simpleDB实验一实验报告

1. exercise1

1.1 设计思路

- TupleDesc类由多个TDItem组成，每个TDItem描述了Tuple中一个属性的名称和类型。TupleDesc的每个属性都有一个名称和类型，可以是可以支持不同类型的数据。TupleDesc的每个属性都有一个名称和类型，这些信息可以在查询和检索数据时用来识别和操作数据。
- Tuple类包含一个属性集合，可以动态添加或删除属性，以适应不同的数据类型和数据格式。Tuple类中的每个属性都也可以是任意类型的数据。
  Tuple类还提供了一些常用的方法来方便地访问和操作属性，例如根据属性名获取属性值、获取属性集合的大小、添加或删除属性等。

1.2 重难点

私有成员变量TDItems的定义。因为刚开始写，不太清楚要自己写一些私有成员变量。

1.3 改动部分

私有成员变量的定义

```
private final TDItem[] tdItems;
```

部分方法的实现

```
    public TupleDesc(Type[] typeAr, String[] fieldAr) {
        // some code goes here
         assert typeAr.length == fieldAr.length;

         tdItems = new TDItem[typeAr.length];
         for (int i = 0; i < typeAr.length; i++) {
             tdItems[i] = new TDItem(typeAr[i], fieldAr[i]);
         }
    }
```

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

```java
    public String getFieldName(int i) throws NoSuchElementException {
        // some code goes here
        if(i < 0 || i > this.numFields())
            throw new NoSuchElementException();
        else
            return tdItems[i].fieldName;
    }
```

```java
    public Type getFieldType(int i) throws NoSuchElementException {
        // some code goes here
        if(i < 0 || i > this.numFields())
            throw new NoSuchElementException();
        else
            return tdItems[i].fieldType;
    }
```

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

```java
    public int getSize() {
        // some code goes here
        int size = 0;
        for (TDItem item : tdItems) {
            size += item.fieldType.getLen();
        }
        return size;
    }
```

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

```
        for(int i = 0; i < td2.numFields(); i++)
        {
            newTypeArr[i + td1.numFields()] = td2.getFieldType(i);
            newFieldArr[i + td1.numFields()] = td2.getFieldName(i);
        }

        return new TupleDesc(newTypeArr,newFieldArr);
    }
```

```
    public boolean equals(Object o) {
        // some code goes here
        if (!(o instanceof TupleDesc)) {
            return false;
        }

        TupleDesc other = (TupleDesc) o;
        if (tdItems.length != other.tdItems.length) {
            return false;
        }

        for (int i = 0; i < tdItems.length; i++) {
            if (!tdItems[i].fieldType.equals(other.tdItems[i].fieldType)) {
                return false;
            }
        }

        return true;
    }
```

```
    public int hashCode() {
        int result = 0;
        for (TDItem item : tdItems) {
            result += item.fieldType.hashCode();
        }
        return result;

    }
```

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

2. exercise2

2.1 设计思路

Catalog类通常包含以下几个方面的功能：

- 数据表和列的管理：Catalog类提供了创建、删除、修改、查询数据表和列的方法。这些方法可以用于定义数据表的结构和属性，例如表的名称、列的名称、数据类型、长度和默认值等。
- 索引的管理：Catalog类还提供了创建、删除、修改、查询索引的方法。这些方法可以用于定义索引的结构和属性，例如索引的名称、类型、列的名称和排序方式等。
- 元数据的存储和检索：Catalog类通常将元数据信息存储在系统表或系统文件中，以便在查询和操作数据时能够快速检索和访问这些信息。

2.2 重难点

对ConcurrentHashMap的使用。

3.3 改动部分

私有成员变量的定义

```
        public final DbFile dbfile;
        public final String tableName;
        public final String pk;
```

部分方法的实现

```
    public int getTableId(String name) throws NoSuchElementException {
        // some code goes here
        Integer id = hashTable.searchValues(1, v->{
            if(v.tableName.equals(name))
                return v.dbfile.getId();
            return null;
            });
        if(id == null)
            throw new NoSuchElementException("Table " + name + " does not
exist");
        return id.intValue();
    }
```

```
    public TupleDesc getTupleDesc(int tableid) throws NoSuchElementException {
        // some code goes here
        Table tab = hashTable.getOrDefault(tableid,null);
        if(tab != null)
            return tab.dbfile.getTupleDesc();
        else
            throw new NoSuchElementException("TableId " + tableid + " does not
exist");
    }
```

3. exercise3

3.1 设计思路

- 缓存页读取：如果用户请求的数据块不在缓存区中，则需要从磁盘上读取该数据块。BufferPool类的getPage方法需要实现磁盘I/O操作，将数据块读取到一个新的缓存页中，并将该缓存页标记为已被占用。

3.2 重难点

缓存页的读取要考虑是否存在。

3.3 改动部分

私有成员变量的定义

```
    private final int numPages;
    private final ConcurrentHashMap<Integer,Page> pageStore;
```

部分方法的实现

```
    public BufferPool(int numPages) {
        // some code goes here
        this.numPages = numPages;
        pageStore = new ConcurrentHashMap<Integer,Page>();
    }
```

```
     public Page getPage(TransactionId tid, PageId pid, Permissions perm)
         throws TransactionAbortedException, DbException {
         // some code goes here
         if(!pageStore.containsKey(pid.hashCode())){
             DbFile dbfile =
Database.getCatalog().getDatabaseFile(pid.getTableId());
             Page page = dbfile.readPage(pid);
             pageStore.put(pid.hashCode(),page);
         }
         return pageStore.get(pid.hashCode());
     }
```

4 exercise4

4.1 设计思路

HeadPage类主要包括以下方面：

- 元数据信息的存储：HeadPage类主要用于存储文件的元数据信息，例如文件的名称、大小、创建时间、修改时间、块的数量等信息。这些元数据信息可以用于识别和管理数据库文件。
- 元数据信息的检索：HeadPage类提供了一些方法，例如getFileSize和getBlockCount等方法，用于检索文件的元数据信息。

- 元数据信息的修改：HeadPage类还提供了一些方法，例如setFileSize和setBlockCount等方法，用于修改文件的元数据信息。

4.2 重难点

空间大小的计算，

4.3改动部分

私有成员变量的定义

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

部分方法的实现

```
    private int getNumTuples() {
        // some code goes here
        //该页能容纳字节数*8/（元组所需字节数*8+1位记录元组是否有效）
        int num = (int)Math.floor((BufferPool.getPageSize() * 8.0)/(td.getSize() * 8 + 1));
        return num;

    }
```

```
     private int getHeaderSize() {

         // some code goes here
         //页头是用来存储每个元组状态（是否被删除）的位向量，
         //其中每个元组都对应位向量中的一个二进制位。
         //每个二进制位表示相应元组的状态，值为1表示元组已被删除，值为0表示元组未被删除。
         //返回单位为字节
         return (int)Math.ceil(getNumTuples() * 1.0 / 8);

     }
```

5 exercise5

5.1 设计思路

- 元数据管理：HeapFile 类需要维护一些元数据信息，例如表的名称、字段的类型、长度、偏移量等信息，以便于数据的读取和写入。
- 页管理：HeapFile 类需要将数据分割成固定大小的页（page），并维护每个页的元数据信息，例如页的编号、大小、数据起始位置等信息。

- 数据访问：HeapFile 类需要提供一系列方法和功能，例如通过主键获取数据、获取指定范围内的数据、添加新的数据、更新已有数据、删除数据等。

5.2 重难点

读取数据，HeapFileIterator类的实现。

5.3 改动部分

私有成员变量的定义

```
    private final File file;
    private final TupleDesc td;
```

部分方法的实现

```
    public Page readPage(PageId pid) {
        // some code goes here
        int pgNo = pid.getPageNumber();
        try {
         RandomAccessFile raf = new RandomAccessFile(file,"r");
         int offset = BufferPool.getPageSize() * pgNo;
         byte[] data = new byte[BufferPool.getPageSize()];
         raf.seek(offset);
         raf.read(data, 0, BufferPool.getPageSize());
         raf.close();
         return new HeapPage((HeapPageId) pid, data);

        } catch (IOException e) {
               throw new IllegalArgumentException("readPage: failed to read page");

        }
    }
```

```
    public class HeapFileIterator implements DbFileIterator {
        private TransactionId tid;
        private HeapFile heapFile;
        private int currentPageNum;
        private Iterator<Tuple> currentTupleIterator;

        public HeapFileIterator(TransactionId tid, HeapFile hf) {
            this.tid = tid;
            this.heapFile = hf;
            this.currentPageNum = -1;
            this.currentTupleIterator = null;
        }

        public void open() throws DbException, TransactionAbortedException {
            currentPageNum = 0;
```

```
            PageId pid = new HeapPageId(heapFile.getId(), currentPageNum);
            HeapPage page = (HeapPage) Database.getBufferPool().getPage(tid,
pid, Permissions.READ_ONLY);
            currentTupleIterator = page.iterator();
        }

        public boolean hasNext() throws DbException, TransactionAbortedException
{
            if (currentTupleIterator == null) {
                return false;
            }
            if (currentTupleIterator.hasNext()) {
                return true;
            } else {
                while (currentPageNum < heapFile.numPages() - 1) {
                    currentPageNum++;
                    PageId pid = new HeapPageId(heapFile.getId(),
currentPageNum);
                    HeapPage page = (HeapPage)
Database.getBufferPool().getPage(tid, pid, Permissions.READ_ONLY);
                    currentTupleIterator = page.iterator();
                    if (currentTupleIterator.hasNext()) {
                        return true;
                    }
                }
            }
            return false;
        }

        public Tuple next() throws DbException, TransactionAbortedException,
NoSuchElementException {
            if (currentTupleIterator == null || !currentTupleIterator.hasNext())
{
                throw new NoSuchElementException();
            }
            return currentTupleIterator.next();
        }

        public void rewind() throws DbException, TransactionAbortedException {
            close();
            open();
        }

        public void close() {
            currentPageNum = -1;
            currentTupleIterator = null;
        }
    }
```

6. exercise6

6.1 设计思路

- 数据访问：SeqScan 类需要提供一种通用的数据扫描方式，可以读取关系表中的所有数据，或者根据指定的查询条件进行筛选。

6.2 重难点

open，next，close，rewind等方法的实现。

6.3 改动部分

私有成员变量的定义

```
    private final TransactionId tid;
    private int tableId;
    private String tableAlias;
    private DbFileIterator it;
```

部分方法的实现

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

```
     public Tuple next() throws NoSuchElementException,
             TransactionAbortedException, DbException {
         // some code goes here
          if(it == null){
              throw new NoSuchElementException("no next tuple");
          }
          Tuple t = it.next();
          if(t == null){
              throw new NoSuchElementException("no next tuple");
          }
          return t;
     }
```

```
    public void close() {
        // some code goes here
        it = null;
    }
```

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

7 实验提交记录