**Interactive SQL Query Generation**

CSE 544 Final Report

Adrian Sham (adrsham) and Lindsey Nguyen (nhlien93)

*University of Washington, Seattle*

**Abstract**

In this paper, we examine an application that helps a user generate complex SQL queries using feedback provided by the user. In order to provide this feedback, the user directly modifies the results of a SQL query and lets the application decide on a possible new query based on the modifications. We discuss some of the implementation details of the application and also some of the challenges faced during development. We will also discuss some the the current work in the area of assisting users with writing SQL queries and some of the different approaches to solving this problem.

**1 Introduction**

The proliferation of relational database systems in almost all aspects of computing has increased the importance of learning SQL for managing data. Various work such as web programming and scientific research of data require the use of SQL for accessing databases. This creates a need for people who are not computer scientists or database administrators to interact with databases, such as researchers or web application developers. As such, for users who are less familiar with SQL, writing complex SQL queries may be difficult.

In general, most users write SQL queries by starting with the simplest form of the query, such as SELECT \* from table. Then, after taking a look at the returned table, users will start thinking of refining the results to suit their needs, such as by adding filter, joins and projections to the query.

After adding the refinements such as filter, joins and projections, the user will usually rerun the query to see if there are any errors and whether they are closer to what they have in mind as the desired result. This process is usually repeated until the user has found the correct query to do what he/she wants.

For users who are more familiar with Office software such as Excel, they may decide to just import the entire table and work with the data using Pivot tables. The first of several issues with this approach is that the user usually does not need the entire dataset, and if the dataset is large, downloading the entire dataset would be slow over the network. Also, if the dataset is large, trying to process it locally may slow down the user’s machine. Finally, it is usually more efficient to do data processing like Aggregation in the database itself and only return the processed result.

Recognizing this problem, we decided we would try to work on an application that would help the user write more complex queries in a intuitive way. Instead of having the user rerun queries and see if the result is closer to what they had in mind, why not let the user directly modify the result to tell the application what the user wants? This way, instead of doing several iterations of modify and run, the application would propose the SQL query and hopefully save the user some time.

We hope that by providing this application, users will feel comfortable with writing complex queries and shift away from downloading entire datasets into software like Excel. This would move more data processing work into the database and likely help save the user some time.

**2 Approach**

As stated previously, the main goal of this project is to find a way to make writing complex SQL queries easier for regular users. In order to achieve this goal, we propose creating an application that helps the user update complex SQL queries by attempting to use the modified results of a SQL query to generate a new query. We will start out with supporting very simple SQL statements and expand on the application depending on how long and difficult it is.

The overall experience we are trying to provide the user can be summarized in the following sequence of actions:

1. Provide the application with a simple query with no sub queries, this initial query is provided by the user and can be simple. An example would be SELECT \* FROM table

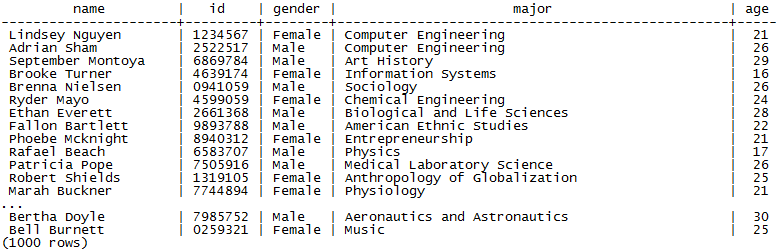
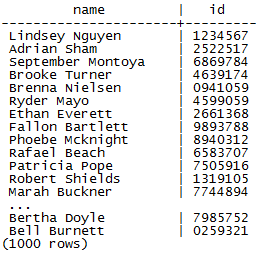
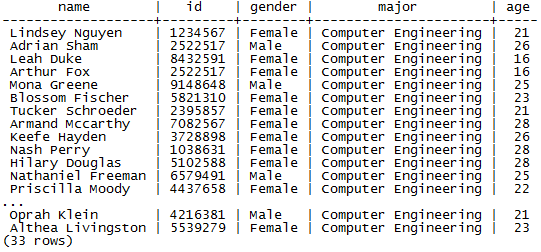
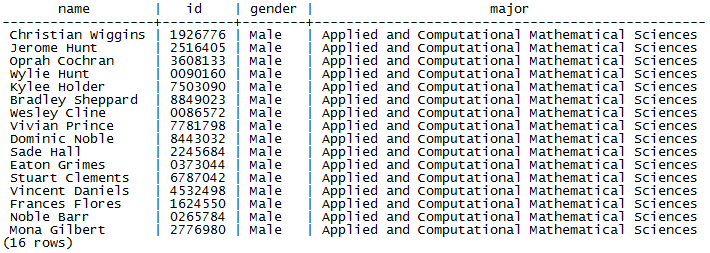
Table 1: Table of results when doing a select \* on the Students table.

Table 2: manually modified query used as input to our program so that an SQL statement can be generated

(a) Changing select statement (b) Changing where clauses



(c) Changing both select statement and where clauses

1. The application then runs the query on an existing Postgres database
2. The application outputs the result of the query
3. The user modifies the result such that it is what the actual user wants and inputs this new set back into the application
4. Based on the modified result, the application will try to generate the corresponding query that will output these exact result

For our project we will be using a generated dataset that contains one table called Students. This table will have fields including name, id, gender, major (text), and age (int), and will have a total of 1000 tuples.

Using the results from the query:  “SELECT \* FROM Students” as displayed in Table 1, the user will be able to input a modified data table such as the one in Table 2a to get the resulting query:

***SELECT name, id FROM Students;***

This is an example of changing the projection operator of a query plan or the select statement inside a SQL query.

The user could input another modified data table as shown in Table 2b which would get the resulting query:

***SELECT \* FROM Students WHERE major=’Computer Engineering’;***

This is an example of changing the select operator of a query plan or the where clause inside a SQL query.

Another good example of what our program can do is change both the select statement and where clause both in one as shown in Table 2c. The result would be the query:

***SELECT name, id, gender, major FROM students WHERE gender = ‘Male’ AND***

***major = ‘Applied and Computational Mathematical Sciences’;***

In the simple examples above we show that an SQL statement can be generated using the changes in the resulting dataset. Our goal is to use the original SQL statement, convert them into a relation algebra query plan, and then use the different operators and it’s order to create the final SQL statement.

For our implementation, we started out with the Project Operator which was relatively simple and mainly used to test our application setup and connection with Postgres. Next we only looked at the where clause of an SQL statement, concentrating on the Select Operator. For this research paper, those two operators were the only implemented one accomplished however if future work is done on this project, other operators such as joins and group bys will be implemented as well.

**3 Implementation**

**4 Evaluation**

In this section of the paper, we will try to give some of the implementation details of the application and evaluate the performance of our application.

2 pages

**5 Accomplishments**

So far, we have been working on a Java command line application that can parse user input, in this case a SQL query into a ZQuery object. This gives us a more sophisticated representation of the information contained within the SQL Query for use in later stages of the project. The application is also able to pass the query to a Postgresql database, the result can then be output to the console or to a file chosen by the user.

Once the user modifies the file, the application can then parse the file back into the ZQuery object, extracting as much information as possible from the file. During the conversion, some information such as the original column names are lost since the column names might be an alias. This information needs to be convert into a ZQuery object so that the query generation code can work with the information.

We have also been working on a algorithm to find the correct Projection based on the information derived from the modified query result and the original query. Our naive implementation of this algorithm has a complexity of O(n3), since it is basically a brute force search across the columns of all available tables, we later improved this to a O(n2) implementation.

**6 Challenges**

Some of the challenges we encountered that we did not expect is just the time it took to get to the interesting part of the project, which is the query generation part. Although we made use of an existing Postgresql database, it took a while to get the parsing of user input and the modified query result file correct. Also, due to our decision to take the results of a query and store it as a String in memory, we had issues when trying to work with results that are large.

There are also a lot of subtle details we need to worry about when parsing SQL queries, such as whether a column name is the alias or the original name and the various possible ways a valid SQL query can be written. We also had to assume a lot of situations won’t occur in order to simplify our application, such as assuming the user won’t make mistakes when modifying the query results, no subqueries and no large query results.

While trying to generate a query for modified results seems relatively straight forward on paper, it has definitely been more challenging than we initially thought, especially when we were trying to come up with algorithms to do Projection.

**7 Related Work**

In our research, we have not been able to find a work that matches exactly what we are trying to accomplish. However, there are a few research papers trying to solve the problem of making SQL query writing easier for users, and are therefore relevant to us. It is interesting to see how they approached the problem and the methods they used.

One paper we found interesting is the paper about Interactive SQL query suggestions [3]. In this paper the authors attempt to make SQL query writing easier for inexperienced users by helping users formulate queries with *queryable templates* to model the structures of SQL queries. These templates are then ranked by their relevance to the keyword query provided by the user, such as “count database author”. Compared to our approach, the paper makes use of keywords provided by the user to generate queries, making it easier for someone totally unfamiliar with SQL to write queries. Our approach asks to user to modify the result to get their desired query and asks the user to provide an initial query, thereby assuming a little bit of SQL knowledge.

Another way to make query writing easier is to use a graphical user interface, such as the one presented in the paper Querying without Keyboards [1]. With the proliferation of next-generation computing devices such as tablets and smartphones, users are increasingly interacting with software without a keyboard. Therefore, it would make sense that some users would be more comfortable with writing queries using a GUI instead of a keyboard interface. In the QWiK database system proposer by the author, if the user brings the tiles for two relations close together, the interface would begin suggesting possible query actions and arrange attributes from both tables such that they are amenable to joining together. This would likely seem more intuitive to users compared to typing a specific SQL statement.

**8 Conclusion**

1 paragraph

**References**

[1] Nandi, Arnab. "Querying Without Keyboards." In *CIDR*. 2013.

[2] Shen, Yanyan, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. "Discovering queries based on example tuples." In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 493-504. ACM, 2014.

[3] Fan, Ju, Guoliang Li, and Lizhu Zhou. "Interactive SQL query suggestion: Making databases user-friendly." In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 351-362. IEEE, 2011.

[4] Stoffel, Kilian, et al. "A graphical tool for ad hoc query generation." *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 1998.

[5] Chen, Zhiyuan, and Tao Li. "Addressing diverse user preferences in sql-query-result navigation." *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007.

[6] Chakrabarti, Kaushik, Surajit Chaudhuri, and Seung-won Hwang. "Automatic categorization of query results." *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004.

[7] ZQL: a Java SQL parser. n.d. Web. 15 Mar. 2015.