keyDB - MS2 Progress Report

keyDB is a console-based Database Management System (DBMS) built in OCaml

Vision

We are still happy with our vision for this project, however we made one (hopefully last) big picture change to our system this sprint. Although we thought we had figured out our data structures in our first sprint, we ran into issues using hash tables as the data structure defining our rows (because our empty row was pointing to the same hash table every time we referenced it). Instead, we've gone on to use association lists to represent our rows. We figured that, although using lists increases complexity for certain functions (like update_cell), our write function (O(n)) is called after every function and is a lower bound on the possible complexity of our entire read-function-write process in executing a SQL query. Thus, using hash tables isn't actually as efficient as we thought.

We've also changed our table data structure to a record to hold the key and column name information, as well as the table itself. We are now feeling very confident about our new data structures. During this sprint, we've also made significant progress in implementing the dbms itself, which we're very happy about.

Summary of progress

Our main goal with this sprint was to add functionality to our system so we could read csv files into the system and convert them to our OCaml table type, and then write from our OCaml table type into a csv file. This enabled us to actually manipulate the database tables, and do things like display or change data as necessary. It will be much easier to build more functionality now that we have a working OCaml table type. Unfortunately, however, we did not get to this for our demo, so we explained our read/write functions and refactoring our project to change our table and row data structures.

After our demo, we started working on some dbms functions (like insert, remove, and update). We were mainly play-testing these functions to see that they worked. After Database.read and Database.write were implemented, the rest of the project is simply manipulating our table type. These core functions are not yet complete, and we plan to finish them during MS3.

Activity breakdown

Kevin is no longer on the team.

Ben:

- Rebuilt row with new association list data structure and converted table to record
- Implemented Database.write, Table.to_csv and helper functions

Donal:

- Implemented Database.read
- Implemented table manipulation and data display functions Table.insert_row, Table.update_cell, Table.remove_row, and Table.select_all

Productivity analysis

We were very productive during this sprint. We got done most of what we wanted to accomplish. During the first week or so, we agreed to not work on the project much (due to prelims and busy schedules). There was also the possibility that we would abandon our project and join other groups because Kevin left. However, despite this initial slow start, morale remained high and we both worked a lot on the project during the second week. Because there are now only two people in the group, decisions are easier to make and it's easier to split up tasks. Overall, this sprint felt more productive to both of us.

Scope grade

We think we achieved good scope for this sprint. We felt like we completed the bulk of the project (reading a csv file and converting the string to a table, applying functions to the table, then converting the table into a string and writing it to the csv file). However, we didn't complete our projected excellent scope (select with conditions). That might be a larger hurdle than we anticipated, since we'll have to lex/parse the conditions and evaluate the result. But other than that, we almost finished the core functionality. Implementing add_column, remove_column, sum_column, count, and count_null should be relatively simple. The log and undo functions should be straightforward in MS3.

Goals for next sprint

Satisfactory:

1. Implement Table.select (with conditions)
2. Implement add_column and remove_column, sum_column, count, count_null

Good:

1. Implement make_log, get_log
2. Allow user to use get_log through toplevel with [log]

Excellent:

1. Implement undo
2. Allow user to use undo through top level with [undo]