SUTD 2021 50.043 SimpleDB Project Part 2 Writeup Report Document

James Raphael Tiovalen / 1004555

Implementation Description

For part 2 of the project, I also simply implemented most of the skeleton methods/functions and classes specified by the handout. I also followed the implementation guide as laid out by the exercises closely, and hence, there are very few deviations from the intended path.

Several design decisions that I have made include:

- Implementing the naive nested loop join algorithm for the join operation as specified by the lab handout. Future work can be done to improve the speed of joins by exploring other algorithms, such as sort-merge join, block nested-loop join, double block nested-loop join, etc.
- Implementing the LRU algorithm as the eviction policy. This is because it is the de-facto standard cache replacement policy used for real workloads of professional production systems worldwide for many years. While it is definitely possible (and easy!) to find certain specific memory access patterns or construct pathological reference use cases whereby alternative behavior such as FIFO, CLOCK, CAR, CLOCK-Pro, ARC, LFU, MFU, MRU, LIRS, random, etc. will be superior, in real life, it depends on actual usage. As such, on average, since LRU has historically given very good performance in overall use, we can simply follow the default and widely used policy chosen by many over the years.

Several challenges that I have faced include:

- Fixing bugs in `src/java/simpledb/storage/HeapFileIterator.java`. There were some subtle bugs that were not caught by part 1's tests, and as such, cropped up when running the tests for part 2.
- The unit tests are sometimes not exhaustive enough to cover all cases that might be detected by the system cases. As described earlier, most of the bugs encountered were caused by a faulty implementation of the `HeapFileIterator`. As such, most of the time was spent debugging the `HeapFileIterator`, even if the unit/system tests in part 2 were meant for other sections/parts of the project.

I did not change the provided API at all. All the custom, additional classes are designed around the current API to support it, instead of subverting, modifying, or redirecting it.

There are no missing or incomplete elements of my code, assuming that all that is being considered is for part 2. Any other extra additional methods will be implemented in future parts, since this provides ease of debugging and separation between different functionalities as the project timeline progresses.