),

County varchar(255),

Postal varchar(32),

Country varchar(255),

Application\_Year varchar(4),

Packet\_Received varchar(3),

Date\_Paid DATE,

First\_Term varchar(6),

Application\_Status varchar(64),

HS\_Core\_GPA decimal(4,3),

Cum\_GPA decimal(4,3),

Total\_Credit int,

Honors\_Eligible varchar(3),

PRIMARY KEY (StudentID),

FOREIGN KEY (AdvisorID) REFERENCES MedOpp\_Advisor(AdvisorID)

);

This is the backbone table holding student demographic and academic data. The foreign key references their MedOpp advisor, and is required as all students will have an advisor no matter their level of participation in the program.

CREATE TABLE Interview (

StudentID varchar(10),

Contacted\_Student varchar(3),

Interview\_Date DATE,

Transmit\_Date DATE,

Committee\_Note TEXT,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID)

);

This table contains information on student interviews, which are conducted by the MedOpp office. If a student has an interview, it will be unique to them, so the foreign and primary keys both reference their student ID from the student table.

CREATE TABLE Interviewer (

InterviewerID SERIAL,

First\_Name varchar(255),

Last\_Name varchar(255),

Email varchar(255),

PRIMARY KEY (InterviewerID)

);

This table contains information on the individuals who conduct the above mentioned interviews.

CREATE TABLE Interview\_Join (

StudentID varchar(10),

InterviewerID int,

FOREIGN KEY (StudentID) REFERENCES Interview(StudentID) ON DELETE CASCADE,

FOREIGN KEY (InterviewerID) REFERENCES Interviewer(InterviewerID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, InterviewerID)

);

This table joins students and interviewers based on foreign keys pulled from the interviewer and interview tables.

CREATE TABLE Letter\_Writer (

WriterID SERIAL,

First\_Name varchar(255),

Last\_Name varchar(255),

Email varchar(255),

Phone varchar(20),

PRIMARY KEY (WriterID)

);

This table contains information on the individuals who write recommendation letters for the students.

CREATE TABLE Letter\_Join (

WriterID int,

StudentID varchar(10),

Reception\_Date DATE,

FOREIGN KEY (WriterID) REFERENCES Letter\_Writer(WriterID) ON DELETE CASCADE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (WriterID, StudentID)

);

This table joins students with their writers based on foreign keys pulled from the student and letter writer tables.

CREATE TABLE Health\_Profession\_Test (

StudentID varchar(10),

Test\_Name varchar(10),

Test\_Date DATE,

Test\_Score int,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Test\_Name, Test\_Date)

);

This table contains information on the health profession tests a student takes before applying to health profession schools. The foreign key links these tests back to the student.

CREATE TABLE HS\_Test (

StudentID varchar(10),

Test\_Name varchar(10),

Test\_Score int,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Test\_Name)

);

This table contains information on the high school tests a student takes before applying to undergraduate college. The foreign key links these tests back to the student.

CREATE TABLE Academic\_Plan (

StudentID varchar(10),

Degree varchar(16),

Academic\_Program varchar(16),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Degree)

);

This table contains information on the academic degree(s) a student is pursuing in their undergraduate studies. Degree name can denote both majors and minors. The foreign key links these degree paths back to the student.

CREATE TABLE Event (

StudentID varchar(10),

Event\_Name varchar(128),

Date\_Completed DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Event\_Name, Date\_Completed)

);

Events are regularly hosted by the MedOpp office, and this table keeps track of student attendance. The foreign key links the event to the student.

CREATE TABLE Health\_Profession\_School (

StudentID varchar(10),

School\_Name varchar(255),

Accepted varchar(16),

Student\_Choice varchar(3),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, School\_Name)

);

The end goal for these students is to be accepted to a health profession school, and this table keeps track of where they are applying, whether or not they were accepted, and their choice of school. The foreign key links the schools back to the student.

CREATE TABLE Honors\_Info (

StudentID varchar(10),

Participating varchar(3),

Credit\_Hours int,

Course\_Count int,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID)

);

This table tracks student participation in the honors college if applicable. The foreign key links the honors info back to the student.

CREATE TABLE Student\_Groups (

StudentID varchar(10),

Group\_Name varchar(128),

Start\_Date DATE,

End\_Date DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Group\_Name)

);

This table tracks what student groups on campus the student is a part of, with the foreign key linking the participation back to the student.

CREATE TABLE Research (

StudentID varchar(10),

Lab\_Name varchar(255),

Start\_Date DATE,

End\_Date DATE,

Mentor\_Last\_Name varchar(255),

Mentor\_First\_Name varchar(255),

Position varchar(255),

Volunteer varchar(3),

Hours\_Per\_Week int,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Lab\_Name)

);

This table tracks academic research the student is performing, with the foreign key linking the research back to the student.

CREATE TABLE Extra\_Curricular (

StudentID varchar(10),

Organization varchar(255),

Start\_Date DATE,

End\_Date Date,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Organization, Start\_Date)

);

This table contains student extracurricular participation, with the foreign key linking back to the student.

CREATE TABLE Language\_Fluency (

StudentID varchar(10),

Language varchar(255),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Language)

);

This table tracks what languages the student is fluent in, with the foreign key linking back to the student.

CREATE TABLE Leadership\_Position (

StudentID varchar(10),

Organization varchar(255),

Position varchar(255),

Start\_Date DATE,

End\_Date DATE,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Organization, Position, Start\_Date)

);

This table tracks what leadership positions the student has held, and the foreign key links this and the student record.

CREATE TABLE Shadow\_Experience (

StudentID varchar(10),

Physician\_Last\_Name varchar(255),

Physician\_First\_Name varchar(255),

Specialty varchar(255),

Total\_Hours int,

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Physician\_Last\_Name, Physician\_First\_Name)

);

This table tracks job shadow experience, and the foreign key links back to the student.

CREATE TABLE Volunteer\_Experience (

StudentID varchar(10),

Organization varchar(255),

Start\_Date DATE,

End\_Date DATE,

Total\_Hours int,

Hours\_Per\_Week int,

Healthcare\_Related varchar(3),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Organization, Start\_Date)

);

This table tracks volunteer experience, and the foreign key links back to the student.

CREATE TABLE Work\_Experience (

StudentID varchar(10),

Employer varchar(255),

Position varchar(255),

Start\_Date DATE,

End\_Date DATE,

Hours\_Per\_Week int,

Healthcare\_Related varchar(3),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, Employer, Position, Start\_Date)

);

This table tracks work experience, and the foreign key links back to the student.

CREATE TABLE Study\_Abroad (

StudentID varchar(10),

School\_Abroad varchar(255),

Start\_Date DATE,

End\_Date DATE,

City varchar(255),

Country varchar(255),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID) ON DELETE CASCADE,

PRIMARY KEY (StudentID, School\_Abroad, Start\_Date)

);

This table tracks any study abroad experience, and the foreign key links back to the student.

CREATE TABLE Packet\_Information (

Application\_Year varchar(4),

Fee decimal(4,2),

PRIMARY KEY (Application\_Year)

);

The MedOpp office charges a small fee for final record processing that goes out to health professional schools. This can change over the years, so this table tracks fee price.

CREATE TABLE Login (

Username varchar(255),

Password varchar(255),

PRIMARY KEY (Username)

);

This table contains the login information for users.

Queries

1. Select students that weren’t present for a given event.

This helps the advisors keep records on which of their students are actually going to which events. This is useful for accountability of the students.

SQL:

SELECT StudentID, Last\_Name, First\_Name

FROM Student

WHERE StudentID NOT IN (

SELECT S.StudentID

FROM Student S, Event E

WHERE E.StudentID = S.StudentID AND E.Event\_Name = ?)

ORDER BY Last\_Name ASC

2. Select students of a specific ethnicity.

This helps give advisors an idea of the diversity of the program. There are often quotas to enforce diversity in university organizations and this helps give insight into that.

SQL:

SELECT StudentID, Last\_Name, First\_Name FROM Student WHERE Ethnic\_Group = ? ORDER BY Last\_Name ASC

3. Select all of the students from a specific application year.

This allows the advisors to segregate students based on when they applied. This gives them insight into the volume of students applying and the way that changes with respect to different academic years.

SQL:

SELECT \* FROM Student WHERE Application\_Year = ?

4. Select which students belong to a specific advisor.

This helps advisors remain clear on which students fall under which advisors jurisdiction. This is helpful in a system with hundreds of students where it may be hard to keep track of which advisor is assigned to which student.

SQL:

SELECT \* FROM Student WHERE AdvisorID = ? ORDER BY Last\_Name ASC

5. Check which advisors have outstanding letters they need to write for students.

Letter writers are often responsible for writing letters for multiple students and because letters have deadlines, this helps advisors keep tabs on which writers still need to submit a letter for a student.

SQL:

SELECT LW.First\_Name as label, COUNT(\*) as y

From Letter\_Writer AS LW JOIN Letter\_Join AS LJ ON LW.WriterID = LJ.WriterID

WHERE LJ.Reception\_Date IS NULL

GROUP BY LW.WriterID

ORDER BY LW.Last\_Name Asc

6. Find students who also have jobs as well as the average time they spend working per week.

This helps the advisors understand how many of their students have obligations outside of school and to what extent that will impede time they can spend on other things like academics and MedOpp events.

SQL:

SELECT StudentID, First\_Name, Last\_Name FROM Student WHERE StudentID IN

(SELECT DISTINCT StudentID FROM Work\_Experience) ORDER BY Last\_Name ASC

SELECT DISTINCT COUNT(StudentID), AVG(Hours\_Per\_Week) FROM Work\_Experience

7. Which students have served in the military.

As advisors, helping with scholarships is important and having knowledge of who has served in the military can help identify which students are available for different benefits.

SQL:

SELECT StudentID, First\_Name, Last\_Name FROM Student WHERE Military\_Service = 'Yes' ORDER BY Last\_Name Asc

8. Find the students who haven’t paid their dues yet.

Dues must be paid by all members of the MedOpp Advising program and keeping records of who hasn’t paid yet helps ensure that all students are paying what they need to.

SQL:

SELECT StudentID, First\_Name, Last\_Name FROM Student WHERE Date\_Paid IS NULL ORDER BY Last\_Name Asc

9. Find the students who have been accepted to health profession schools but have not reported their choice of school.

Students will need to make a decision on where they’re going and students who have been accepted to schools without a choice need to be reminded to make that choice by advisors. This allows them to quickly identify those students.

SQL:

SELECT HPS.studentID, S.First\_Name, S.Last\_Name

FROM Student AS S JOIN Health\_Profession\_School AS HPS ON S.studentID = HPS.studentID

WHERE EXISTS

(SELECT HPS.studentID FROM Health\_Profession\_School AS HPS

WHERE HPS.Accepted = 'Yes')

AND HPS.studentID NOT IN (

SELECT HPS.studentID FROM Health\_Profession\_School AS HPS

WHERE HPS.Student\_Choice = 'Yes' OR HPS.Student\_Choice= 'No')

penis

10. Find first generation students who have been accepted to a health profession school.

Getting admitted to a health profession school as a first generation student is a big achievement for students and often makes them eligible for scholarships. Advisors need to know this to help their students receive benefits.

SQL:

SELECT S.StudentID, S.Last\_Name, S.First\_Name

FROM Student AS S JOIN Health\_Profession\_School AS HPS ON S.studentID = HPS.studentID

WHERE HPS.Accepted = 'Yes' AND S.First\_Generation = 'Yes'

11. Students who have not completed an interview yet.

Interviewing with a committee is a very important part of the MedOpp advising process and keeping track of students who haven’t completed this yet helps advisors remind students of the requirement.

SQL:

SELECT S.StudentID, S.Last\_Name, S.First\_Name, I.Interview\_Date

FROM Student AS S JOIN Interview AS I ON S.StudentID = I.StudentID

WHERE interview\_date IS NULL OR interview\_date > current\_date

12. List the students enrolled in the honors college.

The honors college adds a layer of difficulty to the premed process and keeping track of which students are in the honors college allows advisors to more efficiently gauge which students may struggle with their academic workload.

SQL:

SELECT S.StudentID, S.Last\_Name, S.First\_Name

FROM Student AS S JOIN Honors\_Info AS HI ON S.studentID = HI.studentID

WHERE Participating = 'Yes'

13. View the average test scores of students.

This gives the advisors insight into how students in the MedOpp advising programming are doing on health profession tests.

SQL:

SELECT test\_name as name, to\_char(AVG(test\_score), 'FM999999999.00') AS y, COUNT(\*) as z

FROM Health\_Profession\_Test

GROUP BY test\_name

14. View the health profession schools that have been accepting students from the program.

This gives advisors insight into what schools are accepting students so they can aid students in choosing schools that admit students who have been through the same program.

SQL:

SELECT school\_name as label, COUNT(\*) as y

FROM (SELECT school\_name FROM Health\_Profession\_School WHERE Accepted = 'Yes') AS Yes

GROUP BY Yes.school\_name

15. View the total number of credit hours each student has completed.

This allows advisors to monitor the progress students have made in their academics. It also gives them information on the amount of credits they’ve received from the honors college.

SQL:

SELECT Student.StudentID, Last\_Name, First\_Name, Total\_Credit, Credit\_Hours

FROM Student JOIN Honors\_Info ON Student.StudentID = Honors\_Info.StudentID

16. View the degrees students are working towards.

This gives an idea of the breakdown on how many students are studying towards different degrees. Advisors can then have the insight of what disciplines their students are progressing towards.

SQL:

SELECT degrees.degree, COUNT(\*) FROM (SELECT degree FROM Academic\_Plan

WHERE degree != ' ' AND degree IS NOT NULL) AS degrees GROUP BY degrees.degree

17. View all disadvantaged students.

As advisors, knowing which students are disadvantaged helps them understand which students may be more inclined to need aid to help them through their college experience.

SQL:

SELECT StudentID, Last\_Name, First\_Name FROM Student WHERE disadvantaged = 'Yes'

18. See which students have taken a leadership position.

This helps advisors identify which of their students have become involved as leaders on campus. When applying for jobs and prospective positions advisors with this knowledge will have a talking point to help their students achieve their ambitions.

SQL:

SELECT S.StudentID, Last\_Name, First\_Name, Position, Organization

FROM Student AS S JOIN Leadership\_Position AS LP ON S.StudentID = LP.StudentID

19. See which students in the program speak a foreign language.

Advisors help students who wish to study abroad and knowing which students speak foreign languages helps advisors identify which students may want to study overseas without having to tackle a language barrier.

SQL:

SELECT \* FROM Student WHERE StudentID IN

(SELECT StudentID FROM Language\_Fluency

WHERE language NOT LIKE '%English%' AND language != ' ')

mk20. Find students who have research experience.

Experience is crucial in being accepted for a wide variety of positions and this gives advisors insight on which of their students has experience working in a lab setting.

SQL:

SELECT \* FROM Student S JOIN Research R ON S.StudentID = R.StudentID

Analytics

Query 1:

$sqlWriters = 'SELECT LW.First\_Name as label, COUNT(\*) as y

From Letter\_Writer AS LW JOIN Letter\_Join AS LJ ON LW.WriterID = LJ.WriterID

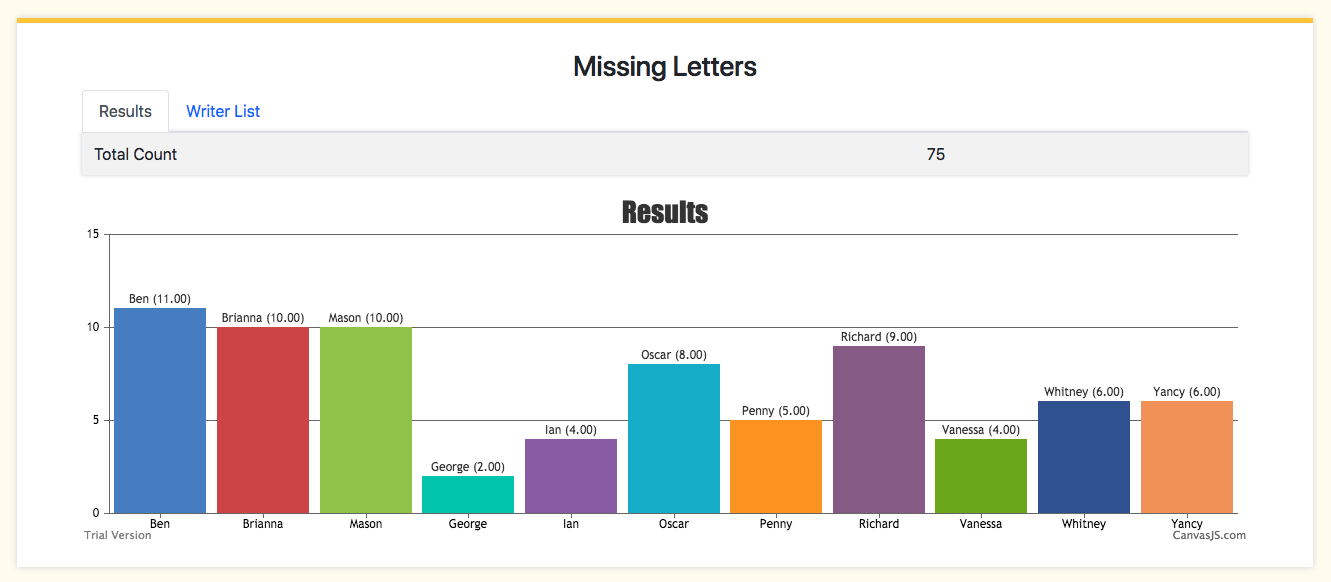
WHERE LJ.Reception\_Date IS NULL

GROUP BY LW.WriterID

ORDER BY LW.Last\_Name Asc';

Natural language = For each letter writer, show how many students have not turned in a letter

Importance: Each student needs a letter, advisors can see what letter writers they should focus on more for encouraging their students to get the letters.

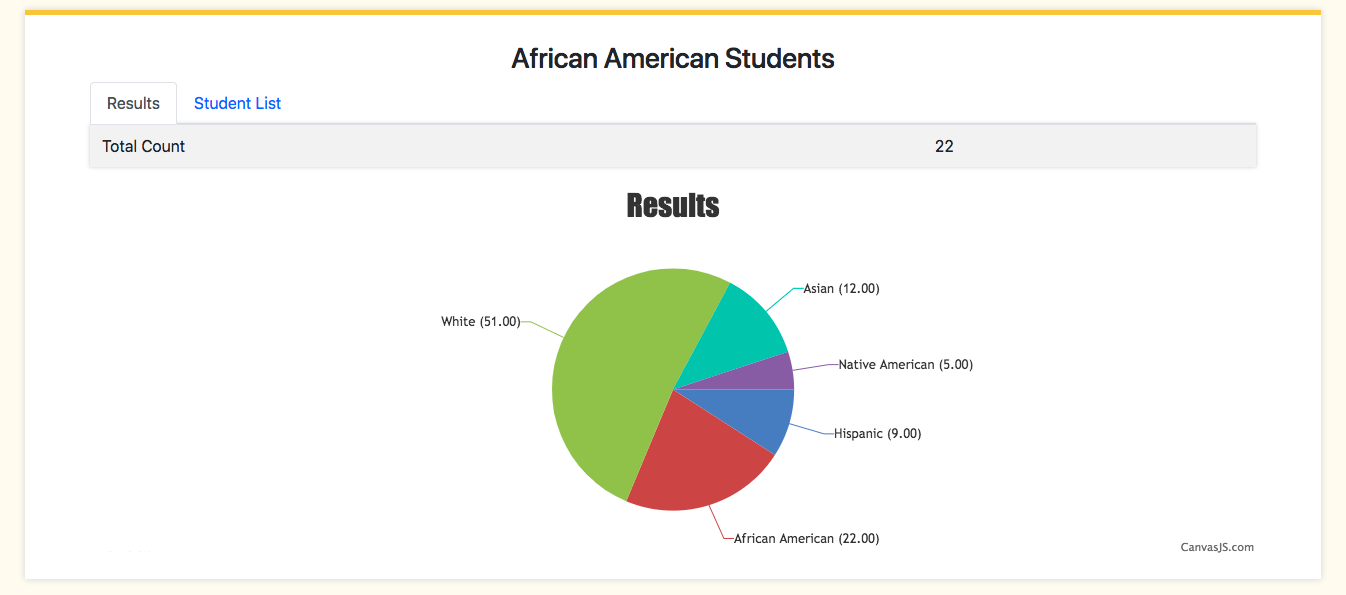


Query 2:

$sqlEthnic = "SELECT COUNT(ethnic\_group) as y, ethnic\_group as label FROM Student GROUP BY ethnic\_group";

Natural language = Show how many students are in each ethnic group

Importance: Diversity is always important. An easy way to track this is by looking at this graph for the ethnic groups inputted from students on the application.

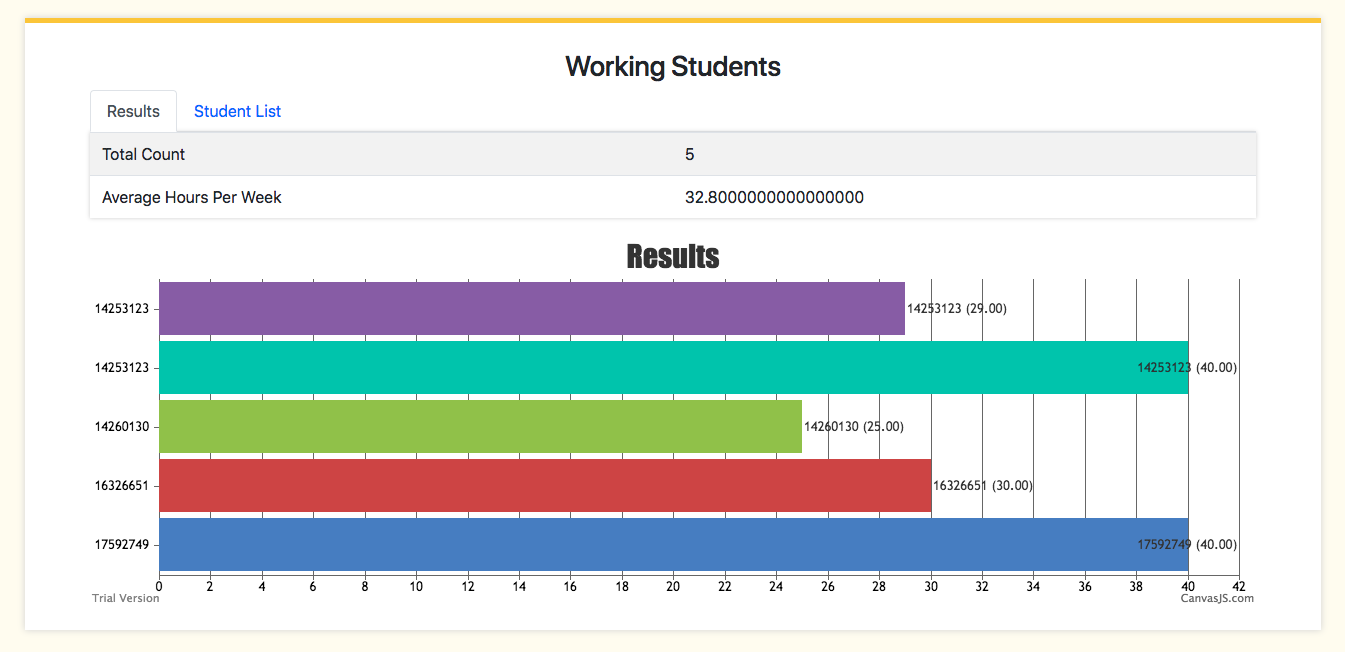


Query 3:

$sqlWorking = 'SELECT DISTINCT StudentID as label, Hours\_Per\_Week as y FROM Work\_Experience';

Natural language = What students are working? What is the Average Hours of all the students?

Importance = This is one of the activities done outside of school, in addition to organizations and leadership positions. We want to see how many hours each student is putting towards work as opposed to others and we can use that to see if it is effecting their progress as a student.



Query 4:

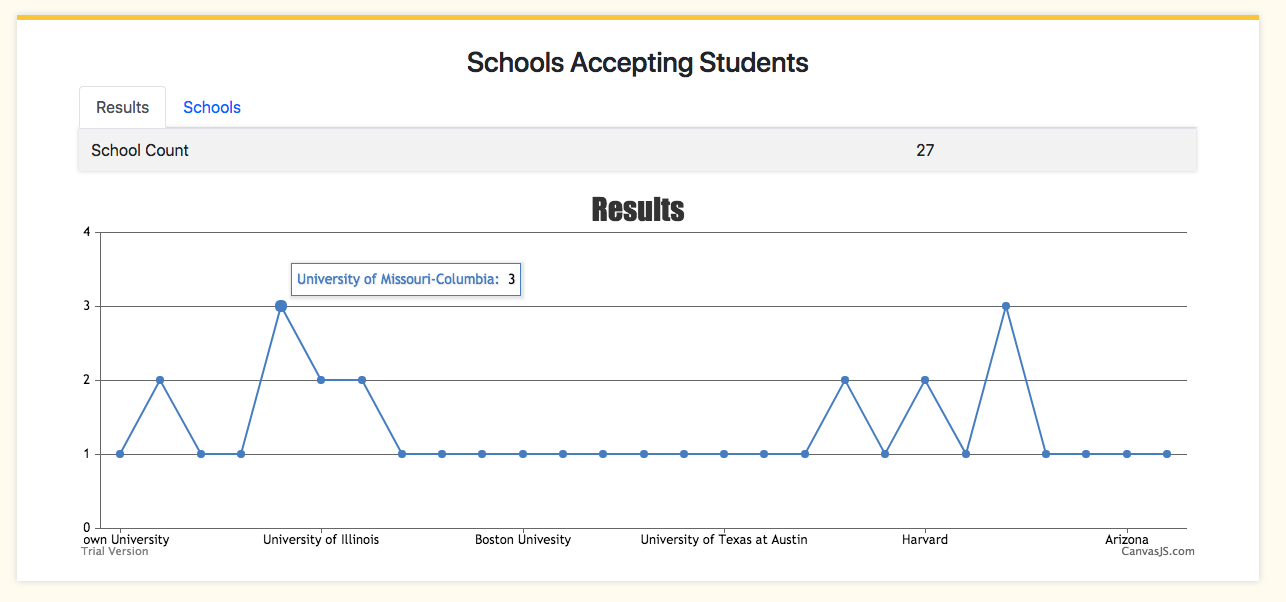
$sql = "SELECT school\_name, COUNT(\*)

FROM (SELECT school\_name FROM Health\_Profession\_School WHERE Accepted = 'Yes') AS Yes

GROUP BY Yes.school\_name";

Natural language = For each health profession school that students are going to, show how many students are going to that school

Importance: We want to see the success rate for our students, as well as where our program and advising is leading our students. For example, Mizzou Engineering can say that a lot of students find opportunity at Cerner, Garmin or Boeing.



Query 5:

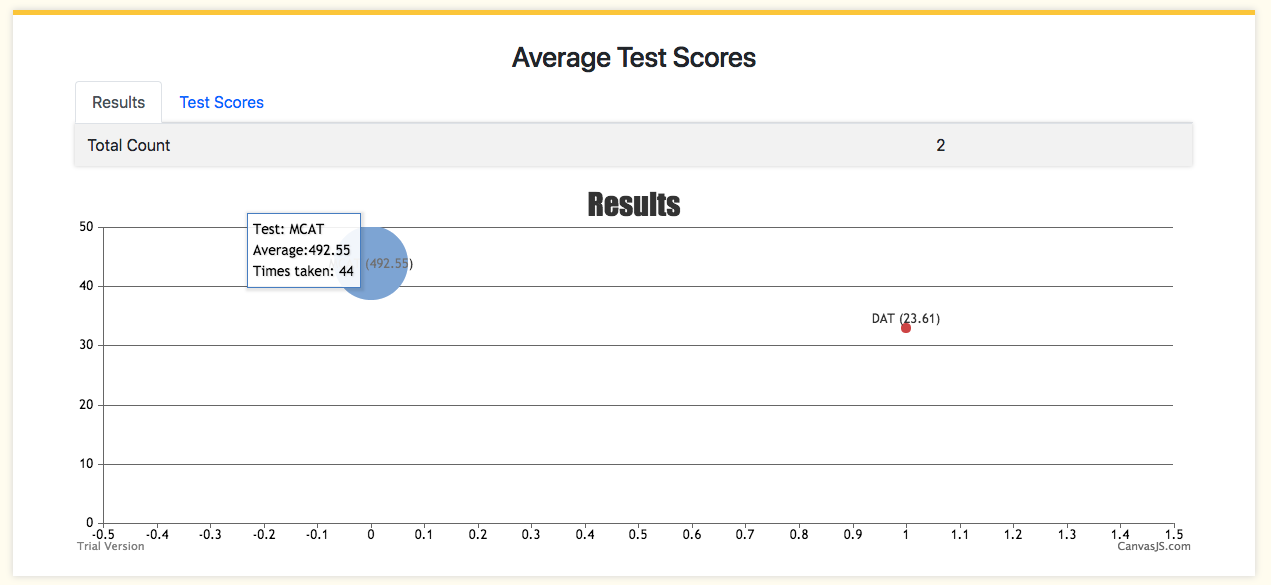
$sql = "SELECT test\_name, to\_char(AVG(test\_score), 'FM999999999.00') AS average, COUNT(\*)

FROM Health\_Profession\_Test

GROUP BY test\_name";

Natural language = Show the different tests (MCAT, DAT), how many times they have been taken, and the average score from the students

Importance: The advising office wants to know how well the students in the program are doing. This can be used to improve test scores in the future.



Normalization

Excepting the MedOpp advisor table, all tables directly related to the student table are optional, but need to be unique to avoid duplication. To do this, primary keys were created composed of the student ID grabbed from the student table, and a collection of fields that uniquely identify the record from other records in the same table. This is especially important as students can fill multiple rows of a single table, but they need to be unique.

An added benefit of using the student ID as part of the primary key of these extraneous tables is that no record will ever be inserted into these tables without a corresponding student ID present in the student table. With cascading deletions, if the student ID is ever removed from the student table, it is also removed from the extraneous tables as that correspondence is no longer preserved.

Indexing Selection

Indexing speeds up the sorting of data in regards to a certain field or collection of fields. By placing an index on ethnic group, the query we setup to identify the percentage breakdown of students belonging to each ethnic group is able to run much faster by presorting the information before the query is run.

CREATE INDEX Ethnicity\_Index

ON Student (Ethnic\_Group);

Optimization & Tuning

Originally the interview table had a serial ID, with that ID being a foreign key in the student and interview join tables. What we realized was that each student has a unique interview to begin with, and the interview table having its own ID was trivial and would make queries more complicated. For example, to link the interviewers and the student would require a join, as the interview join table was using the IDs of the interviewers and the individual ID of the interview itself which would need to be joined to the student table on that interview ID to find the student information. Since students have unique interviews, it made sense to make the ID of the interview table, and by extension the foreign key of the join table, just be the student ID itself so this join would not have to performed every time.

Another tuning method we came up with involved updating the records. All information from all tables relating to the student is displayed when the student’s individual record is pulled up. When the user updates records, we have no way of knowing which tables are actually being updated, and rather than going through the new complicated code to update each table, we use the same method to submit new forms and update forms. We simply check if the form submission is an update, and if it is, we delete the student record associated with that ID, which thanks to ON DELETE CASCADE specifications on all the tables related to the student also wipes those tables clean of that student. We then proceed to insert the form data, which contains all the information that was deleted with whatever updates were submitted, as a “new” form.

Security Settings

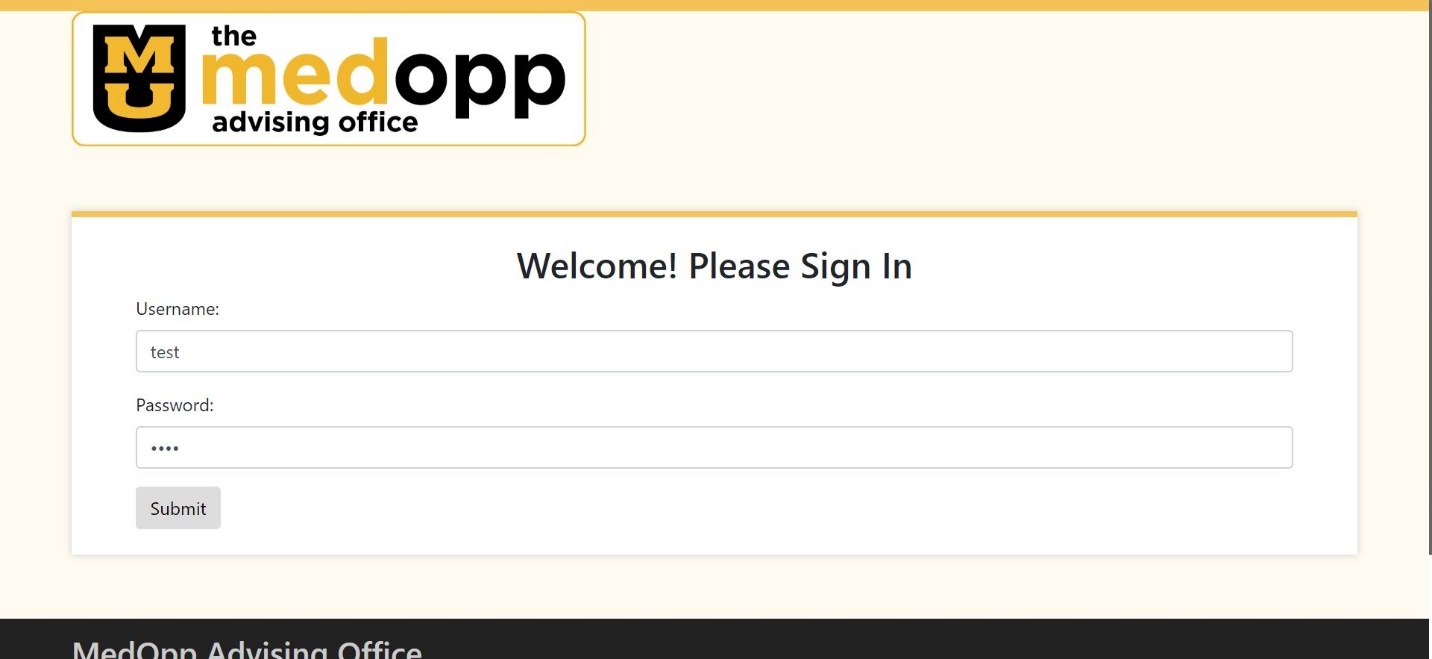
Since the only user of our system would be the advisors themselves, no restrictions were placed on their accounts. We have implemented a login system, however, that will not allow access to any part of the web application without a valid login.

Every query to the database was performed using prepared statements. Before variables are used in queries they are first trimmed of leading whitespace. If the result would be empty strings, the values were instead changed to NULL to avoid complications with data types such as DATE not accepting empty strings. The leading whitespace removal also has the benefit of removing spaces that may have accidentally been placed before the text in the form field.

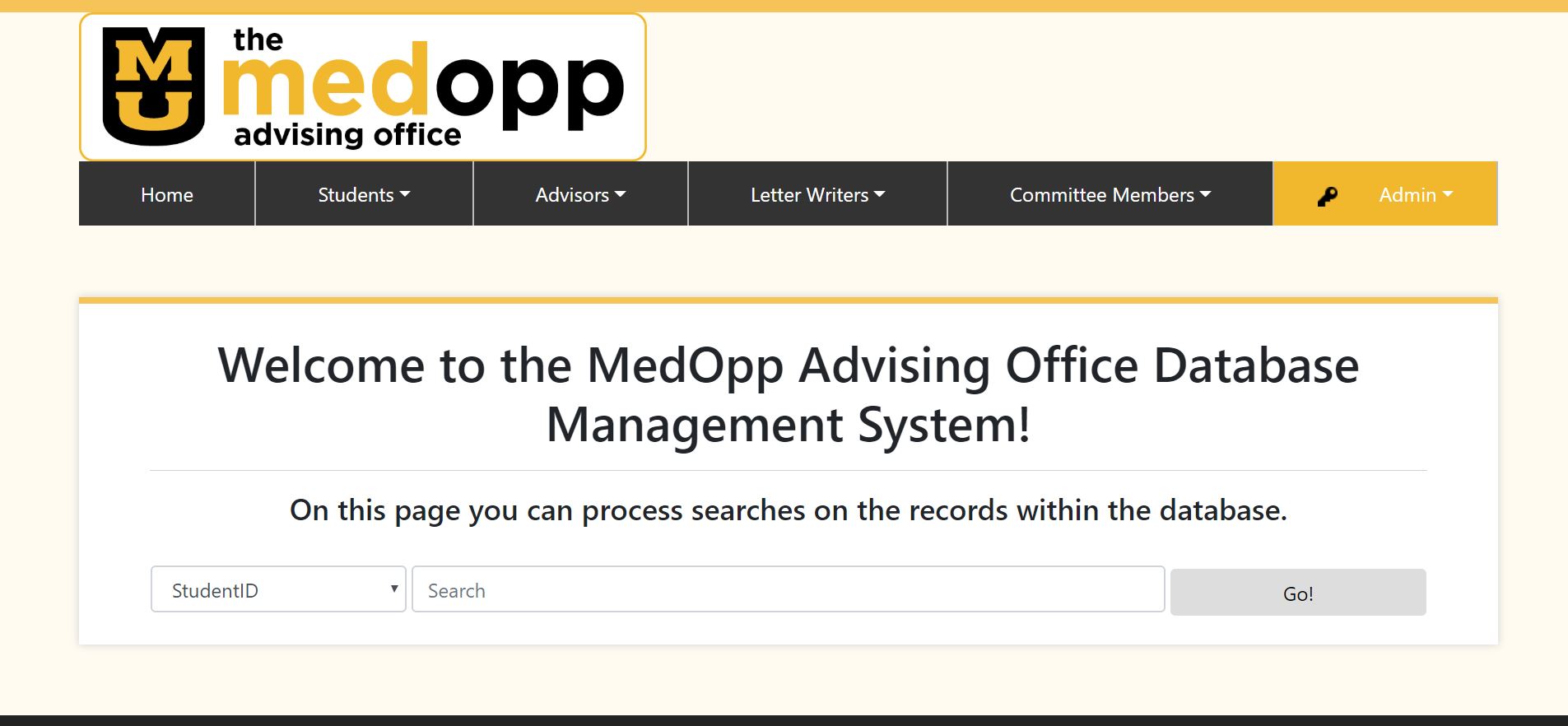
One final security consideration was the use of POST vs GET form data submission. All interactions that communicated sensitive student data, such as form submissions or even searches based on name or student ID, were performed using the POST method, as GET appends the data to the URL, which is unacceptable when dealing with sensitive information.

User’s Manual

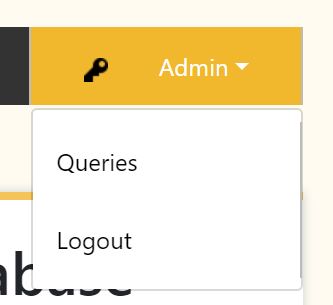
The user starts their session by logging in to the website with their credentials; for the purposes of demonstration, we use the username ‘test’ and the password ‘pass’ as our credentials.



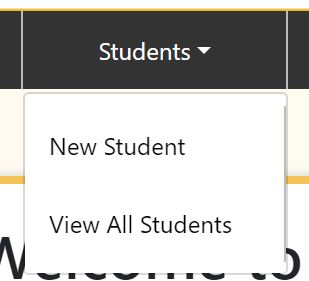
The user will then be directed to the home screen, which will show the different tabs for students, advisors, letter writers, and committee members, as well as the level of login at the top.



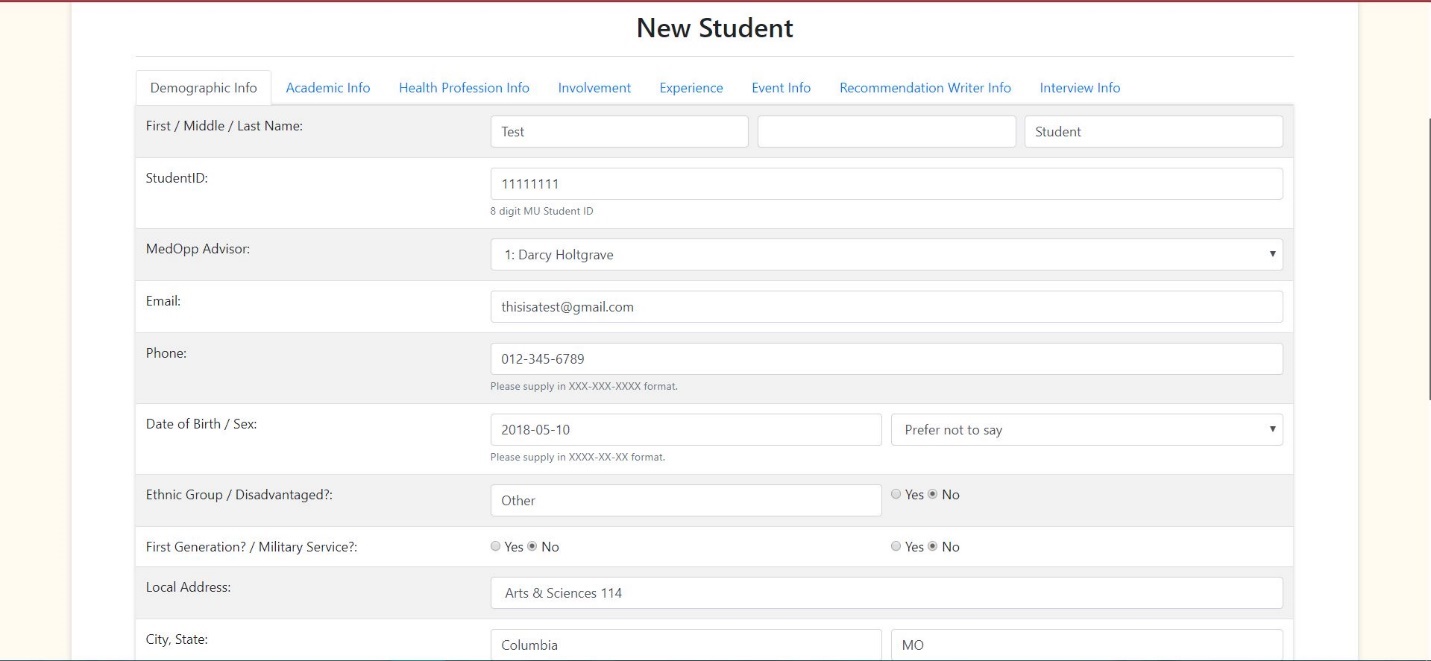
With the credentials provided, the page should show that the user is an administrator at the top (shown as ‘Admin’), and should have options to show queries as well as logout when clicked.

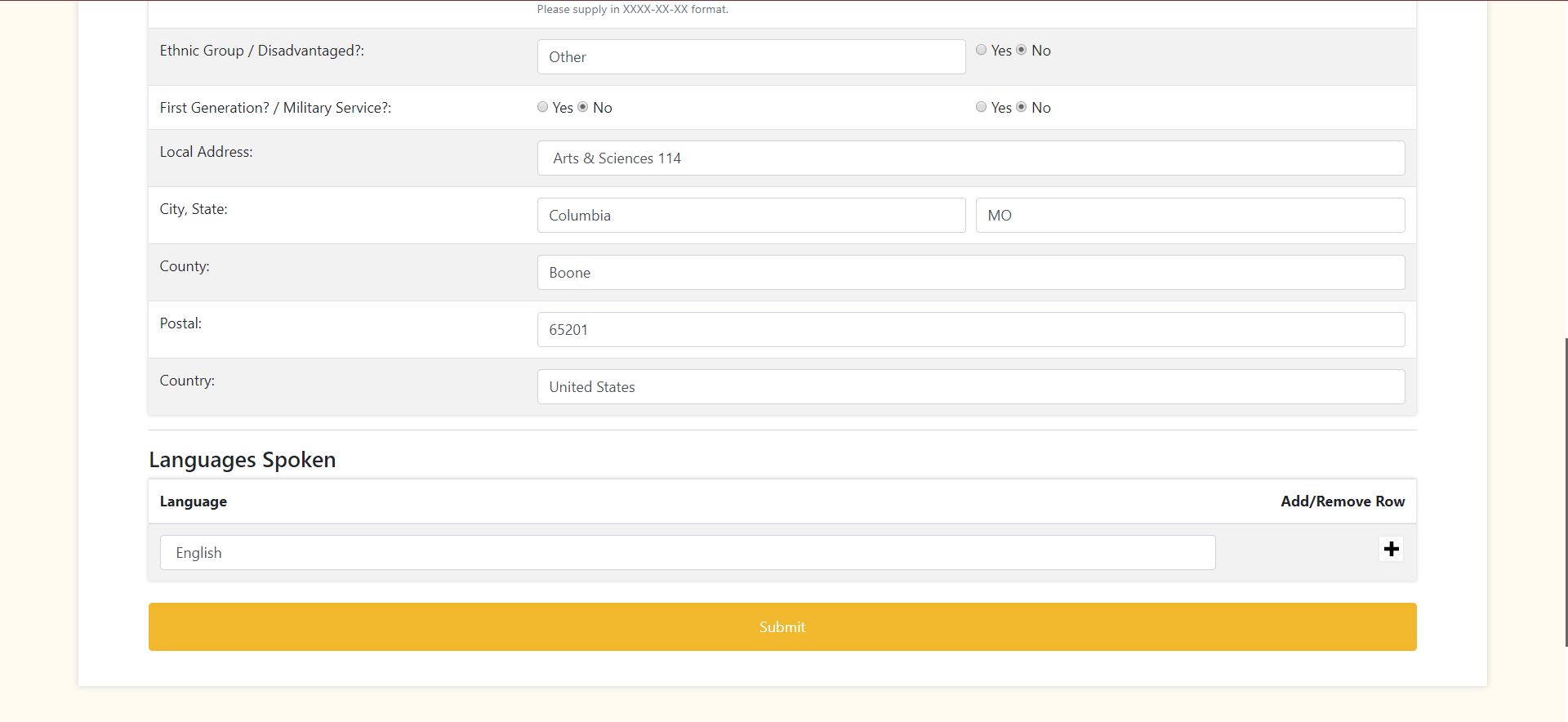


With the access of Admin, the user has the ability to manually add students, advisors, letter writers and committee members to the database. Students can be added by clicking on the Students tab and selecting ‘New Student’.

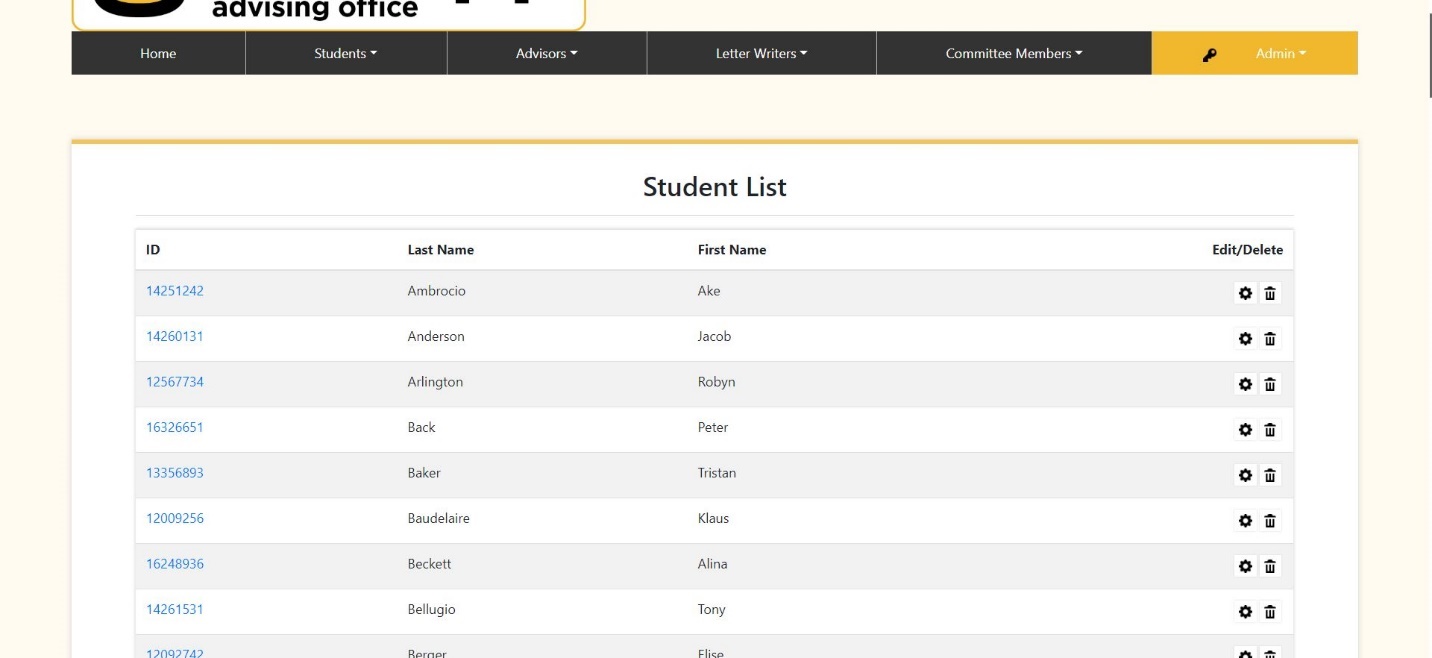


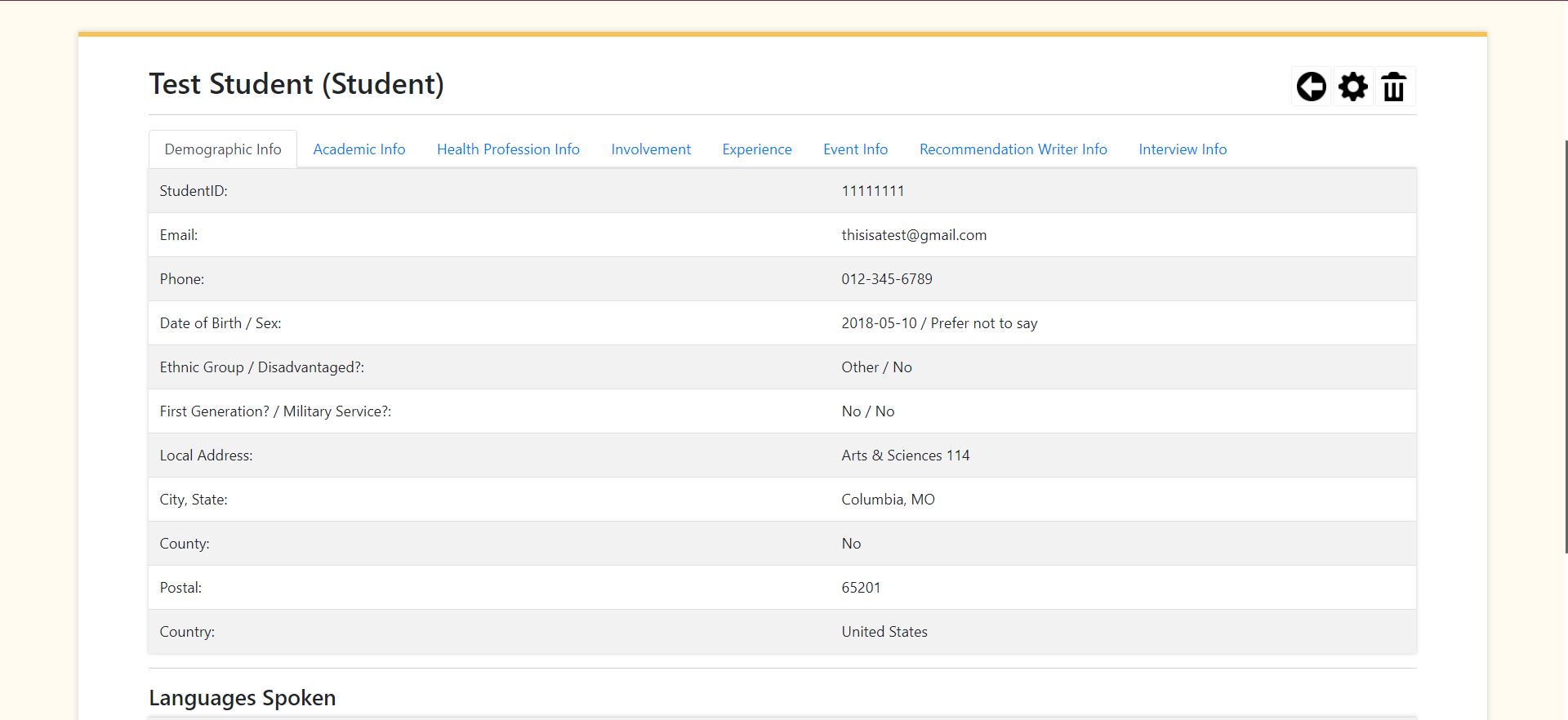
A form will show up with several different fields to input information; some will be required and some will be optional. The user should input any information that is relevant to the student in addition to the required fields, including but not limited to: address, email, phone number, languages, work experience, GPA, application status, etc. Giving this information makes it easier for the user to generate queries related to the students. Once all desired information is given, the user can click ‘Submit’ at any time on the bottom of the page to input the information.



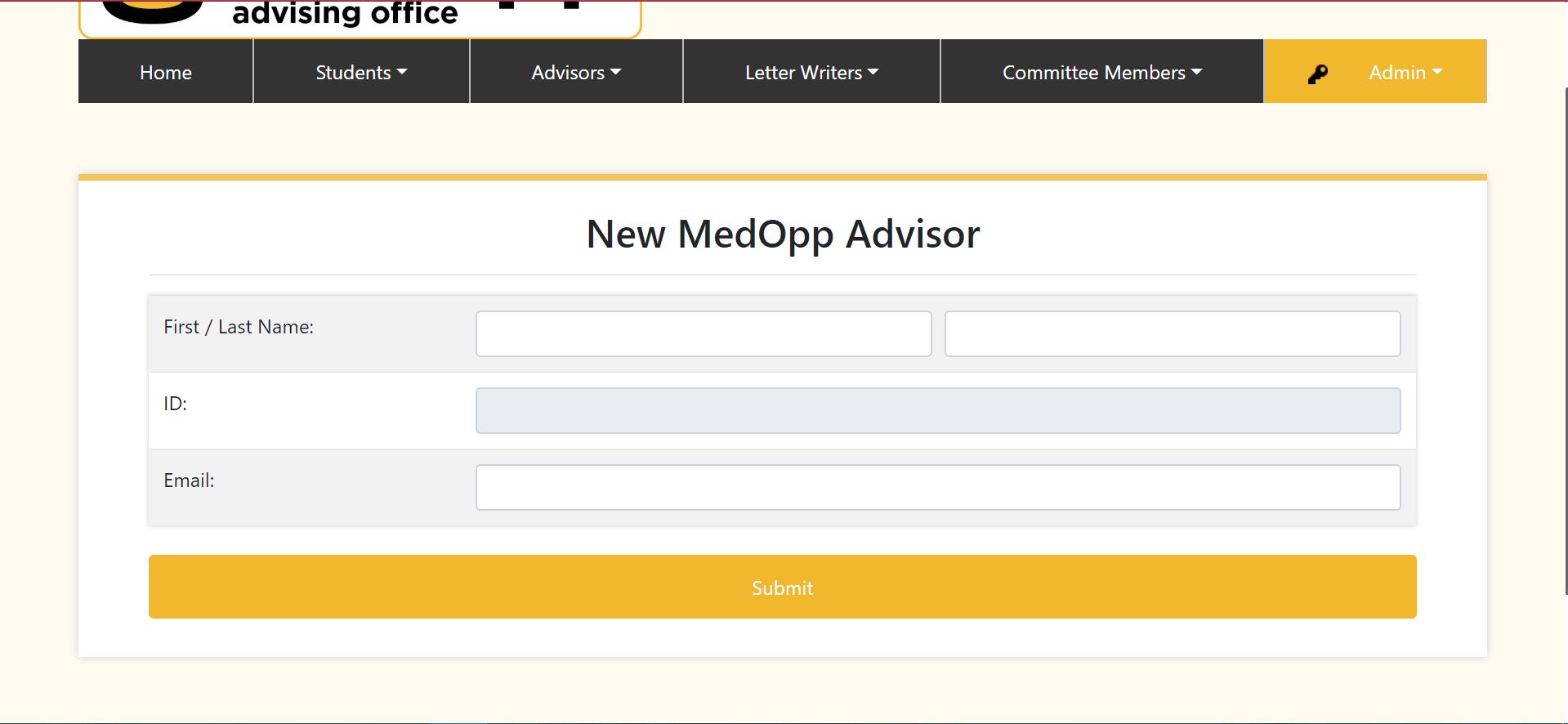


Once the Submit button is clicked, it will bring the user to the ‘View All Students’ page, where all of the students are shown and listed alphabetically. However, the user can edit or delete the student at any time using the edit and delete buttons on the right side of the student’s name. The student can also be clicked on to present the current information by clicking the student number.

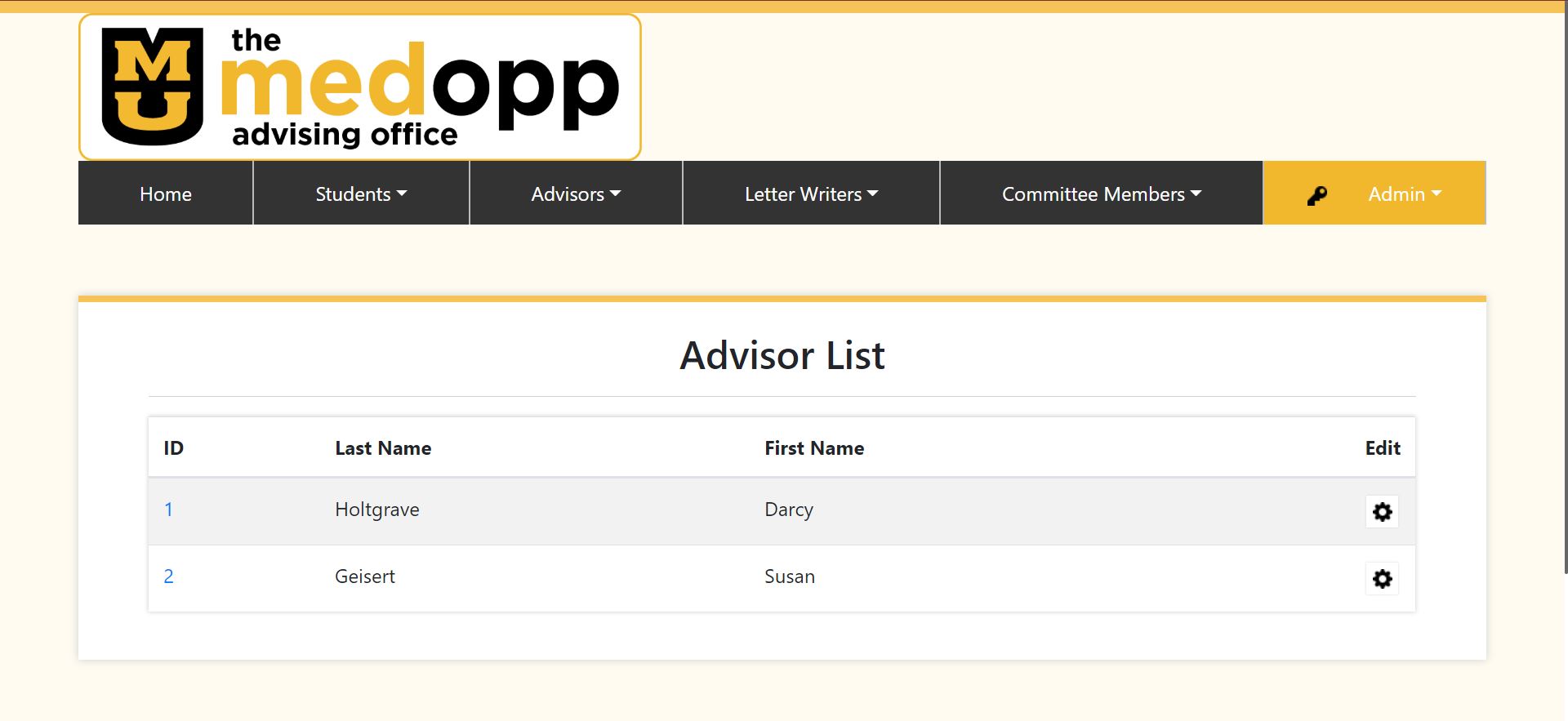


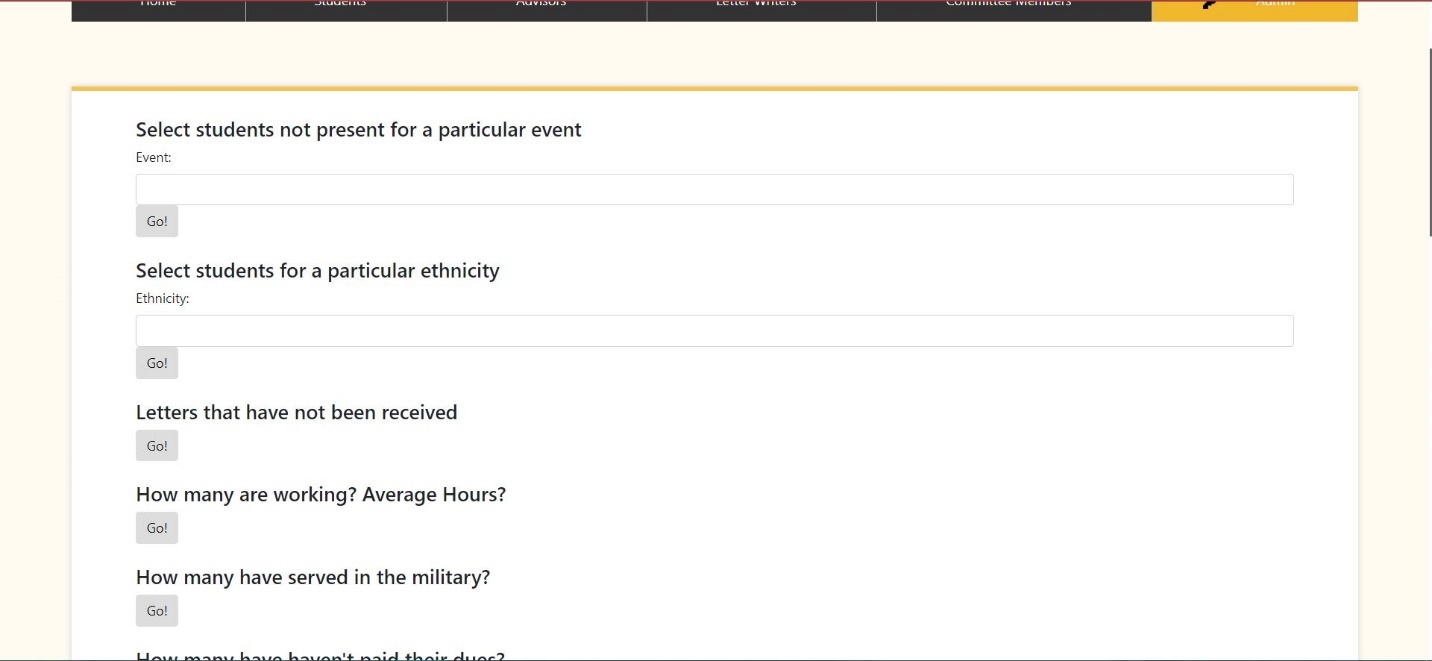


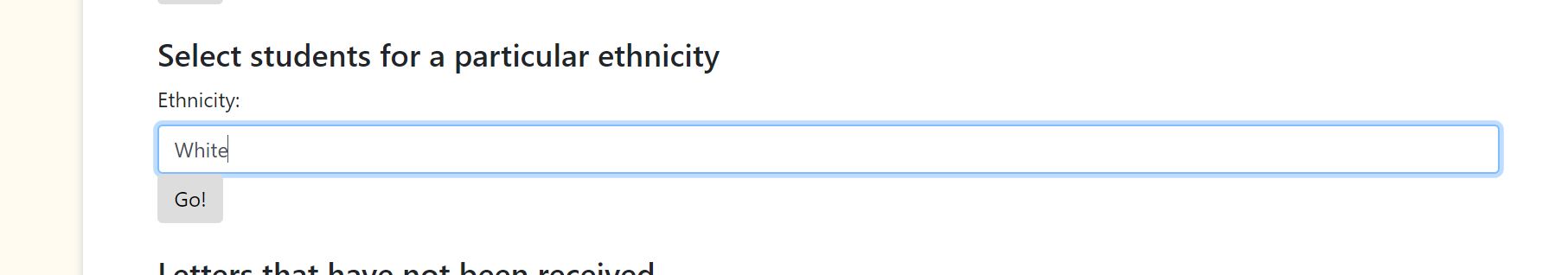
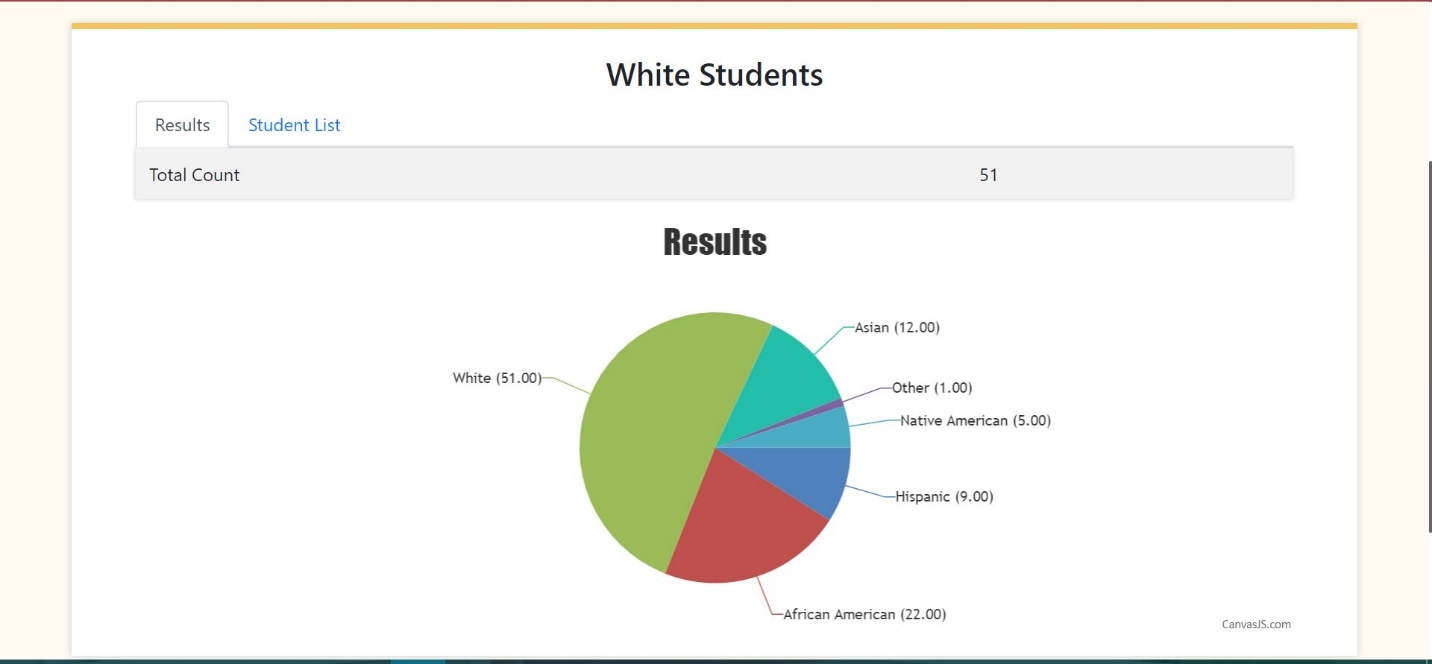
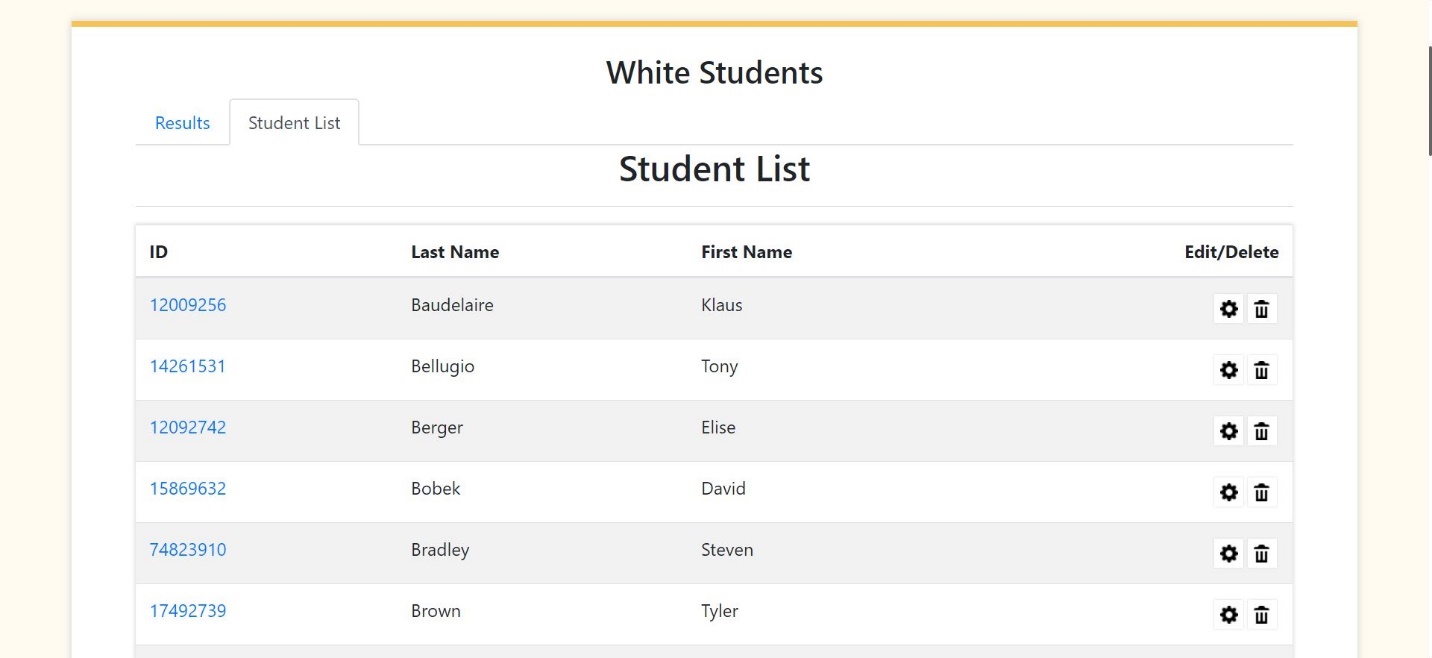
For a new Advisor, Letter Writer, and Committee Member, the form will ask for first and last name, id number, and email. Enter the information into the fields and press Submit to add them to their respective tables.



Note: Be careful about adding and deleting advisors, as students in our tables must be assigned to a table; if students are assigned to an advisor while someone is deleting that advisor, the advisor as well as the students assigned will be dropped from the system. Make sure that if an advisor need to be deleted, no students are assigned to that advisor.



The Queries page can be accessed from the Admin dropdown menu. To search a query, the user can enter the information in the text field provided or simply press ‘Go!’ to generate the results. The number results will appear underneath a graph of the returned data. Some queries will have graphs, others will not. The user can either see the total number of results or the specific students that the query returns.



Conclusion & Final Thoughts

This project has been an endeavor that has taught many lessons on the value of cooperation and communication, as well as planning, and appropriate time allotment. While we may have not been able to implement all of our initial ideas, we have laid a strong framework that can be built on in the future to satisfy all the needs our client may need. Seeing our classmates’ projects have inspired many future avenues for the continuation of this project outside of this class, on to what will be hopefully be an amazing product the MedOpp office can use for years to come.