Rachel Hirsch
Josh Wagner

Advisor: David Chiu
Capstone Project Proposal

1) Project Overview

Prof Chiu's project is an application that runs relational algebra queries. More specifically, we will design a Java API to set up attribute, tuple, and relation classes, then run queries on that data set. This API will be able to run arbitrary relational algebra operations using Java dot notation. We will then create a language to use to input these queries then parse those queries into Java.

Many of the language used in databases (sql, mysql) are based on relational algebra, and knowing the algebra can greatly aid in your understanding of the language and databases. However it can be hard to learn the algebra due to not immediately seeing the effects of your work, like you would with sql. The purpose of this application is to immediately display the results of a relational algebra equation.

This project will be completed in a series of three phases. Phase one is primarily getting the Java API for relationship operators working in the right way. This phase will include at least a week of testing to make sure all operators are correctly implemented and producing the desired output.

Phase two is more difficult. We will create a way to parse relational algebra expressions - written in a new language, like HTML or LaTeX - and turn it into evaluable Java statements using the API from phase one. We will use a language parser tool like Yacc or SableCC to develop input-encoding schemes and context free grammar to interpret relational algebra expressions. This phase will also include at least a week of extensive testing to make sure the application works before moving on to extra functionality. Once we finish phase two, the project will be considered completed and successful.

Phase three includes this extra functionality. If we have time we want to build a way to teach Database Systems students about normalization, and create a way to solve for closures, canonical covers, key, et cetera as well as table decompositions into 2NF through BCNF.

2) Challenge Analysis

Challenges in this project include testing, creating a new language to parse relational algebra into Java, regular meetings, and sharing code. Testing will mostly revolve around figuring out whether legal expressions are being translated correctly from relational algebra to Java. Prof Richards has offered his expertise with compilers and tools like SableCC to help us with the language parser.

No group member on this project, including our project advisor, has any experience using SableCC,so this requires lots of learning on our part - mostly playing with the application over winter break to get a handle on it.

Fortunately the rest of the project will be done in Java, which we are all comfortable with. This lends itself well to saving time to work on the project throughout the semester instead of spending all our time figuring out how to use new programs.

Sharing code will probably be done through GitHub or by passing files back and forth on Slack (which is simple enough because there are only two of us working on the project, not including our project advisor). Part of the challenge will be becoming familiar with GitHub to use it properly, so that it doesn't hinder our progress (mostly Josh, Rachel knows it well). We want to be able to meet at least once per week with David to discuss what needs to be completed each week and what to do next. Slack will also be used for communication between project members and our advisor.

One of the challenges outside of building this project is testing it. We want to make sure that all relational algebra (if it is legal) that is inputted can be executed. We will have to figure out a method to use to test the output (e.g. brute force vs. dynamic, Java class vs. built in IDE test tools) and also how to check if the answer we computed was correct.

Finally, we need to figure out exactly who is going to be using this application. This may just be a grading tool used by Database Systems professors, but depending on how far we get with stage three, this may be used by students to teach them difficult database concepts. So as such, many of the extra features will be tailored for whoever will be using this application.

3) Descriptions of functionality expected

Attribute classes will include name and type variables, plus setter and getter methods. Tuple classes will have a list or map of attribute values, as well as setters and getters. Relation classes will have a name variable, a list or set of Tuples, and methods such as Insert, Delete, Select, Project, Rename, Union, SetKey/SetForeignKey, joins, and grouping functions.

Sample input would be something along the lines of:

```
\project_{“ssn”, “name”}(\select_{ssn=999999999}(user))
```

Which would be translated to Java code like:

```
String[] attrList = {“ssn”, “name”};
user.select(“ssn==999999999).project(attrList);
```

We still need to formalize the format to input and output info, but David was able to give us a fairly thorough walkthrough of what functionality will be expected within each class.

The core functionality of our compiler is that it must be able to take user input, correctly parse it into the Java structure we described above, test if the expression is legal, and correctly execute it.

Additionally, it must be able to output the result of the relational algebra equation in a easily readable manner.

Extra features will include normalizations, table decompositions, solving for closures, keys, GUI elements, and piece by piece explanations of the equation, among others. While this functionality is not required and not critical for the completion of the application, it would be very nice to implement some of them if we have the time.

At the end of this project, at the very least, the user should be able to input relational algebra queries into our application using the HTML/LaTeX-like language we will develop, the application will test if the query is legal, and if it is, execute it and output the results.

4) Timeline

This project will be broken up into three phases, separated by at least one week periods of testing.

Phase 1: ~3 weeks
Testing Phase 1: ~1 weeks

Includes: Building the class structure and methods. Relation class, Tuple class, Attribute class, getters/setters, basic operations (insert, delete, join, rename, etc.) Testing will mostly be making sure that we can execute queries in Java dot notation, and making sure all our methods work together.

This is a pretty straightforward phase, the hard part will be making sure all the classes work together nicely but it is nothing we haven't done before in Java. Phase 1 sets the groundwork for phase 2.

Phase 2: ~4-6 weeks (the timeline for this phase is still unclear - some estimates say just a few weeks, others say the rest of the semester)
Testing Phase 2: ~1-2 weeks

Includes: Formalizing our input and output, converting between relational algebra to Java code, creating a compiler to execute the relational algebra turned java code, implementing language parsers (using SableCC, Yacc, etc), creating I/O for user input (GUI or console). Testing for this phase will be incredibly intensive as we need to make sure that all legal equations can be executed, regardless of their complexity.

This is the most complicated and longest phase, and where the new material we need to learn comes into play. Not only do we need to create a compiler to convert relational algebra to Java, we need to test it intensively to make sure all the conversions are correct. The hardest part of this phase will be converting the input into Java code, and making loops that will execute the code in the correct order.

Phase 3: remaining time after completing Phases 1 and 2
Testing Phase 3: ~1 week, assuming completion of Phase 3

Includes: Making/finalizing extensions to our application.

Assuming we complete phase 2, phase 3 will be about implementing extra features to our program such as additional GUI elements, accessibility functions, normalization, table decompositions, solving for keys, etc. As we develop the core of our application, we will be thinking and writing down what features we think would be most helpful to Database Systems students. Because we are unsure of exactly how long phase 2 will be and thus unsure of how long phase 3 will be, nothing required or critical for the application will be developed in this phase; only additional features will be created. As we figure out exactly who this application is for, we will be able to make features tailored for that purpose.