Relational Algebra Translation Application

iRat: https://github.com/rvhirsch/RelationalAlgebraProject

Rachel Hirsch
University of Puget Sound
1500 N Warner St.
Tacoma, WA 98416
rhirsch@pugetsound.edu

Josh Wagner
University of Puget Sound
1500 N Warner St.
Tacoma, WA 98416
jwager@pugetsound.edu

ABSTRACT

Relational Algebra (RA) is a mathematical way to explain database queries. The purpose of this application is to create a way to graphically view the output of Relational Algebra queries, given a user-inputted series of database tables. This project required a synthesis of Java String parsing, a Java-based GUI, and a number of open-source projects, specifically an SQL-style in-application database by H2 and Camdenre's Equation Editor. The purpose of the this program is to provide a way to test relational algebra strings against a real database without the use of ugly plain text formatting or a markup language. Our program will potentially be used in future database classes in at the University of Puget Sound. This paper will also cover basic use of SQL and Relational Algebra.

KEYWORDS

Relational Algebra (RA), Standard Query Language (SQL), Graphical User Interface (GUI), H2 Database, Relational Database Management Systems (RDBMS)

ACM Reference format:
Rachel Hirsch and Josh Wagner. 2016. Relational Algebra Translation Application. In *Proceedings of Fall Capstone, 2017, Tacoma, WA USA, Fall 2017 (CSCI 440, 2017),* 9 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Relational Algebra (RA) is a mathematical language used by database theorists to explain queries on database tables. However, these queries can be hard for humans to understand and parse through by hand. This project aimed to create a application to translate RA to Structured Query Language (SQL) in order to run these queries on a user-entered database to test their correctness.

Many languages used by database systems (e.g. SQL) are based on Relational Algebra, so knowing RA can greatly aid in the understanding of both these languages. However, it can be difficult to learn RA due to the inability to see the effects of queries, unlike programming in SQL. The purpose of this RA to SQl translator is to display results of RA queries in real-time.

This project is useful for a few kinds of people and research. First off are database theorists who work with complex queries and need an easy way to see if what they are doing is correct. By inputting a sample database and running an RA query through our parser they will be able to see if the output is what they expected. This said, long, complex equations on large datasets are not recommended for this application as it works best on small, simple data.

The largest user-base for this application - and the one we had in mind while creating this project in the first place - is database systems students. These are generally students who only know the basics of how databases work and would benefit greatly from being able to test their RA equations. Thus, this application will allow these students to better both their knowledge of RA and of database theory in general.

This application is composed of two parts, which will be covered in more detail later in this paper:

(1) The back-end: a Relational Algebra parser, converting RA equations to legal, runnable SQL queries.
(2) The front-end: a graphical user interface (GUI) where the user can input a test database and run RA queries.

2 PROJECT BACKGROUND

Information in a database is most often grouped in a series of tables which can be queried using a database query language like SQL, which will be explained later. Each table is composed of rows (a.k.a. tuples) and columns. Each tuple is composed of a series of attributes (i.e. columns). The tables used in this section to explain RA and SQL equations are below:

People:

ID	Name	Sex	Age
0	Martha	F	55
1	David	M	35
2	Brad	M	73
3	Kiona	F	22

Eats:

ID	Pizza	Crust	Loyal
0	Cheese	Thin	True
1	Pepperoni	Thin	False
2	Anchovy	Thick	False
3	Carnivore	Thin	True

2.1 Relational Algebra

As mentioned in the previous section, Relational Algebra (RA) is used as a mathematical way to explain database queries. There are four main types of selections from a database as well as a way to group rows by values in a certain column before running aggregation functions [10]:

(1) $\sigma(tableName)$: "Select all"
This command selects all rows from a given table.
Example: $\sigma(People)$

ID	Name	Sex	Age
0	Martha	F	55
1	David	M	35
2	Brad	M	73
3	Kiona	F	22

(2) $\sigma_{condition}(tableName)$: "Select on condition"
This command selects all rows from a given table which match a certain condition or conditions.
Example: $\sigma_{age>40}(People)$

ID	Name	Sex	Age
0	Martha	F	55
2	Brad	M	73

(3) $\Pi_{columnNames}(tableName)$: "Project"
This command selects all included columns from a given table by column name.
Example: $\Pi_{name,age}(People)$

Name	Age
Martha	55
David	35
Brad	73
Kiona	22

(4) $\gamma_{aggregation}(tableName)$: "Aggregate"
This command selects columns contained in the given aggregation function (i.e. average(), count(), max(), min(), sum()) and runs those functions on the columns.
Example: $\gamma_{max(age)}(People)$

max(Age)
73

(5) $_{group}\gamma_{aggregation}(tableName)$: "Group":
This command groups rows by a certain attribute, then runs the included aggregation functions on each group.
Example: $_{sex}\gamma_{sex,max(age)}(People)$

Sex	max(Age)
F	55
M	73

It is also possible to group tables together in order to return more interesting data upon running these selection queries. Some of the most used joins are Cross Join and Natural Join. Cross Join pairs all tuples in one table with all tuples in the table it is joined with. Natural Join does the same, but only leaves the tuples for which corresponding primary keys of tables match.

(1) table1 $\times$ table2: "Cross Join"
Example: $People \times Eats$

ID	Name	Sex	Age	Pizza	Crust	Loyal
0	Martha	F	55	Cheese	Thin	True
0	Martha	F	55	Pepperoni	Thin	False
0	Martha	F	55	Anchovy	Thick	False
0	Martha	F	55	Carnivore	Thin	True
1	David	M	35	Cheese	Thin	True
1	David	M	35	Pepperoni	Thin	False
1	David	M	35	Anchovy	Thick	False
1	David	M	35	Carnivore	Thin	True
2	Brad	M	73	Carnivore	Thin	True
2	Brad	M	73	Pepperoni	Thin	False
2	Brad	M	73	Anchovy	Thick	False
2	Brad	M	73	Carnivore	Thin	True
3	Kiona	F	22	Cheese	Thin	True
3	Kiona	F	22	Pepperoni	Thin	False
3	Kiona	F	22	Anchovy	Thick	False
3	Kiona	F	22	Carnivore	Thin	True

(2) table1 $\bowtie$ table2: "Natural Join"
Example: $People \bowtie Eats$

ID	Name	Sex	Age	Pizza	Crust	Loyal
0	Martha	F	55	Cheese	Thin	True
1	David	M	35	Pepperoni	Thin	False
2	Brad	M	73	Anchovy	Thick	False
3	Kiona	F	22	Carnivore	Thin	True

2.2 Structured Query Language (SQL)

SQL is a database query language used similarly to Relational Algebra, but is actually able to query a database. The corresponding SQL statements to the RA equations seen in the previous section are as follows:

Selections [11]:

(1) $\sigma(tableName)$:
SELECT * FROM tableName;
Example: $\sigma(People)$
SELECT * FROM People;
(2) $\sigma_{condition}(tableName)$:
SELECT * FROM tableName WHERE condition == True;
Example: $\sigma_{age>30}(People)$
SELECT * FROM People WHERE age > 30;
(3) $\Pi_{columnNames}(tableName)$:
SELECT columNames FROM tableName;
Example: $\Pi_{name,age}(People)$
SELECT name, age from People;
(4) $\gamma_{aggregation}(tableName)$:
SELECT aggregation FROM tableName;
Example: $\gamma_{max(age)}(People)$
SELECT max(age) FROM People;
(5) $_{group}\gamma_{aggregation}(tableName)$
Example: $_{sex}\gamma_{sex,max(age)}(People)$
SELECT sex, max(age) FROM People GROUP BY sex;

Joins [12]:

(1) table1 $\times$ table2
Example: Person $\times$ Eats
Person CROSS JOIN Eats
(2) table1 $\bowtie$ table2
Example: Person $\bowtie$ Eats
Person NATURAL JOIN Eats

2.3 GUI

Our Graphical User Interface uses a combination of Java, JavaFX, HTML, JavaScript, and CSS. The JavaFX library is able to load web pages, which is what we used to display the interactive equation editor.

2.3.1 Languages, Libraries and Tools used.

(1) Java: The good ol' Object Orientated Java that we know and love
(2) JavaFX Library: The successor to and built on top of the Java Swing library, made for creating GUIs in Java, offers a myriad of functional and aesthetic upgrades. You can use traditional Java code, or use a graphical editor to create an FXML file you can load or any combination of the two (recommended). [7]
(3) HTML, CSS, & Javascript: You know what these are
(4) JavaFX Scene Builder: A Graphical editor for creating JavaFX applications built with JavaFX. It lets you see what your working on in a live editor so you don't have to recompile every time you want to see your changes and can be integrated into your IDE of choice. Because it works by generating an FXML form that your main class load, it's generally only used for static content. [8]
(5) Camdenre's Equation Editor: makes up the framework behind ours. While he wasn't kind enough to leave us with documentation, he file structure and code design was meticulously organized so that it was not too hard to figure out how it worked (to an extent). We wanted an simple yet elegant, interactive equation editor and this fit the job perfectly. [2]
(6) H2 Database: is an SQL style database in Pure Java, it's an in-memory database, meaning it lives completely inside our code and there is no way for user to interact with it directly. H2 is only around 2MB and is super lightweight. [4]
(7) JDBC (Java Database Connection) API: The API that H2 and a number of other databases use to comunicate with Java. Because multiple databases support it, there was more than enough documentation and resources to learn how to use it. [9]

3 PREVIOUS RA TO SQL RESEARCH

Relational Algebra to SQL translation has already been done using syntactic and lexical parsing with JLex. In their paper on "Implementation of Relational Algebra Interpreter using another query language"[6], Litoriya and Ranjan describe their system's ability to compile Relational Algebra into SQL, then run those queries on a relational database system.

Before the translate of RA into SQL, the lexical parser checks for syntax errors. If an error is found, a detailed error message about the location of the error in the RA string is forwarded to the user. This is usually a message along the lines of "missing parenthesis at position 7" or "no selection character found". Once the syntax is correct, the RA expression is converted into SQL and executed on a relational database management system (RDBMS).

3.1 Related Applications

We found very few other programs that offered Relational Algebra Calculator technology. We found one called relaX [1], but it is purely an online application and appears to use an incredibly complicated framework, so we were unable to use it as a reference.

There is also a helpful RA translation tool called "Relational Algebra Translator", but we did not have access to the source code, and what little documentation we could find was in Spanish [13]. Again, this could not be used seeing as neither project member speaks Spanish and internet translation software is iffy.

There are a number of projects that convert from SQL to Relational Algebra[3], because it's a good way to visualize the database query statement, but given that SQL to RA goes backwards from what we needed for this project, these tools did not help us much either.

4 IMPLEMENTATION AND ARCHITECTURE

4.1 Parser

The Relational Algebra to SQL parser is written in pure Java using available string splitting and equality functions. Relational Algebra equations are fed into the parser in LaTeX format through the GUI equation editor. The parser works in three steps, which will be explained below using the equation:

- "$\sigma_{age>18}(Person \bowtie Eats)$"

4.1.1 Clean Input String.
Clean the LaTeX input string of special characters such as $\bowtie$, $\cup$, and $\cap$ (see the entire list in Figure 11) and replace them with corresponding SQL phrases ("NATURAL JOIN", "UNION", and "INTERSECTION", respectively). This requires a large number of simple find and replace functions to be run before anything further can be done on the RA string. The only special case of find and replace is the full outer join function ⟗ which requires much more complex lexical parsing [5].

- "\sigma_{age>18}(Person NATURAL JOIN Eats)"

4.1.2 Split String.
Split the string by "{", "(", and "\" into a String array. This separates out each selection type (σ, Π, and γ) as well as what each selection is being operated on (i.e. the text inside the parentheses - the table being selected on - or brackets - the columns or conditions being used for selections).

- ["\sigma_", "{age>18}", "(Person NATURAL JOIN Eats)"]

4.1.3 Parse String.
Read through the split LaTeX string array, if the item in the array is "\sigma", "\sigma_", or "\Pi_" use the corresponding items in the array (generally in the following or next two spots) to create an SQL string representing the RA equation. If there is another selection or join within these items, recurse on those selections

and parse them in the same way until the full equation has been translated.

- SELECT * FROM Person NATURAL JOIN Eats WHERE age>18

If the array item is “\gamma_”, it is clear that a grouping selection is being done and some time is spent by the parser grabbing the preceding column names. For instance:

- “$_{loyal}\gamma_{loyal,count(loyal)}(Person)$” becomes “SELECT loyal, count(loyal) FROM Person GROUP BY loyal”

4.2 Database Connection

At the core of our application is a H2 database object surrounded by a wrapper class. To interact with the H2 object you have the use the JDBC API [9], which has the usual workflow of: make a connection to database ⇒ create query object ⇒ set query object ⇒ execute query object ⇒ get resulting dataset ⇒ parse dataset.

We have a wrapper class that sets up all the necessary connections when it is first instantiated and it includes a large number of functions for automating all the unnecessary code. When the program is given a query, it simply calls the query function with the SQL statement as a parameter. The H2 wrapper has functions for adding or removing tables, rows and columns, for compiling data on a specific table or the entire database, and for importing and exporting data. Any errors or exceptions get passed up to the GUI class that contains the database object class. Abstracting the entire database down to a single object was immensely helpful for building the GUI. Everything is contained in DB.java, but two additional classes are used for queries and information, DBInfo.java holds info and stats about the database and queryResult.java parses results and presents them in simple arrays for the GUI.

4.3 User-Application Interaction

We tried to make running queries and manipulated the database as easy and hassle free as possible. You can break down the core of our program into four main functions:

(1) The ability to view and edit a database
(2) The ability to input Relational Algebra Equation
(3) The ability to run relational algebra queries query
(4) The ability to display results from those queries

To address this we:

(1) Display the database in a table, with two ways to edit data: a simple window, or a text file
(2) Offer an interactive equation maker for creating relational algebra equations without the need for ugly plain text formatting or markup languages.
(3) The GUI is able to take the equation the user wrote, pipe it into the parser which then gets sent to the database where data is manipulated and returned.
(4) Display a simple table for the results, with the ability to see previous results.

In the end, the front end of our program is not very complex, but that's because it doesn't need to be. Initially I wanted to put a menu bar in the top of the application, but couldn't come up with enough front end functions to fill it.

Development for the GUI started with the Java Swing library, but then quickly traded out for the superior JavaFX library when discovered the day after. Except for a couple, very confusing and frustrating quirks, JavaFX looks and works better than Swing library. To help with development was the Scene Builder application. The Scene Builder offered a way to drag and drop GUI elements and edit their attributes easily and assign action handlers to functions (almost ridding the necessity to even write them), making it a lot easier to create the layout of the program.

Most static content was made in the Scene Builder, while the dynamic was in Java, because the GUI it outputs is contained in an fxml file, you can do dynamic content in fxml, but it is a whole lot harder than just Java. Coding the GUI was pretty straightforward, you handle GUI elements like any other objects in Java. The major challenge was designing the layout and functionality. I know how my program works because I wrote it myself, but what about someone who's never seen it before? You have to think about what information is displayed where, and what the flow of the program is (do this first, then this), it should be intuitive and you shouldn't need a manual to learn how to use it. The GUI went through five major revisions before settling on it's current layout.

Some interesting development challenges presented themselves in this project. I have never developed a front end before and certain code, while technically correct and working, caused some erratic behaviour because it was in a GUI. For example, I had a method to remove rows from the database that was querying and updating the database a high number of times very quickly, which, for some weird reason, made the result table lose and gain data seemingly at random. It was huge challenge to hunt down the bug, partly because of how interconnected everything in the GUI is, but also because everything was technically correct and working, if we were working with primitive data in an array instead of a GUI, everything would be fine. It was only when I came back and restructured the method to access the database in a much more methodological way did it work correctly.

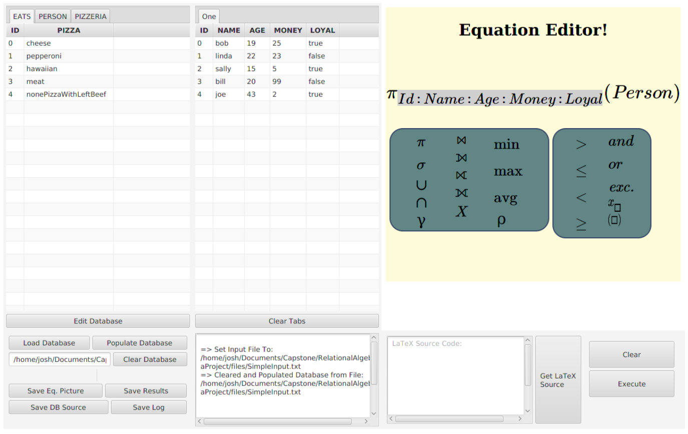
4.4 Interactive Equation Editor

We could not find a way to create or render equations in Java the way we wanted. We did not have the man-hours we needed to develop our own editor and all the ones made in Java took multiple seconds to render equations, making them useless for interactivity. But we found out it is possible to render web content in JavaFX[7], then we used Camdenre's Equation Editor for our editor[2].

Camdenre's original editor has a much different front end and has support for over 150 mathematical symbols (we needed less than 30) and is built on top of the MathJax library. Adopting Camdenre's work was an interesting challenge, as he offered zero documentation and comments were sparse, but everything was meticulously organized, even to a fault. This made it easy to figure out where to look for functions, but hard to figure out exactly what they do. Luckily for us, besides needing to add several symbols of our own, we didn't need change any core functionality. Not to mention Camdenre's GUI front end is made from simple HTML and CSS, so it was relatively simple to develop our own custom front end.

4.5 Equation Editor Usage

A quick run through of the usage of the program, see the appendix for the pictures. Figure 1 is the second newest revision and shows the general layout of the program. On the left you have a table that displays all the data from the database, below that are buttons for loading and saving databases. In the middle you have a table that displays the results and on the right is where you input your equations, where you can simply hit the execute button to the see the results. Below that is a way to get the LaTeX source of the equation if you want to copy it somewhere.



ID	PIZZA
0	cheese
1	pepperoni
2	hawaiian
3	meat
4	nonePizzaWithLeftBeef

ID	NAME	AGE	MONEY	LOYAL
0	bob	19	25	true
1	linda	22	23	false
2	sally	15	5	true
3	bill	20	99	false
4	joe	43	2	true

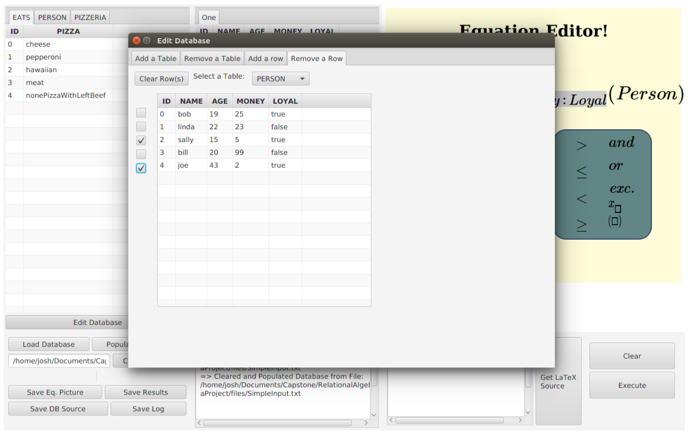
$\pi_{Id:Name:Age:Money:Loyal}(Person)$

/home/josh/Documents/Ca

```
=> Set Input File To:
/home/josh/Documents/Capstone/RelationalAlge
aProject/files/SimpleInput.txt
=> Cleared and Populated Database from File:
/home/josh/Documents/Capstone/RelationalAlge
aProject/files/SimpleInput.txt
```

Figure 1: Older Revision

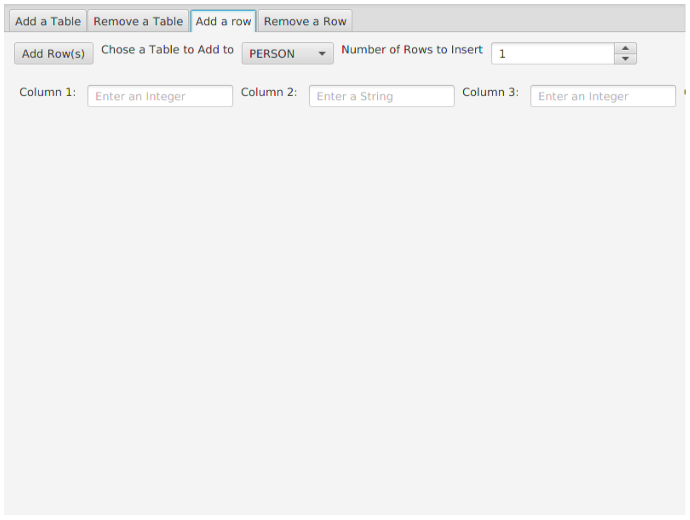
If you press the Edit Database button, it will you the window you see in Figure 2, where you can make manually edits to the database. Figure 5 (appendix) is for removing tables, Figure 3 is for adding data, Figure 6 (appendix) is for removing tables and Figure 4 is for adding tables.



Select a Table: PERSON

	ID	NAME	AGE	MONEY	LOYAL
	0	bob	19	25	true
	1	linda	22	23	false
✓	2	sally	15	5	true
	3	bill	20	99	false
✓	4	joe	43	2	true

Figure 2: Database Editing Window



Chose a Table to Add to: PERSON
Number of Rows to Insert: 1
Column 1: Enter an Integer
Column 2: Enter a String
Column 3: Enter an Integer

Figure 3: Adding Rows of Data

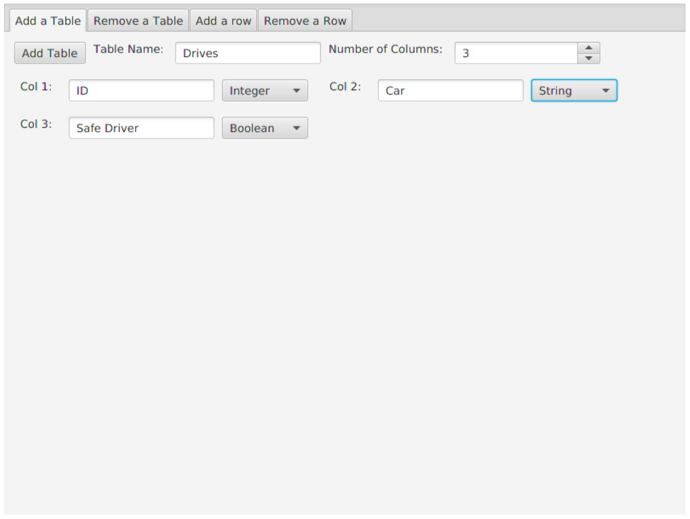


Table Name: Drives
Number of Columns: 3
Col 1: ID Integer
Col 2: Car String
Col 3: Safe Driver Boolean

Figure 4: Adding Tables

4.6 General Architecture

The general architecture of our program is not that complex, a look into our source folder shows

(1) Gui.java - The Main Class, loads the FXML form and associates the guiHandler with it, only about 30 lines long
(2) GuiHandler.java - The Brains of the program, the source of all front end functionality, what links everything together
(3) DB.java - Contains and automates all the functionality for the database
(4) DBInfo.java - Contains information and methods to parse that information about the database as a whole
(5) Parser3.java - Translates a LaTeX String to a SQL String.
(6) Police.java - Contains a whole bunch of functions for testing strings for numbers, letters, spaces, etc for catching user input errors.

(7) RAGUI.FXML - The form generated by the JavaFX SceneBuilder, holds the majority of the static content
(8) Renames.java - Theoretically working but untested backend to add table renaming functionality for relational algebra. We didn't have the time to fully implement it and develop a front end for it, but the backend works for simple queries and it was too much work to delete.
(9) QueryResult.java - Contains the results from a query to the database in simple arrays for easy use for the GUI
(10) Webstuff (folder) - Contains all the web content for our equation editor, explaining the architecture and how the editor works is it's own, much longer paper.

5 EVALUATION, RESULTS, AND ANALYSIS

5.1 Application Development Process

Prior to this project, this group only had elementary knowledge of the inner workings of SQL, Javascript, HTML, and CSS. We were chosen for this project as we had both taken Database Systems and knew the relationship between Relational Algebra and SQLite.

For the first month of the project, on Professor David Chiu's suggestion, we attempted to recreate an SQL style database in Java using a number of objects (i.e. Attributes, Rows, and Tables). We were well on our way to getting SQL-type commands, like Joins and Selects, working with this database, but it quickly became unnecessarily complex. It was also focused entirely on the database end rather than the ability to parse Relational Algebra into SQL. We eventually found the H2 database, which was a SQL-esque database in pure Java, but written by professionals who know what they were doing. Fortunately for us, it works beautifully. [4]

When we were tasked with making an interactive equation editor for our RA parser, we had no luck finding open-source equation editing code in Java. However, a GitHub user named Camdenre created a brilliant euation editor for web pages that we could adapt for our own purposes.[2] This shaved weeks, if not months, of work off our project.

From there it was pretty much smooth sailing, Rachel continued to flesh out the parser, and Josh kept designing and adding functionality to the front end. There wasn't anything too remarkable about the development process for the GUI, except that I learned you have to save all the aesthetics for the very end. If you spend a couple hours making everything look pretty, but then have to change the layout for some reason, you would loose all those hours of work, so this meant that the GUI looked pretty ugly until near the end.

Luckily for us we didn't face too many errors with github. For the most part we were always working on separate files, and when we crossed over to each other's files, it was when we met up and could coordinate to avoid github errors. At the worst, all we had to do was delete a file and pull again.

It was also relatively simple to package everything together, at first we had a little trouble, but we discovered it was because all of our file paths absolute and not relative to the project, after fixing that it was just a simple wizard in Intellij to make the Jar file.

5.2 Accuracy Evaluation - Test Cases

We like how our program turned out. All the simple to moderate equations (akin to the ones used as examples and the equations in the appendix) we tested worked, but we cannot guarantee that more complex equations will work. We think the program is most suited for when your first start learning Relational Algebra, and for doing demos for the class.

We are very happy with how the GUI turned out. If we knew what we know now at the very start of this project, we probably would have written some parts of the application differently. Additionally the program is relatively stable, we covered every exception we could find, and even then, JavaFX is very good at recovering from them. A large amount of safety functions were made that check user input before hand to, so nothing can get too out of hand.

We also did a lot of work creating JUnit tests to testing the parser (see Figure 12). Thus far, all our tests are working, but this still leaves room for the the possibility of creating an RA equation so complex that the parser gets lost amid the parentheses.

Our program is relatively slow, this isn't a problem for an application of this scale, but if we were to scale this up, the loading times would definitely suffer. There are definitely a good number of areas for optimization, however with our timeframe it was difficult to go back to improve previous work.

6 DISCUSSION

6.1 Project Significance

Our project is relatively unique. From our searches, we could only find one other program that wasn't web based, and it was in Spanish and the source code wasn't available. A web based approach would be different in every aspect than a standalone approach, so there is some significance there. Parts of our project, like the GUI and the Parser can be good references for future projects because it encompasses a wide range of design and programming practices.

6.2 Project Difficulties

Only having two people working on this project produced another series of complexities, but was also helpful at times. Because we had so few project members, we were only able to get a small amount of work done, as compared to larger project groups.

However, the small group made scheduling and workflow much simpler. It was usually easy to set up times for the group to meet up and work on our project as there was generally large amounts of open time overlap between the two of us. With more people, scheduling can be tedious and difficult.

Furthermore, we were able to perfectly break up the project into two halves that each group member was able to undertake. One worked primarily on the RA to SQL parser while the other worked almost entirely on the GUI. Because of this we had very few issues with keeping code separate, leading to only one or two GitHub push-related issues all throughout this project. This rarely happens with a larger group.

7 FUTURE WORK

Some ways this program can be made better, or generally added to, are:

(1) Functionality to rename tables
(2) Editing Database from table instead of window
(3) Additional Error Handling
(4) SableCC Support
    This would require massive rewrites of the parser
(5) Live syntax help/spellcheck for inputted equations
(6) Add more datatypes that can be used in database tables
(7) Smoother window resizing
(8) Full documentation of application use

8 CONCLUSION

This application does everything we set out to do, though it has some limitations. As such it is recommended to be used by introductory database systems students to explain the general selections and joins used in Relational Algebra. This hope is this app will give these students a visual representation of mathematical database query equations.

This application is not recommended for use by database theorists, or anyone working with massive datasets. It does its best work on small tables, like in-class example data.

9 ACKNOWLEDGEMENTS

We would like to thank Professor David Chiu for providing us with this project and teaching us what we needed to know about RA and SQL in order to finish it. We also want to thank Professor Brad Richards for keeping us on track and mostly stuck to our timeline all semester. And finally, we want to send a shout-out to everyone on GitHub and Stack Overflow, especially those who allowed us to use their open-source code, for being incredibly helpful while we were debugging our application.

A GUI

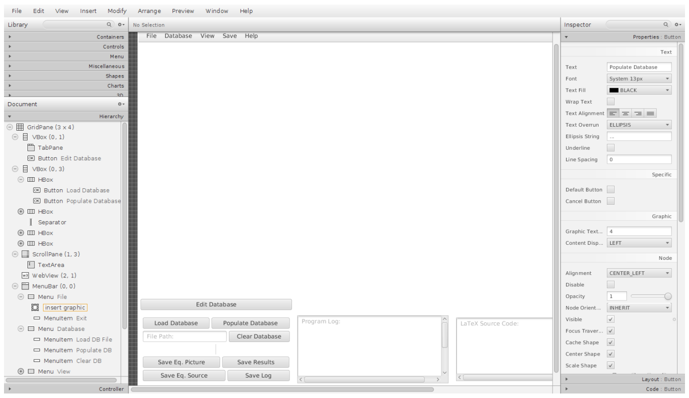
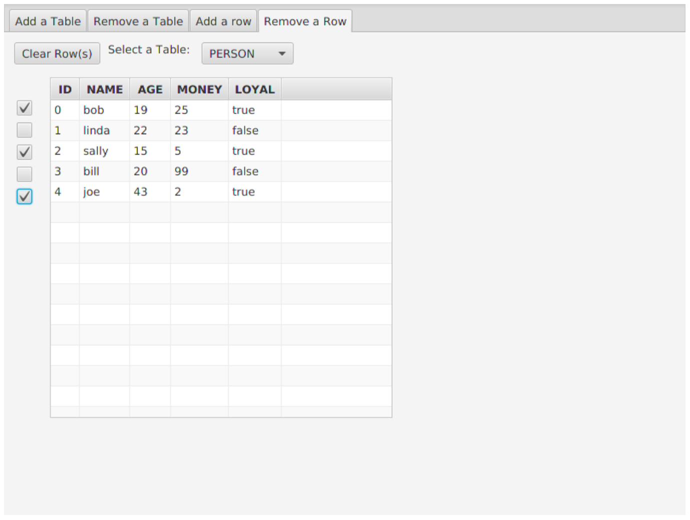


Figure 5: Scene Builder



	ID	NAME	AGE	MONEY	LOYAL
✓	0	bob	19	25	true
	1	linda	22	23	false
✓	2	sally	15	5	true
	3	bill	20	99	false
✓	4	joe	43	2	true

Figure 6: Removing Rows of Data

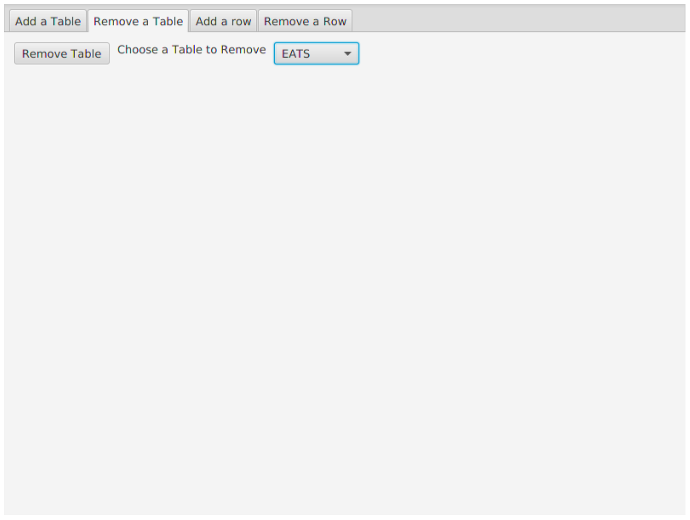
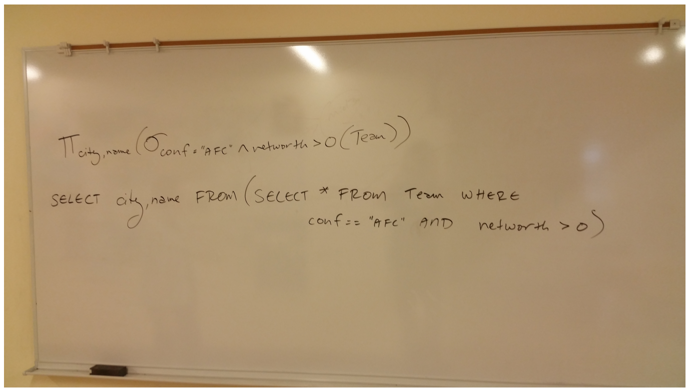


Figure 7: Removing a Table

B WHITEBOARD



$\pi_{city, name}(\sigma_{conf = "AFC" \wedge networth > 0}(Team))$

SELECT city, name FROM (SELECT * FROM Team WHERE conf == "AFC" AND networth > 0)

Figure 8: Sample Equation for Testing

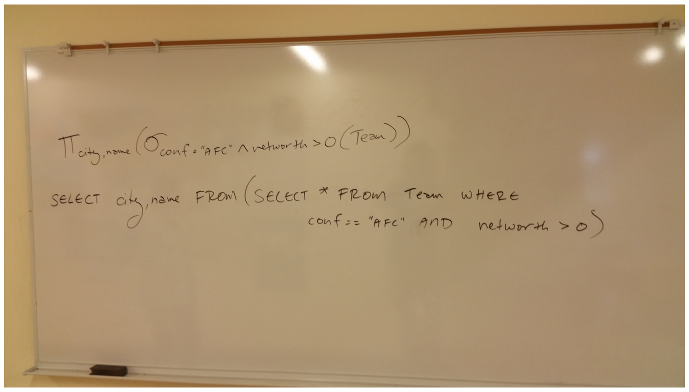

$\Pi_{city, name}(\sigma_{conf="AFC" \wedge networth>0}(Team))$
SELECT city, name FROM (SELECT * FROM Team WHERE conf=="AFC" AND networth > 0)

Figure 9: Another Equation Translated by Hand

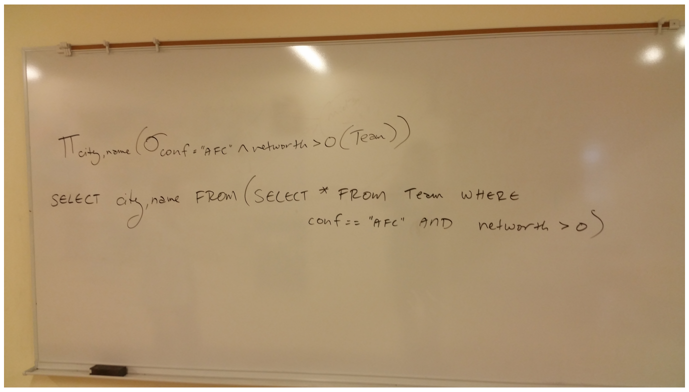

$\Pi_{city, name}(\sigma_{conf="AFC" \wedge networth>0}(Team))$
SELECT city, name FROM (SELECT * FROM Team WHERE conf=="AFC" AND networth > 0)

Figure 10: A Third Hand-Translated Equation

C CODE

Figure 11: All latex commands dealt with by the parser:

```
private final static String PI = "\\Pi_";
private final static String SIGMA1 = "\\sigma";
private final static String SIGMA2 = "\\sigma_";
private final static String AGGR = "\\gamma_";
private final static String NATJOIN = "\\bowtie";
private final static String CROSSJOIN = "\\times";
private final static String UNION = "\\cup";
private final static String INTERSECT = "\\cap";
private final static String AND1 = "\\vee";
private final static String AND2 = "\\land";
private final static String OR1 = "\\wedge";
private final static String OR2 = "\\lor";
private final static String EXCEPT = "-";
private final static String LOJ = "\\loj";
private final static String ROJ = "\\roj";
private final static String FOJ = "\\foj";
private final static String GEQ = "\\geq";
private final static String LEQ = "\\leq";
private final static String MAX = "\\max";
private final static String MIN = "\\min";
private final static String AVG = "\\avg";
private final static String SUM = "\\sum";
private final static String COUNT = "\\count";
```

Figure 12: Example JUnit Test

```
public void testCase1() throws Exception {
    String sampleInput1 = "\\Pi_{name}(Person)";
    p = new Parser3(sampleInput1, d);

    String test = p.runTest(sampleInput1, 1, p);
    String answer = "SELECT name FROM Person";

    assertEquals(test, answer);
}
```

Figure 13: Example GUI Objects

```
final double INV = 2.5;

private DB db;
private DBInfo dbInfo;
private queryResult curQR;

private ObservableList<String> addTableColumnOptions;
private ArrayList<String> tableNameOptions;
private ObservableList<String> tableNameOptionsFX;

//for the edit Database Menu
private Stage editDBStage;
private Scene editDBScene;
private TabPane editDBTabPane;
    private Tab editDBAddTableTab;
    private Tab editDBAddRowTab;
    private Tab editDBClearTableTab;
    private Tab editDBClearRowTab;
```

REFERENCES

[1] Johannes Kessler BSc. *RelaX - relational algebra calculator.* Databases and Information Systems Group at the Institute of Computer Science at the University of Innsbruck. http://dbis-uibk.github.io/relax/
[2] Camdenre. 2015. *Equation Editor.* https://github.com/camdenre/equation-editor
[3] S. Ceri and G. Gottlob. 1985. Translating SQL Into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries. *IEEE Transactions on Software Engineering* SE-11 (April 1985), 324–345. Issue 4. DOI: http://dx.doi.org/10.1109/TSE.1985.232223
[4] H2. 2017. *H2 Database Engine.* http://www.h2database.com/html/main.html
[5] Lasse V. Karlsen. 2009. (January 2009). http://stackoverflow.com/questions/406294/left-join-vs-left-outer-join-in-sql-server
[6] Ratnesh Litoriya and Anshu Ranjan. 2010. Implementation of Relational Algebra Interpreter using another query language. *2010 International Conference on Data Storage and Data Engineering* 1, 1 (February 2010), 24–28. DOI: http://dx.doi.org/10.1109/DSDE.2010.33
[7] Oracle Technology Network. *JavaFX library.* Oracle Technology Network. http://www.oracle.com/technetwork/java/javase/downloads/index.html
[8] Oracle Technology Network. *JavaFX Scene Builder.* Oracle Technology Network. http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html
[9] Oracle Technology Network. *JDBC API.* Oracle Technology Network. https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
[10] TutorialsPoint. *Relational Algebra.* TutorialsPoint. https://www.tutorialspoint.com/dbms/relational_algebra.htm
[11] TutorialsPoint. *SQL SELECT Query.* TutorialsPoint. https://www.tutorialspoint.com/sql/sql-select-query.htm

[12] TutorialsPoint. *SQL Using Joins*. TutorialsPoint. https://www.tutorialspoint.com/sql/sql-using-joins.htm

[13] Universidad Nacional Costa Rica: Escuela de Informatica 2013. *RAT: Relational Algebra Translator*. Universidad Nacional Costa Rica: Escuela de Informatica. http://www.slinfo.una.ac.cr/rat/rat.html