
南開大學
Nankai University

计算机学院

数据库 **SimpleDB** 实验报告

# Lab1

姓名：唐明昊

学号：2113927

专业：计算机科学与技术

2023 年 3 月 17 日

# 目录

# 1 Exercise 1

## 1.1 TupleDesc

TupleDesc 类用来描述一类 tuple，可以理解为是一个表的 schema。

schema 里的每一个 field/attribute 用 TDItem 类描述。该类有两个成员变量分别为 fieldType 和 fieldName，一起用来描述 field。为了保存一个 schema 里的所有 fields，需要声明一个 TDItem[] 数组。

在 TupleDesc 的构造函数中，传入了 typeAr 和 fieldAr 两个数组，分别对应 schema 里每个 field 的 type 和 name，于是可以很自然的完成构造函数。

TupleDesc 构造函数

```
1  public TupleDesc(Type[] typeAr, String[] fieldAr) {
2      // some code goes here
3    tdItems =new TDItem[typeAr.length];//构造数组对象
4    for(int i=0;i<typeAr.length;i++) {
5      //构造数组中每一个元素
6      tdItems[i]=new TDItem(typeAr[i],fieldAr[i]);
7    }
8  }
```

merge 静态函数要求将两个 TupleDesc 对象合并为一个新的 TupleDesc。考虑到构造一个 TupleDesc 需要 typeAr 和 fieldAr 两个参数，于是只需遍历两个 td 对象 fieldType 和 fieldName 并最终合成为一个数组即可完成构造。

类中的其他函数相对来说就比较简单，只需要直接或者经过遍历返回要求的值即可。

## 1.2 Tuple

Tuple 实现一个具体的 tuple。

由构造函数即可看出首先需要一个 TupleDesc 对象作为表头来描述，其次，tuple 中的每个字段同样需要数组来存储，由 setFiled 函数就可以看出，tuple 类需要一个 Field[] 数组。因此定义成员变量 private Field[] fields。

Iterator<Field> fields() 函数要求返回一个 Field 类型的迭代器，调用 Arrays.stream 函数将数组转换为一个顺序流，进而即可直接获取迭代器。

## 1.3 Test


图 1.1: TupleDescTest

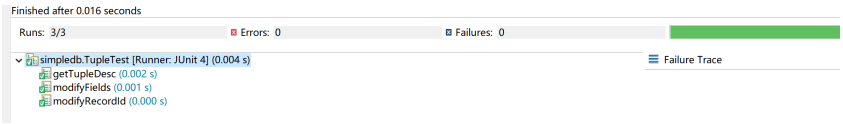


图 1.2: TupleTest

# 2 Exercise 2

## 2.1 Catalog

Catolog 类保存了所有种类的 table 及其 schema，每个 table 对应一个 DbFile。DbFile 是物理层的 Table，它可以获取 page 和迭代 tuple。每个 DbFile 即每个 Table 都有一个特殊的 id。

由 addTable 函数可以看出，每个 Table 由 file，name，pkeyField 三个属性构成。为了代码更好阅读，定义 Table 类，封装这三个属性。

Table 构造函数

```
1  public Table(DbFile dbFile, String name, String pkeyField) {
2    super();
3    this.dbFile = dbFile;
4    this.name = name;
5    this.pkeyField = pkeyField;
6  }
```

Catalog 需要保存所有 Table，用数组或者链表保存不便于查找，考虑到每个 Table 有独立的一个 id，于是使用 Map 数据结构，id 作为 key 值，Table 作为 value 值。

addTable 函数

```
1  public void addTable(DbFile file, String name, String pkeyField) {
2      // some code goes here
3    Table temp=new Table(file,name,pkeyField);
4    //得判断名字冲突
5     for(int key:tables.keySet()) {
6        if(tables.get(key).name.equals(name))
7          tables.get(key).name="";
8       }
9    tables.put(file.getId(), temp);
10 }
```

其余函数的实现均是靠 HashMap 的特性，通过 id 值或遍历获取到相应的 Table，返回需要的值即可。

### 2.2 Test


图 2.3: CatalogTest

## 3 Exercise 3

### 3.1 BufferPool

BufferPool 保存了近期从磁盘中调用来的 pages。

一个 table 由多个 page 组成，每个 page 存储 table 中的一部分数据，即一些 tuple。当需要查询 table 中的数据时，数据库需要在相应的 page 中查找符合条件的数据。

Page 也是一个类，DbFile 从磁盘中读写数据都以 Page 为单位，每个 Page 都有一个与之对应的 PageId，记录该 page 所属的 table。而每个 page 有属于它自己的 id，PageId.hashCode()，BufferPool 同样用 Map 结构保存 pages，用 page 的 hashcode 当作保存 page 的 key 值。

getPage 函数要求返回一个 Page，如果 page 不在 bufferpool 中，则需要将它导入到 bufferpool 中来并返回。

从 bufferpool 找到一个 page 很简单，只需要用 HashMap 查找 key 即可。而当 page 不在 Map 中时，根据提示则需要用到 DbFile.readPage 方法从一个 DbFile 中去获取 Page。

getPage 函数

```
1   public Page getPage(TransactionId tid, PageId pid, Permissions perm)
2       throws TransactionAbortedException, DbException {
3       // some code goes here
4
5     //page有自己独有的id(hashCode),page所属的table也有id(getTableId)
6     if(!pages.containsKey(pid.hashCode())) {//查询的page不在bufferPool中
7       //从文件中读取page，用dbFile
8       //读取文件，catalog.getDatabaseFile()
9       DbFile temp=Database.getCatalog().getDatabaseFile(pid.getTableId());
10      Page page=temp.readPage(pid);
11
12      //读入bufferPool
13      pages.put(pid.hashCode(), page);
14    }
15      return pages.get(pid.hashCode());//lock? insufficient? not necessary for lab1?
16  }
```

# 4 Exercise 4

## 4.1 HeapPageId

HeapPageId 实现 PageId，一个 HeapPageId 对应一个 HeapPage，记录了当前页所属的 table 以及当前页在其 table 中的第几页。

## 4.2 RecordID

一个 RecordID 对应一个 tuple，即 page 中的一个 slot，记录了该 tuple 所属的 page（保存 pageId）和当前 tuple 在 page 中的第几个位置。

## 4.3 HeapPage

HeapPage 实现了 Page，一个 Page 记录 HeapFile 中的部分数据。

getNumTuples() 计算该页中的 Slot 条数，由于每个 tuple 还需要在 header 中保存 1 位数据来记录状态，所以每个 tuple 占用的大小为 td.getSize()*8+1，再用一页的总大小除以它即可获得该页的 tuple 数。

getNumTuple 函数

```
1  private int getNumTuples() {
2  // some code goes here
3      return (int)
           Math.floor(((double)BufferPool.getPageSize()*8)/(td.getSize()*8+1));
4  }
```

getHeaderSize() 计算 Header 占用的字节数，一个 tuple 占一位 bit，用总 tuple 数除以 8 即可得到 Header 占用的字节数。

getHeaderSize 函数

```
1  private int getHeaderSize() {
2      // some code goes here
3      return (int) Math.ceil((double)numSlots/8);
4  }
```

isSlotUsed 函数，需要去看 header 中的对应位，判断当前 slot 的状态。首先需要确定当前 tuple 在 header 的第几个字节，其次在该字节内，还需确定它位于第几位上。每个 byte 上的 low bit 代表先进入 file 的 slot，如 the lowest bit of the first byte 代表的是第一个进入的 slot，同时考虑到 java 虚拟机使用的是大端序保存。

isSlotUsede 函数

```
1  public boolean isSlotUsed(int i) {//应该找到bit上去
2      // some code goes here
3    //大端序，1个byte里8个bit按大端序存
4    //首先要确定在第几个byte
5    int positionOfByte=i/8;
6    //再确定在第几个bit
```

```
        int positionOfBit=i%8;
        //大端序
          return 1==((header[positionOfByte]>>positionOfBit)&0x1);
    }
```

### 4.4 Test

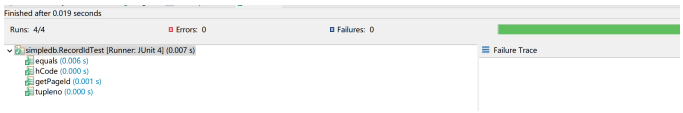


图 4.4: RecordIdTest


图 4.5: HeapPageIdTest


图 4.6: HeapPageReadTest

## 5 Exercise 5

### 5.1 HeapFile

HeapFile 实现 DbFile，一个 table 对应一个 HeapFile，一个 HeapFile 由一系列 HeapPage 组成。由构造函数可以看出，一个 HeapFile 有两个属性：File，TupleDesc，很容易构造。

readPage 函数需要从磁盘上阅读一页并返回。构造一个 HeapPage 需要 pageId 和 data[],PageId 可以由传入的参数 pid 进行构造，而 data[] 需要从文件中进行读取。

从 data[] 中进行随机读取，需要用到 java 的 RandomAccessFile 类。首先计算 page 在文件中的偏移量，再读取固定 page 大小的字节到 data 数组中。有一个坑的地方是，RandomAccessFile 的 read 函数的 offset 偏移量并不是读取位置的偏移量，而是将数据放进数组的偏移量，故需要提前将文件指针进行偏移，否则会出现数组越界等问题。

readPage 函数

```
    public Page readPage(PageId pid) {
        // some code goes here
        //构造一个page需要pageId和data[]
     byte [] data=new byte[BufferPool.getPageSize()];
     int tableId=pid.getTableId();
     int pageNo=pid.getPageNumber();
```

```
        HeapPageId hpid=new HeapPageId(tableId,pageNo);

        //page在文件中有偏移量 random access
        try {
        RandomAccessFile raf=new RandomAccessFile(table,"r");
        //做exercise6出现IndexOutOfBoudsException?
        //read函数偏移量是byte数组的偏移量!! 前面初始化为空
        //不是数据流的偏移量, 数据流得用seek!

        int offset=pageNo*BufferPool.getPageSize();//Number应该是从0开始?
        raf.seek(offset);
        int testRead=raf.read(data,0,BufferPool.getPageSize());

        if(testRead!=BufferPool.getPageSize())//没读对
          throw new IllegalArgumentException("page wrong!");

        raf.close();
        return new HeapPage(hpid,data);

    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
        throw new IllegalArgumentException("page does not exist in this file");
    }
```

iterator 函数需要返回一个 DbFileIterator，该迭代器在一个文件里，一次访问一页到内存，每次都读取这一页的 slots，即每次返回一个 tuple。实现该函数需要用到 auxiliary class，这个内部类实现接口 DbFileIterator。

内部类 HeapFileIterator 需要属性 TransactionId；cursor 记录当前读取到哪一页；file 记录读取的具体文件；该类还维护一个 Iterator<tuple>，从具体的一页中进行读取，每次 next 函数由这个迭代器读取返回一个 tuple。

获取文件中具体的某页，需要用到 BufferPool.getPage 函数，传入具体页的 pageId 即可获取该页。open() 函数，rewind() 函数即把 cursor(pageNo) 置为 0，从 file 中读取第 0 页，再将其迭代器赋给类内维护的迭代器。

在实现 hasNext 函数时，由于每次都是返回 tuple，首先看当前 iterator 所在页是否已经读取完，是则需要进行翻页，cursor 加一，再重新从 bufferpool 中读取 page。

hasNext 函数

```
    @Override
    public boolean hasNext() throws DbException, TransactionAbortedException {
        // TODO Auto-generated method stub
        if(it==null)
            return false;//没有open
        if(!it.hasNext()) {//这一页的tuple已经访问完了
            if(cursor+1>=file.numPages())
                return false;
```

```
            else {
                cursor++;//下一页
                HeapPageId temp=new HeapPageId(file.getId(),cursor);
                it=((HeapPage)Database.getBufferPool().getPage(tid, temp,
                    Permissions.READ_ONLY)).iterator();
                return true;
            }
        }
        else
            return true;
    }
```

### 5.2 Test


图 5.7: HeapFileReadTest

## 6 Exercise 6

### 6.1 SeqScan

SeqScan 实现 OpIterator，从指定的 table 的 page 中读取所有的 tuples。SeqScan 是一个上层迭代器，并不需要它去进行文件读取，它只需要将请求向下传递，再由叶节点的迭代器进行读取返回。所以类中维护一个 DbFileIterator，由该迭代器去获取 tuples。

在 open 函数中，给 iterator 赋值，只需要从 catalog 中找到表所属的 DbFile，再返回 DbFile.iterator() 即可。后续的 next，hasNext 函数等，只需调用该迭代器的相应函数。

getTupleDesc 函数，返回一个表的 TupleDesc，但需要在每个字段前面加上别名。构造一个 TupleDesc 需要 typeAr 和 fieldAr，用 tableId 读取到当前文件的 TupleDesc，遍历它的每个属性，修改字段名后传入两个数组用于构造。

getTupleDesc 函数

```
    public TupleDesc getTupleDesc() {//还需加前缀
      //HeapFile里有TupleDesc
      //TupleDesc需要typeAr和fieldAr去构造
      TupleDesc temp=Database.getCatalog().getTupleDesc(tableId);
      Type[] typeAr=new Type[temp.numFields()];
      String[] fieldAr=new String[temp.numFields()];
      for(int i=0;i<temp.numFields();i++) {
        //typeAr直接得
        typeAr[i]=temp.getFieldType(i);
        //fieldAr得判断field是不是null，tableAlias已经预处理
        StringBuilder toPrefix=new StringBuilder(this.tableAlias+".");
        if(temp.getFieldName(i)==null||temp.getFieldName(i)=="")
```

```
13            toPrefix.append("null");
14          else
15            toPrefix.append(temp.getFieldName(i));
16          fieldAr[i]=toPrefix.toString();
17        }
18          return new TupleDesc(typeAr,fieldAr);
19      }
```

### 6.2 Test


图 6.8: ScanTest

## 7 Git Commit History

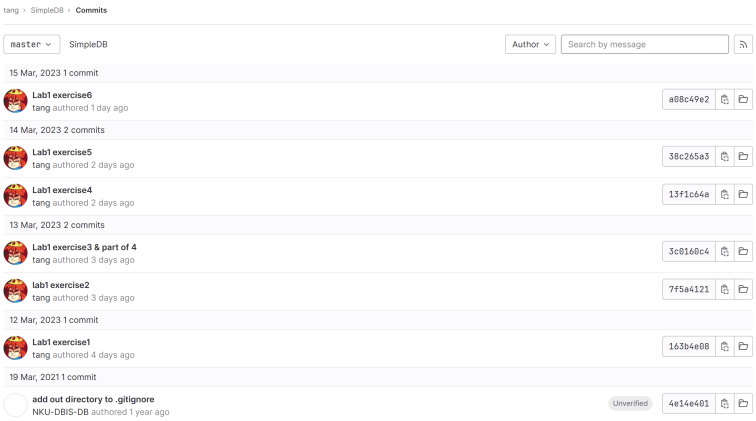


图 7.9: Git Commit History