Csc 8711, Databases and the Web - Programming Project 1

Varchaleswari Ganugapati
dept. of Computer Science
Georgia State University
Atlanta, USA
vganugapati1@student.gsu.edu

Bhagirath Tallapragada
dept. of Computer Science
Georgia State University
Atlanta, USA
btallapragada1@student.gsu.edu

***Abstract*—In this project we implemented a Web-based QBE (Query By Example) Processor for MySQL databases. Functionalities implemented are a subset of the language, specifically performing Select-Project-Join (SPJ) queries.**

***Index Terms*—QBE, GraphQL, MySQL, System software, Web development**

I. Introduction

The below mentioned flow of the model describes the execution flow in accordance with the project requirement notes provided in http://tinman.cs.gsu.edu/ raj/8711/sp21/p1/.

A. Flow of the Model

The *index.html* page is the initial screen in the frontend that allows the user to: 1. Input credentials for mysql login and database. 2. View the tables existing in the input schema 3. Access and create the required QBE interface 4. Submit query using the interface and eventually convert the QBE query to SQL and display the required results.

B. Technologies Used

Querying is done on MySQL database. Backend Web Services are implemented using GraphQL in Python i.e. Graphene and flask server. Frontend interface is built with HTML/Javascript (JQuery).

II. Components and Contribution

A. User Interface (HTML and Javascript) developed by Varchaleswari Ganugapati

Developing the front end interface (comprising of index.html and connect.js files) involved the following tasks: 1. Core HTML page as directed in the project requirement 2. Creation of Dynamic HTML using Javascript to implement parts of the webpage 3. Javascript/Ajax URL binding to the server based on the GraphQL schema.

Developing the front end interface involved the below mentioned Challenges:

1. Abstraction of the entire flow to make sure no values are hard-coded. 2. Retrieving the authentication parameters and then connecting to the database with the right query. 3. A major challenge involved ensuring the correct format of the URL in order for the back end to function as expected. 4. We employed client side validation using regular expressions to alert the user for any errors in the input fields. 4. Creation of a tabulated form for our QBE Interface and further capturing the values, especially for queries involving joins. 5. We used window/global variable(s), namely *"num"* in JavaScript to identify the exact count of the number of tables selected. This solved a major hurdle in iterating through the QBE Interface tables to extract the parameters and send them in proper format to the middleware.

Identify applicable funding agency here. If none, delete this.

B. GraphQL and Server Logic developed by Bhagirath Tallapragada and Varchaleswari Ganugapati

Developing the Server logic/ middleware for this project involved the following: 1. Creation of the Graph QL schema 2. Resolving or unwrapping the URL components to required GraphQL data types 3. Resolver function logic for various use cases 4. Testing

Developing the Server logic/ middleware for this project involved the following challenges mentioned along with our individual contribution:

1. Design and creation of the GraphQL schema present in *graphene_schema.py*, developed by both of us 2. Unwrapping the incoming URL and resolve its components to required GraphQL data types (by Bhagirath) 3. Resolver function logic for the use cases such as(Bhagirath): Quering for single table skeleton, querying for multiple independent table skeletons and querying for multiple tables involving joins 4. Testing execution of each required functionality (Bhagirath and Varchaleswari).

C. Backend Database developed by Bhagirath

Tasks include: 1. Creation of the sample database in MySQL 2. Formulation of SQL queries corresponding to the possible QBE inputs. 3. Testing using customization of sample tables to test the various use cases such as select queries with conditions, Joins etc.

III. A view to the execution flow and Samples (not exhasutive))

The following samples were taken using the classroom database and a custom made database (comprising of tables:

Fig. 1. Simple select query

Fig. 2. Join condition query

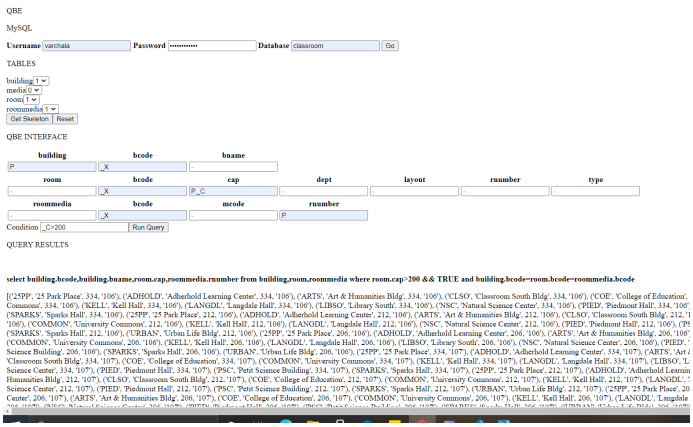

Fig. 3. Join condition query for 3 tables

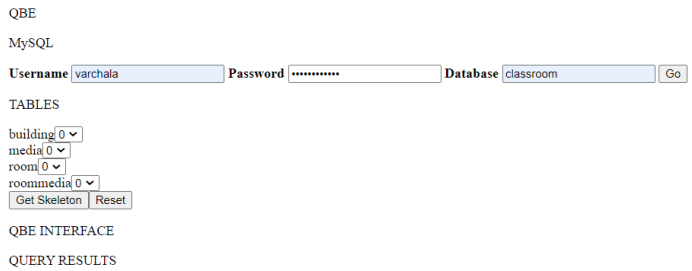

Fig. 4. Reset

Books, authors, book_author and cust_authid) for testing purposes.

1) Simple select query generated for single table selection without specific column parameters, in fig1:

"SELECT * FROM authors"

2) Join condition satisfied for two tables with the foreign key selected in fig 2:

select authors.aId, authors.firstName, authors.lastName, cust_authid.id from authors, cust_authid where cust_authid.id > 12 && TRUE and authors.aId c̄ust_authid.aId

3) Join condition for 3 tables selected in fig 3:

select building.bcode, building.bname, room.cap, roommedia.rnumber from building, room, roommedia where room.cap > 200 && TRUE and building.bcode r̄oom.bcode r̄oommedia.bcode

4) UI After pressing reset as shown in fig 4:

As mentioned in section IV below, snapshots reflect limitations in the beautification of UI, however the data fetched is verified to be consistent with the database entries for the same queries.

IV. Limitation and Scope For Improvement

At this point the developed web application is tested only for S-P-J queries. While the application is tested to ensure that all valid S-P-J queries are consistent, there is possible scope to better identify and iron out the edge cases in client side validations. There is also a lot of scope for design improvement of UX/UI layout as the current implementation does not format the query results in the HTML page. The time constraints forced us to assume the front end beautification as "given".

References

[1] http://tinman.cs.gsu.edu/ raj/8711/sp21/