苏州大学实验报告

院、系	计算机学院	年级专业	19 计科图灵	姓名	张昊	学号	1927405160
课程名称	数据库课程实践					成绩	90
指导教师	赵朋朋	同组实验者	无		实验日期	2021 年 6 月 25 日	

实 验 名 称　SimpleDB 实验

一、实验内容

实现 SimpleDB 对数据的管理。

二、实验过程与结果

Exercise 1. Fields and Tuples

（1）TupleDesc.java

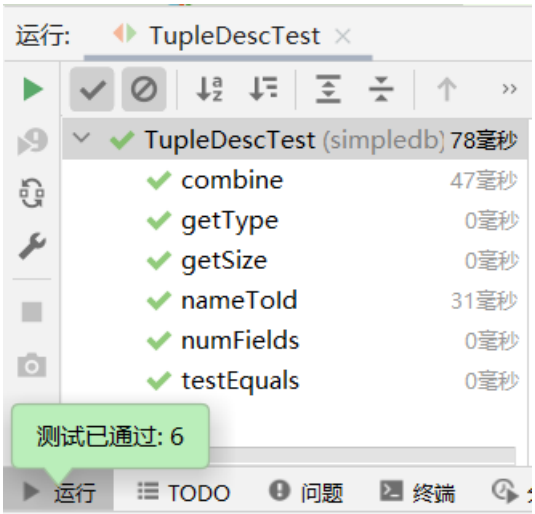
```
public class TupleDesc implements Serializable {
    /**
     * A help class to facilitate organizing the information of each field
     */
    public static class TDItem implements Serializable {
        private static final long serialVersionUID = 1L;
        /**
         * The type of the field
         */
        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 + ")";
        }
    }
    private final TDItem[] items;  // 特定类型的字段
    // 返回字段的迭代器
    public Iterator<TDItem> iterator() {
        // 使用 Java8 流来获取各字段的迭代器
        return Arrays.stream(this.items).iterator();
    }
    // 构造函数，使用字段类型数组+字段名数组初始化
    public TupleDesc(Type[] typeAr, String[] fieldAr) {
        // some code goes here
        // 判断参数是否合法
```

```
        if (typeAr.length <= 0) {
            throw new IllegalArgumentException("typeAr 的长度应大于 0");
        }
        if (typeAr.length != fieldAr.length) {
            throw new IllegalArgumentException("typeAr 和 fieldAr 大小应相同");
        }
        // 构造各字段并初始化
        this.items = new TDItem[typeAr.length];
        for (int i = 0; i < typeAr.length; ++i) {
            this.items[i] = new TDItem(typeAr[i], fieldAr[i]);
        }
    }
    // 构造函数，使用字段类型数组初始化
    public TupleDesc(Type[] typeAr) {
        // some code goes here
        this(typeAr, new String[typeAr.length]);
    }
    // 返回字段数量
    public int numFields() {
        // some code goes here
        return this.items.length;
    }
    // 获取指定索引的字段名，不存在抛出 NoSuchElementException
    public String getFieldName(int i) throws NoSuchElementException {
        // some code goes here
        this.checkIndex(i); // 检查是否存在，若不存在抛异常
        return this.items[i].fieldName;
    }
    // 检查索引值 i 是否合法
    private void checkIndex(int i) throws NoSuchElementException {
        if (i < 0 || i >= this.items.length) {
         throw new NoSuchElementException(String.format("索引值 %d 不合法",i));
        }
    }
   // 获取指定索引的字段类型，不存在抛出 NoSuchElementException
    public Type getFieldType(int i) throws NoSuchElementException {
        // some code goes here
        this.checkIndex(i);
        return this.items[i].fieldType;
    }
    // 根据字段名找其索引，不存在抛出 NoSuchElementException
    public int fieldNameToIndex(String name) throws NoSuchElementException {
        // some code goes here
        for (int index = 0; index < this.items.length; ++index) {
            String fieldName = this.items[index].fieldName;
            if (fieldName != null && fieldName.equals(name)) {
```

```
            // 注意：fieldName 可能为空
            return index;
        }
    }
    throw new NoSuchElementException("字段'" + name + "'未找到");
}
// 返回 TupleDesc 所占字节数
public int getSizeInBytes() {
    // some code goes here
    int total = 0;
    for (TDItem item : this.items) {
        total += item.fieldType.getSizeInBytes();
    }
    return total;
}
// 合并两个 TupleDesc td1+td2
public static TupleDesc merge(TupleDesc td1, TupleDesc td2) {
    // some code goes here
    int length1 = td1.items.length;
    int length2 = td2.items.length;
    int length = length1 + length2;
    Type[] types = new Type[length];
    String[] names = new String[length];
    for (int i = 0; i < length1; ++i) {
        types[i] = td1.items[i].fieldType;
        names[i] = td1.items[i].fieldName;
    }
    for (int i = 0; i < length2; ++i) {
        types[length1 + i] = td2.items[i].fieldType;
        names[length1 + i] = td2.items[i].fieldName;
    }
    return new TupleDesc(types, names);
}
// 判断是否相等
public boolean equals(Object o) {
    // some code goes here
    if (this == o) {  // 同一对象
        return true;
    }
    if (o instanceof TupleDesc) {  // 类型相同才比较
        // 根据文档注释：Two TupleDescs are considered equal if they have
        // the same number of items and the i-th type in this TupleDesc
        // is equal to the i-th type in o for every i.
        // 即相等的条件是：两者字段数相同，对应位置上的字段类型相同
        TupleDesc other = (TupleDesc) o;
        if (this.items.length != other.items.length) {
```

```
                return false;
            }
            for (int i = 0; i < this.items.length; ++i) {
              if(!(this.items[i].fieldType.equals(other.items[i].fieldType))){
                  return false;
              }
            }
            return true;
        }
        return false;
    }
    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");
    }
    public String toString() {  // 格式化为字符串
        // some code goes here
        StringBuilder str = new StringBuilder("TupleDesc: ");
        for (int i = 0; i < this.items.length; i++) {
            str.append(this.items[i]);
            if (this.items.length - 1 != i) {
                str.append(", ");
            }
        }
        return str.toString();
    }
}
```

TupleDescTest 单元测试：



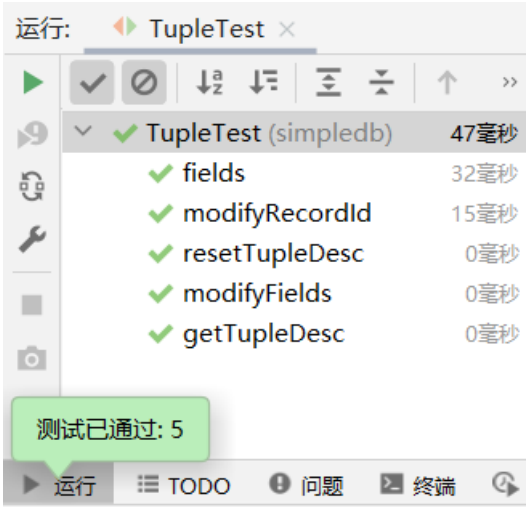
（2）Tuple.java

```
public class Tuple implements Serializable {
    private static final long serialVersionUID = 1L;
    private TupleDesc desc;   // 关系模式
```

```
    private final Field[] fields;  // 元组字段数组
    private RecordId id;    // 该元组的记录编号信息
    // 构造函数，使用模式（TupleDesc）创建
    public Tuple(TupleDesc td) {
        // some code goes here
        this.desc = td;
        this.fields = new Field[td.numFields()];    // 生成对应长度的字段数组
        this.id = null;
    }
    public TupleDesc getTupleDesc() {  // 获取模式
        // some code goes here
        return this.desc;
    }
    public RecordId getRecordId() {  // 获取记录编号信息
        // some code goes here
        return this.id;
    }
    public void setRecordId(RecordId rid) {  // 设置获取记录编号信息
        // some code goes here
        this.id = rid;
    }
    // 通过索引插入字段，索引不存在报错
    public void setField(int i, Field f) {
        // some code goes here
        if (i < 0 || i >= this.fields.length) {
            throw new NoSuchElementException(i + " is not a valid index.");
        }
        this.fields[i] = f;
    }
    // 通过索引获取字段，索引不存在报错
    public Field getField(int i) {
        // some code goes here
        if (i < 0 || i >= this.fields.length) {
            throw new NoSuchElementException(i + " is not a valid index.");
        }
        return this.fields[i];
    }
    public String toString() {  // 格式化为字符串
        // some code goes here
        // 各行格式：column1\tcolumn2\tcolumn3\t...\tcolumnN\n
        StringBuilder str = new StringBuilder();
        for (int i = 0; i < this.fields.length; i++) {
            str.append(fields[i]).append((this.fields.length-1==i)?'\n':'\t');
        }
        return str.toString();
    }
```

```
    // 获取字段迭代器
    public Iterator<Field> fields() {
        // some code goes here
        return Arrays.stream(this.fields).iterator();  // Java8 stream
    }
    // 重置元组模式，仅影响模式，不影响字段
    public void resetTupleDesc(TupleDesc td) {
        // some code goes here
        this.desc = td;
    }
}
```

TupleTest 单元测试：



Exercise 2. Catalog

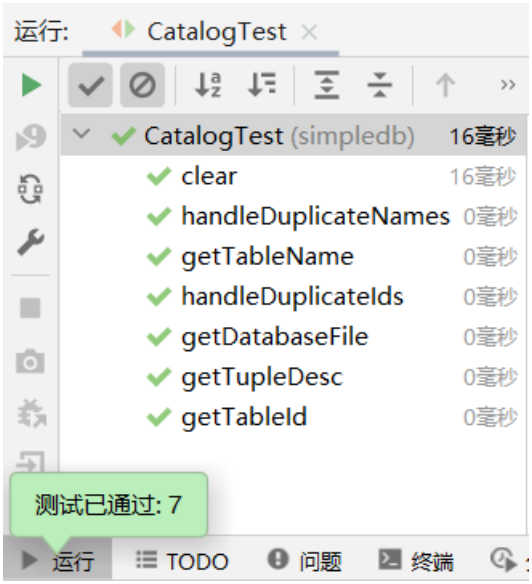
Catalog.java

```
public class Catalog {
    // 为目录建立映射
    private final Map<Integer, DbFile> idFileMap;// map from 表 id 到 DB 文件的映射
    private final Map<Integer, String> idNameMap;// map from 表 id 到表名的映射
    private final Map<Integer, String> idPKeyMap;// map from 表 id 到主键的映射
    private final Map<String, Integer> nameIdMap;// map from 表名到表 id 的映射
    public Catalog() {    // 创建一个新的空目录
        // some code goes here
        this.idFileMap = new HashMap<>();
        this.idNameMap = new HashMap<>();
        this.idPKeyMap = new HashMap<>();
        this.nameIdMap = new HashMap<>();
    }
    //向目录中添加一张新的表，表的详情存储在 DbFile 中，指定表的名称和主键
    public void addTable(DbFile file, String name, String pkeyField) {
        // some code goes here
        // 空指针检查
        if (file == null || name == null || pkeyField == null) {
```

```
            throw new IllegalArgumentException("invalid argument.");
        }
        if (this.nameIdMap.containsKey(name)) {  // 表名已经存在，删除
            int existedId = this.nameIdMap.get(name);
            this.idFileMap.remove(existedId);
            this.idNameMap.remove(existedId);
            this.idPKeyMap.remove(existedId);
            this.nameIdMap.remove(name);
        }
        int tableId = file.getId();  // 获取表 id 并插入
        this.idFileMap.put(tableId, file);
        this.idNameMap.put(tableId, name);
        this.idPKeyMap.put(tableId, pkeyField);
        this.nameIdMap.put(name, tableId);
    }
    public void addTable(DbFile file, String name) {
        addTable(file, name, "");
    }
    public void addTable(DbFile file) {
        addTable(file, (UUID.randomUUID()).toString());
    }
    // 根据表名找表 id，不存在报错
    public int getTableId(String name) throws NoSuchElementException {
        // some code goes here
        if (name == null || !this.nameIdMap.containsKey(name)) {
            throw new NoSuchElementException("表名'" + name + "'未找到");
        }
        return this.nameIdMap.get(name);
    }
    // 根据表 id 找模式，不存在报错
public TupleDesc getTupleDesc(int tableid) throws NoSuchElementException{
        // some code goes here
        if (!this.idFileMap.containsKey(tableid)) {
            throw new NoSuchElementException("表 id=" + tableid + "未找到");
        }
        return this.idFileMap.get(tableid).getTupleDesc();
    }
    // 根据表 id 找 DB 文件，不存在报错
 public DbFile getDatabaseFile(int tableid) throws NoSuchElementException{
        // some code goes here
        if (!this.idFileMap.containsKey(tableid)) {
            throw new NoSuchElementException("表 id=" + tableid + "未找到");
        }
        return this.idFileMap.get(tableid);
    }
    // 根据表 id 找主键，不存在报错
```

```
    public String getPrimaryKey(int tableid) {
        // some code goes here
        if (!this.idPKeyMap.containsKey(tableid)) {
            throw new NoSuchElementException("表 id=" + tableid + "未找到");
        }
        return this.idPKeyMap.get(tableid);
    }
    // 表 id 迭代器，使用 HashMap 中 ketSet 的迭代器实现
    public Iterator<Integer> tableIdIterator() {
        // some code goes here
        return this.idNameMap.keySet().iterator();
    }
    // 根据表 id 找表名，不存在报错
    public String getTableName(int tableid) {
        // some code goes here
        if (!this.idNameMap.containsKey(tableid)) {
            throw new NoSuchElementException("表 id=" + tableid + "未找到");
        }
        return this.idNameMap.get(tableid);
    }
    public void clear() {  // 清空目录，即清空四个映射
        // some code goes here
        this.idNameMap.clear();
        this.idFileMap.clear();
        this.idPKeyMap.clear();
        this.nameIdMap.clear();
    }
    //  loadSchema 方法省略
}
```

CatalogTest 单元测试：



Exercise 3. BufferPool

BufferPool.java 中 getPage()方法（已省略其他不必要的方法和属性）

```
public class BufferPool {
    // 页面缓冲池，至多 numPages 个页面（numPages 由构造函数提供）
    private final Map<PageId, Page> pageCache;
    // 缓冲池中最大页面数量
    private final int PAGES_MAX_NUM;
    /**
     * 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
        this.PAGES_MAX_NUM = numPages;
        this.pageCache = new HashMap<>(numPages);
    }
    // 检索指定页，提供事务 id，页面 id 和读写权限 perm
    // 在 Lab1 中只关心请求的页面 id：pid，其他两个参数暂时用不到
    public  Page getPage(TransactionId tid, PageId pid, Permissions perm)
        throws TransactionAbortedException, DbException {
        // some code goes here
        // 首先检索缓冲池中是否已经存在该页面，若存在（命中）直接返回
        if (this.pageCache.containsKey(pid)) {
            return this.pageCache.get(pid);
        }
        // 若没有命中，需要根据请求的 pid 寻找该页面
        // 首先根据 pid 到目录中获取 DB 文件（表）
        DbFile table=Database.getCatalog().getDatabaseFile(pid.getTableId());
        // 之后从 DB 文件（表）中读取相应的页面
        Page newPage = table.readPage(pid);
        this.addNewPage(pid, newPage);  // 最后将页面添加到缓冲池
        return newPage;
    }
    // 添加页面到缓冲池，如果缓冲池已满需要使用替换算法替换
    // Lab1 中不要求实现替换策略，只需缓冲池已满时抛出 DbException
    private void addNewPage(PageId pid, Page newPage) throws DbException {
        if (this.pageCache.size() < this.PAGES_MAX_NUM) { // 缓冲池空间充足
            this.pageCache.put(pid, newPage);
        } else {  // TODO: 2021-05-21  实现替换策略（当前为抛出异常）
            throw new DbException("缓冲池空间不足");
        }
    }
}
```

无对应单元测试。

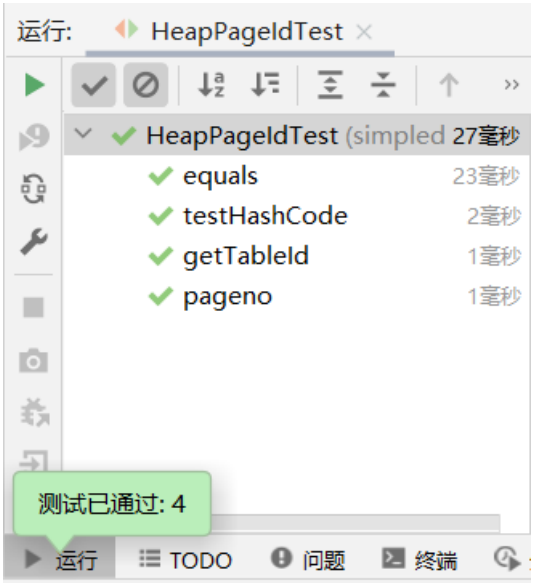
Exercise 4. HeapFile access method (1)

（1）HeapPageId.java

```
/** Unique identifier for HeapPage objects. */
public class HeapPageId implements PageId {
    private final int tableId;  // 特定表
    private final int pageNumber;  // 特定页
    // 构造方法，为特定表的特定页创建一个堆页面 id。根据传入的表 id 和页面号构造
    public HeapPageId(int tableId, int pgNo) {
        // some code goes here
        this.tableId = tableId;
        this.pageNumber = pgNo;
    }
    public int getTableId() {  // tableId Getter
        // some code goes here
        return this.tableId;
    }
    public int getPageNumber() {  // pageNumber Getter
        // some code goes here
        return this.pageNumber;
    }
    // 页面的哈希代码，由表号和页号的连接表示
    public int hashCode() {
        // some code goes here
        // 使用质数 31*表号+页号实现
        return 31 * this.tableId + this.pageNumber;
    }
    public boolean equals(Object o) {  // 判断相等
        // some code goes here
        if (this == o) {  // 同一对象
            return true;
        } else if (o instanceof HeapPageId) {
            // 相同类型，表号和页号都相同才相等
            HeapPageId other = (HeapPageId) o;
            return this.pageNumber == other.pageNumber &&
                   this.tableId == other.tableId;
        }
        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] = getPageNumber();
        return data;
    }
}
```

HeapPageIdTest 单元测试：



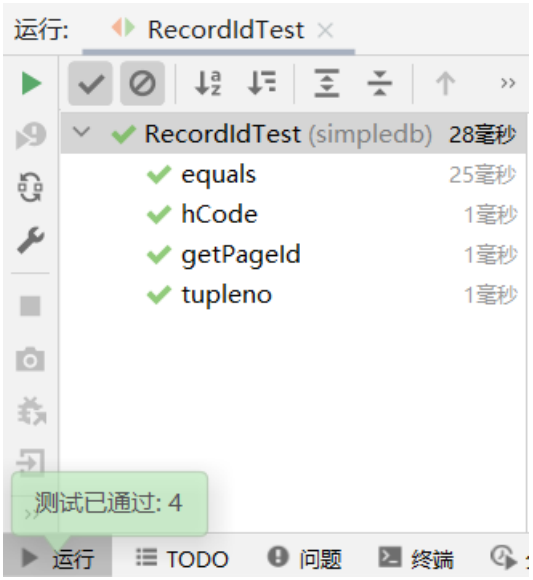
（2）RecordID.java

```
public class RecordId implements Serializable {
    private static final long serialVersionUID = 1L;
    private final PageId pageId;  // 特定页
    private final int tupleNumber;  // 特定元组
    // 构造方法，为特定页的特定元组创建一个记录 id。根据传入的页号和元组号构造
    public RecordId(PageId pid, int tupleno) {
        // some code goes here
        this.pageId = pid;
        this.tupleNumber = tupleno;
    }
    public int getTupleNumber() {  // tupleNumber Getter
        // some code goes here
        return this.tupleNumber;
    }
    public PageId getPageId() {  // pageId Getter
        // some code goes here
        return this.pageId;
    }
    @Override
    public boolean equals(Object o) {  // 判断相等
        // some code goes here
        if (this == o) {  // 同一对象
            return true;
        } else if (o instanceof RecordId) {
```

```
            // 相同类型，页号和元组号都相同才相等
            RecordId other = (RecordId) o;
            return this.pageId.equals(other.pageId) &&
                    this.tupleNumber == other.tupleNumber;
        }
        return false;
    }
    // 记录的哈希代码，由页号和元组的连接表示
    @Override
    public int hashCode() {
        // some code goes here
        // 使用质数 31*页号+元组号实现
        return 31 * this.pageId.hashCode() + this.tupleNumber;
    }

}
```

RecordIDTest 单元测试：



（3）HeapPage.java

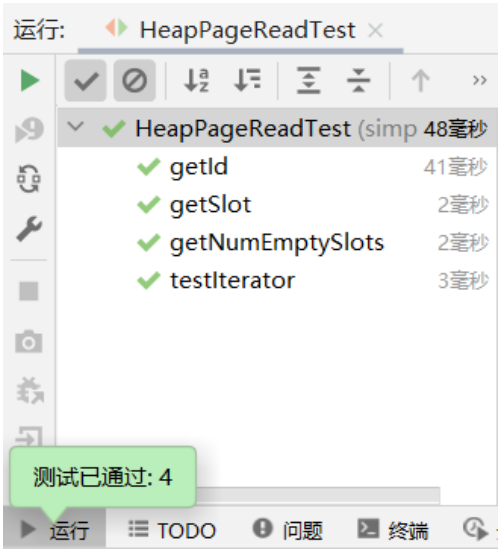
注：数据库中的每个表都有一个 HeapFile 对象。HeapFile 中的每个页面被安排为一组槽，每个槽可以保存一个元组。此外，每个页面都有一个头，它由一个 bitmap 组成，每个元组槽有一个位。如果一个特定元组对应的位为 1，则表示该元组有效；如果为 0，则元组无效。

```
public class HeapPage implements Page {
    final HeapPageId pid;
    final TupleDesc td;
    final byte header[];
    final Tuple tuples[];
    final int numSlots;
    byte[] oldData;
    private final Byte oldDataLock=new Byte((byte)0);
    // 构造函数的实现省略
```

```java
    // 检索可存储在此页上的元组的数目
    private int getNumTuples() {
        // some code goes here
        // 根据构造函数的文档注释：元组的个数等于
        // floor((BufferPool.getPageSize()*8) / (tuple size * 8 + 1))
        return (int) Math.floor((BufferPool.getPageSize() * 8.0) /
                                (this.td.getSizeInBytes() * 8 + 1));
    }
    // 计算 HeapFile 中 header 的字节数，每个元组占用 tupleSize 字节
    private int getHeaderSize() {
        // some code goes here
        // 据构造函数的文档注释：8 位 header 字的数目等于 ceiling(no. tuple slots / 8)
        return (int) Math.ceil(this.numSlots / 8.0);
    }
    public HeapPageId getId() {  // pid Getter
    // some code goes here
        return this.pid;
    }
    // 返回此页上的空槽数
    public int getNumEmptySlots() {
        // some code goes here
        int slots = 0;
        for (int i = 0; i < this.numSlots; i++) {
            // 逐个槽判断是否为空
            if (!this.isSlotUsed(i)) {
                slots++;
            }
        }
        return slots;
    }
    // 判断此页上的相关槽位是否已填满
    public boolean isSlotUsed(int i) {
        // some code goes here
        // 根据 guideline：
        // 1. The low bits of each byte represents the status of the slots
        //   that are earlier in the file.（每个字节的低位表示文件中较早的槽的状态）
        // 2. JVM is big-endian （高位编址，高序字节存储在起始地址）
        // 例如：18 个槽，全部填满，header 为（下面的'.'为分隔符）
        // header:
        // [1.1.1.1.1.1.1.1,  1. 1. 1. 1. 1. 1.1.1, 0.0.0.0.0.0. 1. 1]
        // 槽中 bit 标号:
        // [7.6.5.4.3.2.1.0, 15.14.13.12.11.10.9.8, x.x.x.x.x.x.17.16]
        int idx = i / 8;  // 该槽位在 header 中的分量的下标
        int pos = i % 8;  // 该槽位在这一分量中为第几位
        // 判断 bit 所在的 byte（idx 位置）右起第 pos 位是否为 1 （即是否占用）
        return ((this.header[idx] >> pos) & 0x01) != 0x00;
```

```
    }
    // 遍历本页上所有元组的迭代器，不返回空槽中的元组
    public Iterator<Tuple> iterator() {
        // some code goes here
        return new Iterator<Tuple>() {
            private int usedTuplesCount = 0;    // 遍历过的计数器
            private int index = 0;  // 槽位
            // 所有使用了的槽位总数：槽数-空槽数
            private final int usedTuplesNumber = HeapPage.this.numSlots -
                                                 HeapPage.this.getNumEmptySlots();
            @Override
            public boolean hasNext() {
                // 有后继：槽位小于总槽数 且 遍历过的计数器小于所有使用了的槽位总数
                return this.index < HeapPage.this.numSlots &&
                       this.usedTuplesCount < this.usedTuplesNumber;
            }
            @Override
            public Tuple next() {
                if (!this.hasNext()) {
                    throw new NoSuchElementException();
                }
                while (!isSlotUsed(this.index)) { // 找到非空槽元组
                    this.index++;
                }
                // 返回一个使用过的元组，并更新遍历过的计数器
                this.usedTuplesCount++;
                return tuples[this.index++];
            }
        };
    }
    // 其余方法省略
}
```

HeapPageReadTest 单元测试：



Exercise 5. HeapFile access method (2)

HeapFile.java

```
public class HeapFile implements DbFile {
    private final File file;
    private final TupleDesc tupleDesc;
    private final int numPage;
    // 构造由指定文件支持的堆文件
    public HeapFile(File f, TupleDesc td) {
        // some code goes here
        this.file = f;
        this.tupleDesc = td;
        this.numPage = (int) (file.length() / BufferPool.PAGE_SIZE);
    }
    public File getFile() {  // File Getter
        // some code goes here
        return this.file;
    }
    // 返回唯一标识此堆文件的 ID
    public int getId() {
        // some code goes here
        // 文档注释建议
        return this.file.getAbsoluteFile().hashCode();
    }
    public TupleDesc getTupleDesc() {  // tupleDesc Getter
        // some code goes here
        return this.tupleDesc;
    }
    // 从磁盘读取指定的页
    public Page readPage(PageId pid) throws IllegalArgumentException {
        // some code goes here
        Page page;
        byte[] data = new byte[BufferPool.PAGE_SIZE];  // 存放数据
        // 使用 RandomAccessFile ，try-with-resource 方式读取
        try (RandomAccessFile file = new RandomAccessFile(this.file, "r")) {
            // 计算文件所在页面文件的位置
            int pos = pid.getPageNumber() * BufferPool.PAGE_SIZE;
            file.seek(pos); // 移动文件指针，读取数据
            file.read(data, 0, data.length);
            page = new HeapPage((HeapPageId) pid, data); // 创建页面
        } catch (IOException e) {
            throw new IllegalArgumentException(e);
        }
        return page;
    }
    public int numPages() {  // numPage Getter
        // some code goes here
```

```
        return this.numPage;
    }
    // 返回存储在这个 DbFile 中的所有元组的迭代器。
    // 迭代器必须使用 BufferPool.getPage 而不是 readPage 来遍历页面。
    public DbFileIterator iterator(TransactionId tid) {
        // some code goes here
        return new DbFileIterator() {
            private int pagePos = 0;
            private Iterator<Tuple> tuplesInPage = null;
            // 为 tuplesInPage 提供页面的迭代器
            public Iterator<Tuple> getTuplesInPage(HeapPageId pid)
                        throws TransactionAbortedException, DbException {
                // HeapPage page = (HeapPage) readPage(pid);  // 错误
                // 不能直接使用 HeapFile 的 readPage 方法，
                // 而是要通过 BufferPool 来获得 page
                // readPage 方法只应该在 BufferPool 类被直接调用，
                // 在其他需要 page 的地方需要通过 BufferPool 访问
                // 这样才能实现缓存功能
                HeapPage page = (HeapPage) Database.getBufferPool()
                                    .getPage(tid, pid, Permissions.READ_ONLY);
                return page.iterator();
            }
            @Override
            public void open()
                    throws DbException, TransactionAbortedException {
                this.pagePos = 0;
                int tableId = getId();
                HeapPageId pid = new HeapPageId(tableId, this.pagePos);
                this.tuplesInPage = this.getTuplesInPage(pid);
            }
            @Override
            public boolean hasNext()
                    throws DbException, TransactionAbortedException {
                if (this.tuplesInPage == null) {    // 迭代器已关闭或还未打开
                    throw new IllegalStateException();
                }
                if (this.tuplesInPage.hasNext()) {   // 页面中还有元组某一遍历
                    return true;
                }
                // 检查是否还有其他的页面
                if (this.pagePos < numPages() - 1) {
                    this.pagePos++;
                    HeapPageId pid = new HeapPageId(getId(), this.pagePos);
                    // 转到新的页面
                    this.tuplesInPage = this.getTuplesInPage(pid);
                    return this.tuplesInPage.hasNext();  // 确保该页面也有元组
```

```
            }
            return false;
        }
        @Override
        public Tuple next() throws DbException,
                                   TransactionAbortedException,
                                   NoSuchElementException {
            if (!this.hasNext()) {
                throw new NoSuchElementException();
            }
            return this.tuplesInPage.next();
        }
        @Override
        public void rewind()
                throws DbException, TransactionAbortedException {
            this.open();
        }
        @Override
        public void close() {
            this.pagePos = 0;
            this.tuplesInPage = null;
        }
    };
  }
  // 其余方法省略
}
```

HeapFileReadTest 单元测试：

Exercise 6. Operators

SeqScan.java

```
public class SeqScan implements DbIterator {
    private static final long serialVersionUID = 1L;
```

```
    private TransactionId tid;  // 事务 id
    private int tableId;   // 表号
    private String alias;  // 别名
    private DbFileIterator iterator;  // 文件迭代器，遍历 DB 文件用
    // 作为指定事务的一部分，在指定表上创建顺序扫描。
    // 参数为事务 id，表号，该表的别名（tableAlias，可以为空）
    // 返回的 tupleDesc 应该有名称为 tableAlias.fieldName 的字段
    public SeqScan(TransactionId tid, int tableid, String tableAlias) {
        // some code goes here
        this.tid = tid;
        this.tableId  = tableid;
        if (tableAlias != null){
            this.alias = tableAlias;
        } else {
            this.alias = "null";
        }
        this.iterator = Database.getCatalog().getDatabaseFile(tableid).iterator(tid);
    }
    public String getTableName() {
        return Database.getCatalog().getTableName(this.tableId);
    }
    public String getAlias()    {  // alias Getter
        // some code goes here
        return this.alias;
    }
    // 重置表号和别名
    public void reset(int tableid, String tableAlias) {
        // some code goes here
        this.tableId = tableid;
        if (tableAlias != null){
            this.alias = tableAlias;
        } else {
            this.alias = "null";
        }
        this.iterator = Database.getCatalog()
                                .getDatabaseFile(tableid).iterator(tid);
    }
    public SeqScan(TransactionId tid, int tableid) {
        this(tid, tableid, Database.getCatalog().getTableName(tableid));
    }
    public void open() throws DbException, TransactionAbortedException {
        // some code goes here
        this.iterator.open();
    }
    // 返回 TupleDesc，其中包含来自底层 HeapFile 的字段名，
    // 并以来自构造函数的 tableAlias 字符串作为前缀。
```

```
    public TupleDesc getTupleDesc() {
        // some code goes here
        // 获取原始 TupleDesc
        TupleDesc desc = Database.getCatalog().getTupleDesc(this.tableId);
        int numFields = desc.numFields();
        Type[] types = new Type[numFields];
        String[] names = new String[numFields];
        for (int i = 0; i < numFields; i++) {
            // 添加前缀
            types[i] = desc.getFieldType(i);
            String fieldName = desc.getFieldName(i);
            if (fieldName == null) {
                fieldName = "null";
            }
            names[i] = this.alias + "." + fieldName;
        }
        return new TupleDesc(types, names);
    }
    public boolean hasNext()
              throws TransactionAbortedException, DbException {
        // some code goes here
        return this.iterator.hasNext();
    }
    public Tuple next() throws NoSuchElementException,
            TransactionAbortedException, DbException {
        // some code goes here
        return this.iterator.next();
    }
    public void close() {
        // some code goes here
        this.iterator.close();
    }
    public void rewind() throws DbException, NoSuchElementException,
            TransactionAbortedException {
        // some code goes here
        this.iterator.rewind();
    }
}
```

ScanTest 系统测试：