Shanghai Jiao Tong University
Lab1 Report
Name: Yunxuan Xiao
Sid: 516030901268

Database 2019
April 21st

1 Design Decision

1.1 Field and Tuples

Nothing interesting. Define a subclass *TDItem* to store field names and field types, and use ArrayList to store and index TDItems in *TupleDesc.java*.

1.2 Catalog

The Catalog keeps track of all available tables in the database and their associated schemas. We can search for everything through a predefined *Catalog* object.
Indexing: We can index a table given their *tableid* or *tablename*. So I defined 2 *ConcurrentHashMap* to keep these projections. One for *tableid* to table, another for *tablename* to *tableid*.

1.3 BufferPool

Bufferpool is the most interesting part. As in practice, we cannot load all tuples in a table into main memory, so we have to define a buffer to store constant number of pages for fast indexing.

- **Store**: Define a *ConcurrentHashMap* for pageid to page cache projection, a *ArrayBlockingQueue* for storing an ordered pid sequence in this buffer.
- **Read page**: If the page already in buffer, we simply return it. If it is not, we have to load the page from *heapfile* object.
- **Replacement**: If the bufferpool is full, we need to evict one page and put the new pages in. Here I adopt a LRU(Least Recently Used) eviction policy, which pops the page whose pid is in the tail of the queue.

1.4 HeapFile & HeapPage

- Create a new class *HeapFileIterator* implementing baseclass *DBFileIterator* for traversing the whole table. Actually it returns the tuple iterator in each page.
- Define some bitwise operation to checl whether the i-th slot is empty.

1.5 SeqScan

Nothing interesting, return a *HeapFileIterator* to traverse the whole table.

2 Missing and Incomplete Part

There are several incomplete parts that I need to implement.

1. **HeapPage iterator**: Currently I copyed the tuple objects into a new ArrayList and return the according ArrayList.iterator as HeapPage iterator. However it is not thread-safe since we can simultaneously update the page and traverse the page using iterator, but the iterator runs on a fixed, old version page.

2. **Additional Large Table Test**: The sample query on the course website simply traverses a 3x3 table using *SeqScan*, which is very small and easy to pass. So I create a 10x100 table and a 50x5000 table to test the robustness of my code. In 10x100 version my code works well, while in 50x5000 version it fails. I find that my *HeapPageIterator* cannot find the next page position when the pagenumber exceeds the predefined DEFAULT_PAGE size, i.e. after the page replacement happened. In the next lab I will implement a more stable iterator for large table.

3.

3 Time, Difficulty and Confusion

I spend nearly 3 days in this lab, approximately 10+ hours. The confusion comes from these part:

1. Transaction id. I totally have no idea about what transaction is. There are many tid items through the code so I just copy and store them without any modification.

2. Order of implementation: When I was writing Catalog part, there are some operations involving HeapFile which should be handled later. They mixed together and I try to figure it out within 1 hour.

3. **BUG IN PARSER** In the query example on the website, it write a 3 row table in a .txt file and transform it into .dat file. I run my SeqScan on it but only output the first 2 rows! I takes really a long time and finally figured that a $\backslash n$ character must be typed at the end of file, otherwise it would not be correctly parsed.

4. **BUG IN BUILD.XML** The 46-th row of build.xml should be changed to "srcdir=@srcdir"destdir="@destdir"otherwise it would fail. It dosen't take much time since someone has reported this bug on StackOverflow.