DbSys LAB 1 – Writeup

GROUP 17

AGARWALLA Yash

SENANE Zineb

HAKAM Mouad

12 December 2021

Lab 1(SimpleDB) was a challenging assignment for our team as none of us are fluent in Java programming. Although, the reference materials sent by the professor were helpful, especially "Java iterator", but we faced some difficulties while doing the assignment. In the first session of the lab, we looked at the description of the assignment, understood the tasks, and delegated the tasks amongst us.
We started by trying different environments to perform our tests and settled on using "Eclipse" as it was the easiest one to use. We revised all the basic Java syntaxes such as the variable declaration, making loops, etc. Then we learned some concepts which were fundamental for our lab such as ArrayLists, iterators, hashmaps, etc.

We decided to do the first exercise together, starting with "TupleDesc.java". It took us about 3 days to complete (2 Hours/day). We used an ArrayList instead of a List because we found that ArrayList has many useful predefined methods and uses better indexation as compared to a list. We found some descriptions to be a bit ambiguous. For example, in the method "getSize()", the description says:
"/*** @return the size (in bytes) of tuples corresponding to this TupleDesc.
* Note that tuples from a given TupleDesc are of a fixed size */".
Here, it is unclear if we should return the size of a single tuple or the size of the tupleDesc(list of tuples) as all the tuples of a tupleDesc has a fixed size. So, we decided on taking the size as the sum of the sizes of each tuple
The second file (approx. time: 1.5 hrs), "Tuple.java" was straightforward as it contains only constructors, getters and setters, and two supplementary methods. It just uses the methods implemented on TupleDesc.java. It was the easiest of all the files in this lab. It just took us an hour and a half to finish. Both of our unit tests, i.e, "TupleTest" and "TupleDescTest", passed without any errors.

We started the second exercise (approx. time: 3 hrs) and were greeted with a file named "catalog.java". We declared a class for the tables with constructor and getters and setters. In the "addTable" method, we first checked if the table was already present, otherwise, we create the new table according to the passed parameters.
In the "getTableId" method, firstly we checked if the parameter is valid or not, i.e, if the name is input or not, and throw an exception if not. Then we compared the names of all the tables with the parameter and thus return the ID if matched or throw an exception if not. Similarly, we implement the "getTableName", "getPrimaryKey" and "getDatabaseFile" functions. Hence, we were able to create a new table or get any information associated with the TupleDesc object and pass the unit tests in CatalogTest.

In the third exercise (approx. time: 4 hrs), we begin by declaring the variables in the "BufferPool.java" file. We initialize the declared variable with the parameter and map the pageIds to the corresponding pages using concurrent hash maps in the constructor. In the "getPage" method, we faced many problems because we forgot to consider the case where space is unavailable in the pool. But after some trial and error, we figured it out and worked on it. So, we have begun by checking if the page is already present in the pool, if not, we check if the space is available in the pool. If it is already present, we evict the page, otherwise we create and put the new page into our bufferpool. This exercise was simple, but we wasted

a lot of time figuring out the cases and since the only way to check the correctness was to complete and run the Heapfile tests, we had to wait until the next exercise got finished.

In the fourth exercise, we began with "recordId.java". It was simple to implement as it only involved the usage of constructors, getters and setters and two easy methods, i.e., "equals" and "hashcode". The same goes for "HeapPageId.java" with basic function definitions. However, we faced a lot of difficulties in the "HeapPage.java", especially in the "isSlotUsed" function as we initially tried to just use logical comparison operators and had to do trial and error on pushing the bits since we were getting errors in the "HeapPageReadTest" unit test. The "iterator" function was also a bit time-taking. In the end, we got no errors on the unit tests "HeapPageIdTest", "RecordIDTest", and "HeapPageReadTest". (approx. time: 5 hrs, recordId: 1 hr, HeapPageId: 1hr, HeapPage: 3 hrs)

We then started the fifth exercise (approx. time: 7 hrs) , the "heapfile.java" required an auxiliary class that implements all the methods of DbFileIterator, because when we tried to implement it inside the "heapPage" class we found an issue with the cast from "tupleIterator" to "DbfileIterator", thus failing the "HeapFileReadTest" unit test. We created another file called "HeapFileIterator.java" as asked by the instructor to solve the issue. This file has a private attribute which is a "TupleIterator" having a constructor having parameters heap file and transaction ID and creating a heap File Iterator which is also of type DbFileIterator (as HeapFileIterator implements the DbFileIterator interface). And also, the language was vague in the description, for example, the description of the method “open()” just says “opens the method iterator” which we had to self-analyse. Finally, we passed the tests "HeapFileReadTest" with no errors.

Finally, we reached the last exercise (approx. time: 5 hrs). Here we had to modify the file "SeqScan.java". We encountered difficulty in the "reset()" method because initially, we were not typecasting the database file to heap file. For the "getTupleDesc()" method for formatting "alias.fieldName", we may have increased the time complexity of the whole program by introducing a while loop. We tried different techniques but only the loop was working. Reset of the functions was easy to implement. We copied all the modified files in one PC and ran the System test, we got an error.
We tried to figure out the origin of this error and found out that it was due to the HeapPage.java file. Since we just combined the files without checking their compatibility and each member was responsible for their individual files, our files were not integrated as we all had different ways of implementing methods. We begin with inspecting the "HeapPage.java" file and skimmed through all the methods, analyzing and discussing them. Finally, we discovered that there were some missing constraints such as checking if a table is null and rising an exception, casting to transform the result for an operation to an int, or checking if we have a valid index or not. These small bugs were the reason for the SystemTest error. Once the checking phase is finished, we succeeded to run the simple test correctly. This bug fixing took us around 4 hours.