数据库课程实践期末考试

院、系	计算机学院	年级专业	2020 软件工程	姓名	高歌	学号	2030416018
课程名称	数据库课程实践					成绩	

一. 补全 simpleDB 中的 TupleDesc.java 和 Tuple.java。

注：为了清晰起见，所有的"// some code goes here"都未删去，可根据这段注释定位代码。

（1）TupleDesc 的实现

首先补全 TupleDesc 相关代码。根据代码中给出的注释，可以了解 TupleDesc（TupleDescription）负责描述某个一表中元组的各字段名以及类型信息。

考虑到 TupleDesc 需要存储 TDItem，首先定义 items 属性：

```java
private final TDItem[] items;
```

然后实现构造器：

```java
public TupleDesc(Type[] typeAr, String[] fieldAr) {
    // some code goes here
    items = new TDItem[typeAr.length];
    for (int i = 0; i < typeAr.length; i++) {
        items[i] = new TDItem(typeAr[i], fieldAr[i]);
    }
}

public TupleDesc(Type[] typeAr) {
    // some code goes here
    this(typeAr, new String[typeAr.length]);
}
```

然后实现几个基本方法：

```java
public int numFields() {
    // some code goes here
    return items.length;
 }

public String getFieldName(int i) throws NoSuchElementException {
    // some code goes here
    if (i < 0 || i >= items.length) {
        throw new NoSuchElementException();
    }
    return items[i].fieldName;
}

public Type getFieldType(int i) throws NoSuchElementException {
    // some code goes here
    if (i < 0 || i >= items.length) {
        throw new NoSuchElementException();
    }
```

```
        return items[i].fieldType;
    }

    public int fieldNameToIndex(String name) throws NoSuchElementException {
        // some code goes here
        for (int i = 0; i < items.length; i++) {
            if (items[i].fieldName != null && items[i].fieldName.equals(name)) {
                return i;
            }
        }
        throw new NoSuchElementException();
    }

    public int getSize() {
        // some code goes here
        int size = 0;
        for (int i = 0; i < items.length; i++) {
            size += items[i].fieldType.getLen();
        }
        return size;
    }

    public String toString() {
        // some code goes here
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < items.length; i++) {
            sb.append(items[i].fieldName);
            sb.append("(");
            sb.append(items[i].fieldType.toString());
            sb.append(")");
            if (i != items.length - 1) {
                sb.append(", ");
            }
        }
        return sb.toString();
    }
```

上面这些代码实现起来都比较简单，也没什么太多好谈的。下面将几个较为特殊的方法拿出来单独提一下。

首先是 equals，这是个容易实现但不太容易实现好的方法。一个通用的做法是，先与 null 值判断，接着判断是否与自身指向同一引用，再然后判断是否与自身属于同一个类，再往后才进行属性判断。子这里，要判断两个 TupleDesc 实例是否相等，先判断字段数是否相等，然后判断字段类型是否也依次相等（根据注释，这里的判断相等不要求字段名也对应相等）：

```
    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof TupleDesc)) {
            return false;
        }
        TupleDesc td = (TupleDesc) o;
        if (td.numFields() != numFields()) {
            return false;
        }
        for (int i = 0; i < numFields(); i++) {
            if (!getFieldType(i).equals(td.getFieldType(i))) {
                return false;
            }
        }
        return true;
    }
```

另一个略微复杂的是 merge 的实现，但本质上也非常简单，即简单地创建新的 typeAr 和 fieldAr 将两者合并，然后创建新的 TupleDesc.

```
    public static TupleDesc merge(TupleDesc td1, TupleDesc td2) {
        // some code goes here
        Type[] typeAr = new Type[td1.numFields() + td2.numFields()];
        String[] fieldAr = new String[td1.numFields() + td2.numFields()];
        for (int i = 0; i < td1.numFields(); i++) {
            typeAr[i] = td1.getFieldType(i);
            fieldAr[i] = td1.getFieldName(i);
        }
        for (int i = 0; i < td2.numFields(); i++) {
            typeAr[td1.numFields() + i] = td2.getFieldType(i);
            fieldAr[td1.numFields() + i] = td2.getFieldName(i);
        }
        return new TupleDesc(typeAr, fieldAr);
    }
```

然后是标准的迭代器实现，比较简单，也没什么可多谈的：

```
    public Iterator<TDItem> iterator() {
        // some code goes here
        return new TupleDescIterator();
    }

    private class TupleDescIterator implements Iterator<TDItem> {

        private int curr = 0;
        @Override
        public boolean hasNext() {
            return curr < numFields();
        }

        @Override
        public TDItem next() {
            return items[curr++];
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
```

至此，TupleDesc 实现完毕。

（2）Tuple 的实现

根据注释，Tuple 负责存储表中的元组，它的模式由一个 TupleDesc 实例确定，并且包含一个可选的 RecordId 实例。因此为 Tuple 类添加下面三个实例属性：

```
    private TupleDesc td;
    private RecordId rid;
    private final Field[] fields;
```

然后是构造器：

```
    public Tuple(TupleDesc td) {
        // some code goes here
        this.td = td;
        this.rid = null;
        this.fields = new Field[td.numFields()];
    }
```

接着是几个简单的方法，没什么好多说的：

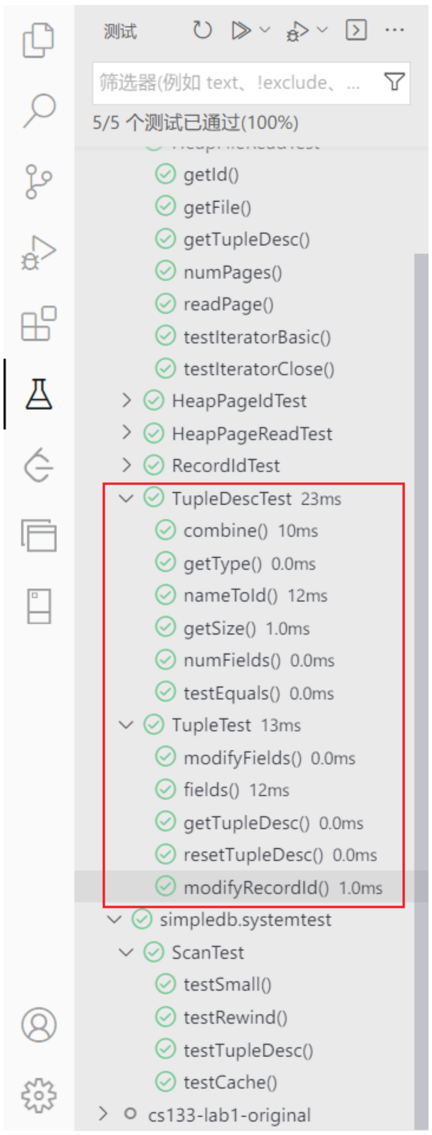
```
    public TupleDesc getTupleDesc() {
        // some code goes here
        return td;
    }

    public RecordId getRecordId() {
        // some code goes here
        return rid;
    }

    public void setRecordId(RecordId rid) {
        // some code goes here
        this.rid = rid;
    }
```

```
public void setField(int i, Field f) {
    // some code goes here
   fields[i] = f;
}

public Field getField(int i) {
    // some code goes here
    return fields[i];
}

public String toString() {
    // some code goes here
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < fields.length; i++) {
        sb.append(fields[i].toString());
        if (i != fields.length - 1) {
            sb.append("\t");
        }
    }
    sb.append("\n");
    return sb.toString();
}

public void resetTupleDesc(TupleDesc td) {
    // some code goes here
    this.td = td;
}
```

然后是迭代器的实现，同 TupleDesc 区别不大：

```
public Iterator<Field> fields() {
    // some code goes here
    return new FieldIterator();
}

private class FieldIterator implements Iterator<Field> {
    private int i = 0;

    @Override
    public boolean hasNext() {
        // some code goes here
        return i < fields.length;
    }

    @Override
    public Field next() {
        // some code goes here
        return fields[i++];
    }

    @Override
    public void remove() {
        // some code goes here
        throw new UnsupportedOperationException();
    }
}
```

（3）单元测试



```
5/5 个测试已通过(100%)
    getId()
    getFile()
    getTupleDesc()
    numPages()
    readPage()
    testIteratorBasic()
    testIteratorClose()
  > HeapPageIdTest
  > HeapPageReadTest
  > RecordIdTest
  v TupleDescTest 23ms
    combine() 10ms
    getType() 0.0ms
    nameToId() 12ms
    getSize() 1.0ms
    numFields() 0.0ms
    testEquals() 0.0ms
  v TupleTest 13ms
    modifyFields() 0.0ms
    fields() 12ms
    getTupleDesc() 0.0ms
    resetTupleDesc() 0.0ms
    modifyRecordId() 1.0ms
 v simpledb.systemtest
  v ScanTest
    testSmall()
    testRewind()
    testTupleDesc()
    testCache()
> cs133-lab1-original
```

*注：事实上写到这里时，所有代码都已经写完了，因此已经通过了全部测试

（4）附：全部代码

```
// src/java/TupleDesc.java
```

```java
package simpledb;

import java.io.Serializable;
import java.util.*;

/**
 * TupleDesc describes the schema of a tuple.
 */
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 + ")";
        }
    }

    /**
     * @return
     *         An iterator which iterates over all the field TDItems
     *         that are included in this TupleDesc
     */
    public Iterator<TDItem> iterator() {
        // some code goes here
        return new TupleDescIterator();
    }

    private class TupleDescIterator implements Iterator<TDItem> {

        private int curr = 0;

        public boolean hasNext() {
            return curr < numFields();
        }

        public TDItem next() {
            return items[curr++];
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    private static final long serialVersionUID = 1L;

    private final TDItem[] items;

    /**
     * Create a new TupleDesc with typeAr.length fields with fields of the
     * specified types, with associated named fields.
     *
     * @param typeAr
     *               array specifying the number of and types of fields in this
     *               TupleDesc. It must contain at least one entry.
     * @param fieldAr
     *               array specifying the names of the fields. Note that names may
     *               be null.
     */
    public TupleDesc(Type[] typeAr, String[] fieldAr) {
        // some code goes here
        items = new TDItem[typeAr.length];
        for (int i = 0; i < typeAr.length; i++) {
            items[i] = new TDItem(typeAr[i], fieldAr[i]);
        }
    }

    /**
     * Constructor. Create a new tuple desc with typeAr.length fields with
     * fields of the specified types, with anonymous (unnamed) fields.
     *
     * @param typeAr
     *            array specifying the number of and types of fields in this
     *            TupleDesc. It must contain at least one entry.
     */
    public TupleDesc(Type[] typeAr) {
        // some code goes here
        this(typeAr, new String[typeAr.length]);
    }

    /**
     * @return the number of fields in this TupleDesc
     */
    public int numFields() {
        // some code goes here
        return items.length;
    }

    /**
     * Gets the (possibly null) field name of the ith field of this TupleDesc.
     *
     * @param i
     *         index of the field name to return. It must be a valid index.
     * @return the name of the ith field
     * @throws NoSuchElementException
     *                     if i is not a valid field reference.
     */
    public String getFieldName(int i) throws NoSuchElementException {
        // some code goes here
        if (i < 0 || i >= items.length) {
            throw new NoSuchElementException();
        }
        return items[i].fieldName;
    }

    /**
     * Gets the type of the ith field of this TupleDesc.
     *
     * @param i
     *         The index of the field to get the type of. It must be a valid
     *         index.
     * @return the type of the ith field
     * @throws NoSuchElementException
     *                     if i is not a valid field reference.
     */
    public Type getFieldType(int i) throws NoSuchElementException {
        // some code goes here
        if (i < 0 || i >= items.length) {
            throw new NoSuchElementException();
        }
        return items[i].fieldType;
    }

    /**
     * Find the index of the field with a given name.
     *
     * @param name
     *             name of the field.
     * @return the index of the field that is first to have the given name.
     * @throws NoSuchElementException
     *                                if no field with a matching name is found.
     */
    public int fieldNameToIndex(String name) throws NoSuchElementException {
        // some code goes here
        for (int i = 0; i < items.length; i++) {
            if (items[i].fieldName != null && items[i].fieldName.equals(name)) {
                return i;
            }
        }
        throw new NoSuchElementException();
    }

    /**
     * @return The size (in bytes) of tuples corresponding to this TupleDesc.
     *         Note that tuples from a given TupleDesc are of a fixed size.
     */
    public int getSize() {
        // some code goes here
        int size = 0;
        for (int i = 0; i < items.length; i++) {
            size += items[i].fieldType.getLen();
        }
        return size;
    }

    /**
     * Merge two TupleDescs into one, with td1.numFields + td2.numFields fields,
     * with the first td1.numFields coming from td1 and the remaining from td2.
     *
     * @param td1
     *             The TupleDesc with the first fields of the new TupleDesc
     * @param td2
     *             The TupleDesc with the last fields of the TupleDesc
     * @return the new TupleDesc
     */
    public static TupleDesc merge(TupleDesc td1, TupleDesc td2) {
        // some code goes here
        Type[] typeAr = new Type[td1.numFields() + td2.numFields()];
        String[] fieldAr = new String[td1.numFields() + td2.numFields()];
        for (int i = 0; i < td1.numFields(); i++) {
            typeAr[i] = td1.getFieldType(i);
            fieldAr[i] = td1.getFieldName(i);
        }
        for (int i = 0; i < td2.numFields(); i++) {
            typeAr[td1.numFields() + i] = td2.getFieldType(i);
            fieldAr[td1.numFields() + i] = td2.getFieldName(i);
        }
        return new TupleDesc(typeAr, fieldAr);
    }

    /**
     * Compares the specified object with this TupleDesc for equality. Two
     * TupleDescs are considered equal if they are the same size and if the n-th
     * type in this TupleDesc is equal to the n-th type in td.
     *
     * @param o
     *            the Object to be compared for equality with this TupleDesc.
     * @return true if the object is equal to this TupleDesc.
     */
    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof TupleDesc)) {
            return false;
        }
        TupleDesc td = (TupleDesc) o;
        if (td.numFields() != numFields()) {
            return false;
        }
        for (int i = 0; i < numFields(); i++) {
```

```java
            if (!getFieldType(i).equals(td.getFieldType(i))) {
                return false;
            }
        }
        return true;
    }

    public int hashCode() {
        // If you want to use TupleDesc as keys for HashMap, implement this so
        // that equal objects have equals hashCode() results
        throw new UnsupportedOperationException("unimplemented");
    }

    /**
     * Returns a String describing this descriptor. It should be of the form
     * "fieldName[0](fieldType[0]), ..., fieldName[M](fieldType[M])", although
     * the exact format does not matter.
     *
     * @return String describing this descriptor.
     */
    public String toString() {
        // some code goes here
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < items.length; i++) {
            sb.append(items[i].fieldName);
            sb.append("(");
            sb.append(items[i].fieldType.toString());
            sb.append(")");
            if (i != items.length - 1) {
                sb.append(", ");
            }
        }
        return sb.toString();
    }
}
```

```java
// src/java/Tuple.java

package simpledb;

import java.io.Serializable;
import java.util.Arrays;
import java.util.Iterator;

/**
```

```
 * Tuple maintains information about the contents of a tuple. Tuples have a
 * specified schema specified by a TupleDesc object and contain Field objects
 * with the data for each field.
 */
public class Tuple implements Serializable {

    private static final long serialVersionUID = 1L;

    private TupleDesc td;
    private RecordId rid;
    private final Field[] fields;

    /**
     * Create a new tuple with the specified schema (type).
     *
     * @param td
     *            the schema of this tuple. It must be a valid TupleDesc
     *            instance with at least one field.
     */
    public Tuple(TupleDesc td) {
        // some code goes here
        this.td = td;
        this.rid = null;
        this.fields = new Field[td.numFields()];
    }

    /**
     * @return The TupleDesc representing the schema of this tuple.
     */
    public TupleDesc getTupleDesc() {
        // some code goes here
        return this.td;
    }

    /**
     * @return The RecordId representing the location of this tuple on disk. May
     *         be null.
     */
    public RecordId getRecordId() {
        // some code goes here
        return this.rid;
    }

    /**
     * Set the RecordId information for this tuple.
     *
     * @param rid
     *            the new RecordId for this tuple.
     */
    public void setRecordId(RecordId rid) {
        // some code goes here
        this.rid = rid;
    }

    /**
     * Change the value of the ith field of this tuple.
     *
     * @param i
     *          index of the field to change. It must be a valid index.
     * @param f
     *          new value for the field.
     */
    public void setField(int i, Field f) {
        // some code goes here
        this.fields[i] = f;
    }

    /**
     * @return the value of the ith field, or null if it has not been set.
     *
     * @param i
     *          field index to return. Must be a valid index.
     */
    public Field getField(int i) {
        // some code goes here
        return this.fields[i];
    }

    /**
     * Returns the contents of this Tuple as a string. Note that to pass the
     * system tests, the format needs to be as follows:
     *
     * column1\tcolumn2\tcolumn3\t...\tcolumnN\n
     *
     * where \t is any whitespace, except newline, and \n is a newline
     */
    public String toString() {
        // some code goes here
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < this.fields.length; i++) {
            sb.append(this.fields[i].toString());
            if (i != this.fields.length - 1) {
                sb.append("\t");
            }
        }
        sb.append("\n");
        return sb.toString();
    }

    /**
     * @return
     *         An iterator which iterates over all the fields of this tuple
     */
    public Iterator<Field> fields() {
        // some code goes here
        return new FieldIterator();
    }

    private class FieldIterator implements Iterator<Field> {
        private int i = 0;

        @Override
        public boolean hasNext() {
            // some code goes here
            return i < fields.length;
        }

        @Override
        public Field next() {
            // some code goes here
            return fields[i++];
        }

        @Override
        public void remove() {
            // some code goes here
            throw new UnsupportedOperationException();
        }
    }

    /**
     * Reset the TupleDesc of this tuple
     * Does not need to worry about the fields inside the Tuple
     */
    public void resetTupleDesc(TupleDesc td) {
        // some code goes here
        this.td = td;
    }
}
```

二. 补全 simpleDB 中的 Catalog.java。

（1）Catalog.java 的实现

根据注释，每个数据库仅有一个 Catalog 实例，用于跟踪数据库中所有模式（即表）。对表的各类操作（添加、删除、修改、获取、查找文件等）都通过 Catalog 来实现。

首先考虑到 Catalog 用于进行对表的操作，因此必然存在一个 Map 对象用于存储表文件；此外考虑到 TupleDesc 并不包含主键相关信息，因此 Catalog 还需额外存储主键信息，这需要另一个 Map 对象。为线程安全考虑，均使用 ConcurrentMap 实现：

```
    private final Map<String, DbFile> dbfiles;
    private final Map<String, String> primaryKeys;

    public Catalog() {
        // some code goes here
        dbfiles = new ConcurrentHashMap<String, DbFile>();
        primaryKeys = new ConcurrentHashMap<String, String>();
    }
```

然后是添加表的方法，这里涉及了三次重载：

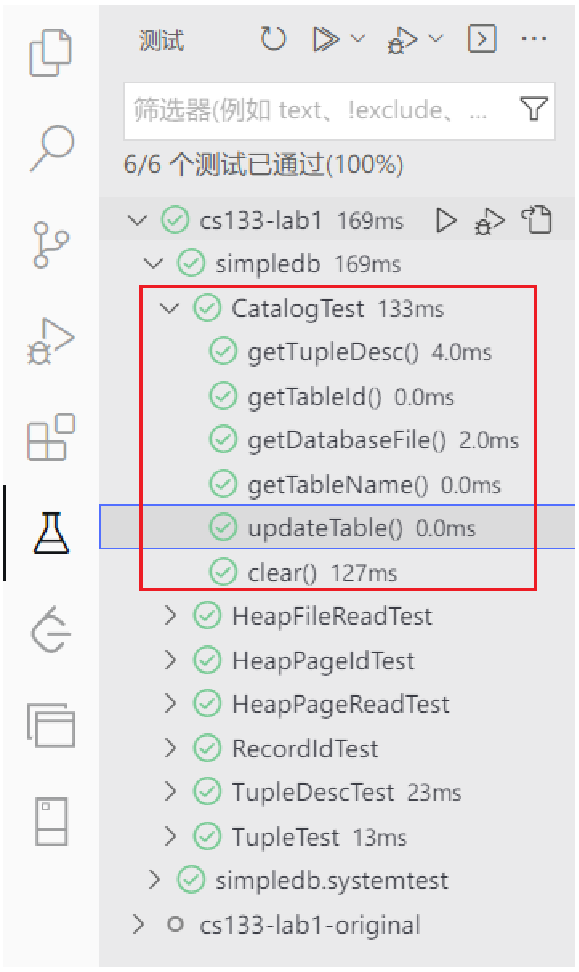
```
    public void addTable(DbFile file, String name, String pkeyField) {
        // some code goes here
        if (name == null) {
            throw new IllegalArgumentException("name is null");
        }
        if (file == null) {
            throw new IllegalArgumentException("file is null");
        }
        if (pkeyField == null) {
            throw new IllegalArgumentException("pkeyField is null");
        }
        dbfiles.put(name, file);
        primaryKeys.put(name, pkeyField);
    }

    public void addTable(DbFile file, String name) {
        addTable(file, name, "");
    }

    public void addTable(DbFile file) {
        addTable(file, (UUID.randomUUID()).toString());
    }
```

然后是一些简单的方法。为了简洁起见，这里部分使用了 Java 8 的 Stream API，因此在 Java 8 以

下版本应当无法运行。

```java
    public int getTableId(String name) throws NoSuchElementException {
        // some code goes here
        if (name == null) {
            throw new NoSuchElementException("name is null");
        }
        if (!dbfiles.containsKey(name)) {
            throw new NoSuchElementException("table " + name + " does not exist");
        }
        return dbfiles.get(name).getId();
    }
    public TupleDesc getTupleDesc(int tableid) throws NoSuchElementException {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return dbfiles.get(getTableName(tableid)).getTupleDesc();
    }
    public DbFile getDatabaseFile(int tableid) throws NoSuchElementException {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return dbfiles.get(getTableName(tableid));
    }
    public String getPrimaryKey(int tableid) {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return primaryKeys.get(getTableName(tableid));
    }
    public String getTableName(int id) {
        // some code goes here
        return dbfiles.entrySet().stream()
                .filter(x -> x.getValue().getId() == id)
                .findFirst().get().getKey();
    }
    public void clear() {
        // some code goes here
        dbfiles.clear();
    }
    public Iterator<Integer> tableIdIterator() {
        // some code goes here
        return dbfiles.values().stream().map(x -> x.getId()).iterator();
    }
```

（2）单元测试


测试

筛选器(例如 text、!exclude、...

6/6 个测试已通过(100%)

```
cs133-lab1 169ms
  simpledb 169ms
    CatalogTest 133ms
      getTupleDesc() 4.0ms
      getTableId() 0.0ms
      getDatabaseFile() 2.0ms
      getTableName() 0.0ms
      updateTable() 0.0ms
      clear() 127ms
    HeapFileReadTest
    HeapPageIdTest
    HeapPageReadTest
    RecordIdTest
    TupleDescTest 23ms
    TupleTest 13ms
  simpledb.systemtest
cs133-lab1-original
```

（3）附：全部代码

```java
// src/java/Catalog.java

package simpledb;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

/**
 * The Catalog keeps track of all available tables in the database and their
 * associated schemas.
 * For now, this is a stub catalog that must be populated with tables by a
 * user program before it can be used -- eventually, this should be converted
 * to a catalog that reads a catalog table from disk.
 *
 * @Threadsafe
 */
public class Catalog {

    private final Map<String, DbFile> dbfiles;
    private final Map<String, String> primaryKeys;

    /**
     * Constructor.
     * Creates a new, empty catalog.
     */
    public Catalog() {
        // some code goes here
        dbfiles = new ConcurrentHashMap<String, DbFile>();
        primaryKeys = new ConcurrentHashMap<String, String>();
    }

    /**
     * Add a new table to the catalog.
     * This table's contents are stored in the specified DbFile.
     *
     * @param file      the contents of the table to add; file.getId() is the
     *                  identfier of
     *                  this file/tupledesc param for the calls getTupleDesc and
     *                  getFile
     * @param name      the name of the table -- may be an empty string. May not be
     *                  null. If a name
     *                  conflict exists, use the last table to be added as the table
     *                  for a given name.
     * @param pkeyField the name of the primary key field
     */
    public void addTable(DbFile file, String name, String pkeyField) {
        // some code goes here
        if (name == null) {
            throw new IllegalArgumentException("name is null");
        }
        if (file == null) {
            throw new IllegalArgumentException("file is null");
        }
        if (pkeyField == null) {
            throw new IllegalArgumentException("pkeyField is null");
        }
        dbfiles.put(name, file);
```

```java
        primaryKeys.put(name, pkeyField);
    }

    public void addTable(DbFile file, String name) {
        addTable(file, name, "");
    }

    /**
     * Add a new table to the catalog.
     * This table has tuples formatted using the specified TupleDesc and its
     * contents are stored in the specified DbFile.
     *
     * @param file the contents of the table to add; file.getId() is the identfier
     *             of
     *             this file/tupledesc param for the calls getTupleDesc and getFile
     */
    public void addTable(DbFile file) {
        addTable(file, (UUID.randomUUID()).toString());
    }

    /**
     * Return the id of the table with a specified name,
     *
     * @throws NoSuchElementException if the table doesn't exist
     */
    public int getTableId(String name) throws NoSuchElementException {
        // some code goes here
        if (name == null) {
            throw new NoSuchElementException("name is null");
        }
        if (!dbfiles.containsKey(name)) {
            throw new NoSuchElementException("table " + name + " does not exist");
        }
        return dbfiles.get(name).getId();
    }

    /**
     * Returns the tuple descriptor (schema) of the specified table
     *
     * @param tableid The id of the table, as specified by the DbFile.getId()
     *                function passed to addTable
     * @throws NoSuchElementException if the table doesn't exist
     */
    public TupleDesc getTupleDesc(int tableid) throws NoSuchElementException {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return dbfiles.get(getTableName(tableid)).getTupleDesc();
    }

    /**
     * Returns the DbFile that can be used to read the contents of the
     * specified table.
     *
     * @param tableid The id of the table, as specified by the DbFile.getId()
     *                function passed to addTable
     */
    public DbFile getDatabaseFile(int tableid) throws NoSuchElementException {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return dbfiles.get(getTableName(tableid));
    }

    public String getPrimaryKey(int tableid) {
        // some code goes here
        if (dbfiles.values().stream().noneMatch(x -> x.getId() == tableid)) {
            throw new NoSuchElementException("table " + tableid + " does not exist");
        }
        return primaryKeys.get(getTableName(tableid));
    }

    public Iterator<Integer> tableIdIterator() {
        // some code goes here
        return dbfiles.values().stream().map(x -> x.getId()).iterator();
    }

    public String getTableName(int id) {
        // some code goes here
        return dbfiles.entrySet().stream()
            .filter(x -> x.getValue().getId() == id)
            .findFirst().get().getKey();
    }

    /** Delete all tables from the catalog */
    public void clear() {
        // some code goes here
        dbfiles.clear();
    }

    /**
     * Reads the schema from a file and creates the appropriate tables in the
     * database.
     *
     * @param catalogFile
     */
    public void loadSchema(String catalogFile) {
        String line = "";
        String baseFolder = new File(new File(catalogFile).getAbsolutePath())
                .getParent();
        try {
            BufferedReader br = new BufferedReader(
                    new FileReader(new File(catalogFile)));

            while ((line = br.readLine()) != null) {
                // assume line is of the format name (field type, field type, ...)
                String name = line.substring(0, line.indexOf("(")).trim();
                // System.out.println("TABLE NAME: " + name);
                String fields = line.substring(line.indexOf("(") + 1,
line.indexOf(")")).trim();
                String[] els = fields.split(",");
                ArrayList<String> names = new ArrayList<String>();
                ArrayList<Type> types = new ArrayList<Type>();
                String primaryKey = "";
                for (String e : els) {
                    String[] els2 = e.trim().split(" ");
                    names.add(els2[0].trim());
                    if (els2[1].trim().toLowerCase().equals("int"))
                        types.add(Type.INT_TYPE);
                    else if (els2[1].trim().toLowerCase().equals("string"))
                        types.add(Type.STRING_TYPE);
                    else {
                        System.out.println("Unknown type " + els2[1]);
                        System.exit(0);
                    }
                    if (els2.length == 3) {
                        if (els2[2].trim().equals("pk"))
                            primaryKey = els2[0].trim();
                        else {
                            System.out.println("Unknown annotation " + els2[2]);
                            System.exit(0);
                        }
                    }
                }
                Type[] typeAr = types.toArray(new Type[0]);
                String[] namesAr = names.toArray(new String[0]);
                TupleDesc t = new TupleDesc(typeAr, namesAr);
                HeapFile tabHf = new HeapFile(new File(baseFolder + "/" + name +
".dat"), t);
                addTable(tabHf, name, primaryKey);
                System.out.println("Added table : " + name + " with schema " + t);
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(0);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Invalid catalog entry : " + line);
            System.exit(0);
        }
    }
}
```

三. 补全 simpleDB 中的 BufferPool.java。

（1）BufferPool.java 的实现

根据注释，BufferPool 用于缓冲，访问数据时首先通过缓冲池获得，若不存在则加入缓冲池。当缓冲池已满时，丢弃第一个页面，若该页面已被更改，则先保存后再丢弃。

因此，需要包含一个 Map 对象用于存储所有 Page，还需要 numPages 变量用于保存 Page 的最大数量。此外，为了在缓冲池满时丢弃页面，还需要一个队列用于存储 pageId 加入的顺序，这里使用数组+一个用于存储数组下标的变量模拟这个队列：

```
    private final ConcurrentHashMap<PageId, Page> buffer;
    private final int numPages;
    private final PageId[] pageIds;
    private int pageIdIndex;

    public BufferPool(int numPages) {
        // some code goes here
        buffer = new ConcurrentHashMap<PageId, Page>(numPages);
        this.numPages = numPages;
        pageIds = new PageId[numPages];
        pageIdIndex = 0;
    }
```

然后根据题目，补全 getPage()方法即可：

```
    public Page getPage(TransactionId tid, PageId pid, Permissions perm)
            throws TransactionAbortedException, DbException, IOException {
        // some code goes here
        if (tid == null) {
            throw new DbException("Transaction is null");
        }
        if (pid == null) {
            throw new DbException("PageId is null");
        }
        if (perm == null) {
            throw new DbException("Permissions is null");
        }
        if (perm != Permissions.READ_ONLY && perm != Permissions.READ_WRITE) {
            throw new DbException("Permissions is not READ_ONLY or READ_WRITE");
        }
        if (buffer.containsKey(pid)) {
            return buffer.get(pid);
        } else {
            Page page = Database.getCatalog().getDatabaseFile(pid.getTableId())
                    .readPage(pid);
            if (page == null) {
                throw new DbException("Page is null");
            }
            if (buffer.size() == numPages) {
                if (buffer.get(pageIds[0]).isDirty() != null) {
                    flushPage(pageIds[0]);
                }
                buffer.remove(pageIds[0]);
                for (int i = 0; i < numPages - 1; i++) {
                    pageIds[i] = pageIds[i + 1];
                }
                pageIds[numPages - 1] = pid;
            } else {
                pageIds[pageIdIndex] = pid;
                pageIdIndex++;
            }
            buffer.put(pid, page);
            return page;
        }
    }
```

（2）单元测试

本题不包含对应的单元测试。

（3）附：全部代码

```
// src/java/BufferPool.java

package simpledb;

import java.io.*;
```

```java
import java.util.concurrent.ConcurrentHashMap;

/**
 * BufferPool manages the reading and writing of pages into memory from
 * disk. Access methods call into it to retrieve pages, and it fetches
 * pages from the appropriate location.
 * <p>
 * The BufferPool is also responsible for locking; when a transaction fetches
 * a page, BufferPool checks that the transaction has the appropriate
 * locks to read/write the page.
 *
 * @Threadsafe, all fields are final
 */
public class BufferPool {
    /** Bytes per page, including header. */
    public static final int PAGE_SIZE = 4096;

    private static int pageSize = PAGE_SIZE;

    /**
     * Default number of pages passed to the constructor. This is used by
     * other classes. BufferPool should use the numPages argument to the
     * constructor instead.
     */
    public static final int DEFAULT_PAGES = 50;

    private final ConcurrentHashMap<PageId, Page> buffer;
    private final int numPages;

    /**
     * TODO for Lab 4: create your private Lock Manager class.
     * Be sure to instantiate it in the constructor.
     */

    /**
     * Creates a BufferPool that caches up to numPages pages.
     *
     * @param numPages maximum number of pages in this buffer pool.
     */
    public BufferPool(int numPages) {
        // some code goes here
        buffer = new ConcurrentHashMap<PageId, Page>(numPages);
        this.numPages = numPages;
    }
```

```java
public static int getPageSize() {
    return pageSize;
}

/**
 * Helper: this should be used for testing only!!!
 */
public static void setPageSize(int pageSize) {
    BufferPool.pageSize = pageSize;
}

/**
 * Retrieve the specified page with the associated permissions.
 * Will acquire a lock and may block if that lock is held by another
 * transaction.
 * <p>
 * The retrieved page should be looked up in the buffer pool. If it
 * is present, it should be returned. If it is not present, it should
 * be added to the buffer pool and returned. If there is insufficient
 * space in the buffer pool, an page should be evicted and the new page
 * should be added in its place.
 *
 * @param tid  the ID of the transaction requesting the page
 * @param pid  the ID of the requested page
 * @param perm the requested permissions on the page
 */
public Page getPage(TransactionId tid, PageId pid, Permissions perm)
        throws TransactionAbortedException, DbException {
    // some code goes here
    if (tid == null) {
        throw new DbException("Transaction is null");
    }
    if (pid == null) {
        throw new DbException("PageId is null");
    }
    if (perm == null) {
        throw new DbException("Permissions is null");
    }
    if (perm != Permissions.READ_ONLY && perm != Permissions.READ_WRITE) {
        throw new DbException("Permissions is not READ_ONLY or READ_WRITE");
    }
    if (buffer.containsKey(pid)) {
        return buffer.get(pid);
    } else {
            Page page = Database.getCatalog().getDatabaseFile(pid.getTableId())
                    .readPage(pid);
            if (page == null) {
                throw new DbException("Page is null");
            }
            if (buffer.size() == numPages) {
                throw new DbException("BufferPool is full");
            }
            buffer.put(pid, page);
            return page;
        }
    }

    /**
     * Releases the lock on a page.
     * Calling this is very risky, and may result in wrong behavior. Think hard
     * about who needs to call this and why, and why they can run the risk of
     * calling it.
     *
     * @param tid the ID of the transaction requesting the unlock
     * @param pid the ID of the page to unlock
     */
    public void releasePage(TransactionId tid, PageId pid) {
        // some code goes here
        // not necessary for lab1|lab2
    }

    /**
     * Release all locks associated with a given transaction.
     *
     * @param tid the ID of the transaction requesting the unlock
     */
    public void transactionComplete(TransactionId tid) throws IOException {
        // some code goes here
        // not necessary for lab1|lab2
    }

    /** Return true if the specified transaction has a lock on the specified page
*/
    public boolean holdsLock(TransactionId tid, PageId p) {
        // some code goes here
        // not necessary for lab1|lab2
        return false;
    }
```

```java
    /**
     * Commit or abort a given transaction; release all locks associated to
     * the transaction.
     *
     * @param tid    the ID of the transaction requesting the unlock
     * @param commit a flag indicating whether we should commit or abort
     */
    public void transactionComplete(TransactionId tid, boolean commit)
            throws IOException {
        // some code goes here
        // not necessary for lab1|lab2
    }

    /**
     * Add a tuple to the specified table on behalf of transaction tid. Will
     * acquire a write lock on the page the tuple is added to and any other
     * pages that are updated (Lock acquisition is not needed for lab2).
     * May block if the lock(s) cannot be acquired.
     *
     * Marks any pages that were dirtied by the operation as dirty by calling
     * their markDirty bit, and updates cached versions of any pages that have
     * been dirtied so that future requests see up-to-date pages.
     *
     * @param tid     the transaction adding the tuple
     * @param tableId the table to add the tuple to
     * @param t       the tuple to add
     */
    public void insertTuple(TransactionId tid, int tableId, Tuple t)
            throws DbException, IOException, TransactionAbortedException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Remove the specified tuple from the buffer pool.
     * Will acquire a write lock on the page the tuple is removed from and any
     * other pages that are updated. May block if the lock(s) cannot be acquired.
     *
     * Marks any pages that were dirtied by the operation as dirty by calling
     * their markDirty bit, and updates cached versions of any pages that have
     * been dirtied so that future requests see up-to-date pages.
     *
     * @param tid the transaction deleting the tuple.
     * @param t   the tuple to delete
     */
    public void deleteTuple(TransactionId tid, Tuple t)
            throws DbException, IOException, TransactionAbortedException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Flush all dirty pages to disk.
     * NB: Be careful using this routine -- it writes dirty data to disk so will
     * break simpledb if running in NO STEAL mode.
     */
    public synchronized void flushAllPages() throws IOException {
        // some code goes here
        // not necessary for lab1

    }

    /**
     * Remove the specific page id from the buffer pool.
     * Needed by the recovery manager to ensure that the
     * buffer pool doesn't keep a rolled back page in its
     * cache.
     */
    public synchronized void discardPage(PageId pid) {
        // some code goes here
        // not necessary for labs 1--4
    }

    /**
     * Flushes a certain page to disk
     *
     * @param pid an ID indicating the page to flush
     */
    private synchronized void flushPage(PageId pid) throws IOException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Write all pages of the specified transaction to disk.
     */
    public synchronized void flushPages(TransactionId tid) throws IOException {
        // some code goes here
        // not necessary for labs 1--4
    }
    /**
     * Discards a page from the buffer pool.
     * Flushes the page to disk to ensure dirty pages are updated on disk.
     */
    private synchronized void evictPage() throws DbException {
        // some code goes here
        // not necessary for lab1
    }
}
```

四. 补全 simpleDB 中的 HeapPageId.java、RecordId.java、HeapPage.java。

（1）HeapPageId.java 与 RecordId.java 的实现

两个 Id 类的实现都比较简单，大同小异，这里就放在一起了。

首先是构造器，根据构造器接受的参数可直接定义所需要的属性：

```
    private final int tableid;
    private final int pgNo;

    public HeapPageId(int tableId, int pgNo) {
        // some code goes here
        this.tableid = tableId;
        this.pgNo = pgNo;
    }
```

```
    private final PageId pageid;
    private final int tupleno;

    public RecordId(PageId pid, int tupleno) {
        // some code goes here
        this.pageid = pid;
        this.tupleno = tupleno;
    }
```

然后是一些简单的 get/set 方法，大同小异，也没什么可说的：

```
    public int getTableId() {
        // some code goes here
        return tableid;
    }

    public int pageNumber() {
        // some code goes here
        return pgNo;
    }
```

```
    public int tupleno() {
        // some code goes here
        return tupleno;
    }

    public PageId getPageId() {
        // some code goes here
        return pageid;
    }
```
然后为了能够放入 Map 中，还需要实现 hashCode()方法。这里的实现比较简单，只是单纯地将两个属性组合：
```
    public int hashCode() {
        // some code goes here
        return tableid * 1000 + pgNo;
    }

    public int hashCode() {
        // some code goes here
        return this.pageid.hashCode() + this.tupleno;
    }
```
接着是 equals 方法，这个上面已经提过了，也不赘述：
```
    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof HeapPageId)) {
            return false;
        }
        HeapPageId pid = (HeapPageId) o;
        if (this.tableid == pid.tableid && this.pgNo == pid.pgNo) {
            return true;
        }
        return false;
    }

    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof RecordId)) {
            return false;
        }
        RecordId rid = (RecordId) o;
        if (this.pageid.equals(rid.pageid) && this.tupleno == rid.tupleno) {
            return true;
        }
        return false;
    }
```

（2）HeapPage.java 的实现

根据注释，表文件存储在堆上（HeapFile），而一个 HeapFile 包含若干页面（HeapPage），一个 HeapPage 包含若干 slot，每个 slot 可以容纳一个元组。在本题中仅需实现 HeapPage.java.

尽管 HeapPage 的实现比较复杂，但由于大多数代码已经补全，因此只需补全少量方法。

首先是一些简单的 get/set 方法：

```
    private int getNumTuples() {
        // some code goes here
        return (int) Math.floor((BufferPool.getPageSize() * 8)
                         / (td.getSize() * 8 + 1));
    }
    private int getHeaderSize() {
        // some code goes here
        return (int) Math.ceil(numSlots / 8.0);
    }
    public HeapPageId getId() {
        // some code goes here
        return pid;
    }
    public int getNumEmptySlots() {
        // some code goes here
        int count = 0;
        for (int i = 0; i < tuples.length; i++) {
            if (!isSlotUsed(i)) {
                count++;
            }
        }
        return count;
    }
    public boolean isSlotUsed(int i) {
        // some code goes here
        return (header[i / 8] & (1 << (i % 8))) != 0;
    }
```

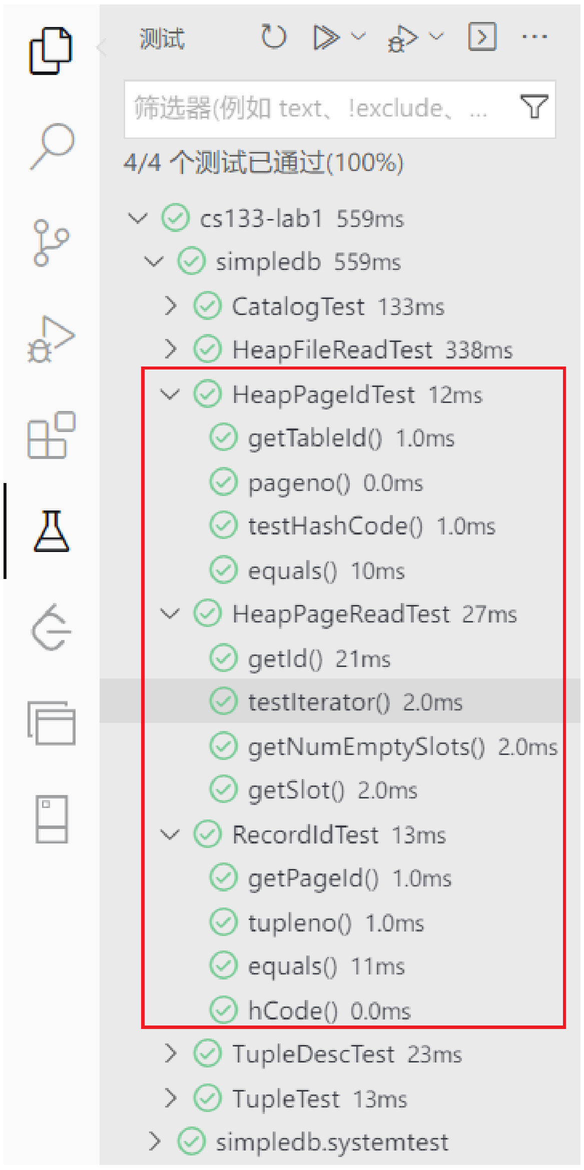
值得注意的是，尽管这些方法代码量很少，但确实需要一番思考。例如 getNumTuples 需要根据 Page 所占的大小除以 TupleDesc 所占大小得到；判断某个 slot 是否被使用需要在 header 数组中精准定

位索引指向的二进制位，由于 header 是一个 Byte 数组，因此每个单元都可以存储八个 slot 的状态，需要较为复杂的判断。

然后是迭代器，这个比较常规，就不多提了。需要注意的仅有判断 hasNext 时还需要判断一下 isSlotUsed.

```
    public Iterator<Tuple> iterator() {
        // some code goes here
        return new HeapPageIterator();
    }

    private class HeapPageIterator implements Iterator<Tuple> {

        private int curr = 0;

        public boolean hasNext() {
            return curr < tuples.length && isSlotUsed(curr);
        }

        public Tuple next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            Tuple t = tuples[curr];
            curr++;
            return t;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
```

（3）单元测试



```
4/4 个测试已通过(100%)
cs133-lab1 559ms
  simpledb 559ms
    CatalogTest 133ms
    HeapFileReadTest 338ms
    HeapPageIdTest 12ms
      getTableId() 1.0ms
      pageno() 0.0ms
      testHashCode() 1.0ms
      equals() 10ms
    HeapPageReadTest 27ms
      getId() 21ms
      testIterator() 2.0ms
      getNumEmptySlots() 2.0ms
      getSlot() 2.0ms
    RecordIdTest 13ms
      getPageId() 1.0ms
      tupleno() 1.0ms
      equals() 11ms
      hCode() 0.0ms
    TupleDescTest 23ms
    TupleTest 13ms
  simpledb.systemtest
```

（4）附：全部代码

```java
// src/java/HeapPageId.java

package simpledb;

/** Unique identifier for HeapPage objects. */
public class HeapPageId implements PageId {

    private final int tableid;
    private final int pgNo;

    /**
     * Constructor. Create a page id structure for a specific page of a
     * specific table.
     *
     * @param tableId The table that is being referenced
     * @param pgNo    The page number in that table.
     */
    public HeapPageId(int tableId, int pgNo) {
        // some code goes here
        this.tableid = tableId;
        this.pgNo = pgNo;
    }

    /** @return the table associated with this PageId */
    public int getTableId() {
        // some code goes here
        return tableid;
    }

    /**
     * @return the page number in the table getTableId() associated with
     *         this PageId
     */
    public int pageNumber() {
        // some code goes here
        return pgNo;
    }

    /**
     * @return a hash code for this page, represented by the concatenation of
     *         the table number and the page number (needed if a PageId is used as
a
     *         key in a hash table in the BufferPool, for example.)
     * @see BufferPool
     */
    public int hashCode() {
        // some code goes here
        return tableid * 1000 + pgNo;
    }

    /**
     * Compares one PageId to another.
     *
     * @param o The object to compare against (must be a PageId)
     * @return true if the objects are equal (e.g., page numbers and table
     *         ids are the same)
     */
    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof HeapPageId)) {
            return false;
        }
        HeapPageId pid = (HeapPageId) o;
        if (this.tableid == pid.tableid && this.pgNo == pid.pgNo) {
            return true;
        }
        return false;
    }

    /**
     * Return a representation of this object as an array of
     * integers, for writing to disk. Size of returned array must contain
     * number of integers that corresponds to number of args to one of the
     * constructors.
     */
    public int[] serialize() {
        int data[] = new int[2];

        data[0] = getTableId();
        data[1] = pageNumber();

        return data;
    }
}
```

```
// src/java/RecordId.java

package simpledb;

import java.io.Serializable;

/**
 * A RecordId is a reference to a specific tuple on a specific page of a
 * specific table.
 */
public class RecordId implements Serializable {

    private static final long serialVersionUID = 1L;

```

```java
    private final PageId pageid;
    private final int tupleno;

    /**
     * Creates a new RecordId referring to the specified PageId and tuple
     * number.
     *
     * @param pid
     *              the pageid of the page on which the tuple resides
     * @param tupleno
     *              the tuple number within the page.
     */
    public RecordId(PageId pid, int tupleno) {
        // some code goes here
        this.pageid = pid;
        this.tupleno = tupleno;
    }

    /**
     * @return the tuple number this RecordId references.
     */
    public int tupleno() {
        // some code goes here
        return tupleno;
    }

    /**
     * @return the page id this RecordId references.
     */
    public PageId getPageId() {
        // some code goes here
        return pageid;
    }

    /**
     * Two RecordId objects are considered equal if they represent the same
     * tuple.
     *
     * @return True if this and o represent the same tuple
     */
    @Override
    public boolean equals(Object o) {
        // some code goes here
        if (o == null) {
            return false;
        }
        if (o == this) {
            return true;
        }
        if (!(o instanceof RecordId)) {
            return false;
        }
        RecordId rid = (RecordId) o;
        if (this.pageid.equals(rid.pageid) && this.tupleno == rid.tupleno) {
            return true;
        }
        return false;
    }

    /**
     * You should implement the hashCode() so that two equal RecordId instances
     * (with respect to equals()) have the same hashCode().
     *
     * @return An int that is the same for equal RecordId objects.
     */
    @Override
    public int hashCode() {
        // some code goes here
        return this.pageid.hashCode() + this.tupleno;
    }
}
```

```
// src/java/HeapPage.java

package simpledb;

import java.util.*;
import java.io.*;

/**
 * Each instance of HeapPage stores data for one page of HeapFiles and
 * implements the Page interface that is used by BufferPool.
 *
 * @see HeapFile
 * @see BufferPool
 *
 */
public class HeapPage implements Page {

    final HeapPageId pid;
```

```java
    final TupleDesc td;
    final byte header[];
    final Tuple tuples[];
    final int numSlots;

    byte[] oldData;
    private final Byte oldDataLock = new Byte((byte) 0);

    /**
     * Create a HeapPage from a set of bytes of data read from disk.
     * The format of a HeapPage is a set of header bytes indicating
     * the slots of the page that are in use, some number of tuple slots.
     * Specifically, the number of tuples is equal to:
     * <p>
     * floor((BufferPool.getPageSize()*8) / (tuple size * 8 + 1))
     * <p>
     * where tuple size is the size of tuples in this
     * database table, which can be determined via {@link Catalog#getTupleDesc}.
     * The number of 8-bit header words is equal to:
     * <p>
     * ceiling(no. tuple slots / 8)
     * <p>
     *
     * @see Database#getCatalog
     * @see Catalog#getTupleDesc
     * @see BufferPool#getPageSize()
     */
    public HeapPage(HeapPageId id, byte[] data) throws IOException {
        this.pid = id;
        this.td = Database.getCatalog().getTupleDesc(id.getTableId());
        this.numSlots = getNumTuples();
        DataInputStream dis = new DataInputStream(new ByteArrayInputStream(data));

        // allocate and read the header slots of this page
        header = new byte[getHeaderSize()];
        for (int i = 0; i < header.length; i++)
            header[i] = dis.readByte();

        tuples = new Tuple[numSlots];
        try {
            // allocate and read the actual records of this page
            for (int i = 0; i < tuples.length; i++)
                tuples[i] = readNextTuple(dis, i);
        } catch (NoSuchElementException e) {
            e.printStackTrace();
        }
        dis.close();

        setBeforeImage();
    }

    /**
     * Retrieve the number of tuples on this page.
     *
     * @return the number of tuples on this page
     */
    private int getNumTuples() {
        // some code goes here
        return (int) Math.floor((BufferPool.getPageSize() * 8) / (td.getSize() * 8
+ 1));
    }

    /**
     * Computes the number of bytes in the header of a page in a HeapFile with each
     * tuple occupying tupleSize bytes
     *
     * @return the number of bytes in the header of a page in a HeapFile with each
     *         tuple occupying tupleSize bytes
     */
    private int getHeaderSize() {
        // some code goes here
        return (int) Math.ceil(numSlots / 8.0);
    }

    /**
     * Return a view of this page before it was modified
     * -- used by recovery
     */
    public HeapPage getBeforeImage() {
        try {
            byte[] oldDataRef = null;
            synchronized (oldDataLock) {
                oldDataRef = oldData;
            }
            return new HeapPage(pid, oldDataRef);
        } catch (IOException e) {
            e.printStackTrace();
            // should never happen -- we parsed it OK before!
            System.exit(1);
        }
        return null;
    }

    public void setBeforeImage() {
        synchronized (oldDataLock) {
            oldData = getPageData().clone();
        }
    }

    /**
     * @return the PageId associated with this page.
     */
    public HeapPageId getId() {
        // some code goes here
        return pid;
    }

    /**
     * Suck up tuples from the source file.
     */
    private Tuple readNextTuple(DataInputStream dis, int slotId) throws
NoSuchElementException {
        // if associated bit is not set, read forward to the next tuple, and
        // return null.
        if (!isSlotUsed(slotId)) {
            for (int i = 0; i < td.getSize(); i++) {
                try {
                    dis.readByte();
                } catch (IOException e) {
                    throw new NoSuchElementException("error reading empty tuple");
                }
            }
            return null;
        }

        // read fields in the tuple
        Tuple t = new Tuple(td);
        RecordId rid = new RecordId(pid, slotId);
        t.setRecordId(rid);
        try {
            for (int j = 0; j < td.numFields(); j++) {
                Field f = td.getFieldType(j).parse(dis);
                t.setField(j, f);
            }
        } catch (java.text.ParseException e) {
            e.printStackTrace();
            throw new NoSuchElementException("parsing error!");
        }

        return t;
    }

    /**
     * Generates a byte array representing the contents of this page.
     * Used to serialize this page to disk.
     * <p>
     * The invariant here is that it should be possible to pass the byte
     * array generated by getPageData to the HeapPage constructor and
     * have it produce an identical HeapPage object.
     *
     * @see #HeapPage
     * @return A byte array correspond to the bytes of this page.
     */
    public byte[] getPageData() {
        int len = BufferPool.getPageSize();
        ByteArrayOutputStream baos = new ByteArrayOutputStream(len);
        DataOutputStream dos = new DataOutputStream(baos);

        // create the header of the page
        for (int i = 0; i < header.length; i++) {
            try {
                dos.writeByte(header[i]);
            } catch (IOException e) {
                // this really shouldn't happen
                e.printStackTrace();
            }
        }

        // create the tuples
        for (int i = 0; i < tuples.length; i++) {

            // empty slot
            if (!isSlotUsed(i)) {
                for (int j = 0; j < td.getSize(); j++) {
                    try {
                        dos.writeByte(0);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                continue;
            }

            // non-empty slot
            for (int j = 0; j < td.numFields(); j++) {
                Field f = tuples[i].getField(j);
                try {
                    f.serialize(dos);

                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        // padding
        int zerolen = BufferPool.getPageSize() - (header.length + td.getSize() *
tuples.length); // - numSlots *

              // td.getSize();
        byte[] zeroes = new byte[zerolen];
        try {
            dos.write(zeroes, 0, zerolen);
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            dos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return baos.toByteArray();
    }

    /**
     * Static method to generate a byte array corresponding to an empty
     * HeapPage.
     * Used to add new, empty pages to the file. Passing the results of
     * this method to the HeapPage constructor will create a HeapPage with
     * no valid tuples in it.
     *
     * @return The returned ByteArray.
     */
    public static byte[] createEmptyPageData() {
        int len = BufferPool.getPageSize();
        return new byte[len]; // all 0
    }

    /**
     * Delete the specified tuple from the page; the tuple should be updated to
     * reflect
     * that it is no longer stored on any page.
     *
     * @throws DbException if this tuple is not on this page, or tuple slot is
     *                     already empty.
     * @param t The tuple to delete
     */
    public void deleteTuple(Tuple t) throws DbException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Adds the specified tuple to the page; the tuple should be updated to reflect
     * that it is now stored on this page.
     *
     * @throws DbException if the page is full (no empty slots) or tupledesc
     *                     is mismatch.
     * @param t The tuple to add.
     */
    public void insertTuple(Tuple t) throws DbException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Marks this page as dirty/not dirty and record that transaction
     * that did the dirtying
     */
    public void markDirty(boolean dirty, TransactionId tid) {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Returns the tid of the transaction that last dirtied this page, or null if
     * the page is not dirty
     */
    public TransactionId isDirty() {
        // some code goes here
        // Not necessary for lab1
        return null;
    }

    /**
     * Returns the number of empty slots on this page.
     */
    public int getNumEmptySlots() {
        // some code goes here
        int count = 0;
        for (int i = 0; i < tuples.length; i++) {
            if (!isSlotUsed(i)) {
                count++;
            }
        }
        return count;
    }

    /**
     * Returns true if associated slot on this page is filled.
     */
    public boolean isSlotUsed(int i) {
        // some code goes here
        return (header[i / 8] & (1 << (i % 8))) != 0;
    }

    /**
     * Abstraction to fill or clear a slot on this page.
     */
    private void markSlotUsed(int i, boolean value) {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * @return an iterator over all tuples on this page (calling remove on this
     *         iterator throws an UnsupportedOperationException)
     *         (note that this iterator shouldn't return tuples in empty slots!)
     */
    public Iterator<Tuple> iterator() {
        // some code goes here
        return new HeapPageIterator();
    }

    private class HeapPageIterator implements Iterator<Tuple> {

        private int curr = 0;

        public boolean hasNext() {
            return curr < tuples.length && isSlotUsed(curr);
        }

        public Tuple next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            Tuple t = tuples[curr];
            curr++;
            return t;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

五、附录

尽管只要求做到 Exercise 4，但实际上全部代码都已经完成，也通过了全部单元测试。这里仅给出这些代码。

（1）Exercise 5

```
// src/java/HeapFile.java

package simpledb;

import java.io.*;
import java.security.Permissions;
import java.util.*;

/**
 * HeapFile is an implementation of a DbFile that stores a collection of tuples
 * in no particular order. Tuples are stored on pages, each of which is a fixed
 * size, and the file is simply a collection of those pages. HeapFile works
 * closely with HeapPage. The format of HeapPages is described in the HeapPage
 * constructor.
 *
 * @see simpledb.HeapPage#HeapPage
 * @author Sam Madden
 */
public class HeapFile implements DbFile {

    private final File file;
    private final TupleDesc tuDesc;

    /**
     * Constructs a heap file backed by the specified file.
     *
     * @param f
     *          the file that stores the on-disk backing store for this heap
     *          file.
     */
    public HeapFile(File f, TupleDesc td) {
        // some code goes here
        file = f;
        tuDesc = td;
    }

    /**
     * Returns the File backing this HeapFile on disk.
     *
     * @return the File backing this HeapFile on disk.
     */
    public File getFile() {
        // some code goes here
        return file;
    }

    /**
     * Returns an ID uniquely identifying this HeapFile. Implementation note:
     * you will need to generate this tableid somewhere ensure that each
     * HeapFile has a "unique id," and that you always return the same value for
     * a particular HeapFile. We suggest hashing the absolute file name of the
     * file underlying the heapfile, i.e. f.getAbsoluteFile().hashCode().
     *
     * @return an ID uniquely identifying this HeapFile.
     */
    public int getId() {
        // some code goes here
        return file.getAbsoluteFile().hashCode();
    }
    /**
     * Returns the TupleDesc of the table stored in this DbFile.
     *
     * @return TupleDesc of this DbFile.
     */
    public TupleDesc getTupleDesc() {
        // some code goes here
        return tuDesc;
    }

    // see DbFile.java for javadocs
    public Page readPage(PageId pid) {
        // some code goes here
        try {
            RandomAccessFile raf = new RandomAccessFile(file, "r");
            raf.seek(pid.pageNumber() * BufferPool.PAGE_SIZE);
            byte[] data = new byte[BufferPool.PAGE_SIZE];
            raf.read(data);
            raf.close();
            return new HeapPage(new HeapPageId(pid.getTableId(), pid.pageNumber()),
data);
        } catch (IOException e) {
            throw new IllegalArgumentException("Page does not exist");
        }
    }

    // see DbFile.java for javadocs
    public void writePage(Page page) throws IOException {
        // some code goes here
        // not necessary for lab1
    }

    /**
     * Returns the number of pages in this HeapFile.
     */
    public int numPages() {
        // some code goes here
        return (int) (file.length() / BufferPool.PAGE_SIZE);
    }

    // see DbFile.java for javadocs
    public ArrayList<Page> insertTuple(TransactionId tid, Tuple t)
            throws DbException, IOException, TransactionAbortedException {
        // some code goes here
        return null;
        // not necessary for lab1
    }

    // see DbFile.java for javadocs
    public ArrayList<Page> deleteTuple(TransactionId tid, Tuple t) throws
DbException,
            TransactionAbortedException {
        // some code goes here
        return null;
        // not necessary for lab1
    }

    // see DbFile.java for javadocs
    public DbFileIterator iterator(TransactionId tid) {
        // some code goes here
        return new HeapFileIterator(tid);
    }

    private class HeapFileIterator implements DbFileIterator {

        private final TransactionId tid;
        private int curPage = 0;
        private HeapPage curPageData = null;
        private Iterator<Tuple> curPageIterator = null;

        public HeapFileIterator(TransactionId tid) {
            // some code goes here
            this.tid = tid;
        }

        @Override
        public void open() throws DbException, TransactionAbortedException,
IOException {
            curPage = 0;
            curPageData = (HeapPage) Database.getBufferPool().getPage(
                    tid,
                    new HeapPageId(getId(), curPage),
                    simpledb.Permissions.READ_WRITE);
            curPageIterator = curPageData.iterator();
        }

        @Override
        public boolean hasNext() throws DbException, TransactionAbortedException,
IOException {
            if (curPageIterator == null) {
                return false;
            }
            if (curPageIterator.hasNext()) {
                return true;
            } else {
                if (curPage < numPages() - 1) {
                    curPage++;
                    curPageData = (HeapPage) Database.getBufferPool().getPage(
                            tid,
                            new HeapPageId(getId(), curPage),
                            simpledb.Permissions.READ_WRITE);
                    curPageIterator = curPageData.iterator();
                    return hasNext();
                } else {
                    return false;
                }
            }
        }

        @Override
        public Tuple next() throws DbException, TransactionAbortedException {
            if (curPageIterator == null) {
                throw new NoSuchElementException("Iterator is not open");
            }
            return curPageIterator.next();
        }

        @Override
        public void rewind() throws DbException, TransactionAbortedException,
IOException {
            curPage = 0;
            curPageData = (HeapPage) Database.getBufferPool().getPage(
                    tid,
                    new HeapPageId(getId(), curPage),
                    simpledb.Permissions.READ_WRITE);
            curPageIterator = curPageData.iterator();
        }

        @Override
        public void close() {
            curPageData = null;
            curPageIterator = null;
        }
    }
}
```

（2）Exercise 6

```java
// src/java/SeqScan.java

package simpledb;

import java.io.IOException;
import java.util.*;

/**
 * SeqScan is an implementation of a sequential scan access method that reads
 * each tuple of a table in no particular order (e.g., as they are laid out on
 * disk).
 */
public class SeqScan implements DbIterator {

    private static final long serialVersionUID = 1L;

    private final TransactionId tid;
    private int tableid;
    private String tableAlias;
    private DbFileIterator fileIter;

    /**
     * Creates a sequential scan over the specified table as a part of the
     * specified transaction.
     *
     * @param tid
     *                   The transaction this scan is running as a part of.
     * @param tableid
     *                   the table to scan.
     * @param tableAlias
     *                   the alias of this table (needed by the parser); the
     *                   returned
     *                   tupleDesc should have fields with name tableAlias.fieldName
     *                   (note: this class is not responsible for handling a case
     *                   where
     *                   tableAlias or fieldName are null. It shouldn't crash if
     *                   they
     *                   are, but the resulting name can be null.fieldName,
     *                   tableAlias.null, or null.null).
     */
    public SeqScan(TransactionId tid, int tableid, String tableAlias) {
        // some code goes here
        this.tid = tid;
        this.tableid = tableid;
        this.tableAlias = tableAlias;
    }

    /**
     * @return
     *         return the table name of the table the operator scans. This should
     *         be the actual name of the table in the catalog of the database
     */
    public String getTableName() {
        // some code goes here
        return Database.getCatalog().getTableName(tableid);
    }

    /**
     * @return Return the alias of the table this operator scans.
     */
    public String getAlias() {
        // some code goes here
        return tableAlias;
    }

    /**
     * Reset the tableid, and tableAlias of this operator.
     *
     * @param tableid
     *                   the table to scan.
     * @param tableAlias
     *                   the alias of this table (needed by the parser); the
     *                   returned
     *                   tupleDesc should have fields with name tableAlias.fieldName
     *                   (note: this class is not responsible for handling a case
     *                   where
     *                   tableAlias or fieldName are null. It shouldn't crash if
     *                   they
     *                   are, but the resulting name can be null.fieldName,
     *                   tableAlias.null, or null.null).
     */
    public void reset(int tableid, String tableAlias) {
        // some code goes here
        this.tableid = tableid;
        this.tableAlias = tableAlias;
    }

    public SeqScan(TransactionId tid, int tableid) {
```

```java
        this(tid, tableid, Database.getCatalog().getTableName(tableid));
    }

    public void open() throws DbException, TransactionAbortedException, IOException
{
        // some code goes here
        fileIter = Database.getCatalog().getDatabaseFile(tableid).iterator(tid);
        fileIter.open();
    }

    /**
     * Returns the TupleDesc with field names from the underlying HeapFile,
     * prefixed with the tableAlias string from the constructor. This prefix
     * becomes useful when joining tables containing a field(s) with the same
     * name.
     *
     * @return the TupleDesc with field names from the underlying HeapFile,
     *         prefixed with the tableAlias string from the constructor.
     */
    public TupleDesc getTupleDesc() {
        // some code goes here
        TupleDesc tupleDesc = Database.getCatalog().getTupleDesc(tableid);
        Type[] fieldTypes = new Type[tupleDesc.numFields()];
        String[] fieldNames = new String[tupleDesc.numFields()];
        for (int i = 0; i < tupleDesc.numFields(); i++) {
            fieldTypes[i] = tupleDesc.getFieldType(i);
            fieldNames[i] = tableAlias + "." + tupleDesc.getFieldName(i);
        }
        return new TupleDesc(fieldTypes, fieldNames);
    }

    public boolean hasNext() throws TransactionAbortedException, DbException,
IOException {
        // some code goes here
        return fileIter.hasNext();
    }

    public Tuple next() throws NoSuchElementException,
            TransactionAbortedException, DbException {
        // some code goes here
        return fileIter.next();
    }

    public void close() {
        // some code goes here
        fileIter.close();
    }

    public void rewind() throws DbException, NoSuchElementException,
            TransactionAbortedException, IOException {
        // some code goes here
        fileIter.rewind();
    }
}
```