For this project, I didn't change any API. All codes that have been modified are those where instructions suggest to do so.

For *TupleDesc*, I use *ArrayList* to store all objects of *TDItem*, each of which contains field name and field type. In this way, I can use method *ArrayList.iterator()* to simply return iterator. Also it is convenient to use *ArrayList* to add *TDItem* in constructors, return length as well as get each *TDItem* through index. For method *equals*, first I check whether object, o is null, belongs to class *TupleDesc* and the length of o. Then each element is checked. For *Tuple*, apply the similar strategy. For *resetTupleDesc* method, I also clear all the data originally stored in fields,

For *catalog*, it contains the name, ID, stored file, primary key name of each table and use *ArrayList* to store all the values. For the methods like *getDatabaseFile()*, just apply for-loop.

For *BufferPool*, the main work focuses on method *getPage()*. First, search the existed pages id that matches the input page id. If there is one exact page id, return the corresponding page. If not, then check whether the number of pages exceeds the limited number. If not, add this page and return, otherwise, throw a *DbException*.

For *hashCode()* method of *HeapPageId*, converge the table id and page number to string, concatenate them and then converge it to integer. *RecordID* is easy. For *HeapPage*, calculate the number of tuples and the number of headers. For *getNumEmptySlots()* method, calculate the number of tuples used which is recorded as 1 in the corresponding position of the specific header. Because 0 in the position of the last header may not have corresponding slot. Then result is the difference of total number of slots and the number of slots used. For method *isSlotUsed()*, calculate and search the corresponding header and the position then check whether the value is 1 or 0. For method *iterator()*, I need create a new class called *TupleIterator* which implements *Iterator<Tuple>*. For method *HeapFile*, the focus is *readPage()*. First multiply page size with page number to calculate the offset. Then skip the offset and read data. For *iterator*, I create a new class called *HFDBFileTerator* which implements *DbFileIterator*. In this class, I mainly use the *iterator* of *HeapPage*. For *hasNext()*, check if iterator of heap page is null and page number is less than maximum page number. For *open()*, *rewind()* and *close()*, just initialize the fields or clear the fields.

For *SeqScan*, apply the similar ways mentioned above.

Drawbacks of my code. Although my project run all the tests successfully, I didn't apply many *Exception* commands so that there may be unexpected errors if the input is not proper. For *BufferPool.getPage(...)*, I didn't know how to use parameters *TransactionId tid, Permissions perm*, so I omit them.

I first learn Java on 4th and start the project on 9th. For java, the interface confused me quite a lot, so it may be better to add more content about it on java tutorial. For the header in heap page, I wonder the order 1/0 corresponding to each slot is stored from right to left or from left to right. When I test the code, I may judge the error from the name of test function, but cannot figure it out totally with debug.