Lab 1: Why Databases?
Part II

Jeff McGovern

13 January 2015

1 Implementation Decision

Ultimately, modifying the existing code to handle the new database file architecture seems to be the most straightforward way to handle this. Minimal changes were made to the overall structure, but additionaly dat structures were built to better handle the new disjoint database files.

1.1 Assumption

One assumption I make about the link between a student and their teacher could have frustrating consequences in the future, primarily due to this footnote:

> "In our test file, there in one teacher per classroom. However, your search algorithm cannot assume that it will always be the case."

If there are multiple teachers per classroom, does that mean that one student has multiple teachers? Nothing in either specification mentions, or even hints at multiple teachers per student. Furthermore, taking the footnote to its fullest extent contradicts the following specification:

> "...the new `schoolsearch` program shall have the same functionality as the `schoolsearch` program, working exactly the same way, i.e., it shall accept the same search commands and produce the same results."

Accepting data with multiple teachers per classroom is impossible with the previous implementation (see the Part I data format specification), making it impossible to adhere to both the footnote and the above specification. Thus, each student has only one teacher in the following implementation.

2 Internal Architecture

2.1 Description

To ease the transition, a merged StudentTeacher object took place of the previous Student object, by containing the exact same functionality, but holding the data, instead, in a Student and a Teacher object, containing the items in each record from `list.txt` and `teachers.txt` respectively.

The structure of the database now consists of five parts:

- DataBase Management System (DBMS)
  - `DBMS.java`
- Merged Record Object
  - `StudentTeacher.java`
- Student Record Object
  - `Student.java`

- Teacher Record Object
  - `Teacher.java`
- Driver
  - `schoolsearch.java`

The Driver calls upon the DBMS to read the provided CSV files of students and teachers. The DBMS then parses each file, where each line in the CSV file becomes a Student object and Teacher object, and stores the resulting records as a `LinkedList<Student>` and `LinkedList<Teacher>` for each for straightforward iteration. Immediately after parsing, the Driver builds a `LinkedList<StudentTeacher>` that merges each Student with its classroom's Teacher into a StudentTeacher object based off of the classroom number.

After building the database, the Driver requests input from the user and responds accordingly. Upon querying for Part I queries, the DBMS searches through the entire list of StudentTeacher objects and returns a new list containing the relevant records. Finally, the Driver prints the results and awaits further input.

2.2 Added Functionality

The ability to query students by their grade existed in Part I, so it is not discussed.

Added functionality to query teachers by their classroom takes advantage of the List kept by the DBMS. A simple iteration through every Teacher while checking for classroom number created a list that, when returned to the Driver, prints to screen.

Finding every teacher for a particular grade is slightly more complex. While iterating through each merged record, once a grade match is found, a HashMap tracks which teachers have been added to the serach results. This required building a key based off of the teacher data (done during parsing) and hashing into the results table. A List of Teachers returns the results to the Driver, which are printe to screen.

2.3 Pitfalls

Pitfalls with the retained architecture from Part I are in the writeup for Part I.

Merging Students with their Teacher is not clearly defined in the specification, as discussed in Section 1.1. This means that if a Student is in a classroom shared by more than one Teacher, then only the first Teacher found in `teachers.txt` becomes the student's teacher. If, however, the Teacher for the Student is in fact the second Teacher that teaches in that classroom, or both, then we have a difference between the new database format and the old database format that needs resolution.

3 Task Log

- Created new data structures to merge separate files
- Verified previous functionality
- Added ability to query teachers by classroom
- Added ability to query teachers by grade

4 Testing

Primary testing came in the form of verifying previous functionality. Verification of queries was the next step. Utilizing the secret "all" command, I performed queries for each query type and ensured that results were as I expected.

A similar "large" database test situation to last time (a database of 60,000) records yielded similar results. Sub-millisecond times for hte database files provided became 1 millisecond times when concatenated upon itself 1000 times.

5 Final Notes

Adding the features for Part II proved straightforward, but required building separate systems to query for a very finite set of queries, and took a while to code. Dynamic queries that include sorting or every possible combination of query type will end up requiring very complex code to handle them effectively.

5.1 Purpose and Takeaways

From this, I learned that the structure of the data very much determines the structure of the database (see Section 1.1). With this style of DBMS creation (using procedural languages), it's important to have a solid model for the data that encompasses every possible way of access to that data. If data is related, there must be a clear relation between that data, or else contradictions arise. Furthermore, adding the relational functionality (between Students and Teachers) while straightforward, took a lot of time to code nd test for.