

CMS Login System  
CST363: Intro to database Systems  
Midterm Project

Mark Mariscal  
Christopher Piwarski  
Wais Robleh

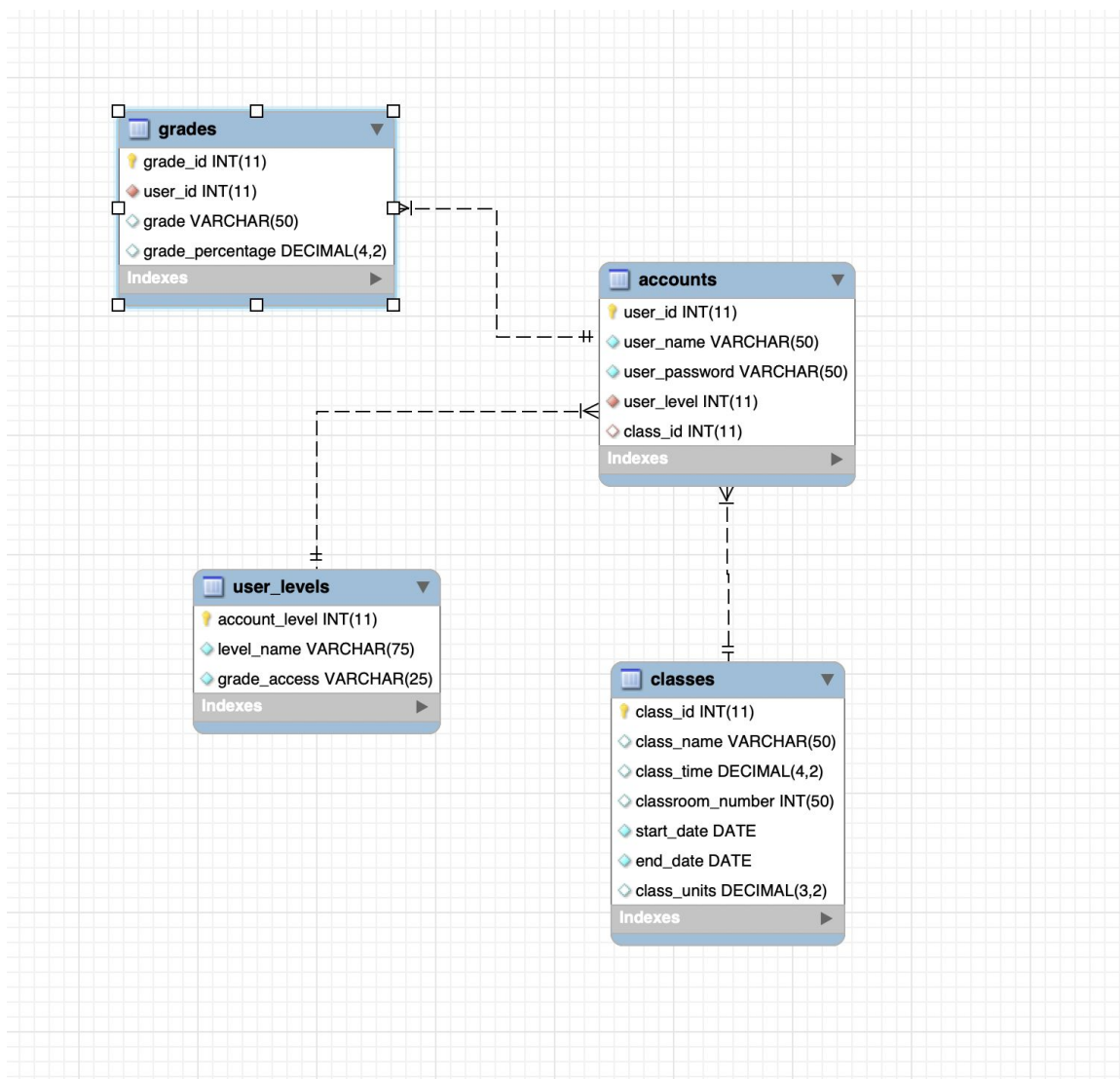
## Midterm Project

For our midterm project we chose to make a course management system (CMS) similar to iLearn. The goal is to create a web application that users log into; based on their user level (student, teacher, administrator), they receive different informational views relevant to their needs. Students will receive access to their course list, and individual grade. Teachers will be able to view grades for all their students. The administrator will be able to create new accounts, but not view courses or grades. The project is centered around displaying different information based on account access levels. This type of project would provide ample application of database technology.

Our application performs several different database operations to provide the user with the before mentioned feature set. SQL queries are used to confirm user identity during login. Passwords are compared against usernames for authentication. For the three other features of our website, queries are used to determine the type of user, and what information they have access to. Once the user type is known, more queries are conducted to display to them the appropriate information. Cookies are implemented so the user does not have to re-authenticate during their session.

The database for this project features a table for account levels, classes, accounts, and grades. The primary and foreign key relationship will be used to connect all the data. An Entity-Relationship Diagram (ERD) is attached to this document to show an outline of our data structure. Also attached are images that display the various pages of our website. Constraints are generated during database creation to guarantee the foreign key relationship. All of the source files can be retrieved from our project github account at [https://github.com/evilmurries/cst363\\_midterm](https://github.com/evilmurries/cst363_midterm).

One of the requirements for this project is that our database adheres to Third Normal Form (3NF). According to the course text, there are three criteria the database needs to conform to in order to be considered in 3NF. Each column should contain a unique scalar value. The columns for each table should only depend on the primary key. Referring to the ERD, one can see that all the column data is unique save for the foreign keys. Each account references another table in order to retrieve data specific to courses, grades, and user privileges.



Entity-Relationship Diagram for our project.

We hope that you enjoy our project as much as we enjoyed creating it. It was an excellent opportunity to implement the lessons we have learned from the previous course modules. Some website features required practicing more advanced SQL skills such as subquery and complex joins. The project was also an opportunity to study and implement HTML and Python Common Gateway Interface (CGI) programming. Python3 was the tool that was used to implement the database interactions beyond the initial HTML homepage. HTML code was used to add style to the various pages as well as create display features such as tables. The project has provided experience that will prove invaluable in our future careers. Attached are screen captures that provide a walk through of our course site. Thank you.

## UPDATE: Feedback for Part One

Our original code appended query strings using string formatting in the Python scripts.

This presents a security vulnerability. Our Feedback was to change our cgi scripts so that they passed the user input in as arguments instead. This change was implemented and verified on 2/1/2019.



Nikola Petkov

—

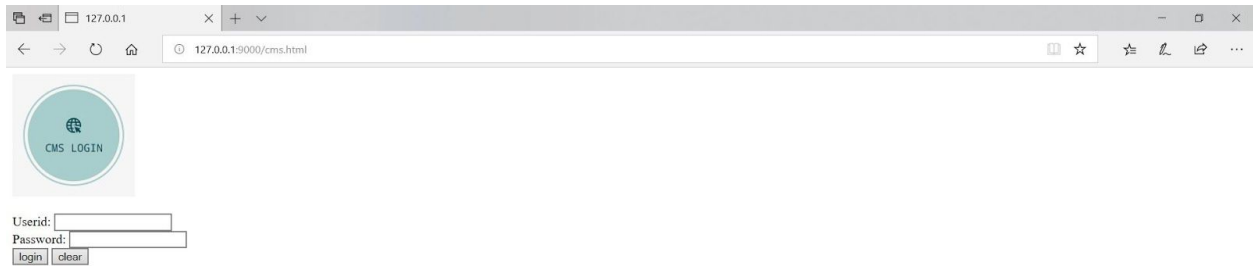
You should never directly append user input to a sql query.

```
ul_query = 'SELECT user_name, user_password FROM accounts WHERE user_name = \''%s\'" % username
cursor1.execute(ul_query)
```

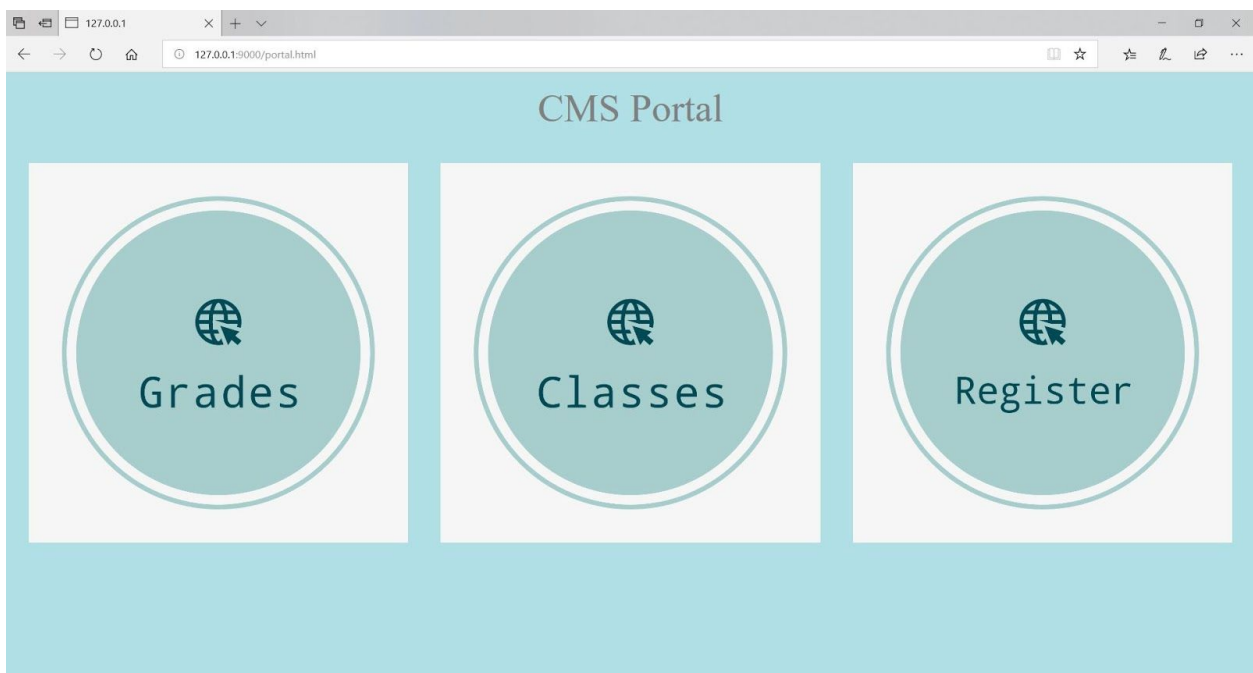
The above should instead be using parameters like you have done in register2.py.

```
ul_query = 'SELECT user_name, user_password FROM accounts WHERE user_name = %s'
cursor1.execute(ul_query, (username,))
```

This is to prevent SQL injection attacks. Please fix these for part 2 of the project.



The login home page - cms.html



The portal page when the user logs in - login.py

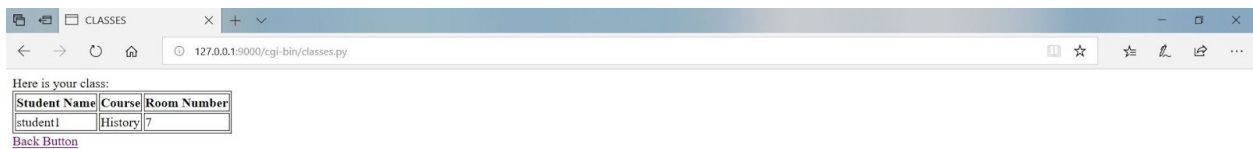


The screenshot shows a web browser window with the address bar displaying "127.0.0.1:9000/cgi-bin/grades.py". The page content includes the text "Here is your grade:" followed by a table with three columns: "Student Name", "Letter Grade", and "Percentage". The table contains one row with the values "student1", "A", and "95.38". Below the table is a link labeled "Back Button".

Student Name	Letter Grade	Percentage
student1	A	95.38

[Back Button](#)

A grade printout for a student user - grades.py



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:9000/cgi-bin/classes.py". The page content includes the text "Here is your class:" followed by a table with three columns: "Student Name", "Course", and "Room Number". The table contains one row with the values "student1", "History", and "7". Below the table is a link labeled "Back Button".

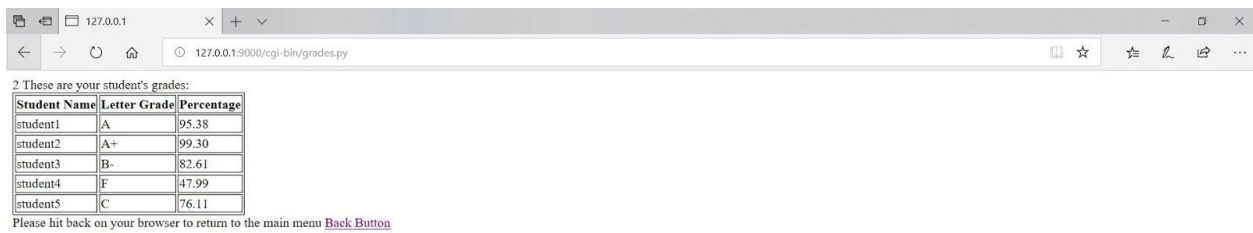
Student Name	Course	Room Number
student1	History	7

[Back Button](#)

A course printout for a student user - classes.py



A student attempting to register new users - register.py



A teacher viewing their student's grades - grades.py





A course print out for a teacher user - classes.py



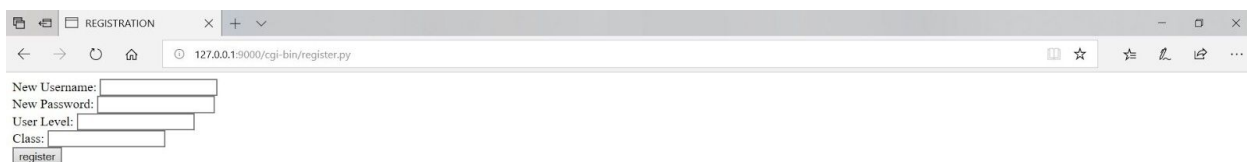
A teacher attempting to access registration - register.py



An admin attempting to view student's grades - grades.py



An admin attempting to view student's classes - classes.py



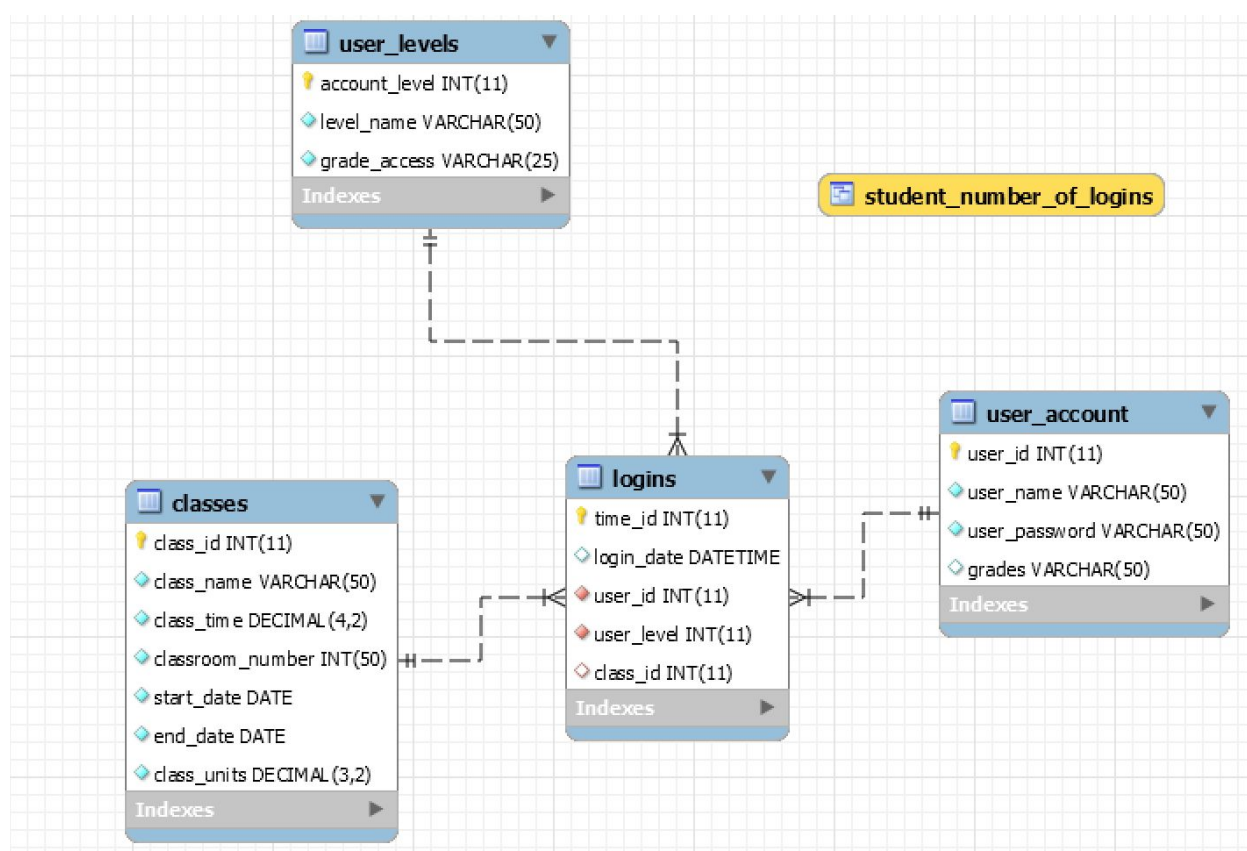
A screenshot of a web browser window. The title bar shows a single tab labeled "REGISTRATION". The address bar contains the URL "127.0.0.1:9000/cgi-bin/register.py". The page content consists of a registration form with the following fields and labels:

- New Username:
- New Password:
- User Level:
- Class:
- 

An admin attempting to register new users - register.py

## Part Two OLAP Design

The second part of the project was to create an OLAP (Online Analytical Processing) database for the OLTP database created in part one. Our approach required a change in our original database in order to provide the kind of information that would be useful to an OLAP user. We incorporated a new table to track cms user logins and generate a time-stamp every time they use the system. This was necessary as it would become the basis for our fact table. The image below is our OLAP database:



Our design features a star schema keeping in line with established theory. The fact table “logins” contains a list of every instance a user logs into our CMS system. There is a **time\_id** primary key, a time stamp for the log in, as well as user id, their user level, and a class id if applicable. From this information, all other details about the logged in user can be accessed via the dimension tables. There are dimension tables for classes, user accounts, and user levels. A

view, "student\_number\_of\_logins," is also provided which displays the amount of times all the students have logged in. This view was designed for a teacher type who would find such information relevant for roll keeping, or an administrator that needs to drop inactive students. Much like an OLAP database whose fact table tracks every purchase, this database tracks every user log in transaction. Individual SQL scripts were written to create the OLAP database, load the data from the OLTP database, and for specific queries a business information (BI) user might use. Our project does not necessarily focus on a "for profit" business model. As a result, the BI queries are more keeping in line with what school administration might need to gauge their effectiveness as an academic institution, or a database administrator might need to properly provide for internet traffic.

Another aspect of this project was to address feedback about part one. As mentioned in the first half of this document, we received feedback requiring us to fix how our Python cgi scripts were handling user inputs. Before conducting any OLAP operations we addressed this feedback first. Our scripts were creating SQL strings in their entirety and passing them onto the cursor for execution. This created opportunities for a "SQL injection" attack. To eliminate this threat we passed all user input in as an argument to the cursor.execute() command instead as instructed.

Part two of the project taught our team about the importance of considering the database warehouse and it's future applications during the design of the operations database. That was the primary challenge we received from part two of the midterm project. Another was generating a fact table primarily filled with foreign keys without causing update issues. In the end we followed the advice of our professor to either disable the update warnings that were generated or to remove the constraints from the database creation script. Another major lesson is to collect more data, no matter how frivolous it may seem at the time. For instance, although you can

determine the semester a course in our system took place in by using the course start and end days, maintaining a semester column would provide a much easier search experience for the business users. It is important to create more options for retrieving data, sometimes to the detriment of storage. These valuable lessons make this part of the midterm project a success in terms of preparing us for the future.