Skyler Kim

25 April 2018

CS4432: Database 2 Project 2

Installation instructions:

1. Unzip contents of “code/” directory.

2. Open Java IDE of choice and import directory as project

-For IntelliJ,

Extensible Hash Index Design:

The extensible hash index was created using tables for each bucket and a higher level table to hold the hashes. The index works by taking the input data value, obtaining the hash code of it, and looking up which bucket it belongs to in the higher level table. This is known as the hash table. The schema for the hash table is the hash it is looking at, the name of the table it points to, the local depth of that bucket, and the number of free spots in that table. Metadata on each bucket was put in the higher level hash table because there was no way to add metadata to TableInfo objects and the tables themselves without adding them directly to pages in memory. There is a separate constant in the ExtensibleHashIndex class that limits how many entries can go into a single bucket. As with normal extensible hash indexes, multiple hashes in the hash table can point to the same bucket. The global depth of the index is stored in the hash table, where the table name is stored as “globalDepth” and the global depth is stored in the “freeSlots” attribute.

Each bucket in the index is treated as a table. These tables are created when they are needed using TableInfo objects. The buckets are searched by opening TableScans on them. Hash buckets are named with their hash plus the name of the index.

Inserting entries is done by looking for the index that the entry would be inserted into and inserting it. This is done by opening a table scan on the hash table to find which table is the correct bucket, then inserting the entry into the correct bucket. If it is detected that there is no free space in the bucket, then the insert raises a flag to split the bucket into two buckets. The first thing that is checked is that whether increasing the local depth by one would make the local depth of the bucket greater than the global depth. If this is true, then the global depth for the whole index needs to be increased. This involves taking all the entries currently in the hash table and duplicating them. Each record in the hash table ends with two variations: one where a 1 bit leads the hash attribute and one where a 0 bit is leading in the hash attribute.

After this check is done, the index scans the hash table and looks for entries that point to the table that will be split. The entries are rewritten to contain the name of a table as if the bucket’s local depth had been increase by one. This is done for every occurrence in the hash table. After this is done, the original bucket is emptied of its records and reinserted into the newly transformed hash index. If all the elements in the bucket still go to the same bucket, then the process is repeated. This is because the call for insert is recursive.

Deleting entries in the index is done by simply finding the bucket that would contain the entry and removing all cases of it from that bucket. Only that bucket needs to be looked at because all duplicates with the same hash should be mapped to the same bucket. Finding the bucket that the hash corresponds to is done by opening a table scan on the hash table. A table scan is then opened on the table that is found and all copies of the search key are deleted. If a table is emptied, the index does not do anything with it or shrink local depths. Shrinking does not happen in case the index needs to have its global depth changed shortly after it has shrunk. It is also impossible to drop a table (completely delete a table) in simpledb.

There are two separate classes relating to the extensible hash index: ExtensileHashBucketRecord and ExtensileHashTableRecord. These are classes that hold the information for records in a bucket and records in the hash table respectively. They are used for storing the records in memory while moving them between buckets and modifying them to accommodate split buckets.

Tests:

Test1: Working Extensible Hash Index  
Ouput: Task2Test1.txt

This is the output of creating an extensible Hash Index and inserting data into it. The index class prints to std out when buckets are being split or when the global depth is being increased. The index also prints how many free slots are in a bucket before trying to split it. This is 0 every time. For this test, the max amount of records allowed in a bucket was 20. The index was tested on by indexing the students table in the CreateStudentDB java file in studentClient/simpledb/ of the project folder.

The main methodology for how the extensible hash index is structured can be found above.

Join Study:

Test 1: 4000 records per table:

Total time for querying test1: 22.0 ms

Total number of I/O’s:

Total time for querying test2: 7.0 ms

Total time for querying test3: 6.0 ms

Total time for querying test4: 6.0 ms

Total time for joining test5 with test1: 1784.0 ms

Total time for joining test5 with test2: 1786.0 ms

Total time for joining test5 with test3: 1779.0 ms

Total time for joining test5 with test4: 1789.0 ms

A larger number of records was tested, but would not complete in reasonable amount of time. Larger record counts for tables also resulted in errors being thrown regarding there not being enough buffers to hold all the necessary data.