Lab 1

Phuoc Huynh - phuynh08

Ivan Zachary Belikov - Ivanb13

For Lab 1, we use the Database class to access various parts of the database. Throughout the lab, we utilize the database class to access static objects within the database, such as the BufferPool and Catalog, to get files, table names, tuple descriptions, pages, etc. In particular, we can use a table Id to retrieve from the catalog the tuple description of the tuples stored within the table referenced by that id. The tuple description describes the schema of the tuple, such as the number of fields, their names, and their types. The tuples themselves consist of field objects that describe the data types stored by that tuple, following the specified schema given by the corresponding tuple description.

As stated prior, we can retrieve the tuple description for a table using its id from the catalog. In the catalog, we store a reference to all of the tables and table schemas stored within the database. Additionally, the catalog has a description associated with each table that notifies the operators within the query plan of the types and number of fields in the table. In our implementation, we store files, containing the tables, and ids linked to those tables separately in hashmaps. We store the ids of the tables as values linked to the name of the table and store the file of the table as values linked to the id of the table. Along with the Catalog, in Lab 1, we implemented the BufferPool, which caches recently read/written pages from disk. Within the BufferPool, there can only be stored a fixed number of pages, which we indicated in our implementation using numPages field, before needing to conduct an eviction policy. In Lab 1, we did not implement an eviction policy, rather a DbException is thrown if we can't cache any more pages. Within the query plan, the operators read and write pages from files on disk through the

BufferPool. In terms of Lab 1, we implemented BufferPool.getPages() in order to work with the SeqScan operator and allow it to read tuples from the pages passed to it from the BufferPool, subsequently caching these pages.

Additionally to implementing tuples and heap pages, we also implemented identifiers for these tuples and heap pages called record Id's and heap page Id's so that during the execution of any query the operators know the specific page or tuple we are looking for or operating on. Associated with a tuple or record id is a page id, which references the page the tuple is stored in, and a tuple number, which refers to the numerical placement of the tuple on the page. Similarly, associated with a heap page id is a table id that is the table being referenced by the page and a page number that acts as an identifier for that heap page. This data can all be retrieved using getter methods implemented in both record id and heap page id.

The sequential scan operator works as an access method that reads each tuple of a table in SimpleDb based on DbFileIterator in HeapFile. The program will interact with SimpleDB through SeqScan as the top of query plan and SeqScan will pass the calls on its children. In lab1, we will need tableId and tableAlias of HeapFile to create a Sequential Scan to read each tuple from that file. All of the functions in SeqScan are based on the DbFileIterator of the HeapFile with the tableId we pass in.

From the DbFileIterator, it will get a Page from BufferPool with a Transactionid, PageId and Permissions. If the PageId is not present in the BufferPool, it will use the readPage() function in the HeapFile to read a Page with PageId from the file on disk and return a Page to Buffer Pool.

Once a Page that DbFileIterator is looking for is in Buffer Pool, we will use the iterator function in HeapPage class to iterate through each tuple in that Page and return the tuple back to SeqScan.

Add Unit Test:

In HeapFileReadTest there is no j-unit test to check to see if the rewind function for the iterator works.

It will be similar to testIteratorBasic() function HeapFileReadTest.java. After line 100, we call it.rewind() and continue to increment the count while it.hasNext() is true. Finally, we call assertEquals(6, count). For this test, we count the number of tuples in the file twice, we count it once until there is no tuple in the file and we reset the iterator with rewind() function and count it again.