Jahnavi Kamireddy

Surya Teja Yellutla

An Online QBE Processor for MySQL

1. **Introduction:** Query by example (QBE) is a declarative query language like SQL and is based on domain relational calculus (DRC). QBE was one of the first graphical query languages with minimal syntax developed for database systems. It is a feature included with many database applications that provides a user-friendly method of running database queries. Structured Query Language (SQL) is a standardized programming language that is used to manage many relational databases. If the syntax is slightly incorrect, it may return wrong results or may not run at all.

   Instead of writing an entire SQL command, the user can just fill in the blanks in the interface provided by QBE to define the query user wants to perform.
   In this project, when a user fills the blanks in QBE interface and executes it, translates into SQL query and displays the results. The SQL translation differs with the database used, we have used MySQL database.

2. **Procedure:**
   - Step 1: User Provides the details and the database name on which he/she wishes to perform QBE.
   - Step 2: After the button "Go" is clicked, it validates the details entered and displays the list of existing tables in the database.
   - Step 3: Table names allows to select the tables on which a user can perform a query.
   - Step 4: Get skeletons button displays the skeletons of the tables selected and a condition box to add conditions to the query.
   - Step 5: Reset Skeletons button provides flexibility to the user to reset the skeletons and to continue with another selection.
   - Step 6: Result button provides the SQL query that is translated from the QBE provided and its results based on the values provided in the skeletons of tables selected.
   - Step 7: A "P." under the table value returns the query result with all column values in that table and a "P." under the column value returns the query result with that particular column values.
   - Step 8: When a same variable is provided with prefix (_) under column names of different or same tables, it returns the query result by matching the same values of those columns (i.e., joins the tables with those columns)
   - Step 9: When conditions are provided in condition box, it returns the query result with the conditions provided.
   - Step 9: Possible values under table names – Null or "P.", in other cases it displays an error.
   - Step 10: Possible values under column names – Null or "P._variable" or "_variable" or Constants ('<value>' – for strings value under single quotes, <number> - for numeric), in other cases it displays an error .
   - Step 11: When a variable is not provided in column values but provides in condition box, it displays an error .

Jahnavi Kamireddy

Surya Teja Yellutla

3. **Technologies Used:**
    - Frontend – Html and Javascript
    - Backend – GraphQL in Python Graphene
    - Database – MySQL

4. **Contributions:**

Jahnavi Kamireddy

- Retrieving the table names from the database selected (resolve_tables function Backend)
- Forming the SQL query by getting the column names to be selected and the table names where the data needs to be retrieved (form_query and columns_from_tab functions)
- Validating the values under table and column names and (validate_values function)
- Retrieving table details from the input provided by the user(getTableDetails function frontend)
- Displaying query results fetched from the input(get_results function)

Surya Teja Yellutla

- Retrieving the column names and its data types for the table selected (resolve_columns function)
- Forming the join conditions and where clause conditions to the query (join_query and cond_query functions)
- Validating the values in condition box (get_results function)
- Binding column data to the skeletons for the tables selected(getSkelletons function)
- Sending data from skeletons to fetch query results(get_results function)
- Applied CSS for tables and designed html part.