keyDB - MS1 Progress Report

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

Vision

Our current vision for the system is largely the same as when we began the project, with the largest change being the data structure we intend on using. We began by using a hash table for our tables to match keys to rows, and an association list for each row to map column names to values. We switched this around to have association lists represent tables, and hash tables to represent the rows. That way, column lookup would be amortized constant time and row lookup would be linear time (instead of the other way around). We are currently considering the upsides and downsides to using maps, the non-imperative alternative to hash tables.

We also underestimated the size of this DBMS project; our original goal for this sprint was a bit ambitious. This could also have been because we lost a teammate and are down to a team of 2. We may need to consider giving up nonessential commands like sum and count and prioritize more fundamental functionality.

Summary of Progress

We wrote all the interfaces for the modules we intended on using. As we continued work on our project, we updated them to match our evolving vision for the system. We have yet to implement the interface for our Log module, but this is scheduled to be part of the excellent scope of our final sprint. In terms of functionality, we have finished implementing the toplevel interface and parsing of commands into the system. This will catch errors with commands and display a message explaining what happened, before allowing you to enter another command.

The commands that have been fully implemented thus far are just to create and drop tables, to show a list of available commands, and to quit. Creating a table will initialize a new file in the databases directory where all tables are held, and dropping it will remove it from the directory. Typing help will display a list of all commands that are available to the user, and calling quit will clear the log and exit the REPL. These functions were what we demoed during discussion. We built unit tests for the create and drop table functions, and play-tested the other functions. Lastly, we got a start on reading and writing tables, which are backend functions that convert files into our data structure for tables (and vice versa).

Activity Breakdown

Ben:

- Initialized git repository and wrote initial files (makefile, _tags, merlin, and test files)
- Implemented the Command module to parse user commands into Command variants
- Implemented the Main module to run the interactive REPL
- Wrote unit tests for Database functions

Donal:
- Wrote some mli files (types and specifications)
- Implemented the Database module's create_table and drop_table functions
- Started the Database module's read and write functions

Kevin:
- Helped write some of the mli files and determined where each function belonged

Productivity Analysis

Although we did not reach our goals, the team was quite productive, especially when working together in-person. Due to the complexities of building a database management system, the early stages required a lot of communication on how exactly we wanted to implement the system. This led to difficulties in working on our own, and as such we could only do so much in between meetings. This had consequences in achieving our planned goals. The overhead required to build the foundation of the system in turn made our initial objectives much further out of reach. We had made our vision far too ambitious and focused too much on the big picture, rather than what was actually needed to be done for the early stages of this project. Nonetheless, the big picture reasoning we did to set our first goals will be helpful in the long run now that the project is off the ground and running.

Scope Grade

I think we achieved good scope for this sprint (even though we only completed our charter's satisfactory scope). Our charter didn't take into consideration the "hidden" functions that were necessary for our commands (like read and write in the Database module). Also, unlike previous assignments, we had to start the project from scratch (including writing the initial files) which took some learning. We made significant progress in deciding the data structures and implementing module-specific functions. The REPL is almost completely finished, which we thought would take a large chunk of a sprint. Overall, we put in more effort than a normal assignment, but didn't finish everything that we wanted to for MS1.

Goals for Next Sprint

Satisfactory:
1. Implement Database.read and Database.write
2. Implement Row.update
3. Implement Table.insert_row, Table.update_cell, and Table.remove_row

Good:
1. Implement Table.select_all and Table.get_columns
2. Allow user to use commands through the toplevel
3. Lex and parse conditions

Excellent:
1. Implement Table.select (with conditions)