Lab1

Zheyuan Xing

zyxing@uw.edu

1.

Lab1 is about implementing the basic structure of simple-db. During lab1, I went through almost most of the essential class in simple-db:

Catalog: **Provide a global map to look for disk information** especially we can directly look up for DbFile location(which in my design, is wrapped up in a `tableInfo class`). That's the most essential role for Catalog. Tables store as DbFile in disk. If we're trying to retrieve it from Heapfile interface, we need to know the information about the location of the file. The only way we can achieve that is by using Catalog.

BufferPool: **Serves as a cache layer to temporally store pages** that we might retrieve again after. In this lab we didn't implement too much functionality of bufferPool. I use a HashMap to cache all the pages we recently retrieved. One thing important thing is that when we implement HeapFile iterator, we make sure that we implemented by using bufferPool.getPage(). This is because we want bufferPool to participate in every query execution no matter the required tuple is cached or not. This will significantly increase the speed by simply add interface call.

Tuple & TupleDescription: The basic unit of data in simple-db. A tuple == A row in table. It's the unit we need to return as the result of executing a query. TupleDesciption is more like a helper class to store information about the field of tuples.

HeapPage & Page: **A basic chunk of data that flow between HeapFile interface and BufferPool. The basic block of cached data that stored in BufferPool.** Tuples are stored in a particular way on pages. It includes pid, td, headers, numSlots ... and other metadata. I think HeapPage provided a way to help us better organize data and create a standard data format that flow between low level simple-db component. The difficult part about implementing HeapPage is implementing iterator, especially dealing with those edge cases. One thing I found is really important is that we need to locate the last-valid-slot before we start iterating. Because we need to know where we stop — where the hasNext() return false. Otherwise I think it's kind of tricky to implement iterator.

DbFile & HeapFile: **The middle man between Disk and high level query execution**. Closely connected with BufferPool. Every page retrieved via HeapFile will flow through and cached in BufferPool. Communicate with SeqScan via Tuples, communicate with Disk and BufferPool via Pages. Responsible for organizing byte data from disk into pages and send to BufferPool. Also responsible for retrieving required tuples from pages(bufferPool or disk) and send to sequencial scan. The difficult part to use java built in file access function to organize byte data into page format. Also, iterator implementation is still tricky(most difficult in lab1). It's because of the nature of HeapFile. It send out Pages to bufferPool but return Tuples to SeqScan via next(). So HeapFile will not simpily iterate Pages or tuples. It iterate pages first then iterate tuples. So we need to make sure that once finishing iterating tuples in a page we'll jump to the next page and start from tuple 0.

SeqScan & Operator: SeqScan is basically a high level custom iterator to retrieve tuples. It communicate with HeapFile interface, directly call the iterator of HeapFile. One thing to mention is that, because SeqScan is extremely depend on HeapFile.iterator(). So whenever we call operations like open() and close() we'll need to make sure to call the same operation on HeapFile.iterator(). Other than that, Heapfile.iterator will do most of the job of SeqScan

2.

Use helper class to gather similiar fields together. I create nested class tableInfo in Catalog so we can better wrap up information in a single HashMap

Use BitSet to programme Byte/Bit related operation

Use RandomAccess to retrieve data from disk.

3.

The test I would recommend to add is a multi-page HeapFileRead Test. I found that we didn't have test for multi-page heapfile read test. The current is to test whether we can pass a "1 page & 3-tuple" read task. We don't whether our heapFile class is good enough to read multiple pages. We have multi-pages reading test in SystemTest. But I think it would be better if we can test it during the implementing of Heapfile.class

4.

I didn't change anything for the given APIs. I did add some System.out.print() statement in Test files. I didn't remove them since they're helpful for my debugging. If that's something not allow to do pls remind me in gradescope comment.

5.

By the time I write this I still can not pass the last systemTest **testCache()** I'm checking my bufferPool cache functionality right now and hopefully I can fix it by the start of lab2