Lab 1: Why Databases?
Part I

Jeff McGovern and Chris Hunt

8 January 2015

1 Initial Decisions

1.1 Programming Language

Java became our language of choice, primarily due to our mutual familiarity with it. Parsing files is straightforward using the methods already built into the Scanner and String classes. The standard library gives us a lot of power to make interactive prompts, pass data structures around, and output the results to the screen.

1.2 Environment

Vim and Unix is our primary build environment. Again, mutual familiarity with both the Vim text editor and Unix operating system makes this a natural choice. Additionally, due to the limited number of files in the project, we did not feel that there was an inherent advantage to using a more complex environment since the number of dependencies was small.

2 Internal Architecture

2.1 Description

The structure of the database consists of three parts:

- DataBase Management System (DBMS)
  - `DBMS.java`
- Single Record Object
  - `Student.java`
- Driver
  - `schoolsearch.java`

The Driver calls upon the DBMS to read the provided CSV file of students. The DBMS then parses the file, where each line in the CSV file becomes a Student object, and stores the resulting records as a `LinkedList<Student>` for straightforward iteration.

After parsing, the Driver requests input from the user and responds accordingly. Upon querying, the DBMS searches through the entire list of Student objects and returns a new list containing the relevant records. Finally, the Driver prints the results and awaits further input.

2.2 Pitfalls

Iterating through the entire database to find relevant queries is not the most efficient way to retrieve the relevant results. Another method is hashing each value into separate tables based off of expected queries.

For example, from the lab requirements, we know that the user will want to search for students in five (5) possible ways:

- Student Last Name
- Teacher Last Name
- Classroom Number
- Grade Number
- Bus Number

The advantage of having five separate hash tables (with each table representing an expected query) is that the DBMS can create each hash table once and then consult the correct table per query, returning results in $\mathcal{O}(1)$ time. However, this DBMS design requires that the implementors know the format of all possible queries from the end user, and so this design is not flexible or easily maintained.

2.3 Justification

Our sanity became priority. Figuring out how to wrangle five different `HashMaps`, we quickly realized, would have been a nightmare. Furthermore, carrying five times as many pointers, depending on the dataset, may have had egregious consequences for memory. Of course, this makes our implementation an $\mathcal{O}(n)$ lookup each time, but computers are fast enough and the database is simple enough that we feel the difference in speed is negligible for this application.

Furthermore, a straightforward implementation allows us to easily add features later on. Since we know we will have a "Part II" to this lab, we felt that keeping our design flexible would allow us to adapt more quickly and easily to future requirements.

3 Task Log

- Chris and Jeff discussed implementation and concluded that a straightforward list would create fewer problems later on down the road.
- Jeff built the Student object with basic getters, constructors, `equals()`, `compareTo()`, and `toString()` methods.
- Chris built the DBMS to read the CSV student file and return relevant lists.
- Chris built the schoolsearch driver to request input and print the relevant results to screen.
- Jeff added ability to do a basic sort of a list of students for a hidden troubleshooting option.
- Jeff modified the print output to have clean columns, exit gracefully, and time the queries.
- Chris added the grade lookup option and reviewed the code.

4 Testing

In order to reduce bugs and keep complexity to a minimum, the project was segregated into three different source files (the DBMS, Student and Driver classes respectively). The `Student.java` class was straightforward as it just had to implement getters, equals, compareTo and toString methods. The `DBMS.java` class was also simple, with the most complex portion being the input csv file parsing. Because of this, both of these source files were relatively bug free (minus a stray typo here or there).

The final driver, `schoolsearch.java`, was much more prone to bugs due to large number of input conditions that needed to be checked (and their associated errors). In order to keep things managable, methods were created to breakout as much functionality as possible, and exceptions were relied on to bail out of these functions as needed.

As for testing the program, each required input (as specified in the program spec) was tried against the program's interactive mode to ensure that all the options worked. Additionally, to ensure that our program

handled incorrect entries the following inputs were attempted: improper spelling, characters, numbers, emptry strings, and escape sequences. Finally, our program was also tested with both a missing and an incorrectly formatted "students.txt" source file.

We performed a simple test for the timing of the database when the database is somewhat extremely large. A simple bash one liner concatenated the `students.csv` file 1000 times into a new file, resulting in a database with 60,000 records in it. This brought query times to 0.010 seconds, as opposed to 0.000 for the smaller database with only 60 records.

5 Final Notes

The program spec stated that our program should handle a query that requested a printout of all students in a specified grade level, however, there was no explicit format for this query in the spec. Because of this, we chose to use the following query format: `G[rade]: <number>`.

Ultimately, our decision to use a straightforward method of holding the records in the database did not hurt our runtime for a large dataset. Such a straightforward design will hopefully still help in the future when we need to add features to possibly handle more complex queries.