Midterm Project Charter

Team members: Daniel Stabile (dis52), Robert Morgowicz (rjm448), Vivian Li (vml39)
Regular status meeting: Monday and Wednesday 11:00 am - 12:00 pm

Project proposal: A database management system that stores a user-generated database schema and allows the user to query their data using standard SQL queries. The user will be allowed to create tables, input data, delete data, and select data with their own parameters. The output will consist of writing the queried data to terminal and an automatically-generated file on the user's local machine.

Key features

- Tracking the state of the database
- Support for basic SQL queries and parameters: INSERT, DELETE, SELECT, WHERE, LIKE, ORDER BY, JOIN
- User interaction with the database through the command line
- Allowing user input and generating the queried output as files on the user's local machine

Narrative description of system

The system that we intend to build is a database management system (DBMS) with certain properties from relational database management systems (RDBMS), following examples such as MySQL and PostgreSQL. Common features and implementations of a DBMS include: storing data in files as opposed to tables; accessing data elements individually; supporting smaller quantities of data; and allowing for data redundancy. In addition to these properties, the DBMS we plan to build will support relations between tables and standard SQL queries, as prevalent in using a RDBMS.

As described in the structure of a DBMS, the user's data and database schemas will be stored in text files. The schema will include tuples of table names, table fields and their required input types. Text files storing the data in the tables will contain tuples that correspond to each row of the database, following the schema set previously. The system will allow the user to create tables and insert data either from the command line or using input files with a sequence of queries. The queries will be parsed and matched against a variant containing the standard SQL queries. The system will support the queries listed above in the key features. The text will then be scanned with the appropriate parameters and the data matching the query will be returned. The data will be relayed to the user either back through the command line or in an automatically-generated text file displayed in the format of a table.

Roadmap:

MS1 (Alpha) Wed, 11/06/19:

1. Satisfactory Scope
    a. Create input folder with schema and example tables in .txt files
    b. Track state of database (write to local machine)
        i. Text files
        ii. Input & output folder
            1. Input: contains database (which contains schema and tables of data) and file with all query commands
            2. Output: print result of query to command window
2. Good Scope
    a. Query parsing (command line)
        i. Create code to parse all SQL codes, even if we don't implement them all
3. Excellent Scope
    a. Implement SCAN operation
        i. SELECT * IN table

MS2 (Beta) Wed, 11/20/19:

1. Create input folder based on create database/table command
    a. Create table (through queries)
2. Implement queries: INSERT, DELETE, SELECT, WHERE, LIKE, ORDER BY
3. Print result of query to .txt file in output folder
4. Change database representation type

MS3 (Release) Mon, 12/09/19:

1. Running multiple queries from a .txt file that is executed line by line
2. Attempt to implement join operation. May create a simplified version that only joins on one feature

Preliminary design sketch:

The two key components of the system are the data itself and the operations (commands) which can be performed on that data.

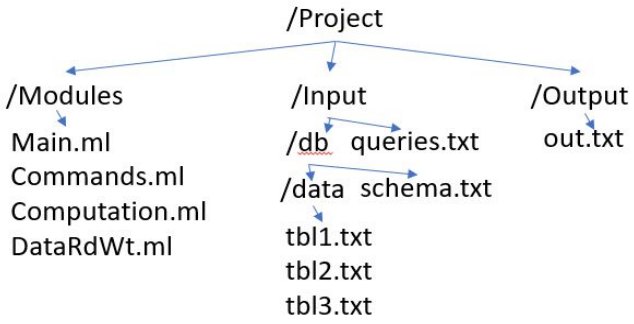
Command Modules:

- Main.ml: This will include an executable main file (similar to A2 and A3) which can read user queries and print output data to the command line. We also plan to add support for file-to-file operations, that is, taking an input file of queries and outputting to an output file of requested data.
- Commands.ml: We will need a parser, capable of reading commands in SQL syntax.
- Computation.ml: Based on the parsed commands, we will need a computation module, which implements the actual function of these commands.

- DataRdWt.ml: In order to operate on data, it will need to be passed into the computational module from the storage files or write to them, so we will need a data reader and writer which can read the files in the format we specify, translate them into Ocaml data structures and back, and modify or generate them as part of functions in the computation module.

Data:

- We plan to store the data tables themselves as text files containing tuples and have a schema file which maps column names to the tuple locations in these files. We can put logic to decode these files based on the schema in the data read/write module, as well as ways of adding to these files or generating new ones.
- Inside of the actual Ocaml modules, the schema can be implemented as a function which takes column names and maps to positions in tuples. The tables can be most easily implemented as lists of tuples, each tuple element being a row of the table. We could also combine both the schema and the tables into a single list of records, but we're leaning towards the former implementation.
- The modules which are run as part of main.ml will be in their own subdirectory alongside input and output directories. A preliminary file organization of the system is pictured below



Libraries:

- We are not planning to use any 3rd party libraries at this time. However, we will be drawing inspiration for parsing and executing SQL queries in OCaml through existing libraries such as sequoia and macaque.

Testing:

- We will be making unit tests for each module as we write them, testing all functions exposed to other modules in the system. The hope is to start from the bottom (files) and work our way up (to main), achieving comprehensive testing of functions in each module before building the next module atop it. Once main is functional, we will be testing simple queries in the top level. The introduction of more complicated queries may require implementing new functions on the lower modules, in which case they will need to be unit tested before being tested in the top level. We will hopefully be writing much of the code as a group, but should any member write code on their own, they will

be responsible for testing and documenting that code and should notify the other members of their changes.