Ethan Chang, Ryan Racine

CS4432

Project 2

**Installation instructions:**

Before running any of the testing code make sure you run the startup code to start the database, the testing code assumes the database is running so it won’t run correctly if it isn’t.

**Queries used for testing:**

The project2Tests file creates 5 tables:

* Test1 has maxSize (currently 10,000) entries and has no indices
* Test2 has maxSize entries and a static hash index
* Test3 has maxSize entries and an extensible hash index
* Test4 has maxSize entries and a b-tree index
* Test5 has maxSize/2 (5,000) entries and no indices

First we test 4 basic queries:

* select a1, a2 from test1 where a1 = 1
* select a1, a2 from test2 where a1 = 1
* select a1, a2 from test3 where a1 = 1
* select a1, a2 from test4 where a1 = 1

Then we test 4 basic joins:

* select a1, a2 from test5 as test5, test1 as test1 where test5.a1 = test1.a1
* select a1, a2 from test5 as test5, test2 as test2 where test5.a1 = test2.a1
* select a1, a2 from test5 as test5, test3 as test3 where test5.a1 = test3.a1
* select a1, a2 from test5 as test5, test4 as test4 where test5.a1 = test4.a1

**Design Description:**

We added type pretty simply by finding everywhere that the index info was used (create index, schema, record file, etc.) and adding in another field (idxtype). To get the update planner to also update indices we switched the implementation to create and use an IndexUpdatePlanner instead of the original BasicUpdatePlanner.

To create the extensible hash we used an Arraylist of Bucket objects as the directory pointing to buckets. These Buckets hold an Arraylist of BucketVal objects representing the records, containing the dataval and RID. One challenge was splitting a full bucket when trying to insert a record. We implemented this by first checking if the global depth needed to increase and double the number of buckets. This would copy and add the existing Arraylist of Buckets to increase number of buckets while the new buckets “point” to the same values. Checking the global depth first allows us to split the same way regardless of the bucket’s local depth. To actually split, we run through all records, hashing them and sorting into two temp Buckets depending whether the hash matches the original bucket or not. We then put the two new Buckets in place of the existing ones with the same index, both with a local depth one greater than the old Bucket. Then we try to add the record again in one of the two new Buckets. To fully integrate this we switched the implementation to create and use a HeuristicUpdatePlanner instead of the original BasicUpdatePlanner.