Web-based QBE Processor for MySQL databases

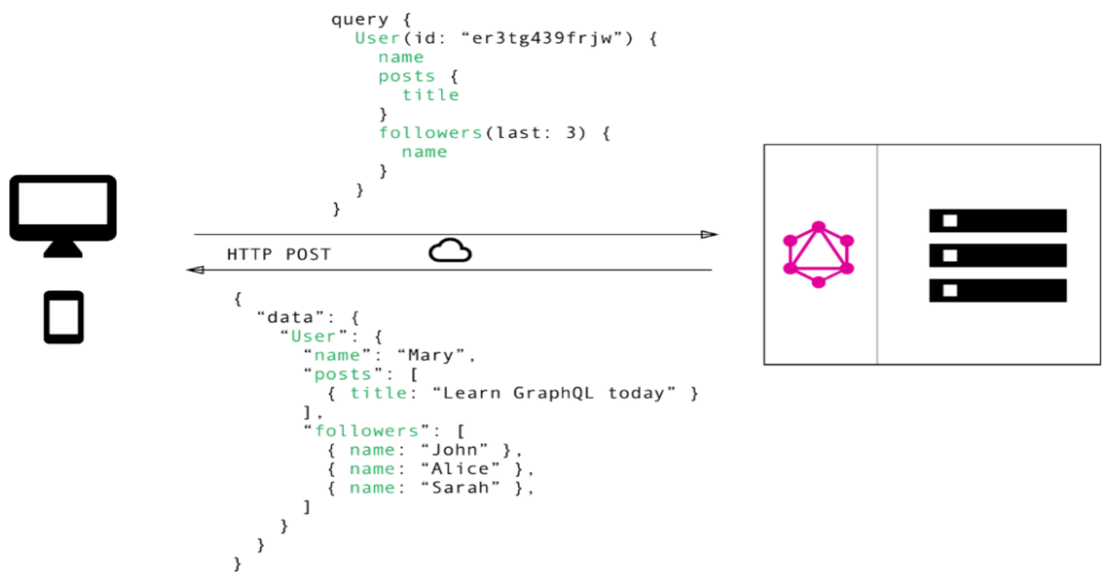
BEJUGAM REVATHI
SAINADH MAKINENI

- **Technologies used:**
    - Frontend –AngularJS and JavaScript
    - Backend – GraphQL in Python Graphene
    - Database – MySQL

- **GraphQL in Python Graphene:**

GraphQL is a query language for APIs. It gives clients the power to ask for exactly what they need and nothing more. With GraphQL, we can simply use one query to access just the information we need.

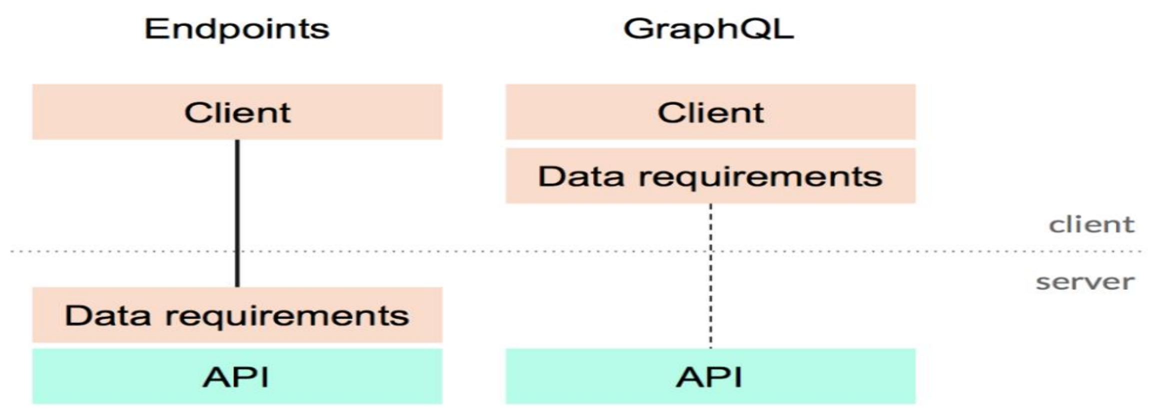
GraphQL API:



```
query {
  User(id: "er3tg439frjw") {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

```
{
  "data": {
    "User": {
      "name": "Mary",
      "posts": [
        { title: "Learn GraphQL today" }
      ],
      "followers": [
        { name: "John" },
        { name: "Alice" },
        { name: "Sarah" },
      ]
    }
  }
}
```

with GraphQL, the data requirements are defined by the client, as shown below:



Web-based QBE Processor for MySQL databases

BEJUGAM REVATHI
SAINADH MAKINENI

- **QBE to SQL Conversion:**

QBE is a graphical query language, and it is based on the domain relational calculus. It has a two-dimensional syntax where system creates templates of relations that are requested by users and Queries are expressed “by example”

SQL stands for Structured Query Language. SQL lets you access and manipulate databases and it became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

SQL and QBE are two commonly used query languages and exist together in several DBMS products. Here in this project when you create and run a QBE Query, the Query Builder will translate this query into SQL. SQL is the language supported by relational databases.

The SQL produced by this translation will differ with the database system used, whether it is Oracle, Sybase, Microsoft or other systems. Here we use Microsoft MySQL Database.

Here is the detailed overview of the project:

Step 1:

In step 1, the User is asked to provide the Username, Password, and Database name for which the user wants to perform further operations of QBE to SQL Parsing. After clicking the “GO” Button, it will trigger the “getTables” function which will collect the details from the text fields and connect to the GraphQL server using Ajax call. Authentication is done and table names are fetched from database in this step.

Step 2 :

Table names are accompanied by the dropdown menu which has a list containing 0,1,2 values. Users can choose an item from the dropdown menu. Upon clicking the "Get Skeleton" button it triggers the “ReadTables” function to retrieve table schemas using multiple ajax calls and connect to the GraphQL server. At the same time, Skeleton tables are created dynamically. It also creates a condition box to make certain validations against the tables.

Step 3:

“Reset Skeleton” button triggers the ResetSkeleton function which will remove the skeletons, reset the skeletons, and provides the user with the flexibility to continue with another selection.

Step 4:

The user has the option to provide the inputs in four formats i,.e P., P._Variable, Constant, and _Variable.

- To display the data in all the columns of a table, put P. under the table name and to display the data in a particular column of a table, put P. under the particular column of the table.
- Conditions can be directly specified under the column names or can be specified via alias in the condition box as follows:
  a. User has the option to directly enter >=100, <=100 or =100 (numeral conditions) under the column name
  b. User has the option to directly enter ‘CLSO’ (string values) under the column name.
  c. User has the option to enter the _Variable under the column name and provide the _Variable along with the condition in the condition box.
- To handle the join condition, The User should enter _Variable under the columns to which the join condition should be specified. Both the columns should have the same variable as '_Variable'
- **Additional Features:**
- d. QBE Parser displays all rows, including duplicate rows, by default if you have just one P. under the column name in your query. To eliminate duplicate rows, specify UNQ. (unique) under the table name in the row with the P. operator
- e. To put rows in a report in ascending order by the values in some column, put AO in that column. The sorting sequence for character data, in ascending order, is as follows:
  i. Special characters, including blanks
  ii. Lowercase letters, in alphabetical order
  iii. Uppercase letters, in alphabetical order
  iv. Numbers, in ascending order
  v. Null values

  To order by more than one column, put AO(1) under the column to be ordered first. Then, put AO(2) under the next most significant column, and so on.
- f. To put rows in a report in descending order by the values in some column, put DO in that column. The sorting sequence for character data in descending order should be in same pattern as of ascending order. To order by more than one column, put DO(1) under the column to be ordered first. Then, put DO(2) under the next most significant column, and so on.

Step 5:

- Upon clicking “Run Query” button, runQuery function in java script is triggered. This function will execute the qbeToSql python function and returns a query and data retrieved from database for corresponding query.

Step 6:

BEJUGAM REVATHI
SAINADH MAKINENI

- Errors handled in the Frontend and Backend:
  a. “You should select at least 1 table or at most 3 table skeletons” : This alert will be generated when you click on “Run Query” which validates and throws an error when there are no skeletons selected or more than 3 skeletons selected.
  b. “Wrong data is passed under "+skeletonIds[i]+". Only P. or P.UNQ is expected in that column” : This alert will be generated if anything other than P. or P.UNQ is given as input to the field under the table name.
  c. “Data type mismatch at "+inputId+". String is not expected” : This error will be generated if string is encountered as input for an integer variable
  d. “Please Check Data. Either it is not valid or there might be more tables with null values” : This alert will be encountered when user generates 3 schemas, but doesn't give any inputs in 2 schemas that are generated.

- **Contributions:**

Revathi Bejugam:

- Function to retrieve the column names and its data types for the table selected, Function to get the final result which calls the Qbe to Sql parser using Graphene.
- QBE to SQL parser function using python.

Sainadh Makineni:

- Functions to retrieve the table names from the database selected using Graphene.
- Ajax calls to get the data from of tables when SQL database is connected, get skeletons for the database tables, handle validations in frontend, retrieve and populate final query, result from backend and UI part using AngularJS.

To run the code with all the packages, please run ‘pip install -r requirements.txt’.