MIT6.830 LAB1 NOTE

WHAT TO DO

SIMPLEDB ARCHITECTURE

- 用于存储的tuple, files, tuple schema等
- 应用predicated和conditions到tuples
- access methods，用来存储关系，提供关系的遍历方式
- select, join, delete等
- buffer pool，用来做缓存
- catalog保存表和其schema的信息

暂时没有SQL front end，这个可以考虑后面自己手写一个。没视图，数据类型小等。而且也没有优化器、没有索引。

DETAIL

exercise 1

实现这两个骨架：

- src/simpledb/TupleDesc.java
- src/simpledb/Tuple.java

Tuple就是filed objects的集合，然后filed类型可以不同。TupleDesc则是Tuple的schema，也就是其元信息。

先看下代码 `/src/java/simpledb/TupleDesc.java`，里面有个 `TDItem`，表示filed，包含其类型和名字：

```java
public class TupleDesc implements Serializable {

    /**
     * A help class to facilitate organizing the information of each field
     * */
    public static class TDItem implements Serializable {

        private static final long serialVersionUID = 1L;

        /**
         * The type of the field
         * */
        public final Type fieldType;

        /**
         * The name of the field
         * */
        public final String fieldName;

        public TDItem(Type t, String n) {
            this.fieldName = n;
            this.fieldType = t;
        }

        public String toString() { return fieldName + "(" + fieldType + ")"; }
    }
```

所以，TupleDesc就是tuple的schema，比如对于表student

```
id(int)  name(string)  sex(string)
1           xxx          m
2           yyy          f
```

那么`(1, xxx, m)`就是一个Tuple，然后TupleDesc是`(id(int) name(string) sex(string))`。

具体看下Type的实现(src/java/simpledb/Type.java)：

```java
public enum Type implements Serializable {
    INT_TYPE() {
        /*...*/
    }, STRING_TYPE() {
        /*...*/
    };

    public static final int STRING_LEN = 128;
    public abstract int getLen();
    public abstract Field parse(DataInputStream dis) throws
ParseException;
}
```

是个枚举类型，目前只有string和int两种类型，然后支持从DataInputStream解析出Field，而Field是一个interface，有IntField和StringField实现。

至此思路很清晰，只需要给Tuple加个 `TupleDesc` 和 `ArrayList<Field>` 成员，并实现相关函数。in

exercise 2

Catelog是Table的集合，然后每个Table有 `DbFile, PrimaryKeyField, Name` ，所以：

```
20      public class Catalog {
21
22          public class Table {
23              private DbFile file;
24              private String name;
25              private String pkeyField;
26
27              public Table(DbFile file, String name, String pkeyField) {
28                  this.file = file;
29                  this.name = name;
30                  this.pkeyField = pkeyField;
31              }
32          }
33
34          private HashMap<Integer, Table> tables;
```

后面增删查改就很容易了。

3.6, 17.45 ok, about 0.5 hour

exercise 3

BufferPool是用来在内存中缓存从disk读入的pages的。有固定的pages大小，所以需要有置换策略，这个在后面的lab会实现。

exercise 4

DbFil是用来read/write数据的interface，有HeapFile和BTreeFile两个实现。

每个HeapFile表示一个表，包含了多个page，然后每个page的size都是固定的。所以，每个page可以分配若干个Tuple，计算方式是：

```
_tuples per page_ = floor((_page size_ * 8) / (_tuple size_ * 8 + 1))
```

之所以需要+1，是因为每个page都会有一个header，这个header里面有个bitmap来表示对应Tuple是否已经in use了。

exercise 5

HeapFile是DbFile interface的实现，包含有一系列HeapPage，是没有顺序的。然后HeapFile刚开始是不会将数据读入内存的，只保存了其handler，只有在readPage的时候，才会根据元信息计算出offset，然后在文件中读取对应的page。

然后还需要给实现Iterator，我们在open的时候不要将整个文件都读入内存，而是遍历的时候，根据position来计算offset，只读入对应的Tuple。所以，需要维护几个变量：`currPageId`，`currPage`，`tupleIter`，`open`等来维护现在查询的状态。然后，我们的page都是从`BufferPool`中读取的，而不是直接去disk读。

在写这个的时候，尝试用`RandomAccessFile`去读文件，然后采用的是`read(buf, offset, len)`函数，offset设置的是在文件中的offset。后来发现，对于这个函数的offset有点误解，应该是在buf中的offset。正确的用法应该是，先`seek(offset)`，然后`read(buf)`。

所以，BufferPool和HeapFile是互相调用的。如果我想要readPage，会去BufferPool里面取，如果内存中没有缓存这个page，BufferPool会调用HeapFile的readPage，然后从disk中，根据计算出来的offset，读取page。

exercise 6

很容易，包装一下HeapFile的iterator就好。