CS 411 SU19 Group 7 Final Report

SQL help

Mrinoy Banerjee (netID: mrinoyb2)
Hassan Goodarzifarahani (netID: hassang2)
Eric Wang (netID: wcwang2)
Steven Zhou (netID: szhou54)

University of Illinois at Urbana-Champaign
Aug 1. 2019.

1. Description:

   Our project helps students practice SQL problems by breaking down their queries into smaller steps, and displaying the output tables from each step. This can guide students by helping them visualize what each step of their query does so they can adjust accordingly if their final result does not match the expected result. The number of steps is determined by their method of querying, which is described later in the advanced function. From the instructor side, we have a question bank manager, where they can modify, add or delete questions for students to practice and a student manager where they can filter and search the students that are signed up to the website.

2. Problem Solved:

   Due to the natural abstraction of SQL queries, beginners may have a hard time visualizing their queries, and not able to understand their final output. Our project serves to break complicated queries into smaller steps, whose result will be displayed accordingly. After brainstorming through various ideas for our advanced functionality, we drew inspiration from the idea of the step by step solutions to math problems from Symbolab and Wolfram Alpha. This closely resonated with our struggles with homework 1 on PrairieLearn, where we struggled to break down questions into smaller steps.

   For instance, the following query

   $$SELECT\ Name\ FROM\ Employees\ WHERE\ Salary > 1000;$$

   will be broken down into multiple steps

   - $SELECT * FROM\ Employees$
   - $SELECT * FROM\ Employees\ WHERE\ Salary > 1000;$
   - $SELECT\ Name\ FROM\ Employees\ WHERE\ Salary > 1000;$

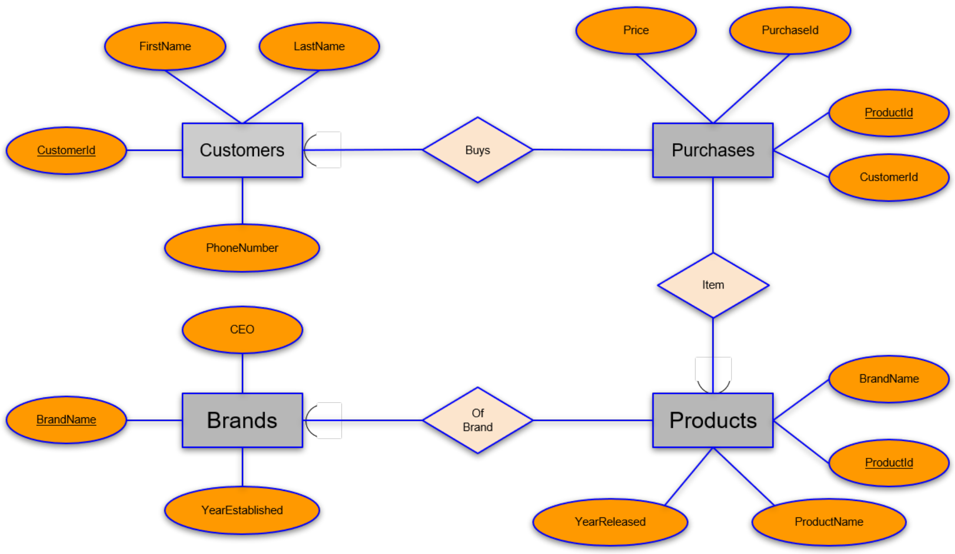
3. Data:

   We created one database to store user information, and the other one to store the tables required for the questions.

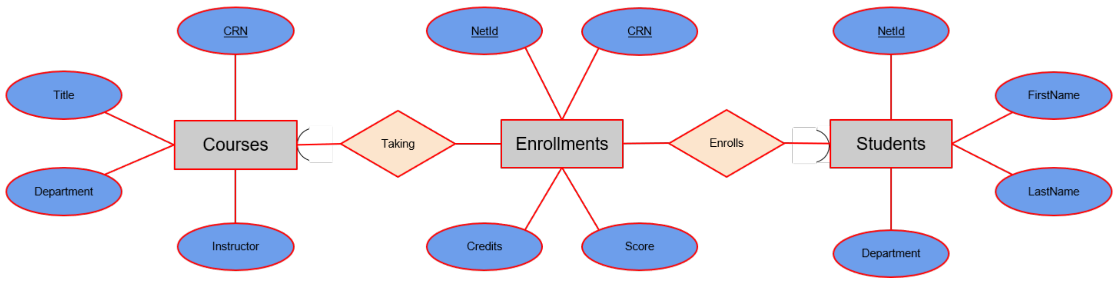
   - User information: this database includes two tables. The first table (users) contains details of every signed up user with basic information including name, email, password, year and major. This information is useful for the instructor to search through the student using the application and get their basic information. The second table (current_question) stores the user ID and the question the user was working on most recently. By joining the two tables, the instructor can monitor the student progress.
   - Question Database: this database includes 8 tables. One of the tables will be served to store the question ID, question and the solution. We also designed an user interface for the instructor to insert, update or delete records from the database. This table originally has 6 questions, which are extracted from the first PrairieLearn assignment, but excluding questions that asks students to update or delete records from the table, because we do not want the student to make changes to our database. The other 7 tables are extracted from the first PrairieLearn assignment, with additional mock data to increase the size of data.

4. ER Diagram & Schema:

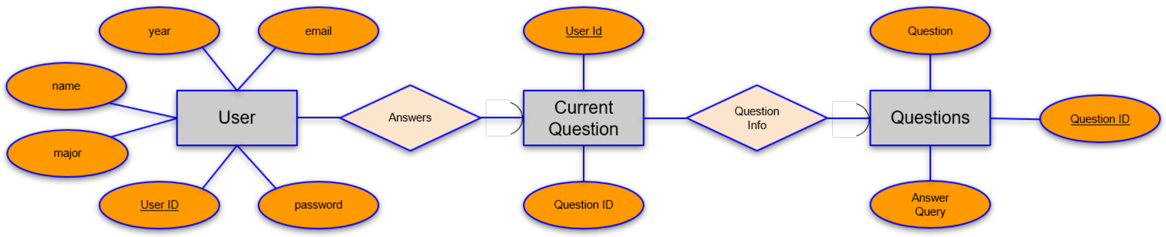
ER diagram for question 1 to 3 in the question bank:



ER diagram for question 4 to 6 in the question bank:



ER diagram for question and user database:



Relational Schema for question 1 to 3 in the question bank:

- Students (NetId, FirstName, LastName, Department)
- Enrollments (Students.NetId, Courses.CRN, Credits, Score)
- Courses (CRN, Title, Department, Instructor)

Relational Schema for question 4 to 6 in the question bank:

- Customers (CustomerId, FirstName, LastName, PhoneNumber)
- Purchases (PurchaseId, Customers.CustomerId, Products.ProductId, Price)
- Products (ProductId, Brand.BrandName, ProductName, Year-Released)
- Brands (BrandName, YearEstablished, CEO)

Relational Schema for question and user database:

- User (UserId, email, password, name, year, major)
- CurrentQuestion (User.UserId, Questions.questionId)
- Questions (QuestionId, Question, AnswerQuery)

5. Data collected:

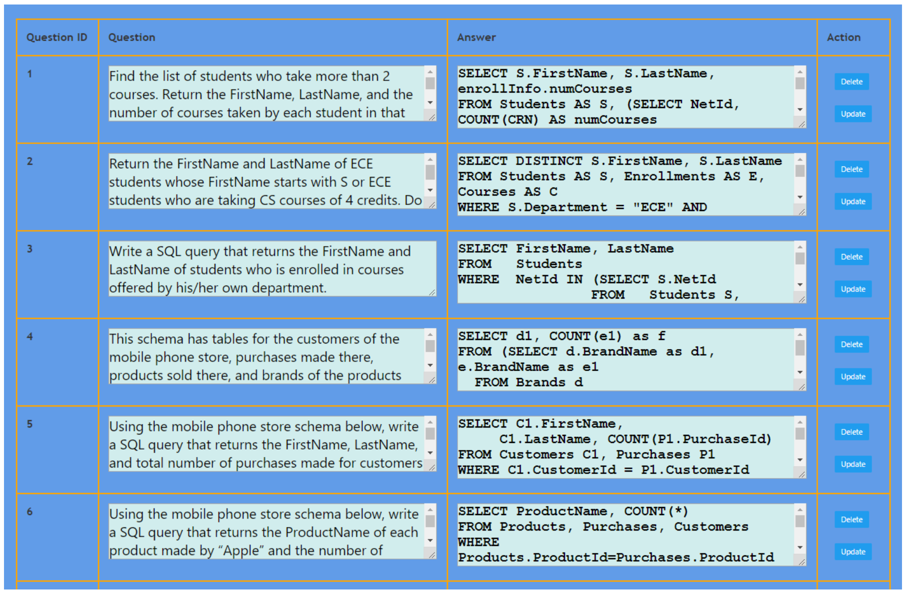
   Since our primary goal was to give useful guidance for SQL beginners in future semester, we decide to work with actual data from the class. Working with problems we are familiar with can enhance our relevancy to the course itself, and develop basic and advanced functions that can directly assist CS 411 students. The data for questions are extracted from the first PrairieLearn assignment, and converted to a proper table and eventually a CSV file. For some of these tables, we included mock entries to make the size optimal. Once these CSV files were created, we imported them to PhpMyAdmin and created tables through them.

6. Functionality:

   SQL helper functions as a tool to break down student queries into smaller logical steps, which allow students to tell where they are diverting from what they are expecting, instead of comprehending the entire query at once. This website includes a login / signup page, parser, question bank and student manager. Students can create an account and sign in using the signup page. The parser is our advanced function site, which includes the question and its schema on top, and a text box for users to input their data. Syntax error will be shown if exists, otherwise the resulting table and the correctness will be shown at the right, followed by the step query / table and the expected table. The student can move to the next question by clicking the next button. The question bank is the site where instructor can insert, delete or update question and answers. The student manager page enables the instructor to navigate the progress of each student, and search certain information using the filter.

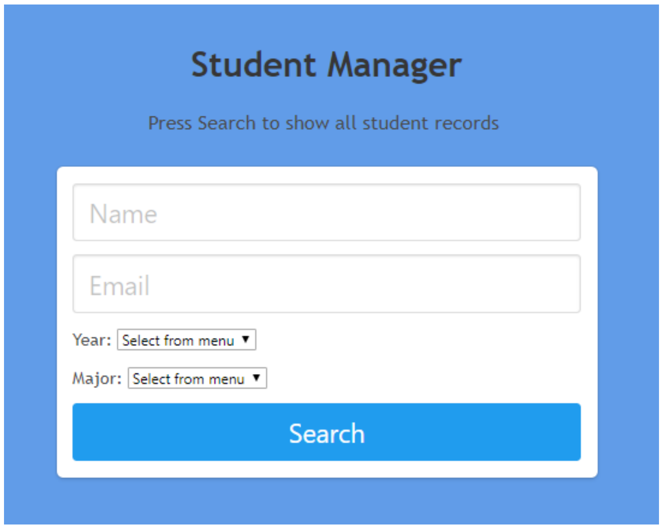
7. Basic Function:

   One of the basic functions we implemented was the question bank page, in which the instructor can insert, update or delete question and answers. For each operation, we will be executing a SQL query to modify our database if needed. We will also make sure that the question ID is always in order, starting from 1, so that it would not cause any conflict when the student tries to navigate to the next question. When the instructor executes a delete operation, it first executes a deletion of that certain row, then decreases the question ID of the questions which have a higher question ID by 1. When the instructor makes an insert operation, it will assign a question ID that is one more than the highest question ID in the question bank. The user interface is as followed:



Question ID	Question	Answer	Action
1	Find the list of students who take more than 2 courses. Return the FirstName, LastName, and the number of courses taken by each student in that	SELECT S.FirstName, S.LastName, enrollInfo.numCourses FROM Students AS S, (SELECT NetId, COUNT(CRN) AS numCourses	Delete Update
2	Return the FirstName and LastName of ECE students whose FirstName starts with S or ECE students who are taking CS courses of 4 credits. Do	SELECT DISTINCT S.FirstName, S.LastName FROM Students AS S, Enrollments AS E, Courses AS C WHERE S.Department = "ECE" AND	Delete Update
3	Write a SQL query that returns the FirstName and LastName of students who is enrolled in courses offered by his/her own department.	SELECT FirstName, LastName FROM Students WHERE NetId IN (SELECT S.NetId FROM Students S,	Delete Update
4	This schema has tables for the customers of the mobile phone store, purchases made there, products sold there, and brands of the products	SELECT d1, COUNT(e1) as f FROM (SELECT d.BrandName as d1, e.BrandName as e1 FROM Brands d	Delete Update
5	Using the mobile phone store schema below, write a SQL query that returns the FirstName, LastName, and total number of purchases made for customers	SELECT C1.FirstName, C1.LastName, COUNT(P1.PurchaseId) FROM Customers C1, Purchases P1 WHERE C1.CustomerId = P1.CustomerId	Delete Update
6	Using the mobile phone store schema below, write a SQL query that returns the ProductName of each product made by "Apple" and the number of	SELECT ProductName, COUNT(*) FROM Products, Purchases, Customers WHERE Products.ProductId=Purchases.ProductId	Delete Update

Another basic function we implemented is in the student manager page. It helps the instructor find students in the database that match the desired filter attributes, including name, email, year and major. The entires also consist fill-in text boxes and drop down menus, in which information is sent to a python script and concatenated in the $WHERE$ clause in the SQL query. The user interface is as followed:



Student Manager

Press Search to show all student records

Name

Email

Year: Select from menu

Major: Select from menu

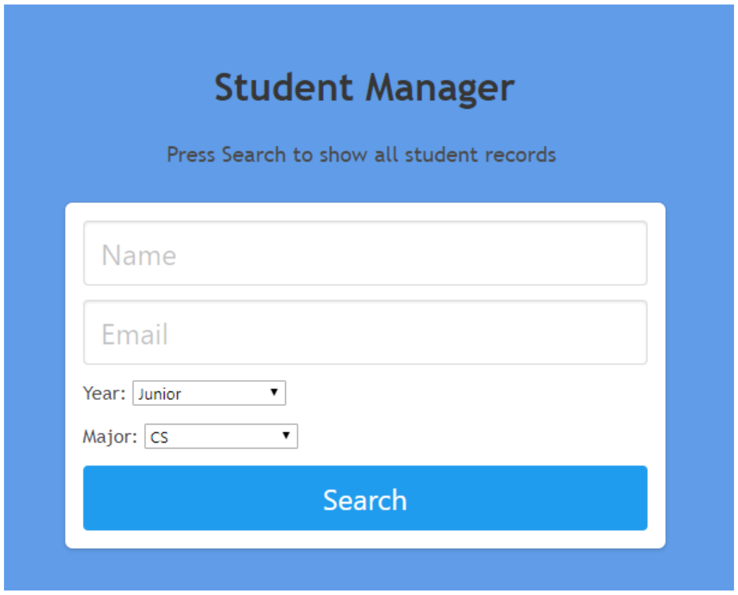
Search

8. SQL Code Snippet:

The following snippet servers to return the student information with certain filters:

```
def get_search (student_name, student_email, student_year, student_major):
    queryresult = ’SELECT name, email, year, major FROM user’
    selection = ”
    if student_name:
        selection += ”AND name = ’” + student_name + ”’ ”
    if student_email:
        selection += ”AND email = ’” + student_email + ”’ ”
    if student_year != ’any’:
        selection += (”AND year = ’” + student_year + ”’ ”)
    if student_major != ’any1’:
        selection += (”AND major = ’” + student_major + ”’ ”)
    if len(selection) > 0:
        queryresult += ’WHERE ’ + selection[4: -1]
    results = run_query(queryresult, ’sequelhelp_users’)
    ret = [ ]
    for r in results:
        ret.append(”name”: r[0], ”email”: r[1], ”year”: r[2], ”major”: r[3])
    return ret
```

For example, if the instructor wants to search for all students that are juniors and CS major, the following drop-down menu will be filled out:



Year: Junior

Major: CS

The following SQL query will be sent from the python script:

$SELECT\ name,\ email,\ year,\ major\ FROM\ user$

$WHERE\ year\ = 'Senior'\ AND\ major\ = 'ECE';$

9. Dataflow:

   As mentioned previously in 7 and 8, the dataflow can be summarized as followed:

   - Student or instructor inputs information into the forms or text boxes from the browser.
   - The information is sent to the back end where the python script processes the information via flask and interacts with the database when necessary.
   - Once the information is processed, the result is sent back to the front end and displayed.

10. Advanced Function:

    We implemented an advanced function which parses through a complicated SQL query into multiple steps. We will be first looking for the *SELECT* clause to determine the number of sub-queries. We will also be indexing the end of each of the sub-query by finding the close parentheses that matches its start. We then determine the scope of each sub-query to obtain the order of execution, and each sub-query will be broken down into sub-steps based on the keywords found in it. These keywords are limited to the ones taught in the class, which include *FROM*, *JOIN*, *WHERE*, *GROUP BY*, *HAVING*, *ORDER BY*, *DESC*, *LIMIT*.

    The most challenging part of the advanced function is to obtain the order of execution of each sub-query. Due to the natural complexity of SQLs, it is hard to determine how the database management system will interpret each query. We developed an algorithm that uses a stack to keep track of which sub-query belongs to which parent-query, and output the order while traversing through the query. We also encountered issues where a sub-query depends on the tables selected by its parent query or queries. We made an engineering decision to omit these type of steps because the sub-query itself would make no sense when ran independently, and should not be served as a step that may confuse students.

11. Technical Challenge:

    In the beginning of the semester, we made a series of bad decisions which severely limited our ideas and delayed our timeline. We initially sought to determine the distance between student SQL submission and the solution SQL, and develop a hinting system that will guide students towards the correct answer. This idea was inspired by professor Abdu who witnessed students making hundreds of attempts without guidence. With the submission data from the first assignment, we planned to use that as our training data for a machine learning model. Unfortunately, due to some legal constrains, Abdu was unable to provide such data.

    Our second idea was to auto generate SQL questions and answers to assist instructors randomizing the exam. While this idea was interesting and useful, we soon realized that it was impossible to implement this given our knowledge and time constraints. When we ultimately gave up on the idea, we were already half way through the semester, so we had to come up with a new idea that is both challenging and doable. We also faced other technical challenges, including not being able to implement Python in WordPress, and our lack of knowledge in front end design. Fortunately, our team collaborated well and spent significant amount of time and effort in the final few weeks, and were able to finish the project.

12. Development Plans:

    Even though We changed our plans for the advanced function and the platform several times throughout the project, we eventually finished up with a fairly satisfying result. We initially planned to use PhpMyAdmin for our database design, and use WordPress as our front end development, but we soon realized that WordPress does not integrate well with Python. We therefore switched to Flask for a simpler and more organized platform.

    We initially sought to use the submission data from the first assignment as our database, and integrate with a API which returns the distance between two SQL queries. However, after we learned that we are unable to receive these data, and the idea turned out to be way more complicated than we have expected, we went back to our original plan of developing an useful tool which can assist SQL begineers.

We shared our thoughts on the first assignment, and we realized that the most challenging part was not why we were not getting the expected table, but instead how we got a result table. We looked for other tutorial websites to see how new materials were first introduced to new learners, and we came up with an idea of breaking a complicated SQL query into pieces, and show them the results step by step. We immediately found this idea intriguing and challenging, and we eventually created a tool that can be extremely useful for students who are struggling with more complicated SQL queries.

13. Division of Labor:

    Overall, our group worked cohesively as a team, and was successful in collaborating with each other and brainstorming ideas throughout the project. We frequently met up as a team for long periods of time to assist with each other's responsibilities over the duration of the semester. Since Hassan and Steven are more experienced in front end development, they spent more time on the user interface design, while Eric and Mrinroy focused on the back end development. A more specific labor division is as followed:

    - Mrinroy collected data from the first PrairieLearn assignment and created the databases necessary for the questions
    - Steven implemented the student manager functionality and provided recommendations and assistance towards other functionalities and their relevance to the databases we created.
    - Eric developed the advanced function algorithm for the parser and assisted with the user interface design.
    - Hassan created the question bank with add, delete and update functionalities as well as user registration and login.