CSE 444 Final Report

Yuanfeng Li

3/18/2021

An overview Description of the architecture:

When I am running a really simple query in my SimpleDB, it will go through different components of its architectures. There are basically 3 major components in my SimpleDB, the 1st component would be the query Processor which parser my query and then give the results to next component Access manager. Access component will access the data and info on where the data is been saved, in SimpleDB it will scan the disk in my computer, and then give the output to another component that would save the necessary data and info into the main memory, so that it can get the fastest processing speed. Meanwhile it will use the concurrency control component to do the manipulation on different files at the same time and write them in log files by using the log component. So the database can be recovered when it is crashed during any emergency or difficult situations.

In summary there are 3 major components in my SimpleDB:

1) Access component – which is responsible for getting the data in disk.
2) Query processing component – which do the parser to analyze the query and tell other components what to do.
3) Memory component –which job is to save the data from the Access component to the main memory and coordinate with other minor component such as log manger, and lock manger, OpIterator to make sure every manipulation is correct and doing their jobs.

It might seem too general when I don't give you an example, now let me show you about the overall architecture by using a simple example:

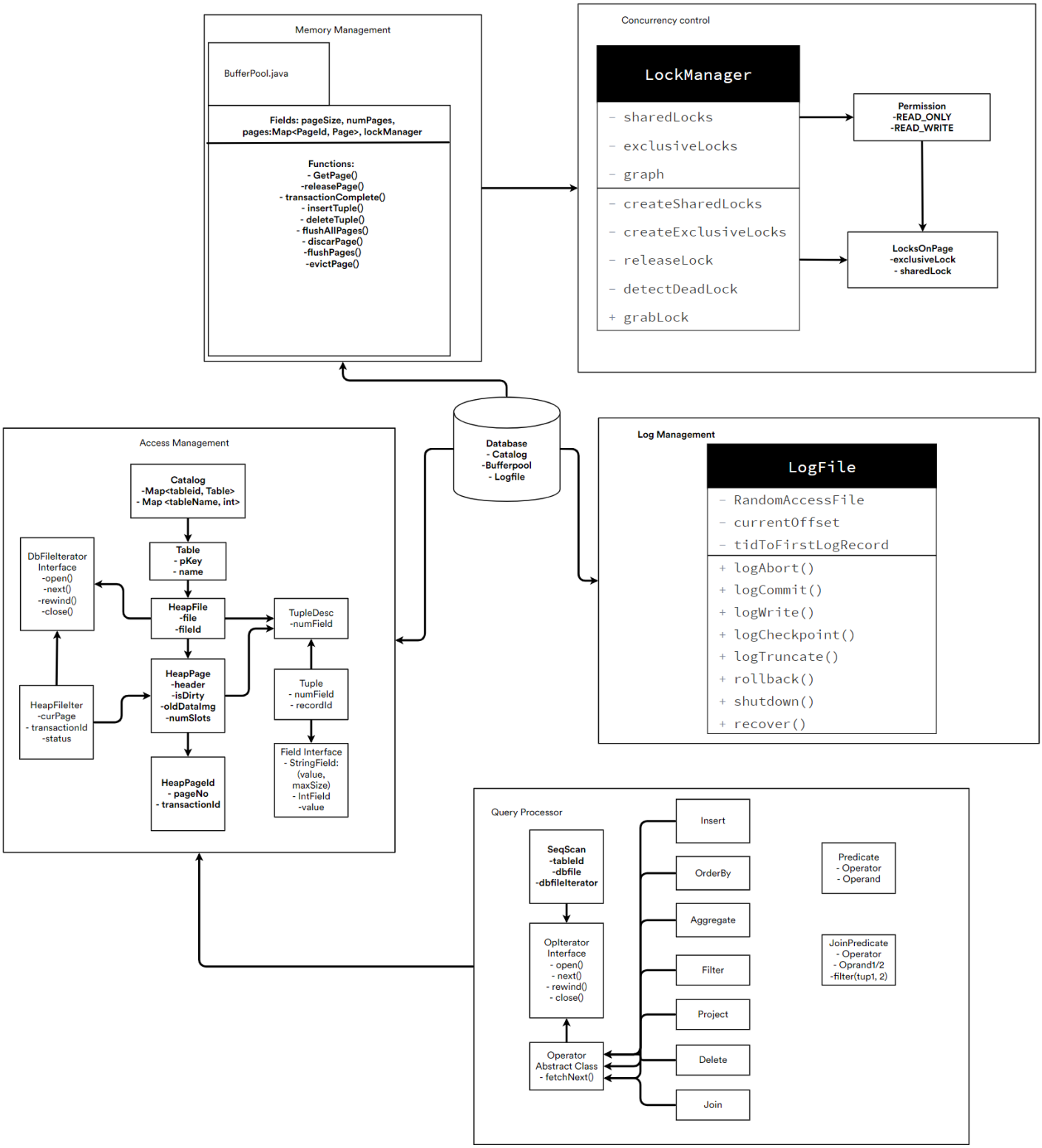
- If I would like to delete a tuple using the following SQL:
  DELETE FROM table1 WHERE id <= 100

  As I described earlier, the first component-Query processing component would parser this query and tell the access component to do the scan over the files on disk to find out the correct tables, and files, and then the memory component will save all the related files into the main memory and ask other components such as lock manger, log manager and iterator components to do their jobs.

A diagram of overall system architecture:

You can see how my architecture really work in the following diagram in the next page. Because it is too big to see clearly, so I put it to next page for better visualization and readability.

The lines and connection in the diagram is not fully correct, and there are more lines that I haven't draw. However, I will discuss about them in the next section.


Memory Management
BufferPool.java
Fields: pageSize, numPages,
pages:Map<PageId, Page>, lockManager
Functions:
- GetPage()
-releasePage()
- transactionComplete()
- insertTuple()
- deleteTuple()
- flushAllPages()
- discarPage()
-flushPages()
-evictPage()
Concurrency control
LockManager
- sharedLocks
- exclusiveLocks
- graph
- createSharedLocks
- createExclusiveLocks
- releaseLock
- detectDeadLock
+ grabLock
Permission
-READ_ONLY
-READ_WRITE
LocksOnPage
-exclusiveLock
- sharedLock
Database
- Catalog
-Bufferpool
- Logfile
Access Management
Catalog
-Map<tableid, Table>
- Map <tableName, int>
Table
- pKey
- name
DbFileIterator
Interface
-open()
-next()
-rewind()
-close()
HeapFile
-file
-fileId
TupleDesc
-numField
HeapPage
-header
-isDirty
-oldDataImg
-numSlots
HeapFileIter
-curPage
- transactionId
-status
Tuple
- numField
- recordId
Field Interface
- StringField:
(value,
maxSize)
- IntField
-value
HeapPageId
- pageNo
- transactionId
Log Management
LogFile
- RandomAccessFile
- currentOffset
- tidToFirstLogRecord
+ logAbort()
+ logCommit()
+ logWrite()
+ logCheckpoint()
+ logTruncate()
+ rollback()
+ shutdown()
+ recover()
Query Processor
SeqScan
-tableId
-dbfile
-dbfileIterator
OpIterator
Interface
- open()
- next()
- rewind()
- close()
Operator
Abstract Class
- fetchNext()
Insert
OrderBy
Aggregate
Filter
Project
Delete
Join
Predicate
- Operator
- Operand
JoinPredicate
- Operator
- Oprand1/2
-filter(tup1, 2)

A more detailed description:

From the architect diagram in page 2, you can now get a clearer idea about the three major components, and other minor components that works together with these three major components.

Major:

1) the catalog class
2) the bufferpool class
3) the log File class

Minor:

1) LockManger class
2) LogFile class
3) DBFile Interface
4) others

For the catalog class and all the java files that makes the access process works such as TupleDesc.java that defines the tupledesc for each table, and tuple.java that defines the component of a tuple. The catalog file is main method and class that lets the SimpleDb reads the files from its disk.

For the bufferpool class, and all the java files that receive the input data from the catalog class and save them into the main memory such as lockManager class and others. They mainly ensure that the query Processor and the catalog class's result can be saved in the main memory for next steps. In my SimpleDb's implementation, the Bufferpool class and LockManager class are all working on the page granularity level, which means that they will save the pages in the main memory and do the manipulations such as insertTuple, deleteTuple on page levels. Before any Transaction may do any activities on any pages, they need to grab a lock on the pages. There are two kinds of locks, one is a shared lock which can be shared by many transactions on the same page for those transactions that is read_only. And an exclusive lock for the transactions that would like to not only read but also write on the pages, and my implementation only allows one exclusive lock on a page with one transaction. Bufferpool works very closely with other classes at all times to read and write the necessary files on disk.

Finally we have our LogFile.java which is mainly to assure better recovery for different transactions. It records the necessary steps such as 'Commit', 'Abort', 'rollback' and others into a file called log, and write it to the disk, so when we need to recover from a crash or emergency situation, we know what to do with the data and tables.

Discussion of the overall performance:

I don't think my overall performance is great, because there are a lot of upgrades I can do, and I will mainly talk about the Join class and LockMnager class in this Section. By analyzing these two classes, I can give you an idea about how my overall performance is, and what kind of upgrade I can do.

1. I think I had a good Join class in my SimpleDb, it is because I have successfully run out the result from the queries in lab3, which is not an easy job. I have developed a hash join method for 'Equal' joins. And its speed is 500 times faster than the ordinary 'nested loop' Join method.

However, my hash join method is only good for the ‘Equal’ joins, which means all other joins such as ‘greater equal’ join, and ‘less equal’ joins are all using the nested loop join method, which is not that fast.

I wish I can design and implement a ‘sort merge’ join method for all kinds of joins. Because I believe ‘sort merge’ join can easily handle more predicates, and maybe it will have faster performance than hash join as well.

2. My deadlock detection is using a timeout method to detect the deadlocks in my lockmanger class. Which is not that efficient compared to the graph method. Because it takes more times to detect all the deadlocks, and it makes the whole database running slow, so I believe using a graph detection can help it to speed up, I hope I can develop this method if I had more time.

In addition to the two methods above that I would modify and updated if I had more time, I would also like to implement the optional lab5 if I had more time. In Summary, I think my SimpleDB’s overall performance is ok, and it definitely has a lot more space to grow!