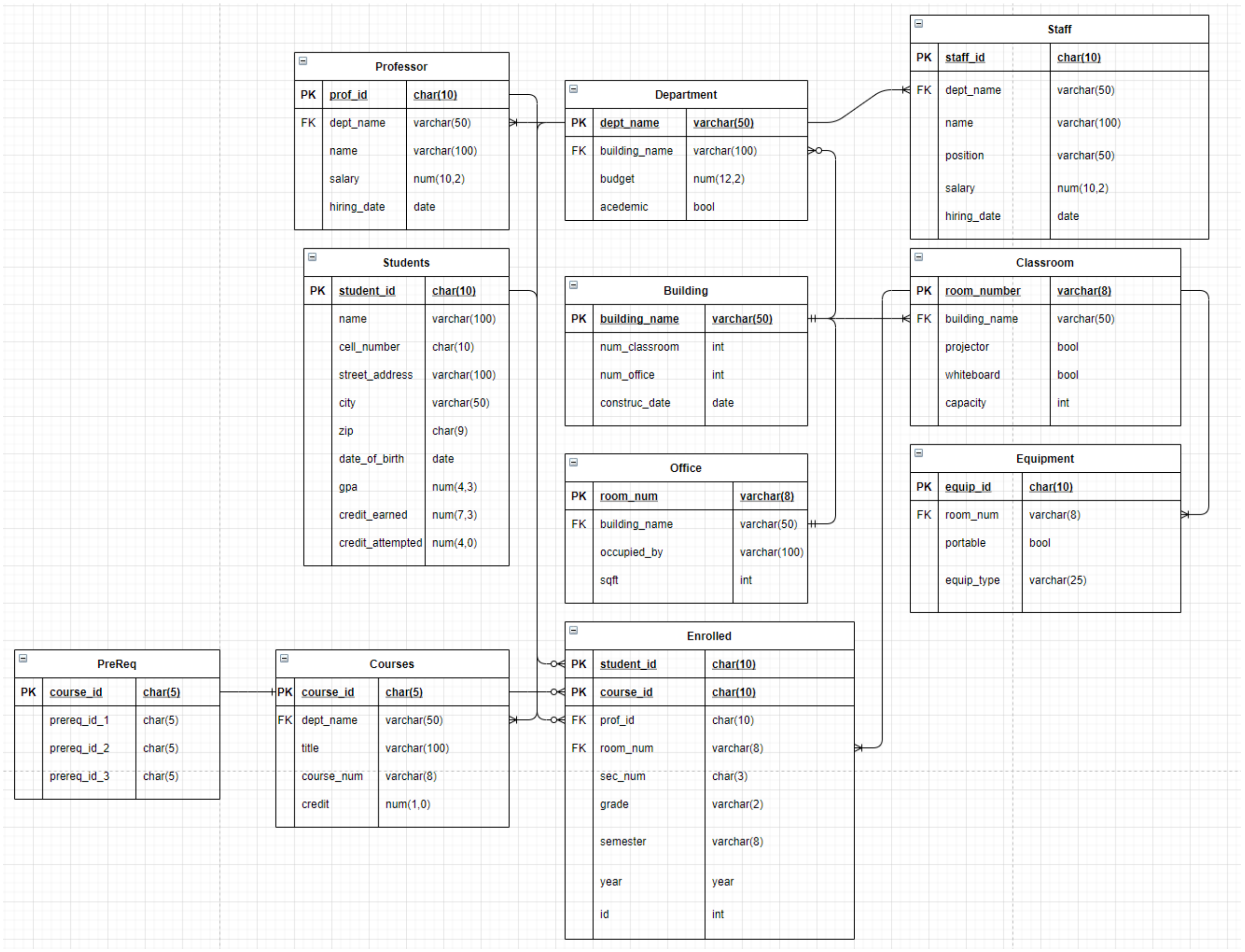
Alec Knutson

EPPS 6354

5-14-21

# Final Project Paper

I have chosen to create a University database for my final project. It consists of eleven relations in order to track University required information. In general, this information includes students, classes, staff, and capital that the university owns. In order to keep everything organized, I used a relational database model when designing the schema, shown below. This paper is broken up into four parts. First, I will discuss the database design. Second, I will briefly mention the data that is housed in each relation. Next, I will discuss the front-end design that I was able to implement. Lastly, we will discuss how the limitations of this database and how this project could be improved upon in the future.



The database is broken up into eleven distinct relations. Each one of these relations is designed to house information about different groups of entities that the University needs to keep track of. In this basic database, three relations are devoted for categories of people. These are the students, professors, and staff members. Four of the relations are devoted to physical capital that the University maintains.  Currently the capital that is being tracked is the buildings and electronics.  The last four relations track more abstract entities.  These are the various departments of the University and the courses that are offered.

One of the first and largest relations in our database is the Student relation.  It contains the basic information of every student that attends our fake university.  This information includes their basic credentials such as their name, cell number, full address of residence, and date of birth.  It also contains so university specific data such as their GPA, credits earned, and the total number of credits attempted.  We also include a student ID as the primary key of this relation that can identify them throughout the system.  Most of these variables are restricted in their scope.  The variable types are exactly what they need to be and nothing more.  For example, the student ID is started as a ten-character variable.  Even though it is technically a number, using it for mathematical operations is impractical.  Thus, we store it as a character.

The next relation we will talk about is a similar one, our Professor relation.  Along the same lines as the Student relation, the Professor relation stores all the information that the University needs to maintain of their professors.  This includes basic information such as their name, salary, and hiring date as well as identifying information such as a Professor ID number and the department for which they are working.  Of these five variables, the professor ID is the most important in terms of the database since it is our primary key.  Similar to the student ID, it is a ten-character string variable since it does not need to support mathematical operations.  Also, it is part of the same system as the student ID so that there are no chances for overlapping ID for students and professors.

The third and final people related relation that we have in this database is devoted to Staff.  We define the staff as anyone who is employed by the University who does not teach a class academically.  For this reason, the relation is the same as the professor relation, with all the same variables, expect for an added position variable.  This way we are able to identify groups of staff members easily by their roles within the university.

The next group of relations is our physical capital relations. The highest-level relation in this class is our Building relation.  It houses all the data the pertains to each individual building on campus.  There are four variables in this basic database.  Those variables are the building name, the total number of classrooms and offices, and the construction date of the building.  The building name is our primary key and it is a character variable.  Since it is a possibility that we may want to do mathematical operations and they will always be whole numbers, we store the number of classrooms and offices as integer.

Below this, we have two more relations that are similar.  The first is our Classroom relation.  It houses the data that pertains to the classrooms in each building.  The number of observations within the Classroom relation should be exactly the same as the sum of all num_classroom variables from the Building relation.  Within the Classroom relation, we house data that describes the characteristics of the room itself.  These are things like whether or not the room has a projector and/or whiteboard and what the capacity of students it can hold.  There is also two identifying variables as well.  These variables identify what building the classroom is in and what room number is assigned to the room. The room number variable is our primary key since it will be unique for all rooms throughout the campus. It consists of an eight-character string. The first four characters are for the abbreviation of the building name and the last four digits are a location number. Of these four digits the first digit is what floor the room is on, the second and third digits are the room number itself, and the last digit is reserved for segmented rooms (such as classrooms with temporary dividing walls).

The other side of this coin is the Office relation. It houses all the data for the various offices that are throughout the campus. This entails the building that the office is in and the room number as well as how big the room is and who occupies it. The room number is once again our primary key and it shares all the same characteristics and layout style of classroom numbers. The utilize the same numbering system so there will be no overlap between the numbers. Both staff members and professors have the ability to occupy offices.

The final relation in our capital group is our Equipment relation. This relation keeps track of the various pieces of equipment that the university owns and utilizes. These could be things such as computers, tablets, projectors, document cameras, etc. Within this relation we have an equipment ID that is used as our primary key. It would ideally be written or labeled on the side of the physical item in our fake university since this is the primary method by which we could identify it in our system. We also keep track of what room this item should be in by storing the room number. Finally, take down the equipment type and whether or not the item is portable. Since the equipment can be a variety of different objects, the equipment type is a open ended character variable while the portable variable is a boolean.

The final group of relations are more abstract in nature. The first and highest order of these is the Department relation. Departments are groups of people that work together to provide as specific set of services to the university. The primary key of this relation, the department name, can be found as a foreign key in a few other relations as well. It is through the departments that the university is mainly organized. Also, within this relation, we store what building the department mainly operates from and what is their yearly budget. Naturally, the budget is stored a numeric variable that has two decimal points, while the name is stored as a character variable. Our last variable in this relation is a boolean that indicates whether or not the department is an academic department. We define an academic department to be one whose primary focus is to an individual discipline, as well as prepares and instructs classes within said discipline. This would include departments such as Mathematics, Economics, Computer Science, Physics, Music, etc. A non-academic department is a division of the university which has the primary purpose of organization of the business or the upkeep of facilities. These would include departments such as Advising, Admissions, Information Technology, Security, etc.

The other three relations are devoted to course management. The Courses relation houses a list of all courses offered by the university. Within, there is an ID variable which is our primary key. This is a five-digit number where each new course that is added will be assign a unique identifying number. This is mainly for internal use. The variables that house the department name, a foreign key, and the course title and number are more useful for the average person to identify the different courses. The title is a character string that is the full-length name of the course.  There is another possibility for the primary key, namely the course number.  This number consists of a similar numbering scheme to the room number.  The first four characters of the eight-character variable are a department abbreviation while the final four characters are reserved for a unique number assigned to each course, the first of which indicates level.  For each course, we also store how many credits the completion of this class will earn a student.

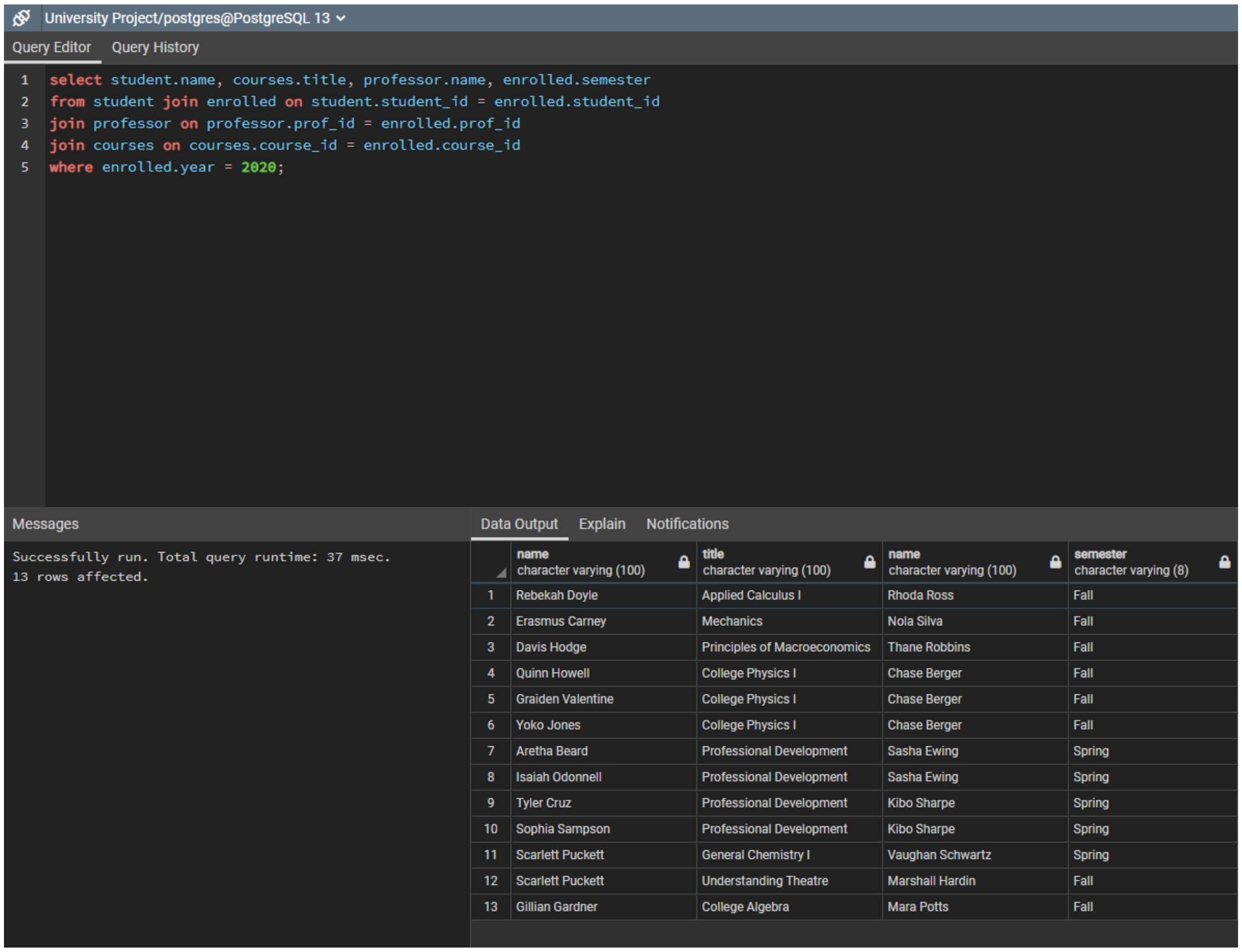
A highly related relation is the Prereq relation.  It is highly dependent on the courses relation for interpretation.  Within this relation, we use the course ID as the primary key again since every course will have a maximum of one entry.  The other three variables are other course ID's from the Courses relation that are required by the University to have been taken before the course in question.

Last but not least is the Enrolled relation.  This relation may be the most complex out of the eleven.  It contains a list of all the courses that students have taken or are taking.  For this reason, we are using two primary keys, those being the student ID and the course ID.  We also house vital information about each type of course for operation within this relation.  These include the instructor's ID, the room number where the class is taught, the section number in case of there are multiple classes being taught, and the semester/year that the class was taught. We also store the student's current grade in this relation in the form of a two-character variable. This allows for a letter grade that includes pluses and minuses.   As a last-minute edition, I was required to add a generic id variable to the list in this relation.  It is an auto-incrementing integer variable that is simply used as an identifier for Django.  Postgres is set up in order to use both the course ID and the student ID as the primary keys.  However, Django does not allow two primary keys when defining a model.  Thus, so that Django can identify and display this relation, a unique id the is only for use by Django must be added.

Obtaining the data for the database was not super straight forward.  Besides the ID used by Django in Enrolled, no other variable is auto generated by Postgres.  Thus, all other data had to not only come from outside the database, but also be realistic.  We need it to be realistic in order to see if any problems would come up in a real-life scenario.  In order to accomplish this, I utilized a free open source online data generation tool call generatedata.com for about half of the data.  This fantastic online tool lets you generate a verity of different types of data such as names, address, zip codes, and cell numbers.  It also lets you format the data in a way you define, such as dividing a phone number with dashes or slashes.  Because of how the site exports its data as SQL, I had to rewrite most of the code myself since I wanted to use very specific data types in my database.  Also, I had already designed and implemented all the tables and thus could not utilize their code generation.  By this method, I was able to generate realistic data for my Student relation and names for the staff/professors.

As for the rest of the data, I either entered it in myself or obtained it from real universities.  For example, when it came to the grades and credits of the students, I was forced to simply make up numbers in the most random but realistic way I could.  This also includes variables such as room numbers, whether or not a classroom as a white board or projector, and who occupies which offices.  The variables that utilize department or building names or abbreviations, I obtained for real universities.  I gather a verity of different name from various universities in order to get a unique set of buildings that was still realistic. A very similar method was taken when constructing the Course relation. All courses are real courses, with real titles, course numbers, and number of credits from UTD's coursebook.

Having implement all this, we can now test the Postgres server. I ran several SQL queries to make sure I could access all the relations. Having succeed, I tried various joins and filtering's. The final one of note is the SQL below. All the queries successful pulled the information required without duplicates and without missing any observations.


University Project/postgres@PostgreSQL 13

Query Editor | Query History

```sql
select student.name, courses.title, professor.name, enrolled.semester
from student join enrolled on student.student_id = enrolled.student_id
join professor on professor.prof_id = enrolled.prof_id
join courses on courses.course_id = enrolled.course_id
where enrolled.year = 2020;
```

Messages

```
Successfully run. Total query runtime: 37 msec.
13 rows affected.
```

Data Output | Explain | Notifications

| | name character varying (100) | title character varying (100) | name character varying (100) | semester character varying (8) |
|---|---|---|---|---|
| 1 | Rebekah Doyle | Applied Calculus I | Rhoda Ross | Fall |
| 2 | Erasmus Carney | Mechanics | Nola Silva | Fall |
| 3 | Davis Hodge | Principles of Macroeconomics | Thane Robbins | Fall |
| 4 | Quinn Howell | College Physics I | Chase Berger | Fall |
| 5 | Graiden Valentine | College Physics I | Chase Berger | Fall |
| 6 | Yoko Jones | College Physics I | Chase Berger | Fall |
| 7 | Aretha Beard | Professional Development | Sasha Ewing | Spring |
| 8 | Isaiah Odonnell | Professional Development | Sasha Ewing | Spring |
| 9 | Tyler Cruz | Professional Development | Kibo Sharpe | Spring |
| 10 | Sophia Sampson | Professional Development | Kibo Sharpe | Spring |
| 11 | Scarlett Puckett | General Chemistry I | Vaughan Schwartz | Spring |
| 12 | Scarlett Puckett | Understanding Theatre | Marshall Hardin | Fall |
| 13 | Gillian Gardner | College Algebra | Mara Potts | Fall |

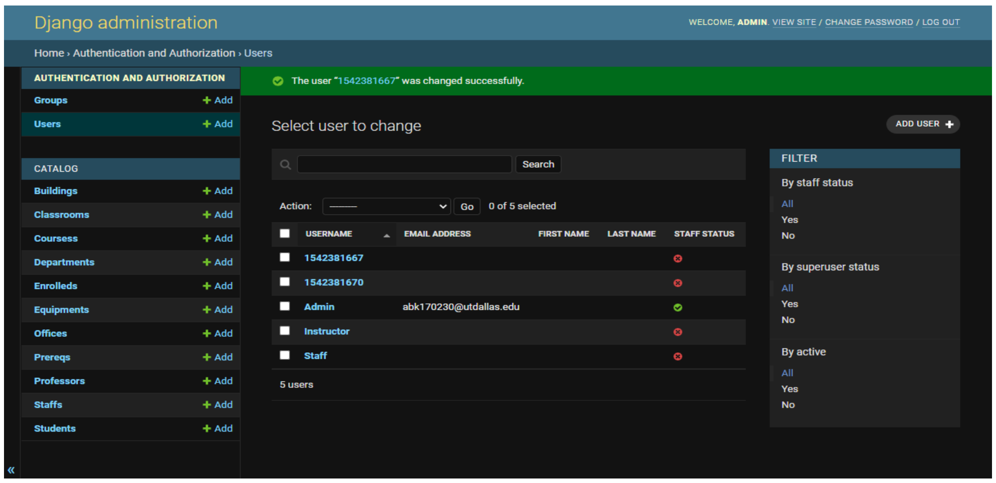
This brings us to the topic of front-end implementation. We utilized python and Django in order to implement a front-end user interface for the database. The building of our Django server was a mostly typical process. We utilized a number of different fields in order to mimic the Postgres server exactly. This includes utilizing a foreign key field. For this database, cascade is enabled for all foreign keys so that the modification or deletion of one part of the database is changed throughout the entire database. This protects against inconsistent information in case a course is renamed or in case of human error when creating a new observation. One important model to point out is the Enrolled model. As we were talking before, this relation has two primary keys and since Django does not support this feature a generic id variable needed to be created for Django's use only. I set this ID variable up as an auto incrementing integer on Postgres' side, so in the Django model we must use the auto field. Also, to ensure consistency when making changes to the database, we need to add another line to the Meta class. This line forces the variables of student ID and course ID to be unique together when submitting changes to the database. This should limit the number of errors we receive back from the server since Postgres will also enforce these variables are our primary keys.

```python
class Enrolled(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey('Student', on_delete=models.CASCADE, db_column='student_id')
    course_id = models.ForeignKey('Courses', on_delete=models.CASCADE, db_column='course_id')
    prof_id = models.ForeignKey('Professor', on_delete=models.CASCADE, db_column='prof_id')
    room_num = models.ForeignKey('Classroom', on_delete=models.CASCADE, db_column='room_num')
    sec_num = models.CharField(max_length=3)
    grade = models.CharField(max_length=2, null=True)
    semester = models.CharField(max_length=8)
    year = models.IntegerField()

    class Meta:
        managed = False
        db_table = 'enrolled'
        unique_together = (('student_id', 'course_id'),)
```

After implementing all eleven relations into Django, I wanted to implement a user interface as well. There are several considerations we must take into consideration when building this graphical user interface as well. Firstly, the type of data we are storing needs to be utilizable by a great many user, most of which may not have a lot of computer science background. For this reason, setting up a series of url's that someone has to type in to view the data is not practical. Also, there is enough data stored that we can't just put it up on one screen. It needs to be split apart into different tables that a user can select from a list. Finally, a lot of this information is user specific. Thus, in order to maintain user privacy, we need to also have a user authentication system to place some data behind a login wall.

To start off, we need to build the user login system. Django allows for user authentication inherently. We can mange the users, there passwords, some of their permissions for the traditional administer website. From this site, we can also lock users out of the administration site so they do not have access to our servers back end without permission. This option is called staff status and is step one in security.



The user "1542381667" was changed successfully.

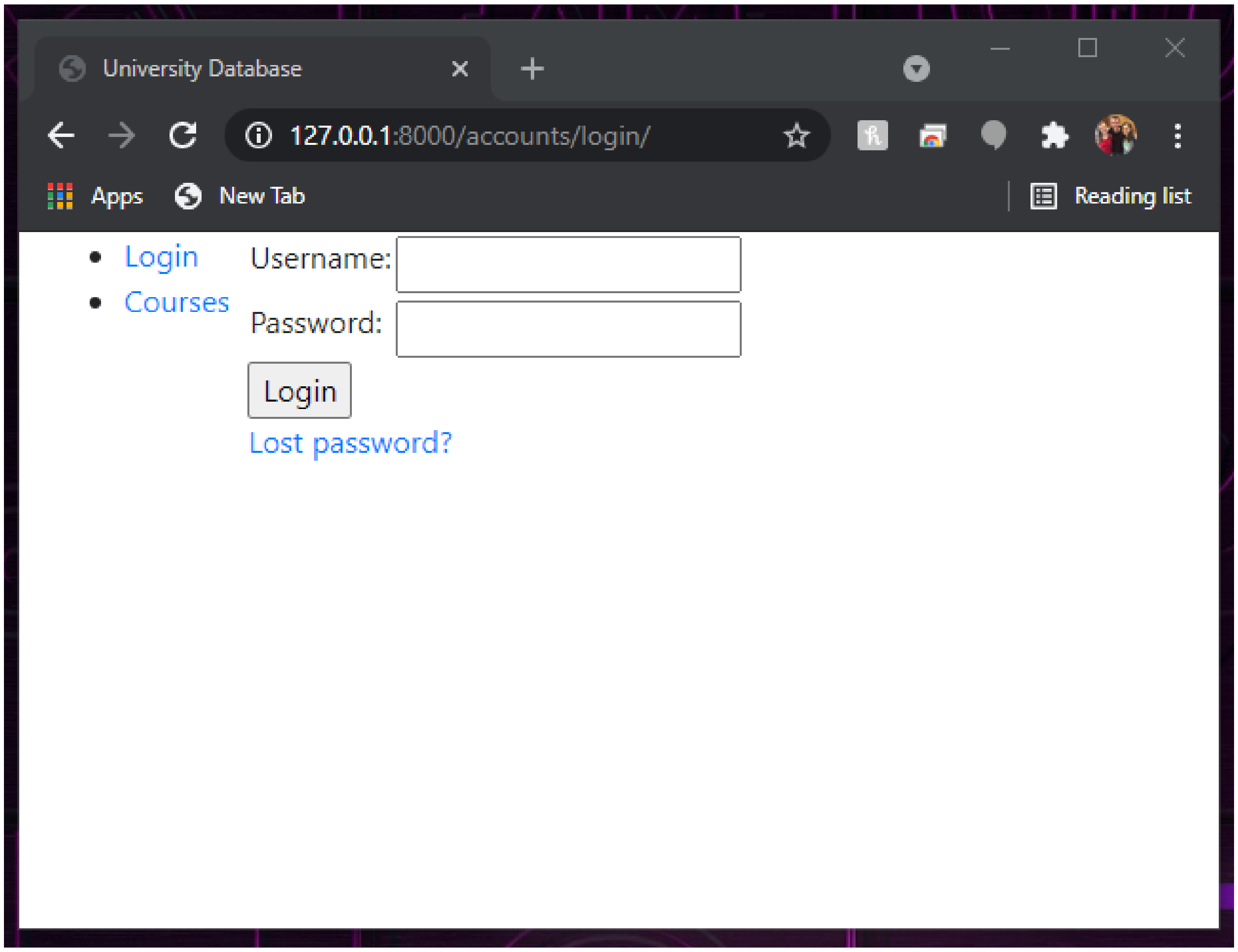
Select user to change

0 of 5 selected

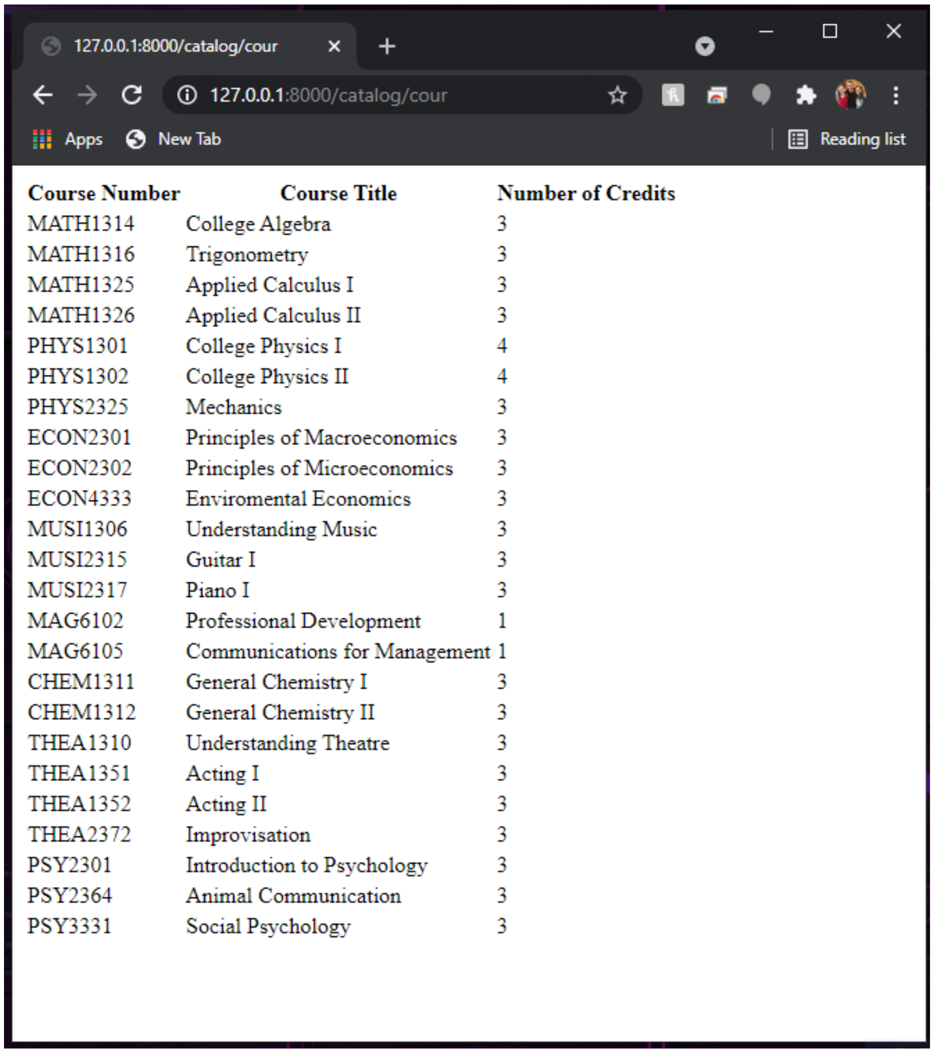
| USERNAME | EMAIL ADDRESS | FIRST NAME | LAST NAME | STAFF STATUS |
|---|---|---|---|---|
| 1542381667 | | | | ✗ |
| 1542381670 | | | | ✗ |
| Admin | abk170230@utdallas.edu | | | ✓ |
| Instructor | | | | ✗ |
| Staff | | | | ✗ |

5 users

Since most users cannot use the administration site, we need to build a general access front end for most users. We do this by adding some html files in a registration folder within our project. These files house templates and commands for what to do login in, login out, and when a password needs to be reset. Because of the scope of this project and because this server is only local, the password reset email system is not completed. However, the files are there for display purposes and so that Django can reference them if needed. This provides us with a very basic front for our database website.

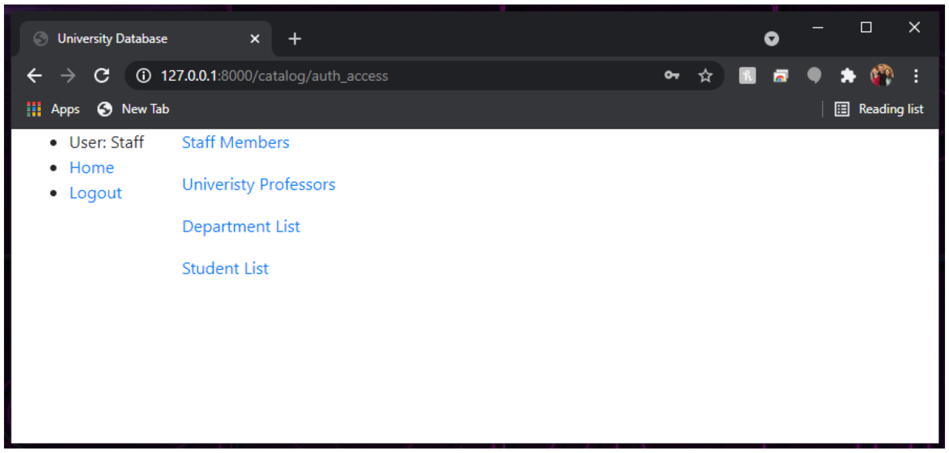


This is our next level of security. On our front page, we can have information about our university or other general access information. For our example, I have included a tab that will take us to a course catalog. This will allow non-users to see what classes this university offers without needing to be logged in.



| Course Number | Course Title | Number of Credits |
|---|---|---|
| MATH1314 | College Algebra | 3 |
| MATH1316 | Trigonometry | 3 |
| MATH1325 | Applied Calculus I | 3 |
| MATH1326 | Applied Calculus II | 3 |
| PHYS1301 | College Physics I | 4 |
| PHYS1302 | College Physics II | 4 |
| PHYS2325 | Mechanics | 3 |
| ECON2301 | Principles of Macroeconomics | 3 |
| ECON2302 | Principles of Microeconomics | 3 |
| ECON4333 | Enviromental Economics | 3 |
| MUSI1306 | Understanding Music | 3 |
| MUSI2315 | Guitar I | 3 |
| MUSI2317 | Piano I | 3 |
| MAG6102 | Professional Development | 1 |
| MAG6105 | Communications for Management | 1 |
| CHEM1311 | General Chemistry I | 3 |
| CHEM1312 | General Chemistry II | 3 |
| THEA1310 | Understanding Theatre | 3 |
| THEA1351 | Acting I | 3 |
| THEA1352 | Acting II | 3 |
| THEA2372 | Improvisation | 3 |
| PSY2301 | Introduction to Psychology | 3 |
| PSY2364 | Animal Communication | 3 |
| PSY3331 | Social Psychology | 3 |

Finally, our third type of user is a member of this university. They should have access to a verity of university specific information that is not general public information. Thus, once you have been logged in as a member, I have implemented four pages as an example of what type of information we can offer.



- User: Staff
- Home
- Logout

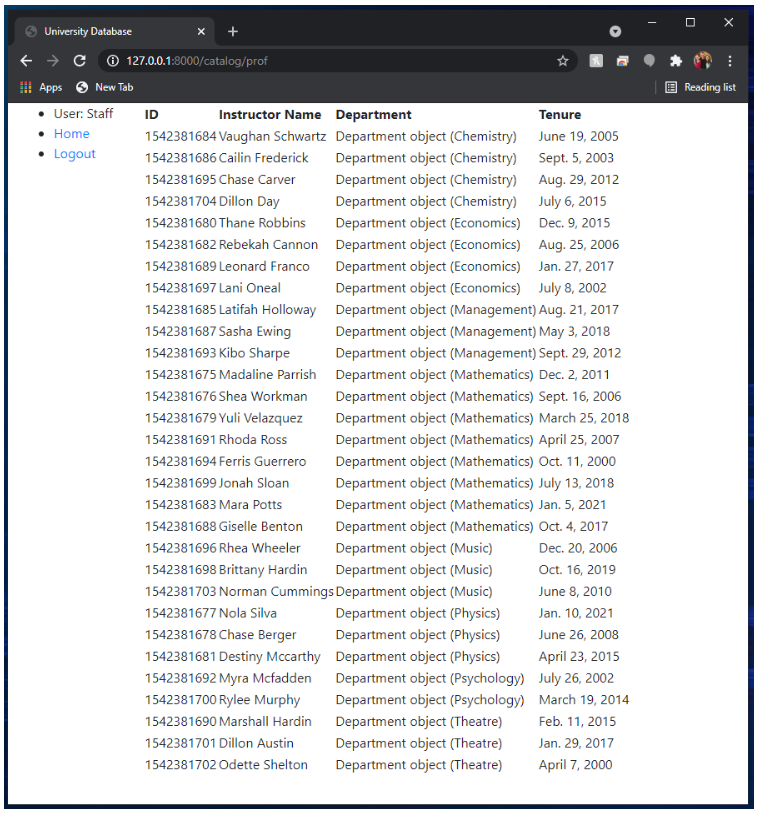
Staff Members

Univeristy Professors

Department List

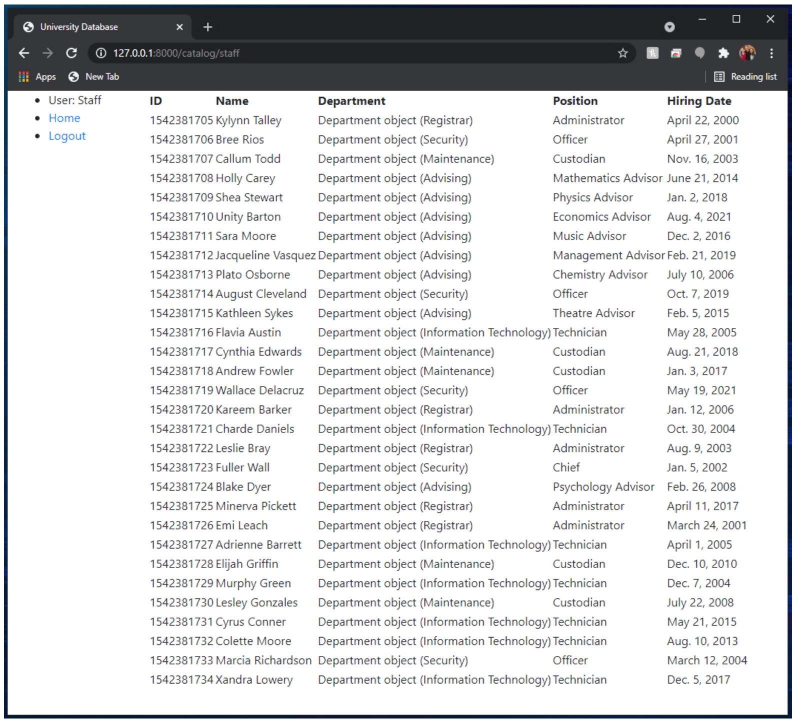
Student List

The first two of these tables houses the information in our Staff and Professor relations. This is information that can be utilized as an employee look up. Since we don't have that many employee observations currently, we simply have them listed out in tables for reference. The first image is our professors and the second is the staff list.



- User: Staff
- Home
- Logout

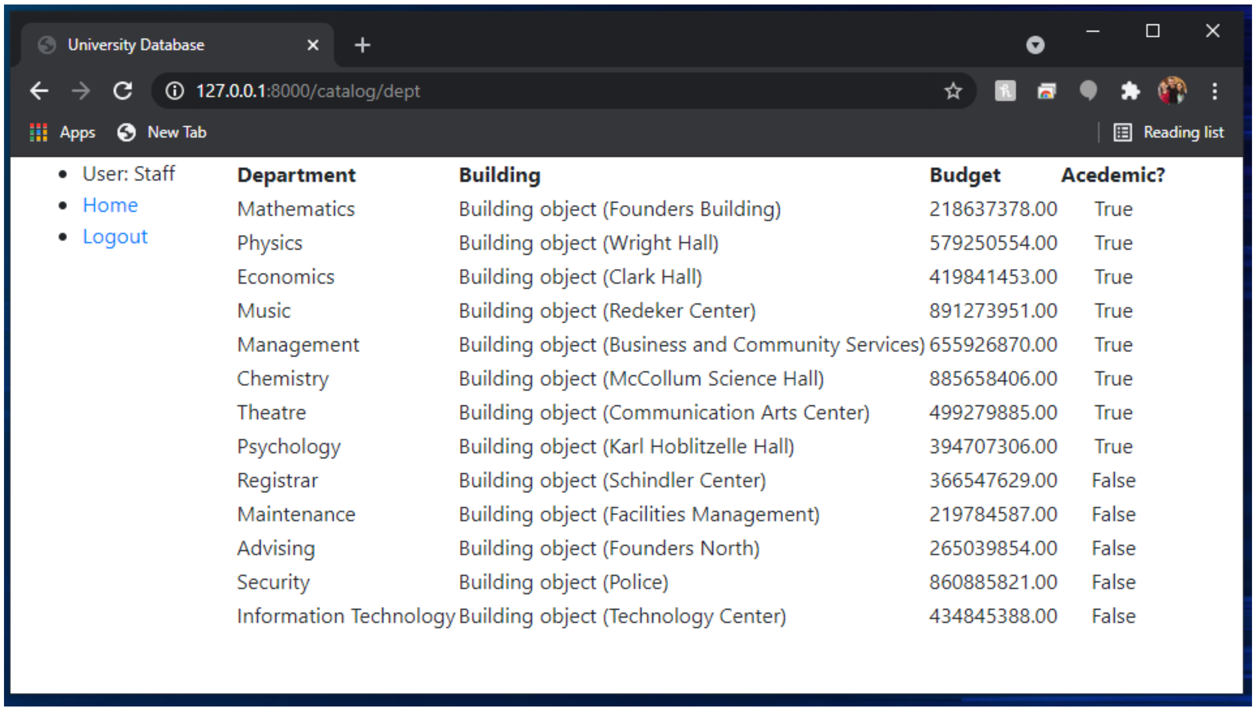
| ID | Instructor Name | Department | Tenure |
|---|---|---|---|
| 1542381684 | Vaughan Schwartz | Department object (Chemistry) | June 19, 2005 |
| 1542381686 | Cailin Frederick | Department object (Chemistry) | Sept. 5, 2003 |
| 1542381695 | Chase Carver | Department object (Chemistry) | Aug. 29, 2012 |
| 1542381704 | Dillon Day | Department object (Chemistry) | July 6, 2015 |
| 1542381680 | Thane Robbins | Department object (Economics) | Dec. 9, 2015 |
| 1542381682 | Rebekah Cannon | Department object (Economics) | Aug. 25, 2006 |
| 1542381689 | Leonard Franco | Department object (Economics) | Jan. 27, 2017 |
| 1542381697 | Lani Oneal | Department object (Economics) | July 8, 2002 |
| 1542381685 | Latifah Holloway | Department object (Management) | Aug. 21, 2017 |
| 1542381687 | Sasha Ewing | Department object (Management) | May 3, 2018 |
| 1542381693 | Kibo Sharpe | Department object (Management) | Sept. 29, 2012 |
| 1542381675 | Madaline Parrish | Department object (Mathematics) | Dec. 2, 2011 |
| 1542381676 | Shea Workman | Department object (Mathematics) | Sept. 16, 2006 |
| 1542381679 | Yuli Velazquez | Department object (Mathematics) | March 25, 2018 |
| 1542381691 | Rhoda Ross | Department object (Mathematics) | April 25, 2007 |
| 1542381694 | Ferris Guerrero | Department object (Mathematics) | Oct. 11, 2000 |
| 1542381699 | Jonah Sloan | Department object (Mathematics) | July 13, 2018 |
| 1542381683 | Mara Potts | Department object (Mathematics) | Jan. 5, 2021 |
| 1542381688 | Giselle Benton | Department object (Mathematics) | Oct. 4, 2017 |
| 1542381696 | Rhea Wheeler | Department object (Music) | Dec. 20, 2006 |
| 1542381698 | Brittany Hardin | Department object (Music) | Oct. 16, 2019 |
| 1542381703 | Norman Cummings | Department object (Music) | June 8, 2010 |
| 1542381677 | Nola Silva | Department object (Physics) | Jan. 10, 2021 |
| 1542381678 | Chase Berger | Department object (Physics) | June 26, 2008 |
| 1542381681 | Destiny Mccarthy | Department object (Physics) | April 23, 2015 |
| 1542381692 | Myra Mcfadden | Department object (Psychology) | July 26, 2002 |
| 1542381700 | Rylee Murphy | Department object (Psychology) | March 19, 2014 |
| 1542381690 | Marshall Hardin | Department object (Theatre) | Feb. 11, 2015 |
| 1542381701 | Dillon Austin | Department object (Theatre) | Jan. 29, 2017 |
| 1542381702 | Odette Shelton | Department object (Theatre) | April 7, 2000 |



- User: Staff
- Home
- Logout

| ID | Name | Department | Position | Hiring Date |
|---|---|---|---|---|
| 1542381705 | Kylynn Talley | Department object (Registrar) | Administrator | April 22, 2000 |
| 1542381706 | Bree Rios | Department object (Security) | Officer | April 27, 2001 |
| 1542381707 | Callum Todd | Department object (Maintenance) | Custodian | Nov. 16, 2003 |
| 1542381708 | Holly Carey | Department object (Advising) | Mathematics Advisor | June 21, 2014 |
| 1542381709 | Shea Stewart | Department object (Advising) | Physics Advisor | Jan. 2, 2018 |
| 1542381710 | Unity Barton | Department object (Advising) | Economics Advisor | Aug. 4, 2021 |
| 1542381711 | Sara Moore | Department object (Advising) | Music Advisor | Dec. 2, 2016 |
| 1542381712 | Jacqueline Vasquez | Department object (Advising) | Management Advisor | Feb. 21, 2019 |
| 1542381713 | Plato Osborne | Department object (Advising) | Chemistry Advisor | July 10, 2006 |
| 1542381714 | August Cleveland | Department object (Security) | Officer | Oct. 7, 2019 |
| 1542381715 | Kathleen Sykes | Department object (Advising) | Theatre Advisor | Feb. 5, 2015 |
| 1542381716 | Flavia Austin | Department object (Information Technology) | Technician | May 28, 2005 |
| 1542381717 | Cynthia Edwards | Department object (Maintenance) | Custodian | Aug. 21, 2018 |
| 1542381718 | Andrew Fowler | Department object (Maintenance) | Custodian | Jan. 3, 2017 |
| 1542381719 | Wallace Delacruz | Department object (Security) | Officer | May 19, 2021 |
| 1542381720 | Kareem Barker | Department object (Registrar) | Administrator | Jan. 12, 2006 |
| 1542381721 | Charde Daniels | Department object (Information Technology) | Technician | Oct. 30, 2004 |
| 1542381722 | Leslie Bray | Department object (Registrar) | Administrator | Aug. 9, 2003 |
| 1542381723 | Fuller Wall | Department object (Security) | Chief | Jan. 5, 2002 |
| 1542381724 | Blake Dyer | Department object (Advising) | Psychology Advisor | Feb. 26, 2008 |
| 1542381725 | Minerva Pickett | Department object (Registrar) | Administrator | April 11, 2017 |
| 1542381726 | Emi Leach | Department object (Registrar) | Administrator | March 24, 2001 |
| 1542381727 | Adrienne Barrett | Department object (Information Technology) | Technician | April 1, 2005 |
| 1542381728 | Elijah Griffin | Department object (Maintenance) | Custodian | Dec. 10, 2010 |
| 1542381729 | Murphy Green | Department object (Information Technology) | Technician | Dec. 7, 2004 |
| 1542381730 | Lesley Gonzales | Department object (Maintenance) | Custodian | July 22, 2008 |
| 1542381731 | Cyrus Conner | Department object (Information Technology) | Technician | May 21, 2015 |
| 1542381732 | Colette Moore | Department object (Information Technology) | Technician | Aug. 10, 2013 |
| 1542381733 | Marcia Richardson | Department object (Security) | Officer | March 12, 2004 |
| 1542381734 | Xandra Lowery | Department object (Information Technology) | Technician | Dec. 5, 2017 |

The department list is very similar to the previous two. It lists the information in our department relation. This includes relevant items for a university employee such as the building where to find the department, its budget, and whether or not it employs any professors (indicated by the academic column.



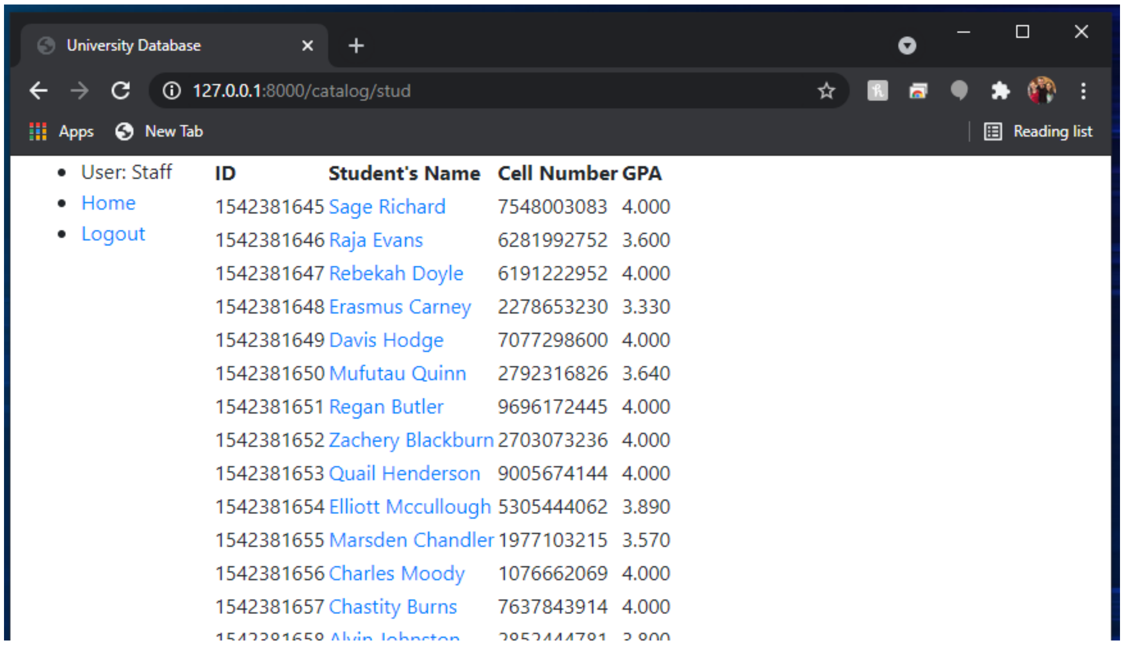
University Database

127.0.0.1:8000/catalog/dept

- User: Staff
- Home
- Logout

| Department | Building | Budget | Acedemic? |
|---|---|---|---|
| Mathematics | Building object (Founders Building) | 218637378.00 | True |
| Physics | Building object (Wright Hall) | 579250554.00 | True |
| Economics | Building object (Clark Hall) | 419841453.00 | True |
| Music | Building object (Redeker Center) | 891273951.00 | True |
| Management | Building object (Business and Community Services) | 655926870.00 | True |
| Chemistry | Building object (McCollum Science Hall) | 885658406.00 | True |
| Theatre | Building object (Communication Arts Center) | 499279885.00 | True |
| Psychology | Building object (Karl Hoblitzelle Hall) | 394707306.00 | True |
| Registrar | Building object (Schindler Center) | 366547629.00 | False |
| Maintenance | Building object (Facilities Management) | 219784587.00 | False |
| Advising | Building object (Founders North) | 265039854.00 | False |
| Security | Building object (Police) | 860885821.00 | False |
| Information Technology | Building object (Technology Center) | 434845388.00 | False |

Last but not least, is our student list. This page is special as it is an example of how the navigation of our site could work more effectively. The list of students is fairly standard. It allows staff to look up the student, find their contact information, their student ID number and also their current grade point average.



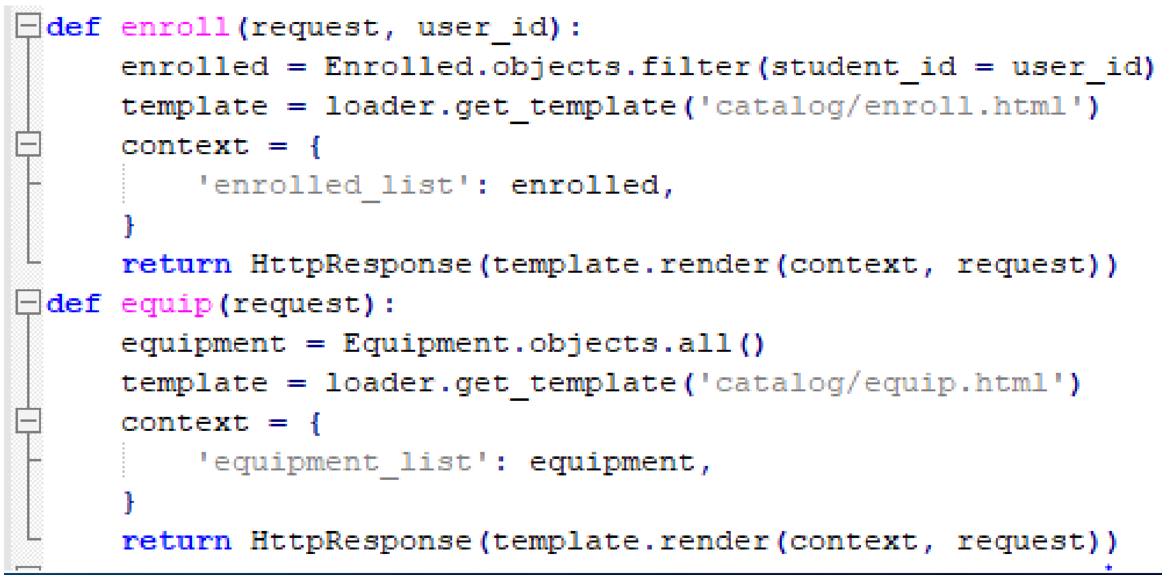
University Database

127.0.0.1:8000/catalog/stud

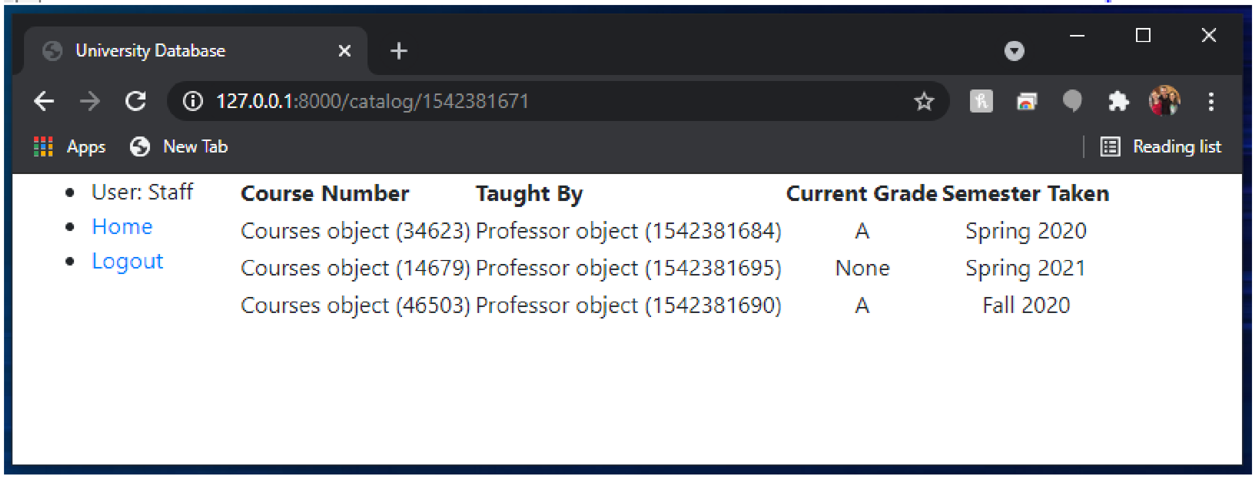
- User: Staff
- Home
- Logout

| ID | Student's Name | Cell Number | GPA |
|---|---|---|---|
| 1542381645 | Sage Richard | 7548003083 | 4.000 |
| 1542381646 | Raja Evans | 6281992752 | 3.600 |
| 1542381647 | Rebekah Doyle | 6191222952 | 4.000 |
| 1542381648 | Erasmus Carney | 2278653230 | 3.330 |
| 1542381649 | Davis Hodge | 7077298600 | 4.000 |
| 1542381650 | Mufutau Quinn | 2792316826 | 3.640 |
| 1542381651 | Regan Butler | 9696172445 | 4.000 |
| 1542381652 | Zachery Blackburn | 2703073236 | 4.000 |
| 1542381653 | Quail Henderson | 9005674144 | 4.000 |
| 1542381654 | Elliott Mccullough | 5305444062 | 3.890 |
| 1542381655 | Marsden Chandler | 1977103215 | 3.570 |
| 1542381656 | Charles Moody | 1076662069 | 4.000 |
| 1542381657 | Chastity Burns | 7637843914 | 4.000 |
| 1542381658 | Alvin Johnston | 2852444781 | 3.800 |

Beyond this listing of our Student relation, we can also look at more specific information about an individual student. Each one of our student's names is a clickable link that will take us to a filtered subset of our Enrolled relation. We do this by filtering the observations we pull from our enrolled list by a user id so that we only access from the database relevant information.



```
def enroll(request, user_id):
    enrolled = Enrolled.objects.filter(student_id = user_id)
    template = loader.get_template('catalog/enroll.html')
    context = {
        'enrolled_list': enrolled,
    }
    return HttpResponse(template.render(context, request))
def equip(request):
    equipment = Equipment.objects.all()
    template = loader.get_template('catalog/equip.html')
    context = {
        'equipment_list': equipment,
    }
    return HttpResponse(template.render(context, request))
```



- User: Staff
- Home
- Logout

| Course Number | Taught By | Current Grade | Semester Taken |
|---|---|---|---|
| Courses object (34623) | Professor object (1542381684) | A | Spring 2020 |
| Courses object (14679) | Professor object (1542381695) | None | Spring 2021 |
| Courses object (46503) | Professor object (1542381690) | A | Fall 2020 |

Overall, this database and other like it is primarily of the organization and quick reference of data. None of the information we have simulated and stored needs to be charted or displayed and very little of it is useful for analysis. However, quick reference of this data is vital for the operation of the business. There are a few limitation in the current design that are worth mentioning.

First, because the Enrolled relation uses course ID and student ID as primary variables, this allows students to only take courses once in their entire student careers. Most universities allow for a student to take a class around three times. This could be fixed by utilizing the general id variable we add for Django as our primary key in Postgres. However, a far better option is to create a new relation that stores all previous class information. This would require a trigger at the end of each semester where all currently enrolled classes are moved from the Enrolled relation to a Taken relation, for example. We could then clean up some of the data as we move it over, for example deleting the section number of the class since it will likely not be needed. This would enable us to use a generic ID for the Taken relation but still utilize our Enrolled relation as is.

The next improvement I would like to mention is in regard to the user interface. All of these aspects are do able with more time and know how. Beyond the addition and organization of the website, there are a few key authentication and visual aspects that need to be changed. Firstly, because I utilized foreign key fields in Django, when I display the entries of my database, it includes the object type beforehand. For example, above you can see all professors are listed as: Professor object (##########). It would be much better to just list the professor id number instead. Similarly, while I can join relations in Postgres with SQL, it does not seem as easy in Django. Thus, doing a three relation join in order to also display the course and professor's name in the example above is quite challenging. Ideally, all these observations would list only information useful to the human reader, thus replacing ID for names most of the time. Lastly, as I currently have it, the user authentication system does not discriminate between users. This means that students and staff members can view the same pages. For security and practicality reasons, this should not be the case. I was not able to solve this issue in the time that I had, but I know, of course, it is possible and would like to add it in the future.