Milestone 2
CS165, Fall 2016
Kevin Zhang

1 Goals

The goal of this milestone is to build a system which uses indexes to improve query performance. Indexes must be either clustered or unclustered and organized the column in sorted order or in a B-tree. Clustered indexes will have a leading column that all columns in the cluster are sorted according to. We assume that all clustered indices will be created before any data is inserted.

2 Technical Description

To accomplish this milestone, there are a few things we need to be able to create:

- An initial unclustered, un-indexed copy of the data upon loading and insertion.
- An unclustered, sorted index by maintaining a single file with a list of pointers to rows in the original dataset.
- An unclustered, B-tree index by maintaining a single file with a tree that contains pointers to rows in the original dataset.
- A clustered, sorted index by maintaining an entire copy of the original dataset, where every column in the table is sorted on disk according to the index, and the index is a sorted list on disk.
- A clustered, B-tree index by maintaining an entire copy of the original dataset, where every column in the table is sorted on disk according to the index, and the index is a B-tree on disk where each node contains a single data value and a row index.

The size of nodes in our B-trees will be a single page, to minimize the number of pages accessed. We have to figure out exactly how many values will be stored in each node; this can be done by using system calls to retrieve the page size and calculate the size of a `Node` struct, and ensure that the `Node` takes no more space than that of a single page.

We also need to be able to update all existing indexes, which means we need to add information about indexes to our catalog file. Our catalog will therefore contain an array of `DbIndex` objects, each of which has a boolean flag for clustered vs. clustered and a boolean flag for sorted vs. B-tree, along with the database name, table name, and column name.

Maintaining indexes will be done by keeping track of the updates to files that need

to be written to disk; if `shutdown` performance does not matter, we might write all data to disk regardless of whether significant changes to data have occurred. To optimize performance, we keep track of updated values and only write the necessary row changes to disk.

Retrieving values becomes fairly simple; we will have B-tree and sorted column objects and write functions to search through them as necessary. We have a few choices for doing tuple reconstruction:

- Find all qualifying values from the index first, then iterate through the other columns and reconstruct one column of the tuple at a time (late reconstruction).
- Reconstruct tuples one at a time while searching through the index (early reconstruction).

These strategies have pros and cons; depending on the number of qualifying values, the number of columns we need to reconstruct, and more, one will be better than the other (a hybrid approach may be even better). Experimentation will be done to figure out the best approach; in addition, we may calculate statistics about our data in order to make informed decisions about the correct approach to use in each case.

3 Evaluation

Not implemented yet!

4 Challenges & Open Items

Haven't begun implementation yet.