南开大学
Nankai University

南　　开　　大　　学

计 算 机 学 院

数据库实验报告

Lab1

学号：2113839

姓名：林帆

2023 年 3 月 20 日

目录

一、 实验过程

(一) exercise1

1. TupleDesc 类的实现

TupleDesc 是表头类，由一行字段构成，因而在类内首先需要实现字段 TDItem 类。每个字段有两个属性——字段的类型 fieldType（Type 类型）和字段的名字 fieldName（String 类型），在 field 类中声明变量，在其构造函数中赋值即可。

TupleDesc 要存储多个 field 信息，需要用数组实现，即需要在类内定义 TDItem 类型的数组 tdItems 和该数组对应的迭代器 iterator。考虑链表 List 类自身有迭代器，因而实现 tdItems 的迭代器时，只需要通过 Arrays.asList() 函数，将 tdItems 由数组转化成 List 即可。iterator 返回语句如下所示：

iterator 返回语句

```
return (Iterator<TDItem>) Arrays.asList(tdItems).iterator();
```

定义 TupleDesc 的构造函数时需要考虑，对每一个 field，都需给出其 fieldType 和 fieldName。而 field 是顺序存储在 tdItems 中的，因而可按照对应顺序给出类型数组 typeAr 和段名数组 fieldAr 作为构造函数的两个参数，并按顺序生成 tdItems[i]。当未给出段名数组时，tdItems[i] 设置为空字符串即可。含双参数的构造函数如下所示：

TupleDesc 类的构造函数

```
public TupleDesc(Type[] typeAr, String[] fieldAr) {
    //typeAr为表头结构的类型数组
    //fieldAr为表头的Name数组
    tdItems = new TDItem[typeAr.length];
    for(int i=0;i<typeAr.length;++i){
        tdItems[i] = new TDItem(typeAr[i],fieldAr[i]);
        //根据typeAr[i]和fieldAr[i],依次生成filed[i]
    }
}
```

对于一些实现获取字段属性值功能的函数，只需要根据参数返回对应的属性值即可。但也要注意，如果有索引越界等情况出现，应 throw 对应类型的错误。部分函数功能和返回值如下所示：

函数	返回值	获取属性
getFieldName(int i)	tdItems[i].fieldName	第 i 个 filed 的 name
getFieldType(int i)	tdItems[i].fieldType	第 i 个 filed 的 type

表 1: TupleDesc 类中获取字段属性的函数

fieldNameToIndex(String name) 函数是用字段名来确定对应 field 在 TDItems 中第一个出现的位置下标。使用 for 循环遍历，一一比较即可求解。两个 String 类型的字符串比较是否相同时，一般使用其自带的 equal 函数，即 str1.euqal(str2)。但要注意，参数中的 name 有可能为 null，因而 fieldName 在前，name 在后。

name 是否相同的判断语句

```
if(tdItems[i].fieldName.equals(name))
```

getSize() 函数返回 tuple 的大小（字节数）。由于每个 tuple 的构成类型都与 TupleDesc 相同，所以需要遍历 TupleDesc 中的 field，每个 field 的 size=fieldType 的长度，将其累加后得到整个 tuple 的大小。getSize() 函数的实现如下所示：

TupleDesc 类中的 getsize() 函数

```
public int getSize() {
    int size=0;
    for(int i=0;i< tdItems.length;i++)
        size+=tdItems[i].fieldType.getLen();
    return size;
}
```

merge(TupleDesc td1, TupleDesc td2) 函数可将两个表头合并，实现方式是创建新表头 td3，将 td1 和 td2 中的 field 信息依次添加，最终返回 td3。

equals(Object o) 用于判断 o 与该 TupleDesc 结构是否相同。函数体内判断步骤如下所示，若其中有一个 step 不满足，就返回 false；三个全部满足时返回 true。

TupleDesc 类中 equal() 函数的判断步骤

```
step1:用instance of判断类型是否相同
step2:判断长度是否相同
step3:设置次数为tdItems.length的for循环，依次判断field类型是否对应相同
```

toString() 函数中，创建可变型字符串 StringBuilder，通过 append() 函数将表头中所有 field 的类型和名字添加至字符串中,最终以"fieldType[0](fieldName[0]), ..., fieldType[M](fieldName[M])"的形式展现。

2. Tuple 类的实现

Tuple 是实际的数据行，类的内部共有三个基本变量：

变量名	变量类型	含义
tupleDesc	TupleDesc	表头
recordId	RecordId	该行在磁盘中的位置
fields	Field[]	行中的字段数组

表 2: Tuple 类中的三个基本变量

类中有一些必要的 get/set 某属性的函数，实现过程中，根据函数要求，对变量赋值或返回相应变量即可。除此以外，还有 Field 数组的迭代器，实现方式与 TupleDesc 类中 tdItems 的迭代器一样，都是通过 Arrays.asList() 函数将数组转化成 List，并返回 List 的 iterator。

(二) exercise2

1. Catalog 类的实现

Catalog 作为数据库中 table 的目录, 类中需要实现的主要工作是: 实现 table 类和修改 table 的函数，并创建相应的哈希表 hashTbale 便于查询。

table 类中的主要成员如下所示:

变量	变量类型	含义
tableName	String	表名
dbFile	DbFile	存储表信息的文件
pkeyField	String	表中主键的 fieldName

表 3: Table 类中的三个基本变量

在Catalog 类中，实现添加 table 的函数是 addTable(DbFile file, String name, String pkeyField)；函数的参数是新表的三个属性，需要据此调用 table 的构造函数创建新表。在添加新表前，需要遍历目录中的旧表（可利用 hashTable 的 iterator 进行键值 key 的遍历），通过 Key 值判断是否有和新表重名的旧表。若存在，则将旧表改名为 null，再添加新表；若不存在，则直接添加新表。添加可通过 hashTable.put() 实现。addTable() 函数的实现思路如下所示:

Catalog 类中的 addTable() 函数

```
public void addTable(DbFile file, String name, String pkeyField) {
    //判断是否已经有该名字的表
    int old_key=-1;
    Enumeration<Integer> hashKeys=hashTable.keys();
    while(hashKeys.hasMoreElements()){
        Integer key=hashKeys.nextElement();
        Table temp_table=hashTable.get(key);
        if(temp_table.tableName.equals(name))
            old_key=key;
    }
    判断old_key是否为-1（在循环中是否被修改）;
    （修改旧表名）创建新表，hashTable.put()添加新表
}
```

hashTable 存储每个 table(value) 对应的 key(哈希值)，查找效率非常高。Catalog 中的一些 get 属性的函数，大多都是通过 hashTable 寻找 DBfile 中信息得到的。

例如，查询 tableID 的函数 getTableId(String name)，需要遍历哈希表的 keys，找到对应 value(table)，然后判断 tableName 是否相同，如果相同则返回 table.dbFile.getId()。

Catalog 类中的 getTableId() 函数

```
Table temp_table=hashTable.get(key);//通过Key遍历到的当前table
    if(temp_table.tableName!=null&&temp_table.tableName.equals(name))
            return key;//Key即为所求table的Id
```

getTableId() 函数实现后，每个已知 tableName 的 table 都可通过 getTable(tablename)，得到在 hashTable 中的 Key, 继而通过 hashTable.get(Key) 直接确定 table(value), 然后查询 DBfile 该表的表头 td；若已知 tableId，则可直接使用 hashTbale.get(Key)，例如:

getTupleDesc(int tableid) 函数的实现

```
Table temp_table=hashTable.get(tableid);//通过get(key)确定table
解决冲突
return temp_table.dbFile.getTupleDesc();//查询dbFile, 获得表头
```

Catalog 类中的 getDatabaseFile(int tableid) 函数，getPrimaryKey(int tableid) 函数和 getTableName(int id) 函数，实现方式也都和 getTupleDesc(int tableid) 函数相同，故不作详解。

考虑 Catalog 的迭代器 iterator 时，也可直接调用 hashTable 的内部函数。返回语句如下：

Catalog 的迭代器

```
return hashTable.keySet().iterator();
```

为将 catalog 中的 table 全部清空，可直接调用 hashTable.clear();

(三) exercise3

1. BufferPool 类的实现

缓冲池用来存储最近读取的 page 的信息，因而需要一个变量 pageStore，类似于 Catalog 中的 hashTbale，来存储已读页，其中 page 为 value；也需要一个变量 numpages 记录缓冲池 BufferPool 中存储的页数。默认的 pagenums 和 pageSize 在类中已给出。

本实验中重点需要实现的函数为 getPage() 函数，返回类型为 Page, 三个参数为：请求事务的 id——ransactionId tid, 想要得到的页的 Id——PageId pid, 访问权限——Permissions perm。但实现过程中只用到了参数 pid。

实现思路为：通过 pageStore.get(key)，得到目标页 Page。（key 的计算用到了 exersice4 中的 PageId 类的函数，可参考下文。）若 pageStore 中有该页，则直接 return 即可；若 pageStore 中没有找到该页，则找到该页的 DBfile，读取相应信息，将该页添加进 pageStore。具体如下所示：

getPage() 函数的实现

```
若BufferPool中有该页
    key=pid.hashCode();//确定键值
    return pageStore.get(key);//返回该页
若BufferPool中没有该页
    获取该页的DBfile
    读取该页
    通过pageStore.put()函数将该页添加进BufferPool
```

(四) exercise4

1. HeapPageId 类的实现

HeapPageId 从 PageId 继承而来。每一个 table 对应多个 page，每个 HeapPageId 类中，既有对应 table 的 Id——tableId，也有该页在对应 table 中的 pageNumber——pgNo。类中的 getTableId() 函数和 getPageNumber() 函数可通过 return 直接返回相应变量。

在 exersice3 实现 BufferPool 的过程中，我们知道每一个存储在 pageStore 的 page 都需要有其对应的 key，这个 key 便可以在 HeapPageId 类中的 hashCode() 函数中构造。考虑每个页

由其 table 在磁盘中的位置，和页本身在 table 中的偏移位置唯一确定。因而可综合两者构造字符串，进而利用 String 类的自带函数 hashCode 直接构造哈希值。具体的实现如下所示：

HeapPageId 类中 hashCode() 函数的实现

```
String str = "" + tableId + pgNo;
    return str.hashCode();
```

在比较两个页是否相同时，传统的逐一数据判断比较的方法时间复杂度过高，我们可以在 HeapPageId 中构造 equals() 函数，通过判断两个页的 hashCode() 相同，或判断两个页的 tableId 和 pgNo 全都相同。在实验过程中选了后者实现。判断过程中也是首先用 instance of 判断类型，再比较属性是否相同。

2. RecordID 类的实现

RecordID 是确定一个具体 tuple 位置的类，用该 tuple 所在页的 ID，和其在该页的偏移量确定，故该类的基础变量为 pid（PageId 类型）和 tupleno（int 类型）。

getTupleNumber() 函数和 getPageId() 函数分别用来获取对应属性。

构造 tuple 的 hashCode，页可通过 pid 和 tupleno 唯一确定。pid 的类型为 PageId，可将其 hashCode 提出，与同为 int 型的 tupleno 相加，唯一确定 tuple 的 hashCode。具体的实现如下所示：

RecordID 类中 hashCode() 函数的实现

```
String str=""+pid.hashCode()+tupleno;
    return str.hashCode();
```

equals() 的实现思路与 HeapPageID 中的 equals() 函数相同，即判断类型相同后，判断两个 tuple 的 pid 和 tupleno 是否相同。

3. HeapPage 类的实现

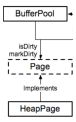

图 1: HeapPage,page,BufferPool 间的关系

如图所示，HeapPage 由 Page 继承而来，是将 BufferPool 中的 Page 具体化实现的子类。HeapPage 中的基础成员变量如下所示：

变量	变量类型	含义
pid	HeapPageId	该页的 id
td	TupleDesc	该页所在 table 的表头
header	byte[ ]	记录该页的 tuple 是否有效（填入）的字节数组
tuples	Tuple[ ]	该页的 tuples
numSlots	int	该页的行（tuples）数

表 4: HeapPage 类中的基本变量

numSlots 需要赋值，因而可通过构造 getNumTuples() 函数实现。考虑每个 Page 的 Size 固定，由 numSlots 个 tuple 和一个 header 构成。每个 tuple 的字节数为 td.getSize()，乘 8 即为比特数；header 的比特数即为 tuple 的个数；每页的 PageSize（单位为 bit）可以通过 BUfferPool 中的 getPageSize() 函数获取。通过 $\lceil Page_size/(td.getSize()*8+1)\rceil$，其中向上取整可通过 ceil 实现，得到结果。因而函数体如下所示：

getNumTuples() 函数的实现

```
1    private int getNumTuples() {
2        //通过字节计算,整个页面的行数（已用行+空行）
3        int page_size=BufferPool.getPageSize()*8;
4        int Tuple_num=(int)Math.ceil(page_size/(td.getSize()*8+1));
5        return Tuple_num;
6    }
```

实现 getHeaderSize() 函数计算 header 的大小 (单位为字节) 时，考虑 header 的每个 bit 位顺序对应一个 Tuple，因而 header 的一个字节位存储了 8 个 bit，即 8 个 Tuple。所以 $HeaderSize = \lceil numSlots/8\rceil$。

isSlotUsed(int i) 函数用于判断该页下标为 i 的 Tuple 是否被用过。只需找到该 Tuple 在 header 中的对应 bit 位，判断是否为 1 即可。需要先确定 Tuple[i] 在 header 数组中的字节位 index，然后确定在字节位中的偏移量 pos。根据 $1byte = 8bits$ 的关系，并通过 >> 的方法找到所求比特位。实现方法如下所示：

isSlotUsed() 函数的实现

```
1    //判断header的bit数
2        int index=i/8;//确定byte位置
3        int pos=i%8;//确定bit位置
4        return 1==((header[index]>>pos)&0x1);//返回字节位与1的比较结果
5    }
```

getNumEmptySlots() 函数用来计算该 page 中空 Tuple 的数量。通过 for 循环遍历该页的每个 tuple，然后对 isSlotUsed(i) 进行条件判断，最后返回不满足该条件的 tuple 的个数。

HeapPage 类内 Iterator<Tuple> 类型的迭代器，要求每次指向下一个非空的 tuple。实现思路是构建 ArrayList 类对象，for 循环对每一个 tuple 进行 isSlotUsed(i) 的条件判断。若符合条件非空，则用 ArrayList 的 add() 函数将该 tuple 添加，最后得到一个非空 tuple 构成 List，返回 List 类型自带的 iterator 即可。实现方式如下所示：

页内 Iterator<Tuple> 的实现

```
1    ArrayList<Tuple>used_tuples_array=new ArrayList<Tuple>();
2        for(int i=0;i<numSlots;i++)
3            if(isSlotUsed(i))//被用过了
4                used_tuples_array.add(tuples[i]);
5        return used_tuples_array.iterator();//直接返回数组的迭代器
```

（五） exercise5

1. HeapFile 类的实现

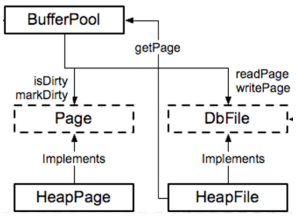

图 2: HeapFile,DbFile,BufferPool 间的关系

如图所示，HeapFile 由 DbFile 继承而来，是将 BufferPool 中的 Page 的 Dbfile 具体化实现的子类。HeapFile 中的基础成员变量有两个：对应页的 file——f（File 类型）和该页对应表的表头 td（TupleDesc 类型）。类中的 get 属性的函数通过 return 即可实现。在该实验中，需要实现的最主要的两个函数分别为 readPage() 和 iterator(TransactionId tid)。

readPage() 的参数为 PageId pid，返回类型为 Page。实现思路如下所示：

readPage() 的实现

```
1 getTableId()和getPageNumber()获取pid信息
2 创建新的HeapPageId
3 定义每页的字节数组bytes
4 使用RandomAccessFile定义输入流myfile
5 myfile.seek(起始)定位输入流的读入位置
6 myfile.read(bytes,页头，页尾)将0~page_size数据读入bytes
7 用HeapPageId和bytes创建HeapPage并返回
```

iterator() 需要通过创建内部类 HeapFileIterator 实现, 后者内部的主要成员变量如下所示：

变量	变量类型	含义
heapFile	HeapFile	当前的 heapfile
tid	TransactionId	请求的事务 id
permissions	Permissions	访问权限（读写）
bufferPool	BufferPool	当前库中的缓冲池
iterator	Iterator<Tuple>	每页的 tuple 迭代器
num	int	当前页的序号

表 5: HeapFileIterator 类中的基本变量

open()，close() 和 rewind() 分别实现开启，关闭和重置 iterator。open() 只需设置 iterator 的指向为 Tuple[0], close() 需要设置 iterator 指向 null, rewind() 内部先后调用 cloes() 和 open() 即可完成。

而 iterator() 指向该页 Tuple 的具体操作, 需要 getPageTuple(int pageNumber) 函数的实现。首先判断 pageNumber 是否在 0 numPages 之间, 然后获取 tableId, 通过 BufferPool().getPage() 函数找到目标 HeapPage（权限设为只读），返回该类自带的 iterator()。寻找目标 HeapPage 并返回其迭代器的代码如下所示：

getPageTuple() 的实现

```
int tableId = heapFile.getId();
HeapPageId pageId = new HeapPageId(tableId, pageNumber);
HeapPage temp_page=(HeapPage)Database.getBufferPool().getPage
                    (tid, pageId, Permissions.READ_ONLY);
```

hasNext() 函数判断 HeapFileIterator 是否还有下一个指向，需要对 Tuple 的 Iterator 指向进行分类讨论。分类讨论的思路如下所示:

hasNext() 的实现思路

```
若Iterator为空:
    返回false
若Iterator不空:
    若Iterator.next()不为空:
        返回ture
    若Iterator.next()为空: //看下一页的<Tuple>Iterator
        若下一页的numPages()指向超界:
            返回false
        若下一个的numPages()指向不超界:
            num+1
            将Iterator设置为下一页的Iterator
            返回true
```

next() 函数返回类型为 Tuple，返回 iterator.next(); 即可实现。

HeapFileIterator 类实现完成后，HeapFile 类的 iterator() 只需执行一条返回语句，如下所示;

HeapFile 类中 iterator() 的返回语句

```
return new HeapFileIterator1(this, tid);
```

(六) exercise6

1. SeqScan 类的实现

SeqScan 类主要实现了迭代器的封装，调用或返回前五个 exercise 中的 get 函数和不同类型的迭代器 iterator 即可完成。

二、 总结

本次实验通过补充实现数据库中一些基本类的代码，更加深入地理解了数据库的底层构造。在实验过程中，通过阅读注释，查看继承的父类实现，以及搜寻资料，不断构建 DataBase 的基础框架。学习过程中也对 java 语言有了一定的掌握，并能学会使用 git 终端和 gitLab。

参考文献