CS4432-Project2

Name: Dian Chen /Yang Gao

**Instructions for Simple DB project - Eclipse**

1. Unzip project2`.zip

2. Import into Eclipse

3. Go to src folder and open sampled

4. you will see package server under it

5. Right click "Startup.java" under the server, and click "Run As" -> "Run configuration". Add a variable, which will be the name of the working directory of the SimpleDB server you are about to start

6. Then click "Run" You console will show our print out and "database server ready" at the end. if it runs successfully

**To run our test Program:**

* We create a createtesttable class under StudentClient/simpledb folder called createtesttable.java to test our program.
  + To run it, right click and go "Run as" -> "Run configuration" Add a variable, which will be the name of the working directory of the SimpleDB server you are about to start. And click "Run"
  + Then you will see some create table and select queries.
* The simpleDB contains a class called createstudent under StudentClient/simpledb folder called createstudent.java to test the program too.
  + To run it, right click and go "Run as" -> "Run configuration" Add a variable, which will be the name of the working directory of the SimpleDB server you are about to start. And click "Run"
  + Then you will see some students been created
* Also, you can switch to simpledb to see whether our Extensible Hash Index is working correctly

**Extensible Hash Index Design:**

We create table for each bucket and a linked table to hold the hashes. Basically, how our hashing working is that we will take the input value, and keep what the simple hash code and look up the corresponding bucket and look for the linked table, which is the usual hash code working. The hash table schema contains hash, name, local depth, and empty space. In the Metadata data, we have TableInfo class, we stored those metadata in the linked table of those bucket. And multiple linked table can point to the same bucket as usual hash code. Each bucket itself is also counted as a table, and naming with hash and their type, such as extensible hash or simple hash. In ExtensibleHashIndex class, we have a max entry setting that limit the entries.

For inserting entries, it will look for the index and find out which entry that would be and then insert in to it. Table scan will work for looking at which table is the correct bucket for entry. If error detected, such as no empty space, then there is a flag raised to split the bucket. In addition, the local depth will be checked whether it’s greater than the global depth; I f so, the global depth will be increased. This will involve, the duplicate entries, with either 0 leading hash attribute or 1 leading. After the check, index scanner will scan the hash table to look for entries that point to splitted bucket. The entries are re-grouped by the name of the table. After this is done, the original bucket is reinserted. Situation like repeating process for local depth greater than glabal depth, until it is smaller.

For deleting entries, it will done by simply finding the bucket that would contain the entry and remove all the file in the bucket. Only the certain bucket need to be looked at, because all duplicates with the same hash should be mapped to the same bucket. Finding the bucket will be same process we had for insertion. Then after it found, table scan will open the table and delete all required file. If the table is empty, either nothing will done, or decrease the local depth.

There are two other 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 store the record in the memory while the insertion and deletion is processing.