**INSTALLATION**

Extract .zip and add as a project in Eclipse. Run the server startup as usual. Performance tests can be run from simpledb.datagenerator. Otherwise, the default studentClient database functions just as well.

**TESTING QUERIES**

The test queries from the previous project (pokemonTest.simpledb.\*, studentClient.simpledb.\*) as well as the performance testing queries were used in order to test whether the database still functioned properly.

**DESIGN**

Index

The Index data structure was updated throughout simpleDB in order to accommodate the index type based on user input; “bt” and “eh” will create B-tree and extensible hash indices respectively, and any other input (or lack thereof) will make a static hash index.

Extensible Hashing

The Extensible Hash index (simpledb.index.exhash) expands on the functionality of the original hash index class. Initially, 4 buckets are created and the global depth is set to 2 (2-bit checks). Buckets are now a new class rather than an abstract integer and hold an ID number, local depth, contents, and capacity. The contents can not be larger than 4. Buckets are held in a hashmap to match ID numbers to buckets during hashing.

A bitmask is used to determine the bucket to send a record to (2 => b11, 3 => b111, etc). For each record, the bitmask is initalized based on the global depth for safety and a bucket number is generated with the search key's hash and the bitmask. If the local depth of this bucket is not equal to the global depth, the bitmask and bucket number are updated before continuing. This is how our design handles multiple local depths; only the necessary number of bits are checked.

If a bucket is not full, the record is added to its contents and the index is updated. Otherwise, the local depth (and the global depth if need be) are incremented. The hashed, unmasked value of the record, as well as this bucket, are then passed to a bucket updater for splitting and proper hashing.

The bucket updater first applies the bitmask to the hashed value, then uses this to make a new bucket. The contents of the old bucket are removed and rehashed to update these two buckets, and finally, the record that caused this mess is put into its proper bucket and the index is updated.

SmartMergeJoin

Several changes within simpleDB’s classes needed to be changed to support SmartMergeJoin. First, TableInfo needed to keep track of whether a table has been sorted or not, and a Boolean flag was added to record this along with appropriate getters and setters. Classes that implemented the UpdateScan interface had inherited a new method that would set the isSorted field under TableInfo to false as an update may have occurred. This also occurs in plans where a delete, insert, or update may take place on a table. As a scan is passed into the SortPlan class, the scan interface also received a new method that would retrieve the scan’s RecordFile, as we chose to replace the original table’s RecordFile with the temporary table’s RecordFile. By using this method, the original table’s RecordFile is passed into SortScan, through a new constructor method, and the temporary table’s RecordFile is copied into the original table’s RecordFile within the constructor method. Once this has been done, the isSorted flag is set to true.

One final change needed to be made in the MergeJoinPlan class to make use of these new pieces of data. When creating a new MergeJoinPlan, the constructor method first checks the table’s TableInfo to see if the data is sorted for each table involved in the MergeJoin. If it is already sorted, a SortScan is not called on the table and the original Scan passed into the constructor is used. If it isn’t sorted, a new SortPlan is created for that table (the default action).

ExploitSortQueryPlanner

**TESTING**

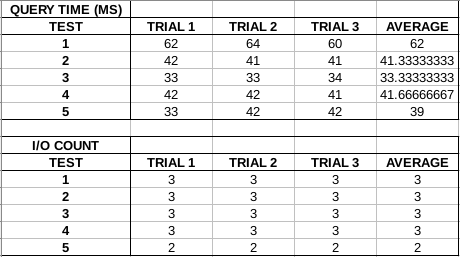
Performance tests were run with 1,000 queries per table (anything higher did not complete after several hours, despite running on a very powerful machine).

Extensible Hashing

The following query was run on the 5 tables, where *i* is the table number.

Select a1, a2 from test*i* where a1=275

This query gave 2 results for tables 1-4 and 1 result from table 5. The query time and I/O counts are as follows:

Interestingly enough, static hashing seems to have slowed the queries (1). B-Tree indexing resulted in the fastest query times (3), and extensible hashing had a very minor improvement over no index at all (2 compared to 4). The I/O counts appear to be invalid; we were unsure as to how to test these and are fairly sure we did so incorrectly.

Joins were attempted but could not be completed in time, as simpleDB does not support the “table.attribute” query format and there was not enough time to rebuild the database with alternate attribute names for table 5.

SmartMergeJoin

**BUGS**

We are unaware of any bugs in this application. However, we are not sure if our IO counter for select statements works as intended.