SIQL

Simple Interactive Query Language

Daniel Trujillo
dtru@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Travis McKenzie
travism@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Edward Funger
funger@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Gary Menezes
gmenezes@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Susan Davidson
susan@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

The purpose of this document is three-fold. First, it presents background information on the topic of relational databases and the current state of database management software. Second, it proposes a new, conceptually simplified, interface for database management. Third, it explores such a project's architecture, implementation projections, and criteria for evaluation.

The proposed application will simplify relation and query generation by replacing SQL syntax with graphical elements, and will further simplify query generation with some auto-completion / prompting heruristics to ease the process of handling multiple relations in a single query.

1. INTRODUCTION [3]

Relational databases were developed in the 1970s, with Oracle 2.0 coming out in 1979. A *relational database* is a database that conforms to the *relational data model* proposed by Codd in 1970 [1]. In this model, data is conceptualized as a set of *relations*, each of which is a table in the database. Each relation is composed of a set of tuples(rows) that span a set of attributes(columns/fields) that each has a domain(type). For example, a 'user' relation might have a 'password' attribute of domain 'string'. The tuples in a relation are distiguishable by their *key*; a set of attributes that is, collectively, unique for each tuple in the relation. In practice, a key is usually an 'id' attribute of domain integer. Relations are either *entity* or *relation* sets, where entities are representative of nouns and relations are representative of verbs. For example, a database might have 'user' and 'message' entity sets along with a 'has' relation set. The 'has' relation would associate messages with users by associating their keys.

Relational databases are operated upon using Codd's relational algebra and tuple relational calculus. The relational algebra and relational calculus are mathematical operations over relations, founded in first order logic. They are logically equivalent. The relational algebra is used to query relational databases and its operations include projection, selection, union, and join.

The most commonly used implementation of the relational calculus is the Structured Query Language, SQL. SQL is a superset of the relational calculus and can also be used to create databases that conform to the relational data model.

Relational database management is conceptually difficult and, for non-trivial data sets, is only accessible to experienced database administrators. A few specific difficulties for the layman are designing effective relations, writing efficient queries in SQL, and understanding pre-existing databases enough to query them for complex relations(closures).

An interface that is visually and conceptually simpler than existing relational database management systems would increase the accessibility of this already popular database model.

2. RELATED WORK

There is an abundant supply of graphical user interfaces (GUIs) designed to manage databases. The most popular database administration tool today is almost certainly phpMyAdmin [8]. This tool provides a web interface as its GUI, allowing a user to manually enter SQL commands or generate basic commands on existing database tables. A user must be aware of all of the settings that they wish to use in their database, but they can choose these settings from lists of options rather than strictly writing SQL. SQL queries are generated as a result of any action taken in phpMyAdmin, so a user can get an intuition for SQL syntax without actually writing SQL. Additionally, imports are allowed from SQL or CSV files, and various export methods are also supported.

QBE, or Query By Example, was the first graphical query language developed for relational databases. Initially this language was designed solely for accessing existing databases, but this has changed over time. Today, generalized QBE (GQBE) projects have drawn ideas from QBE and incorporated user-friendly GUIs [4]. However, these implementations work on top of prexisting databases and lack the ability to generate arbitrary databases and data relations.

Various graphically-based applications have expanded upon the ideas found in the applications mentioned above. One implementation, WWW SQL Designer, provides a simple, low-level abstraction that relies entirely on foreign keys [10]. A user can create data relations by dragging and dropping visual representations of database tables which are all connected by foreign keys. While this can take care of many simple data relations, the scope of this application is limited by its simplicity. For example, with hundreds of connections between data tables, the visual representation of the database includes all of these connections together on the screen. Thus it is easy to see that this is not a scalable solution as the number of connections between data fields increases. Additionally, this implementation relies on user familiarity with SQL.

A more complete, mathematically justified, graphical query language called Picasso was developed to ”remove some artificial constraints in query formulation and help users to represent their thought process naturally” [5]. Picasso is database agnostic in the sense that it uses universal relation theory to support arbitrary SQL types. However, this comes with a price. Picasso relies on the hypergraph data model, which introduces implementation complexities as this model is still controversial.

Currently the state-of-the-art interfaces for SQL management act as interactive development environments (IDEs) which allow users to generate SQL and refine that SQL manually. This requires users to attain only basic competency and allows them to accomplish advanced tasks relatively efficiently. The most notable open-source example is MySQL Workbench [2]. One initial drawback is that this project is only designed for MySQL, though migration from other major SQL database types, such as Microsoft SQL Server, is supported. MySQL Workbench functions as both an IDE complete with syntax highlighting and SQL execution history, as well as a visual tool for demonstrating data relations. A database agnostic counterpart to MySQL Workbench is FlySpeed SQL Query [9]. FlySpeed SQL Query allows drag and drop capabilities to generate queries, as well as other advanced features, but its focus is data processing rather than database design and creation. While this is valuable for users who inherit databases that they must access, there exists an implicit assumption that a well-versed database administrator configured the database for that end user.

One new product that goes beyond the current IDE approach is Microgen DBClarity Developer [7]. This product is database agnostic, allowing the user to develop database logic for an arbitrary SQL implementation. Key features include a proprietary method of representing database logic through an interactive GUI, full-feature database schema diagrams that give the user control over more than just foreign keys, a debugging system, and modularization of database logic for use in multiple projects. While all of these features provide the user convenience and efficiency, there still exist basic assumptions about the user’s general familiarity with programming. The native syntax of the application is relatively simplistic in nature, but it is arguable that a less technical, more human-readable language could be developed to describe database components. Furthermore, it is also possible that a method more intuitive than a 50-button GUI could readily be developed to describe database logic.

3. PROJECT PROPOSAL

For our project, we expect to be able to implement a graphical query language that will be hosted on a server. This web application will help non-technical users create and interact with a database, without having to understand the complexities of SQL. We are pursuing this approach because we recognize the success of graphical programming languages like Scratch, that teach people how to program. With this project, we hope that valuable programmer time will not be wasted writing SQL queries for non-technical users in a collaborative environment.

3.1 Anticipated Approach

Our project will create a web application that allows users to create an account. From that point, they will be able to create new tables and columns for the database, as well as query the database through a graphical interface. Because of ActiveRecord and the many different database adapters available, the app will be database agnostic. Most relational, as well as some non-relational databases are supported.

To achieve this goal, we will create a controller, model and view that are responsible for interacting with the database. The view is where the interface will be implemented. We plan on using HTML5 canvas to create an interactive page, with simple graphics similar to Scratch(see Fig. ??). On the controller,there will be actions that are responsible for the different types of database interactions such as creating a table, modifying a column, or retrieving a value. Inside of the model will be the methods responsible for parsing, composing, and executing queries. First, the user will build the query in the view by selecting tables and entering values. Once the user is finished constructing, the query will be sent to to controller as JSON, and a parser will deconstruct the request, and map each part to the appropriate ActiveRecord call. Then, each part will be composed together, executed, and the app will return the result to the view.

Because of the way that Rails forces the programmer to follow conventions, it will be a challenge to figure out how to run migrations, which create/remove tables and columns, from the client side. Another issue will be covering the entire domain of SQL. (option to execute actual sql). There are many complicated queries that will be difficult to represent visually. The unparsed queries will also have to be sent to the controller. We will have to create an intermediate format between the graphical representation of the query, and the well-formed query, because we cannot do any heavy processing on the client side of the application. For each user, we will have to create a new database, or a virtual database inside our application. This may involve spinning up instances on a cloud platform which could be very challenging. Finally, the application must be simple enough that our target audience is able to use it with minimal help. This will be hard because we are all Software Engineers, and do not have the same perspective as a non-technical user.

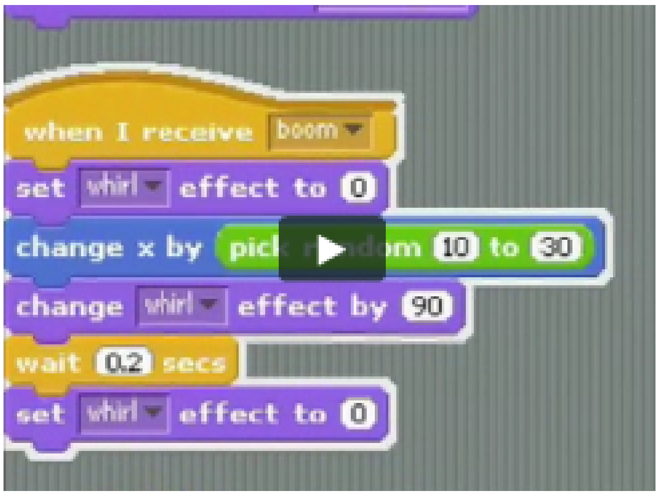
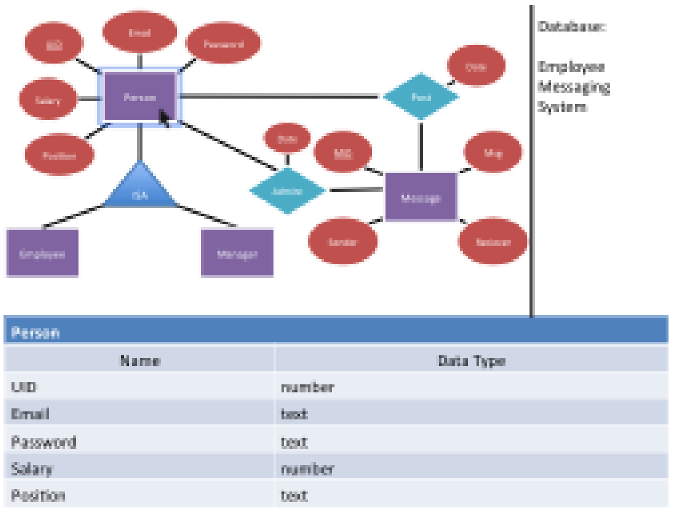


Figure 1: Scratch Tool

3.2 Evaluation Criteria

Our project will facilitate the creation and use of databases for users in a simpler and more intuitive way than other tools that currently exist today. The interface for creating a database will be much simpler than ever before by abstracting the SQL specific details that would normally confuse the user(see Fig. 2). Specific datatypes such as VARCHAR, CHAR, and STRING would be hidden from the user and would be replaced by a single easy to understand type such as TEXT. This would deter the user from worrying about what specific datatype would be best suited for their use-case. Other specific UI functions including the specific use of color on each element of the database and specific types of shape would help the user differentiate between tables and relations.



Person	
Name	Data Type
UID	number
Email	text
Password	text
Salary	number
Position	text

Figure 2: Sample Interface for Database Creation

In addition to the overall simplification of the entire database model, this app would provide the advantage of not requiring any local resources since the app would be hosted online. Most graphical database design tools are Operating System specific applications that require local storage to work. Our application would provide hosted storage that would be accessed using a RESTful API. This provides the advantage of having a central location for storage, so that the user can utilize the created database in multiple applications (including mobile, web and desktop).

Other open source or paid projects are hosted on the user's personal machine. On top of that most of the GUIs used in these applications are generally technical in the sense that it would require some knowledge of ER diagrams or SQL to truly understand what is being created(see Fig. 3). The datatypes that are denoted in these projects are specific database types that might confuse the user and provides a generally unfriendly GUI.

However, the main differentiation between our project and other services out there will be the way that we query our database. Other services that provide these Graphical Design tools for databases still rely on SQL to make their queries work. Although these services attempt to facilitate the creation of databases through a GUI, they still require the same level of understanding of SQL to use the actual database. This seems counter intuitive since these projects

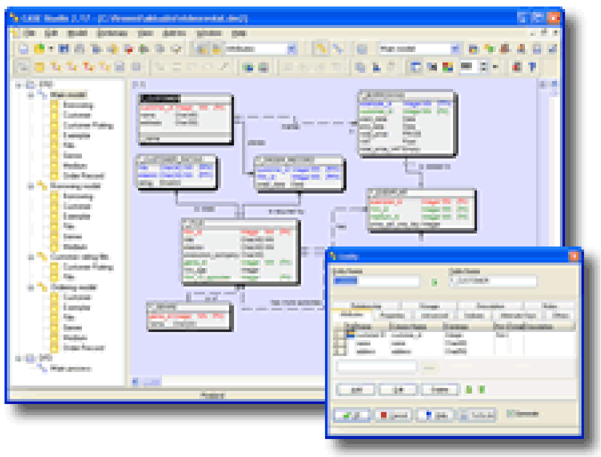


Figure 3: ToadDataModeler Database Graphical Design Tool

are attempting to utilize a UI to make database interaction simpler. Which begs the question, "Why not just learn SQL?"

This project would facilitate this process by creating a Scratch (see Fig.1) like tool that would simplify query creation for the user. It would use specific keywords to perform certain actions that would be placed together like puzzle pieces. This would facilitate the creation of functions like GetAllEmployees by creating a single connected piece that would COUNT ALL on TABLE Employees that is much simpler for a user to read than the SQL equivalent stating: SELECT COUNT(*) FROM Employees. This simplified querying language would allow users to relate their queries to be closer to the english language than SQL. This could even utilize text entry that would use this type of keyword search to create queries for users. Other tools like LOGOS [6] convert SQL to text but don't actually convert text to SQL.

In effect this project would combine two different solutions to facilitate database creation and use for non-technical users. It would simplify the entire database creation process through our GUI based database creation tool, that would abstract any complexities of SQL. Furthermore, this would create a new interface for query creation based on Scratch and a form of keyword search will make it easier for users to properly utilize their databases.

4. RESEARCH TIMELINE

We hope to complete the basic functionality of the project by the end of this semester. Next semester we hope to be able to expand our implementation to be more complete. Our work flow will iterative, where at the end of each sprint, we will have completed a block of functionality that will have passing tests.

- Current Task: Preliminary research, Work on graphical interface, research hosting providers.
- prior-to thanksgiving : Complete sketch ups of graphical interface, begin to implement interface. Begin building api that interacts with the database.

- PRIOR-TO CHRISTMAS : GUI completed for set of queries possible, api supports set of simple queries.
- COMPLETION TASKS : Write unit, functional, and integration tests to verify correctness of program, find non-technical users to evaluate if program is easy enough to learn, complete write-up.
- IF THERE'S TIME : begin to expand the possible queries in the api as well as the graphical interface, continue testing, optimize query building.

5. REFERENCES

[1] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[2] Oracle Corporation. Mysql workbench features. `https://www.mysql.com/products/workbench/features.html`.

[3] Z. Ives. Upenn cis550 lecture notes. `http://www.seas.upenn.edu/ zives/cis550/schedule.html`.

[4] B. E. Jacobs and C. A. Walczak. A generalized query-by-example data manipulation language based on database logic. *IEEE Trans. Softw. Eng.*, 9(1):40–57, January 1983.

[5] Hyoung-Joo Kim, Henry F. Korth, and Avi Silberschatz. Picasso: a graphical query language. *Softw. Pract. Exper.*, 18(3):169–203, March 1988.

[6] Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. Logos: a system for translating queries into narratives. In K. SelÃğuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *SIGMOD Conference*, pages 673–676. ACM, 2012.

[7] Microgen. Microgen dbclarity developer. `http://www.microgen.com/uk-en/products/microgen-dbclarity-developer`.

[8] phpMyAdmin. About. `http://www.phpmyadmin.net/home_page/index.php`.

[9] Active Database Software. Flyspeed sql query: Free sql query tool for any database. `http://www.activedbsoft.com/overview-querytool.html`.

[10] Ondrej Zara. Www sql designer: Project home. `http://code.google.com/p/wwwsqldesigner/`.