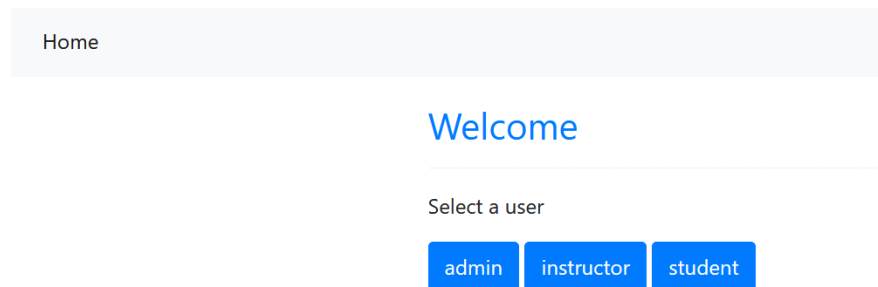


Home page:

For this task we had to create a homepage that allows the user to pick which type of user they were. The home page is the opening page of the website, to get to it the user has to go to `http://127.0.0.1:8000/home/`. The home page has 3 buttons, one for each user. When the button is clicked it will bring the user to a page specific to that type of user. The home page is created by using the html file `homepage`. Using html format for the button when a given button is clicked it will go to a function that will render the next page html.

```
<button class="btn btn-primary" onclick="window.location.href='/admin1'">admin</button>
<button class="btn btn-primary" onclick="window.location.href='/instructor'">instructor</button>
<button class="btn btn-primary" onclick="window.location.href='/student'">student</button>
```

We added extra formatting to make the page look more appealing by making the buttons blue with rounded edges. When the html file is called it will look like the image below.



Feature 1:

For this feature we implemented 3 buttons under our admin page, labeled Name, Department, and Salary. There is a text above them stating “Click button to get list of name by name, department and salary”. When each button is clicked, it runs a query against our MySQL database to get a list of names of professors ordered by whichever button you clicked. The code for generating these buttons as well as the resultant output is shown below.

```
<p> Click button to get list of name by name, department and salary</p>
<button class="btn btn-primary" onclick="window.location.href='/name'">Name</button>
<button class="btn btn-primary" onclick="window.location.href='/dept_name'">Department</button>
<button class="btn btn-primary" onclick="window.location.href='/salary'">Salary</button>
```

Click button to get list of name by name, department and salary

[Name](#) [Department](#) [Salary](#)

Ordered by Name

Name
Hou
Khondker
Lambert
Lynch
Martin
NEW MA
New Prof.
Ramsdell
Thorpe
White
Yao

Each buttons on click called a different function, either name() dept_name(), or salary(), for displaying our list as ordered by name, department, and salary> Shown here is our name function, the other two are similar with a different order by statement:

```
def name(request):
    mydb = mysql.connector.connect(
        host="128.153.13.175",
        port="3306",
        user="group_c",
        passwd='ChaBraKatMik',
        auth_plugin='mysql_native_password',
        database="university_group_c",
    )

    mycursor = mydb.cursor()

    mycursor.execute('Select name from instructor order by name asc;')

    table_data = '<table style="width:400px"><th>Name</th></tr>'
    for (name) in mycursor:
        name1 = str(name[0])
        row = '<tr><td>{</td></tr>'.format(name1)
        table_data += row
    table_data += '</table>'

    mycursor.close()
    mydb.close()
    context = {'table_data': table_data}
    return render(request, 'name.html', context)
```

Feature 2:

For feature 2, we created 3 buttons called Minimum, Maximum and Average. The image below shows the html format for the tree buttons.

```

<br><b>Salary:</b></br>
    <button class="btn btn-primary" onclick="window.location.href='/minumumTable'">Minimum</button>
    <button class="btn btn-primary" onclick="window.location.href='/maximumTable'">Maximum</button>
    <button class="btn btn-primary" onclick="window.location.href='/averageTable'">Average</button>

{% block table %} {% endblock %}

```

Each button code had a “onclick=“window.location.href=‘/minumumTable’” line which tell the html that when that button is clicked it has to go to that function which is in the view.py file. The image below shows the function for the minimum Table. The function for the maximum function and average are the same except the queries are different.

```

def minumumTable(request):
    mydb = mysql.connector.connect(
        host="128.153.13.175",
        port="3306",
        user="group_c",
        passwd='ChaBraKatMik',
        auth_plugin='mysql_native_password',
        database="university_group_c",
    )

    mycursor = mydb.cursor()

    mycursor.execute('select dept_name, min(salary) as min from instructor where salary is not null group by dept_name;')

    table_data = '<table style="width:400px"><th>Department Name</th><th>Minimum Salary</th></tr>'
    for (dept_name, min_salary) in mycursor:
        row = '<tr><td>{</td><td>{</td></tr>'.format(dept_name, min_salary)
        table_data += row
    table_data += '</table>'

    mycursor.close()
    mydb.close()
    context = {'table_data': table_data}
    return render(request, 'min_table.html', context)

```

The functions will connect to the database and then call the query. The query will return the given values that will then be imputed into a table format for html. Which is being done in the for loop. Once that is done the database is disconnected and the data is rendered to the html called min_table. min)table is an html file that will display the table and is an extension of the admin page html allowing the table to show on the admin page. The image below shows the web page display of feature 2.

Click button to get list of minimum, maximum, and average salaries by department:

Minimum

Maximum

Average

Minumum Salaries by Department

Department Name	Minimum Salary
CHE	150000
CS	95000
ECE	100000
MA	89000
PH	98000

Feature 3:

For this feature, three drop down menus, Name, Year, and Semester were added to our admin page. The data contained within the dropdowns was populated by information from the database, as well as hard coded years for our year dropdown. There is also a submit button next to the drop downs, which when clicked, generates the resultant table from putting the drop down parameters into the appropriate SQL queries. The resultant table has 5 columns, Name, #Sections, #Students, Funding, and Papers Published. This table is intended to be a representation of the performance for a given professor's name, the year, and semester. Below the dropdown options and resultant table are displayed:

Name: -- Select Name -- Year: -- Select Year -- Semester: -- Select Semester -- Submit

-- Select Name --

- Hou
- Lambert
- Thorpe
- Lynch
- Martin
- Ramsdell
- Yao
- Khondker
- NEW MA
- White
- New Prof.

Name	#Sections	#Students	Funding	Papers Published
Hou	2	2	11500000	1

This feature required 4 separate queries as well as model data pulled from models.py. The values the user input for Name, Year, and Semester are obtained as post requests, and the drop downs were populated using the objects() method from our models.py file, as shown here:

```
name = request.POST.get('name', '')
year = request.POST.get('year', '')
semester = request.POST.get('semester', '')
names = Instructor.objects.all()
years = Takes.objects.values('year').distinct
semesters = Takes.objects.values('semester').distinct
```

This data was then fed into the 4 queries used to retrieve the column data, with one value being retrieved for each column as shown below:

```
query1 = 'select count(sec_id) from instructor join teaches on teaches.teacher_id = instructor.id where name = ' \
        '%s and year = %s and semester = %s;'
query2 = 'select count(distinct(student_id)) as "#Students" from instructor join teaches on teaches.teacher_id ' \
        '= instructor.id right join takes on takes.course_id = teaches.course_id ' \
        'where name = %s and takes.year = %s and takes.semester = %s group by takes.semester, takes.year;'
if (semester == 2):
    query3 = 'select sum(amount) from investigator as v join grantaward as g on v.award_id = g.award_id and v.agent ' \
            '= g.agent join instructor on id = teacher_id where name = %s and ((startYear = %s and startMonth <= 6) ' \
            'or (startYear >= %s and endYear>= %s) or (endYear = %s and endMonth <= 6));'
else:
    query3 = 'select sum(amount) from investigator as v join grantaward as g on v.award_id = g.award_id and v.agent ' \
            '= g.agent join instructor on id = teacher_id where name = %s and ((startYear = %s and startMonth <= 12) ' \
            'or (startYear >= %s and endYear>= %s) or (endYear = %s and endMonth > 8));'

if semester == 2:
    query4 = 'select count(title) from publication join instructor on id = teacher_id where name = %s and ((year = ' \
            '%s and month < 6) or (year > %s));'
else:
    query4 = 'select count(title) from publication join instructor on id = teacher_id where name = %s and ((year = ' \
            '%s and month <= 12) or (year > %s));'
```

If the page was being loaded for the first time with a GET request, just the dropdowns with their relevant database data would be loaded. If a POST request occurred however, this would mean someone selected options in the dropdowns and hit submit, so our table would be rendered, as shown in this code:

```

table_data = '<table style="width:600px"><tr><th>Name</th><th>#Sections</th><th>#Students</th><th>Funding</th><th>Papers Published</th></tr><tr><td>{</td><td>{</td><td>{</td><td>{</td><td>{</td></tr>'.format(name, result1, result2,
                                                                    result3, result4)
table_data += '</table>'

mycursor.close()
mydb.close()
if request.method == 'POST':
    context = {'table_data': table_data, 'names': names, 'years': years, 'semesters': semesters}
    return render(request, 'f3.html', context)
else:
    context = {'names': names, 'years': years, 'semesters': semesters}
    return render(request, 'base.html', context)

```

Instructor Home Page:

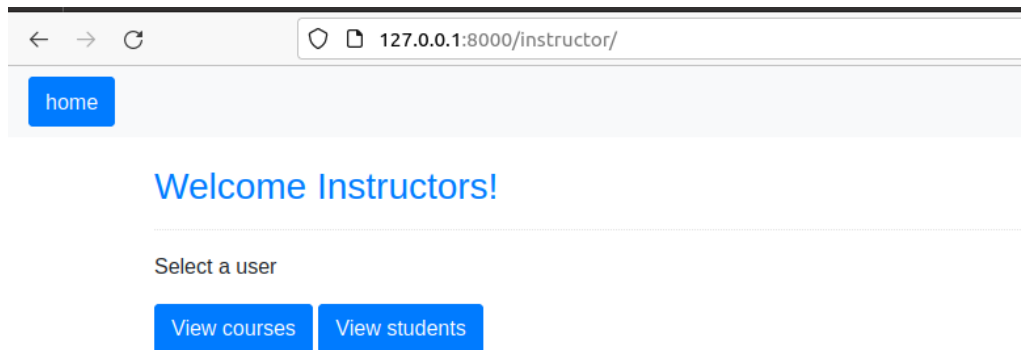
Since there were two features for the Instructor tab, we implemented a page with two buttons to reroute to each feature. The code for the buttons is shown below. The “View courses” button sends the user to Feature 4, while “View students” sends them to Feature 5.

```

<button class="btn btn-primary" onclick="window.location.href='/instrr'">View courses</button>
<button class="btn btn-primary" onclick="window.location.href='/instructor2'">View students</button>

```

The page shows up for instructors as the following image, with a home button to return to the homepage.



Feature 4:

For Feature 4, we had to create a form so that instructors could enter their ID, a year, and semester and get a table of the classes they taught that semester with the number of students that were in each section. To implement this, we created a HTML form for the instructor to input all the information, shown in the following code.

```

</div>
<form method="POST" action="{% url 'instr' %}">
    {% csrf token %}
    <label for="name">ID:</label>
    <input type="text" id="name" name="name"><br><br>
    <label for="year">Year:</label>
    <input type="number" id="year" name="year"><br><br>
    <label for="semester">Semester:</label>
    <input type="number" id="semester" name="semester"><br><br>
    <input type="submit" name="submit" value="Submit">
</form>

```

When the submit button is clicked, the 'instr' function is called. This function then uses the input to get the course sections and number of students for that given professor that year and semester. The query we used to perform this search is shown in the image below.

```

mycursor = mydb.cursor()
query = 'select course_id, sec_id, numStus from teaches left join (select course_id as course, sec_id as sec, semester as sem, year as yr, count(studen
t id) as numStus from takes group by course, sec, sem, yr) as t2 on course_id = t2.course and sec_id = t2.sec and semester = t2.sem and year = t2.yr where seme
ster = ' + semester + ' and year = ' + year + ' and teacher_id = ' + name + ';'

```

The data is then compiled into a table which will then be displayed on a new page. The new page features a home button along with a back button, so instructors can easily maneuver out of the current page.

The image below shows how the interface looks for instructors once the "View courses" button is clicked. Along with the form, it has a home button to return to the homepage and a back button to go back to the instructor home page.

back home

Welcome Instructor!

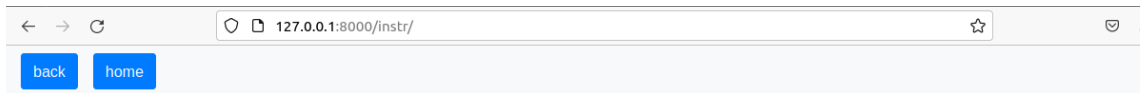
ID: 13342

Year: 2019

Semester: 1

Submit

Upon submitting their information, instructors will be shown their results in a table like the following, with buttons again to go a page back and home.



Your Courses

Course ID	Section	Number of Students
CS141	01	1
CS141	02	2

Feature 5:

For Feature 5 we had to create a way for an instructor to be able to input an instructor name, a course ID and the semester. For this we created 3 input textboxes. The image below shows the html code to create and the input textbox. The input from this code will be sent to feature 5 by designating it at the top by `action="{% url 'feature5' %}"`.

```
<!-- style= ... -->
<form action="{% url 'feature5' %}" method="post">
    {% csrf_token %}
    <div>
        <label for="instructor">Enter an instructor name:</label>
        <input type="text" id="instructor" name="instructor">
    </div>
    <div>
        <label for="course1">Enter a course id:</label>
        <input type="text" id="course1" name="course1" required>
    </div>
    <div>
        <label for="semester">Enter a semester (1 for fall, 2 for spring):</label>
        <input type="text" id="semester" name="semester" required>
    </div>
    <input type="submit" value="Submit">
</form>
{% block feature5 %} {% endblock %}
</div>
```

When the submit button is clicked the code calls feature 5 function. In function5 the code opens the database and imports the input from the html. The inputs are then entered into a query.


```
q = 'SELECT s.name FROM takes as t JOIN Student as s ON t.student_id = s.student_id JOIN section as sec ON ' \
    't.course_id = sec.course_id AND t.sec_id = sec.sec_id AND t.semester = sec.semester AND t.year = sec.year ' \
    'JOIN teaches as tch ON sec.course_id = tch.course_id AND sec.sec_id = tch.sec_id AND sec.semester = tch.semester ' \
    'AND sec.year = tch.year JOIN instructor as inst ON tch.teacher_id = inst.id WHERE t.semester = %s AND ' \
    'inst.name = %s AND t.course_id = %s;'
mycursor.execute(q, (semester, instructor, course))
```

The data from the query is then stored into a table that will be rendered to an html to be displayed on the page. If the user leaves the course id or the semester blank a message will show saying that something has to be typed into the box. The image below shows the instructor page for students.

[home](#)

Welcome Instructor!

Enter an instructor name:

Enter a course id:

Enter a semester (1 for fall, 2 for spring):

As stated before, once the user has entered the instructor name, course id and the semester. The user will press submit. After the submit button is clicked it will say "Student" and then list the three inputs entered. If the three inputs are valid and have students in those sections, a list of students will appear below. If there are no students in the course or a mistake is typed into the input box. It will only display what the user typed in but no list of students will be displayed.

Welcome Instructor!

Enter an instructor name:

Enter a course id:

Enter a semester (1 for fall, 2 for spring):

Students [Hou, EE468, Fall]

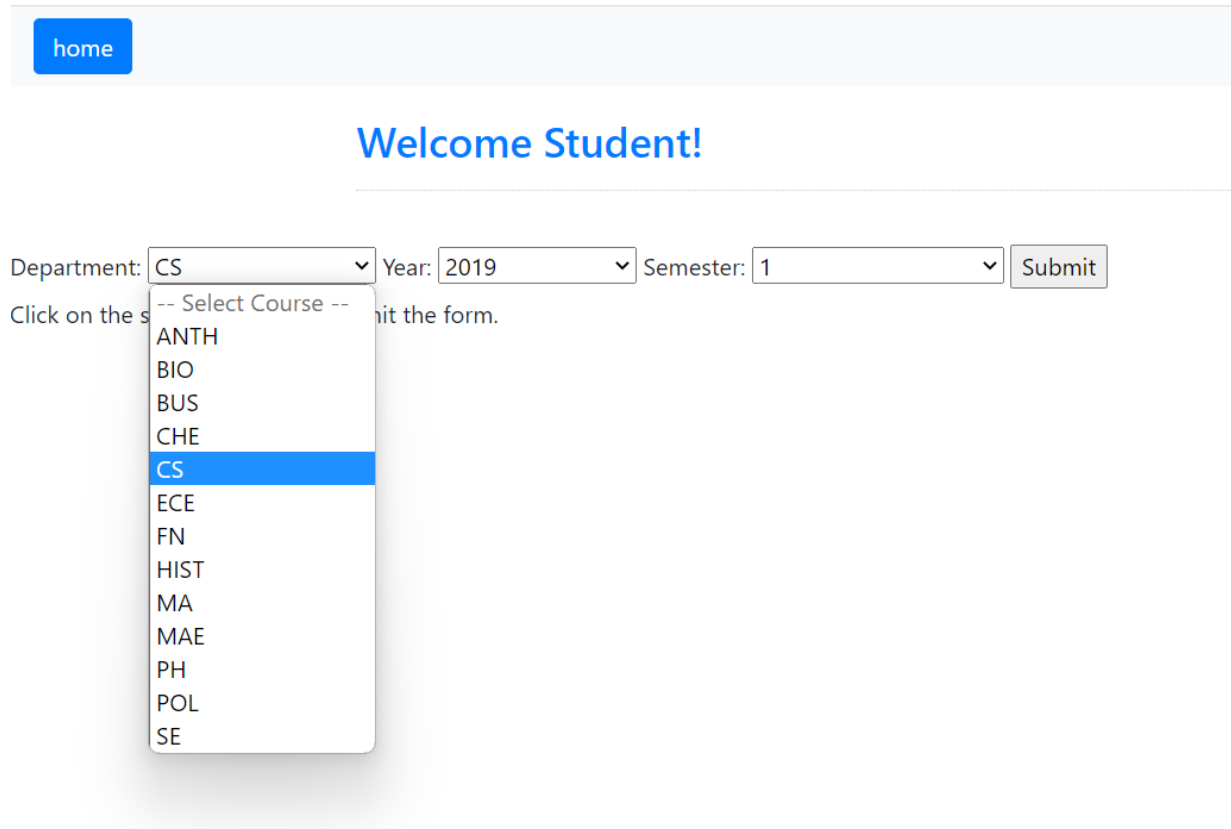
Alex

Bourikas

The instructor page also has a back and home button. When the button is clicked it will call a function that will either take it back to the homepage and render the homepage html or it will go to the previous page depending on which button is clicked.

Feature 6:

For this feature, a student page was created to allow for student users to query the list of course sections offered by a given department in a given year and semester. The page consists of three dropdown menus for Department, Year, and Semester along with a submit button for the user to click when they've made a selection for each dropdown menu. The design of the student page can be seen in the picture below:



The screenshot displays a web interface for a student query. At the top, there is a blue button labeled "home". Below it, a large blue heading reads "Welcome Student!". A horizontal line separates the header from the form area. The form contains three dropdown menus: "Department:" with "CS" selected, "Year:" with "2019" selected, and "Semester:" with "1" selected. To the right of these is a "Submit" button. Below the "Department:" dropdown, a list of departments is visible: ANTH, BIO, BUS, CHE, CS (highlighted in blue), ECE, FN, HIST, MA, MAE, PH, POL, and SE. To the left of the dropdown list, the text "Click on the s" is partially visible. Below the "Year:" dropdown, the text "hit the form." is partially visible.

Upon clicking submit, the user will be redirected to a new page which contains a table of the course sections offered by the given department in the given year and semester. A sample result of a student query can be seen in the image below:

[Back](#) [home](#)

Welcome Student!

Course	Section	Semester	Year	Building	Room	Capacity
CS141	01	1	2019	CAMP	194	60
CS141	02	1	2019	CAMP	194	60

If the user wants to perform another query, there is a “Back” button on the navigation bar, which will take the user back to the student page.

To populate the dropdown menus with data from the database, a file `models.py` was generated using the following command:

```
python manage.py inspectdb > models.py
```

After running this command, each table becomes a model object which can be referenced within `views.py` to populate the webpage with information from the database.

The function `student()` is then updated in `views.py` in order to fetch the data from the database to be used in the dropdown menus. The updated `student()` function can be seen in the image below:

```
def student(request):
    depts = Department.objects.all()
    years = Section.objects.values('year').distinct()
    semesters = Section.objects.values('semester').distinct()
    context = {'depts' : depts,
               'years' : years,
               'semesters' : semesters
              }
    return render(request, "student.html", context)
```

By updating the `student` function to include context whilst rendering, we can then update the HTML within `student.html`. Using Django’s template language, we are able to use for loops to iterate through the departments, years, and semesters. The updates made to `student.html` in order to iterate through departments can be seen in the image below:

```

<form action="{% url 'f6' %}" method="post">
{% csrf_token %}
    <label for="dept">Department:</label>
    <select name="dept" id="dept">
        <option disabled="true" selected>-- Select Department--</option>
        {% for r in depts %}
            <option>{{ r.dept_name }}</option>
        {% endfor %}
    </select>

```

The same changes shown in the image above were applied for the dropdown menus for year and semester.

Once the user clicks the Submit button on the page, a POST is sent to the server which is used to generate the table. This functionality is accomplished in the f6() function within views.py. First the POST for each dropdown is assigned to a variable name as can be seen in the image below:

```

def f6(request):
    dept = request.POST.get('dept', 0)
    year = request.POST.get('year', 0)
    semester = request.POST.get('sem', 0)

```

These variables are then used to complete the query shown in the following image:

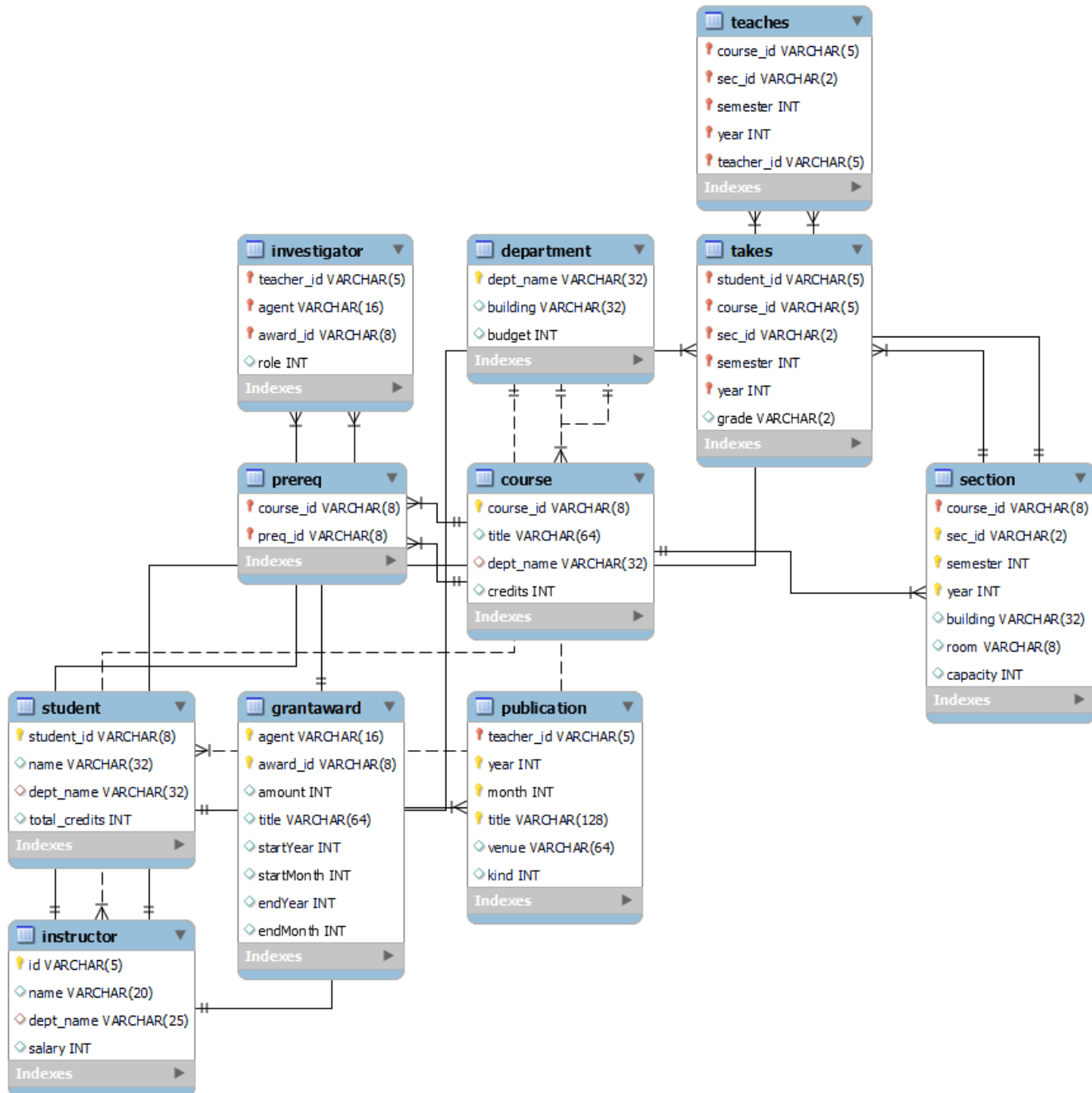
```

query = "select s.course_id, s.sec_id, s.semester, s.year, s.building, s.room, s.capacity from section s join " \
        "course c on s.course_id = c.course_id where c.dept_name = '" + dept + "' and s.semester = '" + semester + "' \
        "and s.year = '" + year + "'"

```

This query is then used to generate the table shown after pressing the submit button.

E-R diagram:



Security Assurance Process:

One security measure that was taken into account when developing the web-application was the prevention of SQL injection, where an attacker can modify a query to see data not meant for them. For the features that take advantage of dropdown menus populated from the database, we prevent the risk of possible SQL injection from potential attackers. An optimal implementation of our current design would utilize dropdown menus for all features that currently operate using text inputs.

While our current design simply uses buttons displayed on the homepage to authenticate a user as an admin, instructor, or student, an optimal implementation would take advantage of an actual login system to authenticate users.