Lab 1 Report

Design Decisions

Exercise 2: `Catalog`

- We created one inner class named `Table` that is used to represent a table associated with Database file, table name and primary key field.
- We also created two private fields `tableNameMap` and `tableIdMap` that keep track of the table IDs and names.

Exercise 3: `BufferPool`

- We declared two attributes required for the initialisation of the `BufferPool`; an integer `numPages` to store the number of pages in the BufferPool and a ConcurrentHashMap `pagesHashMap` to that maps the `PageId` to the `Page` objects to store the pages in the BufferPool.
- A ConcurrentHashMap is used as it is threadsafe; we cannot read or insert a page into the page at the same time.

Exercise 5: `HeapFile`

- To create an iterator which iterates through the tuples of each page in the `HeapFile`, we added an inner class `HeapFileIterator` which implements `DbFileIterator` interface.

Non-trival Part

Exericise 1

- `Tuple`: We use an `ArrayList<Field>` called `tupleFields` to store a list of fields in one tuple.
- `TupleDesc`: We use an `ArrayList<TDItem>` called `tdItemList` to store a list to `TDItem` in a schema.

Exercise 2

- `Catalog` constructor initializes two data containers:
  - `tableNameMap`: A hash map that maps table name to table ID
  - `tableIdMap`: A hash map that maps table ID to a table object

Exercise 3

- `getPage()`
  - If the page does not exist in the pool, we have to fetch it using `Database.getCatalog().getDatabaseFile(pid.getTableId()).readPage(pid)`
  - Additionally, we have to check if the size after adding the `newPage` exceeds the max number of Pages `numPage` allowed. Since eviction is not implemented, we throw a `DbException` if so.

Exercise 4

- `getNumTuples()`
  - requires calculating the page size in bits and dividing it by the tuple size in bits with an addition of the header bit, according to the formula
    `_tuples per page_ = floor((_page size_ * 8) / (_tuple size_ * 8 + 1))`
- `isSlotUsed()`
  - requires tracing the value of the bits that corresponds to the index of the slot to determine whether the slot is used.

Exercise 5

- `readPage()`
  - We use `RandomAccessFile` to random access to the file. We skip `pgNo * pageSize` number of bytes to read the page data.
  - If the `pid` reaches the end of the file, we create a new empty page at the end of this file and increase the number of pages by 1.
- `iterator()`
  `HeapFileIterator` class which implements `DbFileIterator` interface is returned.

Exercise 6

- `getTupleDesc()` : In order to handle the case where `tableAlias` or `fieldName` are `null`, we used a for-loop with if-else inside to determine if they are `null` and convert `null` to string so that the resulting name can be `null.fieldName`, `tableAlias.null`, or `null.null`.

Changes on API

Exercise 1

- Added one contructor in `TupleDesc.java`: `TupleDesc(ArrayList<TDItem> tdItemList)`

Incomplete Part

BufferPool:

- Currently has no eviction policy; throws a `DbException` when size of pool exceeds max size.
- Yet to account for permissions for `getPage` method.