Mike Calder, Nathan Longnecker
CS542 - Database Management Systems

Project 4 - Undo/Redo Logging

We implemented undo / redo logging in our database. It is set up so each time you create or modify a transaction, that change is written to the log before the change is written to the database. We extended our project three and the logging component from project one to create this one. Because we reused code from project 3, our original data is read directly in csv format and the updates are done on the data in csv format. The relation is piped one tuple at a time through the update function, and the output is written to a file as updates are applied. When the data is recovered, the database puts the original data into our custom Relation class, and partitions it into separate blocks. Then when each update is applied, the relation reads the appropriate block from the disc, updates it, and then writes it back. The updating portion of our code could be extended to use this relation class in the future to improve the consistency and extensibility of the program.

Using our our undo/redo logging database

The code in our repository contains only the classes necessary to do the update requested by the assignment. However, the program is still compatible with the other classes used in our previous assignment that allowed users to do selections, projections, and joins, so the program could support a queryable database and allow updates with undo/redo logging as well. It would be a relatively simple matter to add classes to allow dropping tables or creating tables and inserting rows.

To use the program, the user should first convert their data to csv format. Then they should create a main method that creates readers to read in the data from their csv data files, and pass those readers to the in parameter of their update operation. The update will create a transaction object and add each change that was made. When those changes are made, the changes are also written to the log. The log file can then be copied over to a second database and the database can use the syncWithLog() function to execute the transactions in the log.

Our SampleMain provides an example of this behavior. The sample main updates the population of each country and city by 2%, and then copies the log to a second database and restores that database from the log.

Assumptions made

We assumed that the user had their data in CSV format. The database could be extended to support other data formats relatively easily by adding an interpreter to the initial reader (and/or

final writer) to convert it to or from CSV, which is the format used when passing data between each step through the PipedReaders and PipedWriters.

We assumed that the relations were not indexed for simplicity. A more complex database could be written that makes use of indices if they exist.

We assumed that the entire relation would fit into memory. This assumption was purely for simplicity. When creating a new relation, if it has not been partitioned into blocks yet, the program will read the entire relation into memory, partition it into blocks, and write the blocks back to memory. From then on, the relation class never needs to have more than one block in memory at a time. To allow our relation class to support larger relations, we could modify the initialization of a relation to pipe data from the source file and write it to blocks as it gets it, so it never needs to have the whole relation in memory at once.

How it works

Recovery
The program makes use of logs to allow the database to recover or bring databases into sync by reading the logs. Each transaction and each update in each transaction is written to the log so it can be read and applied to a database. The database will redo transactions that were already committed and undo transactions that have not yet been committed to bring the database into a consistent state.

Testing
We have created a few JUnit 4 tests to test the logger classes. Running the sample main on the world database data set served as our stress test for this application. These tests are visible in the test folder of our github repository.