Database Management Systems Implementation

Report Lab1: SimpleDB

Massimiliano Pronesti
Federico Tiblias
Giulio Corallo

Instructor: **Paolo Papotti**

December 14, 2021

1 Design and Implementations

In this section, we're going to briefly describe our main design and implementation choices. Please notice that, since this first lab required implementing the core modules to access stored data on disk, the design choices we were required to make are very minimal, with the slight exception of the *HeapFileIterator* class, as described in **section 1.9**.
No change was made to the provided APIs of SimpleDB.

1.1 Tuple

To properly link the tuple with its descriptor, this class exposes a *TupleDesc* reference and a classical array containing the *Fields*. We opted for a classical array as the size is fixed and known apriori, so that we didn't resort to Java Collections.

1.2 TupleDesc

To avoid duplicating code, we introduced two helper methods: *__tupleDesc* and *__containsSameTDItems* to, respectively, initialize the tuple and check if two tuple descriptors contain the same items. Please notice that the method *hashCode* was not implemented, as we opted not to use the *TupleDesc* as a key for the *HashMap*.

1.3 Catalog

Two hash maps were employed: the first one takes the id as key and the *Table* object as value, whereas the second uses the name of the table as a key and the id as a value. This allows to search more conveniently the id and the name of a *Table*, avoiding linear time complexity in one of the two cases.

1.4 BufferPool

Pages are referenced through a *HashMap*. At every *getPage()* call a new page is added and a counter *freePages* is decremented. When it reaches zero and a new page is requested, a *DbException* is thrown.

1.5 HeapPage

Various methods were implemented employing bitwise computations. The iterator is created by first gathering all tuples in an *ArrayList* and then returning its built-in iterator.

1.6 HeapPageId

The class consists almost entirely of getters and setters. The *hashCode* method was implemented concatenating the tableId and pageNumber as suggested.

1.7 RecordId

The same considerations of *HeapPageId* remain valid here. The *hashCode* method was implemented concatenating the *pageId* and *tupleNumber*, as suggested.

1.8 HeapFile

The random access in *readPage* is managed through the class *RandomAccessFile*. Iterator returns a *HeapFileIterator*, described in **section 1.9**.

1.9 HeapFileIterator

This class was implemented from scratch, implementing (in a Java sense) the *DbFileIterator* interface. A counter keeps track of the current page while Tuples inside each page are returned with the page iterator previously written.

1.10 SeqScan

This class implements the *OpIterator* interface. Exposes a *DbFileIterator* reference to implement *open, hasNext, next, close* and *rewind* methods.

2 General comments

In this section, we're gonna describe the time spent in this lab, our split of the work as well as our subjective evaluation of the effort in terms of difficult and/or confusing aspects of the requirements.
The work was split equally, following the suggested order of the implementations and employing a github workflow to make sure the provided tests actually passed on a gh-provided runner. The entire work took overall, approximately, 4h per person.
Since we already took other courses of Java, the programming language was not a barrier for us. Broadly speaking, we encountered no particular setbacks. The only thing worth to mention is an initial confusion regarding the usage of the *Database* class to obtain *Catalog*s and *Table*s. We didn't notice it had static getters thus we were a little confused about how to fetch the information we needed.