Query Analyzer - QueryViz

Scope Change from Stage 1 Proposal

Query Analyzer (QueryViz) is an interactive web application to practice SQL. The goal of this application is to aid students to learn SQL by understanding the step-by-step execution of a SQL query. This tool visualizes the results of the intermediate steps during query execution. The application also provides a dashboard for the instructors to see the various aspects of the students' performance on a set of predefined SQL practice questions.

The product built was in line with this overall vision and scope. It incorporated intermediate query visualization and student activity metadata generation. There are a few aspects changed, which have been highlighted in detail in the functionalities section.

Usefulness

The proposed target audiences were students and instructors. The purpose was to enhance student learning experience and instructor's ability to get insights into student queries.

Achievements

1. Considering the students to be SQL beginners, we wanted to create a simple interface showing order of execution of clauses and the results of each which was achieved.
2. The ability for students to track the status of their completion along with login and authentication were implemented.
3. We currently provide instructors with the average number of clauses used to solve a question and the average number of attempts made by students before a successful submission.

Improvements (Failed to achieve)

1. We were not able to find the relative execution time for each clause
2. We do not currently provide a separate student/teacher profile, though the total set of functionalities is implemented with the instructor profile.
3. We could not achieve the processing of uncorrelated nested queries in FROM, WHERE, and HAVING clauses which were mentioned in the initial proposal.

Change in Schema or Source of the Data

We did not change the Schema proposed in Stage 2. Also, we used student submissions data across 10 SQL queries for the previous semester that were shared to us by the Instructor.

Change in ER diagram

There were no changes in the ER diagram proposed in Stage2. However, we introduced a new entity called MalQueries to store the student submissions that involved DML commands DROP, DELETE, UPDATE, INSERT, ALTER.

Change in functionalities

We added a new functionality to prevent SQL injection through the submission interface. Whenever a student submits the query, it is checked for words like DELETE/UPDATE/DROP/ALTER and if found the submission id is captured in a malicious query table along with the user id. We were unable to capture the relative execution time of the intermediate queries and work with subqueries. This was due to lack of time as we had to develop user state maintenance and defense against unwanted queries which took more time than planned

Advanced Database Programs

The stored procedure contains advanced queries which compute insights based on submissions on a per-question level. Initially, we were executing individual queries from the node server and doing data integration. However since the results are input agnostic, and can be calculated purely based on the database contents, we decided to move this task to a stored procedure in the database. Doing so would enable other applications sharing this database to also reuse functionality. Additionally, it allows us to make a single DB call instead of two, thereby optimizing on network latency.

The trigger provides a crucial security feature to the application. It prevents SQL injection through the submission interface. Whenever there is an insertion into submissions, the query is checked for words like DELETE/UPDATE/DROP/ALTER and if found the submission id is captured in a malicious query database. For any downstream processing the first check is to ensure that the submission is not flagged malicious. This prevents bad actors from having an impact on the application data. The trigger also enables the tracking of bad actors.

Technical Challenges and Advice

Rajiv (rajivr3)

Technical Challenge: I found executing SQL queries asynchronously through the Node js server to be challenging. The main issue was that the server was trying to retrieve the results from the SQL compiler before the query completed execution (before the Promise was resolved). This resulted in the server returning *null* values to the client which made troubleshooting a daunting task.

Advice: My advice is for programmers to thoroughly understand the concept of handling side effects through async/await methods in Javascript before delving straight into the code. It is also good practice to implement some proof-of-concept projects to cement our knowledge in this topic.

Siqi Du (siqidu3)

Technical Challenge: As I handled the authentication, I found it is difficult to implement browser single sign-on (SSO) in our application. In particular, I found it difficult to set up the required cloud infrastructure using a verified identity provider. Despite my best efforts, I wasn't successful in implementing cookie based authentication and resorted to storing the user details such as username and password hash in the database.

Advice: I would strongly advise programmers to understand the latest authentication protocols such as OAuth and OpenIdConnect to securely verify user identity and further provide role-based access controls. It is recommended that programmers inspect network traffic and look at the different Http requests to discern for themselves the logic behind these algorithms.

Vidya (vidyak2)

Technical Challenge: My main technical challenge was parsing the SQL query and breaking it down into intermediate queries based on the different clauses. Initially, I tried to develop a script from scratch to parse this query, however I ran into some issues related to query formatting. It was only later that I got to know about an existing library called sqlparse in python.

Advice: My main advice is to do some research before implementing the features. In short, do not reinvent the wheel. Secondly, I recommend reducing the scope and starting with simple queries to verify the correctness of our algorithm. We should not try to write a universal parser that can work with all queries in the beginning, rather start slow, make incremental progress and gradually increase the complexity by introducing correlated queries and sub-queries.

Yeshwanth (ys58)

Technical Challenge: Displaying the different intermediate tables each having a different structure was particularly challenging for me. Specifically, I faced issues with updating the different components in the front-end based on the results fetched from the Node js server. I found myself replicating a lot of code and introducing redundancies which ultimately made the codebase disorganized. It was only after reading about the state management library called Redux was I able to clean up the front-end and produce an elegant UI.
Advice: My advice is for programmers to prioritize writing clean code. I encourage everyone to surround their code with *try/catch* blocks for adequate error handling. In addition, I suggest programmers maintain a single global state of the application (single source of truth) in the browser rather than maintaining several local states that need to communicate with each other. This makes re-rendering changes fast and responsive.

Additional Changes

There are no additional changes from the proposal, other than the functional changes described above.

Future Work

As an improvement / extension to the current Application, we would like to address the following aspects in the future.

1. Separate Master (Application data) and testing data (Student, enrollments, courses).
2. Support the visualization of subqueries.
3. Handle Concurrent Submissions: We use the max id to find the submission id, which will be a problem during concurrent submissions.
4. Server Side Pagination
5. Cookie based authentication, store passwords in a hashed manner
6. Reliability using replication of database
7. Load balancing of network traffic
8. Have several instances of synthetically generated data to test user queries for correctness instead of just a static single instance

Work Division and Team Management

We used the work division plan as mentioned in Proposal 1 to ensure the person responsible for the task took the lead to communicate with other team members to

ensure the completion of the task. Rajiv being the team Captain, made sure to track the progress of the project. We as a team worked together twice in a week for 1-2 hours on Monday / Wednesday after the class. Then we used to assign individual tasks for the rest of the week.
While Yeshwanth and Sarah took the lead on developing the Frontend (ReactJS). Rajiv and Vidya took the responsibility of developing the backend (node.js, python). Everyone in the team contributed equally for the database setup, design, data insertion, indexing, Writing SQL queries for the application, and developing advanced database programs.
With this plan, we were able to achieve all the stages of the project well before the deadline.

Task	**Responsibility**
Initial Environment setup	Yeshwanth
Establish GCP to server connection	Rajiv
Establish client server connection	Vidya
Schema design (meta data)	Siqi
Query parsing to identify the different clauses	Rajiv
Intermediate query generation	Yeshwanth
Chaining the Intermediate results	Siqi
Business logic (server side)	Vidya
UI for Login screen	Yeshwanth
UI for Query Visualiser	Siqi
UI for Students Question screen	Vidya
UI for Instructor view (Dashboard)	Rajiv