Milestone 1
CS165, Fall 2016
Kevin Zhang

1 Goals

In this milestone, we seek to build a basic column-store database, with create, insert, load, and select functionality:

- Create: databases, tables with any number of columns, columns with integer data.
- Insert: arbitrary integer data.
- Load: load data from a file into a database.
- Select: perform basic range select queries on the database.

2 Technical Description

We will need to implement a variable pool and a database catalog as the two main components of this milestone. I have taken the liberty of reorganizing the codebase and modifying some of the API so that things make more sense; in particular, I've refactored a lot of the parse and execution code.

Our data will be organized on disk in folders; a `data` folder will contain a folder for each database, each of which will contain a folder for each table, each of which contains a file for each column. We organize our data in this way because most queries will operate on specific columns, and so our data on disk should be organized to optimize retrieval time of data in a single column. Spreading a single column across multiple files incurs unnecessary IO overhead, and squeezing multiple columns into a single file causes our code to load too much data to access a single column. This structure may change later on when we have clustered indexes. I will have a IO interface that handles all file interaction.

Insertion will be done by creating data in memory; data will then be written to disk on shutdown, or when we run out of memory (when `malloc` returns `NULL`). I also have to decide how to store data in a file; for a simple scan operation, this does not matter that much, but when I'm implementing a B-tree scan, I think it may be advantageous to store a page's worth of data on each line at once. I need to test this theory when I get to that point.

Our database catalog will be a struct defined as a `DbCatalog` object; it will contain information on the current database, what tables are available, and what columns are available in each table. This catalog will be written to and read from disk from a top-level `catalog` file.

Selects will be performed by reading column data into memory and doing a simple scan over it; I will design it to read in chunks of data at a time and aggregate data as it goes, so that we don't overflow memory from excessively large columns. This will be useful for parallelization later on.

Variables will be organized into `Result` objects, each of which will contain the name of the intermediate variable and a pointer to a `GeneralizedColumn` object which contains the number of tuples, the type of data, and an array of the tuples (the provided API does not follow this structure; I will modify it to match). I will have a `ClientPool` object that consists of an array of `Result` objects for each file descriptor, and a single `VariablePool` object that consists of an array of `ClientPool` objects.

3 Evaluation

Not finished yet!

4 Challenges & Open Items

The biggest challenge for this milestone so far has been understanding and reorganizing the provided code.