

SimpleDB Lab1 实验报告

2011269 王楠舟

1.元组

元组是关系数据库中的基本概念，关系是一张表，表中的每一行就是一个元组，用户实际就是在对元组进行操作。为了实现数据库元组，SimpleDB 中设计了 TupleDesc 和 Tuple 两个类。

TupleDesc 类： TupleDesc 的对象是用于描述一个 Tuple 对象的模式。 TDItem 是其维护的一个内部类，有 fieldType 和 fieldName 两个属性，用于描述元组中字段的类型与名称，在数据库里为对应的属性的类型及属性的名称。 TupleDesc 内维护了一个 TDItem 动态数组成员，能达到描述一个元组各个字段的属性名称及数据类型的作用。	
构造函数 TupleDesc(Type[]typeAr,String[] fieldAr)	过传入 Type 类型数组和 String 数组对维护的 TDItem 进行初始化，其中 fieldAr 的 String 数组可以缺省，代表数据段没有名称，默认为 NULL。
部分重点成员函数的实现	
fieldNameToIndex(String name)	返回属性名为 name 的字段在 tditems 中的下标 注意，传入的参数 name 可能为 null，需要我们判断。
getSize()	返回该元组的大小，即将各个字段的类型的长度累加在一起，通过 getFieldtype().getLen()方法获取字段类型的长度。
merge(TupleDesc td1, TupleDesc td2)	将两个 tupledesc 对象合并到一起。
equals(Object o)	由于传入参数是 Object 父类，需要判断对象是否为 null，再判断 o 是不是 TupleDesc 对象（用 getClass 或 is instance of）。

Tuple 类： Tuple 是用于数据库中描述一个元组的内容。 内部成员变量维护了一个 TupleDesc 的对象，用于描述该元组的模式； 以及一个 Field 类型的数组对象 fields，用于储存每个字段的信息。	
构造函数 Tuple(TupleDesc td)	传入的参数是一个 TupleDesc 的对象，用于初始化 Tuple 的内部成员 tupledesc， 同时还要注意 fields 数组的初始化，必须为数组赋初值 null。
部分重点成员函数的实现	
resetTupleDesc(TupleDesc td)	重新设置元组的 tupledesc，同时需要改变 fields 数组的大小。

--	--

2. 目录

为了在 SimpleDB 中查询我们能访问到的表，设计出 Catalog 目录类，注意在调用中直接使用 DataBase.getCatalog()就能得到这个数据库的目录对象。

Catalog 类:

Catalog 的对象是数据库中的一个目录，用于保存在数据库中访问到的表。

成员变量：为了保存一张数据库中的表，我们需要 DBfile，以及表的名称、这张表的主键，如果我们为这三个变量都创建一个数组，那么在后续查询操作中变得十分繁琐。为了简化，我选择创建一个内部类 Table，其维护着 dbfile、name、pkeyField 三个成员变量。在 Catalog 中，我们只需要维护一个 Table 类型的数组 tables 就能访问到目录中所有的表。此外，我们还需要维护一个 TableId 数组，用于记录各个表的 Id，便于后续在给出 tableid 时，只需要在 TableId 中快速查找到相应下标，就能在 tables 数组中找到对应的内容，不需要遍历 tables 数组去判断。

部分重点成员函数的实现

addTable(DbFile file, String name, String pkeyField)

因为要求一张表对应的 DBFile 是唯一的，目录中不能出现重复的 file，所以我们先在 tableIds 查找是否有相同的 file.getId()，如果出现了重复的 DBFile，则在 tables 数组中的对应下标处重新设置新传入的 table；如果没有，我们再去查找需要 addtable 的 name 是否已经在 tables 中出现过，如果发现，就将这个同名的 table 替换成我们新传入的 table；如果既没有同 DBFile 或同名的 table，我们直接将新表加入目录即可。

3. 缓冲池

因为数据库的表在保存在内存文件中，每次访问都需要访问内存的时间开销很大。所以 SimpleDB 设计一个缓冲池，用于保存我们近期访问过的 Page。

BufferPool 类:

BufferPool 是一个缓冲池类，负责缓存最近在内存读或写过的 page，SimpleDB 通过缓冲池向磁盘读或写文件。其内部成员包含了 page 的大小、BufferPool 内部最多存多少页面等；此外其维护一个成员变量：numPages，用于记录 BufferPool 存 page 的数量；还维护了一个哈希表 ConcurrentHashMap<Integer,Page> pageConcurrentHashMap，用于查找、插入 page。

Page 类：一个需要被实现的接口类，Page 内部储存了当前页面内的消息，以 Byte 数组形式保存。每一个 Page 都有自己相对应的 PageId。

部分重点成员函数的实现

getPage(TransactionId tid, PageId pid, Permissions perm)	通过 PageId 参数,在哈希表中查找有没有对应 PageId 的 hashCode,如果在哈希表中查找到了,直接返回 hashmap 中键值对应的 Page;如果查找失败说明此时 BufferPool 中不包含这个 Page, Database.getCatalog().getDatabaseFile(pid.getTableId()) 通过数据库目录找到 Page 所在的 DBFile,然后用 dbfile.readpage()方法,返回得到我们想要的 Page 对象,并将这个新的 Page 和他的 pageId 的 hashCode 组成的键值对,保存到 hashmap 中。
--	---

4.堆文件类型

在 SimpleDB 中,每一个表都有一个 HeapFile 对象,HeapFile 用于管理 HeapPage 的集合,Page 类型被储存在缓冲池中,由 HeadFile 类对其进行读写。

<p>HeapPageId: HeapPage 对象特用的 PageId,实现了 PageId 接口。内部成员有 tableid (这个 page 对应的表的 id)和 pgNo (第几个 page)</p> <p>RecordId: 用于保存这个表所保存在的 page 的 pageid 以及特定元组在该 page 中的位置,内部维护一个 hashCode 函数,将 pid 和 tupleno 转化成 hashCode:</p> <pre>String hash=pid+" "+tupleno; return hash.hashCode();</pre>
--

HeapPage: 堆页面类,实现了 Page 接口。
内部维护了 HeapPageId、TupleDesc、一个用于描述页头信息的 Byte 数组 header[]、Tuple 数组 tuple[]、numSlots (槽的数量)。

部分重点成员函数的实现	
getNumTuples()	td.getSize()返回了这个元组的大小,乘 8 转化为 bit 大小,因为每一个元组需要一个槽对应保存在头数组中,所以一个元组的大小为 td.getSize()*8+1, BufferPool.getpagesize 是页的总大小。(BufferPool.getPageSize()*8)/(td.getSize()*8+1)就能返回该页能保存的最多元组数量。
getHeaderSize()	返回这个页头数组的大小。 return numSlots/8+(numSlots%8==0?0:1); 若不为整数,需要向上进位。
isSlotUsed(int i)	判断第 i 个槽是否为 1,首先计算 Byte 数组的偏移量 byte_va=i/8;再计算 bit 的偏移量 bit_va=i%8;大端序存储,所以高位在低地址,byte_num=header[byte_va]读出一个 Byte,(byte_num>>bit_va)&1 右移 bit_va 位然后和 1 做与运算,结果若等于 1,说明槽被占用,为 0 则说明槽没有被占用。

Iterator<Tuple> iterator()	返回一个包含所有非空的元组，所以遍历成员 tuples 元组数组，若该元组对应的槽被占用，说明元组非空，将其加入到新的临时数组中，最后返回这个 Tuple 类型 ArrayList 的迭代器。
-------------------------------	--







































































<p>HeapFile: 堆文件类。</p> <p>管理一些 HeapPage 对象的集合，实现了 DBFile 接口。</p> <p>内部成员: File 对象 file 和 TupleDesc 对象 td 用于描述该 HeapFile 在内存具体的文件和在文件中储存的表的模式。</p>	
<p style="text-align: center;">部分重点成员函数的实现</p>	
readPage(PageId pid)	<p>通过 pageid 参数，将 page 读出来返回一个 HeapPage 对象。byte[] page_data=new byte[BufferPool.getPageSize()]构造好一个 Byte 数组，用于将内存中数据读入并构造 HeapPage 对象。</p> <p>因为这个 page 不一定是从文件头开始的，需要用 RandomAccessFile 对象来跳转访问文件。</p> <p>st 是通过 page 的号码和 page 大小计算出来该 page 在文件中的偏移量。</p> <pre>RandomAccessFile randomAccessFile=new RandomAccessFile(getFile(),"r"); int st=pid.getPageNumber()*BufferPool.getPageSize(); randomAccessFile.seek(st); randomAccessFile.read(page_data,0,BufferPool.getPageSize());</pre> <p>最后将传入参数的 pid 强制转化成 HeapPageId 类型，用 HeapPage 构造函数，返回需要的 HeapPage。</p> <pre>res=new HeapPage((HeapPageId) pid,page_data);</pre>
DbFileIterator iterator(TransactionId tid)	<p>返回一个 DBFileIterator。DBFileIterator 是一个接口，里面有 open、hasnext、next、rewind、close 函数。</p> <p>为了实现这个迭代器，为 HeapFile 实现一个内部静态类 HeapFileIterator。该静态类实现了 DBFileIterator 接口，内部成员有 HeapFile、TransactionId、currentPage（记录当前页数）、Tuple 实例化的 Iterator。</p> <p>构造函数：将 heapfile 和 transactionid 赋初值。</p> <p>getTupleIterator(): 设置 tupleiterator。通过 HeapPageId 构造函数 new HeapPageId(heapFile.getId(),currentPage) 获取一个当前 page 的 id</p> <pre>return((HeapPage)(Database.getBufferPool().getPage(transactionId,pagId,Permissions.READ_ONLY))).iterator();</pre> <p>再通过 DataBase 的缓冲池调用 getpage 函数，读取出 Page，强制转化成 HeapPage 类型，返回其 iterator，迭代器关联的是该 Page 中非空的元组。</p> <p>open(): 设置当前 page 页为 0，调用 getTupleIterator 函数</p>

	<p>hasNext：判断迭代器是否有下一项。分为两种情况：1、当前的 <code>tupleiterator.hasNext</code> 为 <code>TRUE</code>，说明在这个 <code>page</code> 里的表还有下一行元组，返回 <code>TRUE</code>；2、当前的 <code>tupleiterator.hasNext</code> 为 <code>False</code>，但 <code>currentPage</code> 不是最后一个，此时进行一个翻页操作，就是将 <code>currentPage++</code>，然后重新设置 <code>tupleiterator</code>，返回 <code>tupleiterator.hasNext()</code></p> <p>next()：首先判断当前迭代器 <code>tupleiterator</code> 是否为空，若空，抛出异常；如果 <code>tupleiterator.hasNext()</code> 为真，则直接返回 <code>tupleiterator.next()</code>，注意这里不需要再进行翻页的判断和操作，因为翻页已经在 <code>hasNext()</code> 中进行了。</p>
--	---

5.Seqscan 运算符

<p>Seqscan：实现了 SimpleDB 中的运算符接口，Seqscan 实现的是一个用于遍历元组的类。SimpleDB 中的运算符内部只要有一个 <code>DBFileIterator</code> 对象，由于其他所有的子类都是实现了 <code>DBFileIterator</code> 接口的，就能借助 <code>DBFileIterator</code> 对象在不同 <code>DBFile</code> 子类下的内容进行访问运算。</p>	
<p>部分重点成员函数的实现</p>	
<code>open()</code>	<p>通过访问数据库的目录查找到需要访问的表所在的 <code>DBFile</code>，然后返回其 <code>iterator</code> 即 <code>Datebase.getCatalog().getDatabaseFile(tableId).iterator(transactionId)</code> 再调用 <code>dbFileIterator.open()</code></p>
<code>hasNext()</code> 、 <code>next()</code> 、 <code>close()</code> 、 <code>rewind()</code>	<p>都直接使用在 <code>open</code> 中得到的 <code>dbfileiterator</code> 对应的函数即可。</p>

6.Submit History 截图

Stupid-wangnz > SimpleDB > Commits		wangnz	
master	SimpleDB	Author	Search by message
24 Mar, 2022 1 commit			
 lab1 Done	Stupid-wangnz authored 3 days ago	1ae7214f	 
20 Mar, 2022 1 commit			
 lab1 Done	Stupid-wangnz authored 1 week ago	5db7911d	 
16 Mar, 2022 1 commit			
 lab1 补全并修正了tupledesc类	Stupid-wangnz authored 1 week ago	c2fbd316	 
13 Mar, 2022 13 commits			
 lab1-Exercise6(pass)-2022-3-13	Stupid-wangnz authored 1 week ago	3194c9fa	 
 Merge remote-tracking branch 'origin/master'	Stupid-wangnz authored 1 week ago	8f39a743	 
 lab1-Exercise6(pass)-2022-3-13	Stupid-wangnz authored 1 week ago	f65d27e2	 
 lab1-Exercise6(pass)	Stupid-wangnz authored 1 week ago	701f3514	 
 lab1-Exercise6(pass)-2022-3-13	Stupid-wangnz authored 1 week ago	f65d27e2	 
 lab1-Exercise6(pass)	Stupid-wangnz authored 1 week ago	701f3514	 
 lab1-Exercise6(pass)	Stupid-wangnz authored 1 week ago	f044f12d	 
 lab1-Exercise6(pass)	Stupid-wangnz authored 1 week ago	869d087f	 
 lab1-Exercise5(all pass)	Stupid-wangnz authored 1 week ago	f306b448	 
 lab1-Exercise4(all pass)	Stupid-wangnz authored 1 week ago	eab1e3a4	 
 lab1-Exercise4(RecordId)	Stupid-wangnz authored 2 weeks ago	68544308	 
 lab1-Exercise4(HeapPageld)	Stupid-wangnz authored 2 weeks ago	483d2e58	 
 lab1-Exercise3(未过test)	Stupid-wangnz authored 2 weeks ago	597a3de1	 
 lab1-Exercise2(all pass)	Stupid-wangnz authored 2 weeks ago	cec63cc3	 
 lab1-Exercise2(all pass)	Stupid-wangnz authored 2 weeks ago	8a6c8974	 
12 Mar, 2022 3 commits			
 lab1-Exercise1(all pass)	Stupid-wangnz authored 2 weeks ago	f50afc40	 
 lab1-Exercise1(完成)	Stupid-wangnz authored 2 weeks ago	55c61a14	 
 lab1-Exercise1(未完成)	Stupid-wangnz authored 2 weeks ago	7855ba51	 
19 Mar, 2021 1 commit			
 add out directory to .gitignore	NKU-DBIS-DB authored 1 year ago	Unverified 4e14e401	 
06 Mar, 2020 2 commits			
 Update .gitignore	NKU-DBIS-DB authored 2 years ago	Unverified 6a0540ea	 
 SimpleDB NKU 2020	DBIS NKU authored 2 years ago	1aa5633b	