1. Design Decisions
    a. Catalog.java
        i. Because table IDs are used frequently throughout the functions of the Catalog class, we decided to store all of them in a local vector, mirroring the vectors containing the table files, their names, and their primary key fields. Although this takes up more memory and likely has negligible performance ramifications, it creates a more convenient and clean interface through which we can access table IDs when they are needed. In latter labs when dealing with potentially larger data sets, the costs may outweigh the perceived benefits, which could mean removing this implementation altogether.
    b. HeapFileIterator.java
        i. A separate file containing the class which implements the DbFileIterator interface to create a functional iterator for HeapFiles. The iterator function in HeapFile.java then just returns a dynamically allocated instance of the HeapFileIterator class.
2. Changes to API
    a. Although no changes were made to the API with regards to function or argument names, we did decide to have certain functions throw exceptions whose skeletons did not do so initially. This is because the arguments for these functions have constraints, so any circumstance that leads to incorrect arguments being supplied should be known about, with exception messages being especially useful. We also attempted to do this with the Tuple class constructor (which has constraints on its initializing arguments), but because the exception was not caught in some of the project's files which we did not wish to alter (and this caused errors inhibiting compilation), we decided to use assert(clause) instead. The following functions now throw exceptions:
        i. Tuple.java
            1. setField(int, Field)
            2. getField(int)
            3. resetTupleDesc(TupleDesc)
        ii. Catalog.java
            1. addTable(DbFile, String, String)
            2. getPrimaryKey(int)
            3. getTableName(int)
3. Missing or Incomplete Elements
    a. The code we have written encompasses all of the functionality necessary for Lab 1, and passes all of the available unit tests. However, our liberal use of exceptions without the accompanying try-catch mechanisms could be categorized as an incomplete element. Additionally, due to difficulty understanding the instructions regarding hash codes and inexperience in implementing them, it is

possible that our hash code functions may not be complete enough for foolproof use without improving their implementations.

4. Difficulty
    a. The lab as a whole took around 14 hours of work. Neither of us had any issue with Java or its syntax. However, both being new to the underlying implementation of database management systems, understanding the existing code and learning how to manipulate and imitate its structure to fill out the necessary functions was a challenge. More specifically, the use of iterators was a source of difficulty as neither of us have had much experience with that type of data structure. Additionally, figuring out how to utilize a hash map for the buffer pool was also difficult. But, some of the challenge came from the instructions, or lack thereof. Figuring out the implementation of some of the functions (especially hash codes) was hard because we were not sure what was expected, and the use of terms like "concatenation" in a non-literal sense caused some confusion. We feel that more concise instructions could be beneficial, but recognize that coming to understandings and making decisions on our own offer us experience in a skillset all programmers need.