CSE 444: Lab 1

Aakash Srazali (aaksra) | Cherry Winarto (cwinarto)

In your own words, describe what Lab1 was about: Describe the various components that you implemented and how they work. This part needs to demonstrate your understanding of the lab! If your description looks like a copy paste of the instructions or a copy-paste of the provided documentation you will lose points.

In Lab1, we built up SimpleDB's architecture by enabling the SeqScan operator to retrieve tuples in a chained-fashion. For instance, the retrieval of tuples occurs with the following chained events: SeqScan invokes a method call in HeapPage, which invokes another method call in HeapFile which invokes another method call in BufferPool. Depending whether or not the page containing the requested tuple is in BufferPool, it invokes another method call in HeapFile to read the page containing the tuple from disk. Eventually, the retrieved tuple is propagated back from the chained method calls to the SeqScan operator.

The TupleDesc and Tuple classes tell Java how to define a "tuple" by explicitly specifying what the names and data types are for each data stored within a tuple. The Catalog keeps track of all tables in the database, hence, for each new table that we want to add, it should be added to the Catalog upon creation. Each table is represented by a HeapFile, which consists of a collection of HeapPages to form a table that stores tuples in the same order as on disk.

A HeapPage has a unique HeapPageID, header, and tuples, with the header indicating whether slots for tuples are filled (i.e., it keeps track of dirty bits for tuples/slots). A Tuple contains a RecordId, TupleDesc (schema), and an array of Field objects representing data values. The RecordId class is used for uniquely identifying/addressing tuples.

The BufferPool enables caching mechanism by storing HeapPages for quicker access. When HeapFile requests for a page that is not in BufferPool, the HeapFile is responsible for retrieving those pages from disk and then giving it to the BufferPool. Because of limited space in BufferPool, we will add the mechanism for removing pages via some eviction page policy to be able to store other HeapPages.

The SeqScan operator scans through specific tables in the database and retrieves tuples using its HeapFile iterator for use by higher-level operators in the query plan.

Describe any design decisions you made. These may be minimal for Lab 1.

We implemented Catalog.java with 3 HashMaps to make all the necessary relations. The first HashMap maps the table name to table id. We would use this map in the getTableId(String name) function which will provide us with a table name and returns the table id. The second HashMap is used to map a table id to the table class. We can use this map structure for the getTupleDesc(int tableid), getDatabaseFile(int tableid), getTable(int tableid) which will provide us with a table id and return some sort of information regarding the table. The third HashMap is to map the lowercase name to its original name – when dealing with unique names, it is easier to work with all-lowercase in the case of duplicates. We use this HashMap for the getTableName(int tableid) which will require the original name of the table instead of the lowercase table Name which we store in our Table class.

For HeapPage, we made a design choice involving the iterator where the iterator should return non-empty tuples when the next() function is called, therefore it should skip over any empty tuple slots within. An integer variable is used to keep track of the index of the next non-empty slot. When the iterator is initialized in the constructor, this variable is set to the index of the first non-empty slot, ready for the next next() call. If there are no more tuples to return within the page, this integer value would serve as an indication that it is out of bounds.

Also for HeapFile, the design decision involved the iterator – due to the HeapFile being made up of multiple HeapPages, the HeapFile Iterator “chains” all its HeapPages’ iterator. For example, when the HeapFile Iterator is initialized on open(), it gets the iterator of the first Heap Page. The HeapFile will then continue to use this Heap Page until it reaches the end of the tuples, then the HeapFile iterator will take the next HeapPage iterator to continue iterating through tuples until there are no more HeapPages containing tuples.

For RecordID and HeapPageID, we added fields that we already included in the constructor. We also referenced them in the get methods. For the hashcode() function in HeapPageID, we multiplied the page number and table id with prime numbers (which is a good rule of thumb for dealing with hashcodes) and concatenated them as Strings to be passed into the Java String hashcode() function which should lead to fewer collisions.

Give one example of a unit test that could be added to improve the set of unit tests that we provided for this lab. You do not need to write the code of the unit test, only describe it in the write-up. If you wrote one or more especially useful new unit-tests, you can submit a pull request and we will consider integrating your tests into the lab for next year. Pull requests won't get extra credit, only eternal glory if the tests get integrated.

Something we realized while debugging was that in the HeapFileReadTest, the numPages() test, it isn’t testing for integer division. We calculate the number of Pages using the mathematical function ceil(FileSize / PageSize). Thus, the difference of double division and integer division would make a difference when calculating different values.

Discuss and justify any changes you made to the API. You really should not change the API. If you plan to make a change, ask the TAs first.

We did not make any changes in the API

Describe any missing or incomplete elements of your code.

We do not have any missing or incomplete elements of our code.

If you have any feedback for us on the assignment, you can add it to the writeup, send us an email, or make a note and paste your note in the course evaluation form at the end of the quarter.

N/A