simpleDB实验二实验报告

1. exercise1

1.1 设计思路

- Predicate类和JoinPredicate类中实现基本的运算符
- filter类中实现迭代器
- join类中实现迭代器，需要考虑两组字段

1.2 重难点

- filter方法的实现
- 迭代器的重写

1.3 改动部分

私有成员变量的定义

```
private final int field;
private final Op op;
private final Field operand;

```

```
private final int field1;
private final int field2;
private final Predicate.Op op;
```

```
private final Predicate p;
private OpIterator child;
```

```
private final JoinPredicate p;
private OpIterator child1;
private OpIterator child2;
private Tuple leftTuple = null;
```

部分方法的实现

```
 public String toString() {
        // some code goes here
        String str = String.format("f = %s op = %s operand = %s",
((Integer)field).toString(), op.toString(), operand.toString() );
        return str;
    }
```

```
public boolean filter(Tuple t) {
        // some code goes here
        Field f = t.getField(field);
        if(f.compare(op, operand))
            return true;
        return false;
    }
```

```
public boolean filter(Tuple t1, Tuple t2) {
    // some code goes here
    if(t1.getField(field1).compare(op, t2.getField(field2)))
        return true;
    return false;
}
```

```
protected Tuple fetchNext() throws TransactionAbortedException, DbException {
     // some code goes here

     Tuple rightTuple = null;
     while(child1.hasNext() || leftTuple != null)
     {
         if(child1.hasNext() && leftTuple == null)
         {
             leftTuple = child1.next();
         }
         while(child2.hasNext())
         {
             rightTuple = child2.next();
             if(p.filter(leftTuple,rightTuple))
             {
                 Tuple t = new Tuple(this.getTupleDesc());
                 int numFields = leftTuple.getTupleDesc().numFields() +
rightTuple.getTupleDesc().numFields();
                 t.setRecordId(leftTuple.getRecordId());
                 for (int i = 0; i < numFields; i++)
                 {
                     if (i < leftTuple.getTupleDesc().numFields())
                     {
                         t.setField(i, leftTuple.getField(i));
                     } else
                     {
                         t.setField(i, rightTuple.getField(i -
leftTuple.getTupleDesc().numFields()));
                     }
                 }
                 //System.out.println(t.toString());
```

```
                    return t;
                }
            }
            child2.rewind();
            leftTuple = null;
        }
        return null;


}
```

2. exercise2

2.1 设计思路

- IntegerAggregator用于对类型为各种INT_TYPE的字段进行求和、计数等操作。
- StringAggregator用于对类型为STRING_TYPE的字段进行计数操作。
- Aggregate通过调用上述两种类来实现对各种字段的聚合处理。

2.2 重难点

- 对字段进行分类、分操作处理。
- 迭代器类的实现。

2.3 改动部分

私有成员变量的定义

```
private int gbfield;
private Type gbfieldtype;
private int afield;
private Op what;

// running SUM,MIN,MAX,COUNT
private Map<Field,Integer> groupMap;
private Map<Field,Integer> countMap;
private Map<Field,List<Integer>> avgMap;
```

```
private int gbfield;
private Type gbfieldtype;
private int afield;
private Op what;

private Map<Field, Integer> groupMap;
```

```
private  OpIterator child;
private final int afield;
private final int gfield;
private final Aggregator.Op aop;

private Aggregator aggregator;
private OpIterator it;
private TupleDesc td;
```

部分方法的实现

```
public void mergeTupleIntoGroup(Tuple tup) {
    // some code goes here
    IntField afield = (IntField)tup.getField(this.afield);
    Field gbfield = this.gbfield == NO_GROUPING ? null :
tup.getField(this.gbfield);
    int newValue = afield.getValue();
    if(gbfield != null && gbfield.getType()!=this.gbfieldtype){
        throw new IllegalArgumentException("Given tuple has wrong type");
    }
    // get number
    switch(this.what){
        case MIN:
            if(!this.groupMap.containsKey(gbfield))
                this.groupMap.put(gbfield,newValue);
            else

 this.groupMap.put(gbfield,Math.min(this.groupMap.get(gbfield),newValue));
            break;

        case MAX:
            if (!this.groupMap.containsKey(gbfield))
                this.groupMap.put(gbfield, newValue);
            else
                this.groupMap.put(gbfield, Math.max(this.groupMap.get(gbfield),
newValue));
            break;

        case SUM:
            if (!this.groupMap.containsKey(gbfield))
                this.groupMap.put(gbfield, newValue);
            else
                this.groupMap.put(gbfield, this.groupMap.get(gbfield) +
newValue);
            break;

        case COUNT:
            if (!this.groupMap.containsKey(gbfield))
                this.groupMap.put(gbfield, 1);
            else
                this.groupMap.put(gbfield, this.groupMap.get(gbfield) + 1);
```

```
            break;

        case SC_AVG:
            IntField countField = null;
            if (gbfield == null)
                countField = (IntField)tup.getField(1);
            else
                countField = (IntField)tup.getField(2);
            int countValue = countField.getValue();
            if (!this.groupMap.containsKey(gbfield)) {
                this.groupMap.put(gbfield, newValue);
                this.countMap.put(gbfield, countValue);
            } else {
                this.groupMap.put(gbfield, this.groupMap.get(gbfield) +
newValue);
                this.countMap.put(gbfield, this.countMap.get(gbfield) +
countValue);
            }
        case SUM_COUNT:

        case AVG:
            if (!this.avgMap.containsKey(gbfield)) {
                List<Integer> l = new ArrayList<>();
                l.add(newValue);
                this.avgMap.put(gbfield, l);
            } else {
                // reference
                List<Integer> l = this.avgMap.get(gbfield);
                l.add(newValue);
            }
            break;
        default:
            throw new IllegalArgumentException("Aggregate not supported!");
    }
}
```

```
public void mergeTupleIntoGroup(Tuple tup) {
    // some code goes here
    StringField afield = (StringField) tup.getField(this.afield);
    Field gbfield = this.gbfield == NO_GROUPING ? null :
tup.getField(this.gbfield);
    //String newValue = afield.getValue();
    if (gbfield != null && gbfield.getType() != this.gbfieldtype) {
        throw new IllegalArgumentException("Given tuple has wrong type");
    }
    if (!this.groupMap.containsKey(gbfield))
        this.groupMap.put(gbfield, 1);
    else
        this.groupMap.put(gbfield, this.groupMap.get(gbfield) + 1);
}
```

```java
class AggregateIterator implements OpIterator {

    protected Iterator<Map.Entry<Field, Integer>> it;
    TupleDesc td;

    private Map<Field, Integer> groupMap;
    protected Type itgbfieldtype;

    public AggregateIterator(Map<Field, Integer> groupMap, Type gbfieldtype) {
        this.groupMap = groupMap;
        this.itgbfieldtype = gbfieldtype;
        // no grouping
        if (this.itgbfieldtype == null)
            this.td = new TupleDesc(new Type[] {Type.INT_TYPE}, new String[]
{"aggregateVal"});
        else
            this.td = new TupleDesc(new Type[] {this.itgbfieldtype,
Type.INT_TYPE}, new String[] {"groupVal", "aggregateVal"});
    }


    @Override
    public void open() throws DbException, TransactionAbortedException {
        this.it = groupMap.entrySet().iterator();
    }

    @Override
    public boolean hasNext() throws DbException, TransactionAbortedException {
        return it.hasNext();
    }

    @Override
    public Tuple next() throws DbException, TransactionAbortedException,
NoSuchElementException {
        Map.Entry<Field, Integer> entry = this.it.next();
        Field f = entry.getKey();
        Tuple rtn = new Tuple(this.td);
        this.setFields(rtn, entry.getValue(), f);
        return rtn;
    }

    @Override
    public void rewind() throws DbException, TransactionAbortedException {
        this.it = groupMap.entrySet().iterator();
    }

    @Override
    public TupleDesc getTupleDesc() {
        return this.td;
    }

    @Override
```

```
    public void close() {
        this.it = null;
        this.td = null;
    }

    void setFields(Tuple rtn, int value, Field f) {
        if (f == null) {
            rtn.setField(0, new IntField(value));
        } else {
            rtn.setField(0, f);
            rtn.setField(1, new IntField(value));
        }
    }
}
```

```
public Aggregate(OpIterator child, int afield, int gfield, Aggregator.Op aop) {
// some code goes here
    this.child = child;
    this.afield = afield;
    this.gfield = gfield;
    this.aop = aop;


    Type gfieldtype = gfield == -1 ? null :
this.child.getTupleDesc().getFieldType(this.gfield);

    if(this.child.getTupleDesc().getFieldType(this.afield) ==
(Type.STRING_TYPE)){
        this.aggregator = new
StringAggregator(this.gfield,gfieldtype,this.afield,this.aop);
    }else{
        this.aggregator = new
IntegerAggregator(this.gfield,gfieldtype,this.afield,this.aop);
    }
    this.it = this.aggregator.iterator();
    // create tupleDesc for agg
    List<Type> types = new ArrayList<>();
    List<String> names = new ArrayList<>();
    // group field
    if (gfieldtype != null) {
        types.add(gfieldtype);
        names.add(this.child.getTupleDesc().getFieldName(this.gfield));
    }
    types.add(this.child.getTupleDesc().getFieldType(this.afield));
    names.add(this.child.getTupleDesc().getFieldName(this.afield));
    if (aop.equals(Aggregator.Op.SUM_COUNT)) {
        types.add(Type.INT_TYPE);
        names.add("COUNT");
    }
    assert (types.size() == names.size());
```

```
    this.td = new TupleDesc(types.toArray(new Type[types.size()]),
names.toArray(new String[names.size()]));

}
```

```
public void setChildren(OpIterator[] children) {
// some code goes here
     this.child = children[0];
     List<Type> types = new ArrayList<>();
     List<String> names = new ArrayList<>();
     Type gfieldtype = gfield == -1 ? null :
this.child.getTupleDesc().getFieldType(this.gfield);
     // group field
     if (gfieldtype != null) {
         types.add(gfieldtype);
         names.add(this.child.getTupleDesc().getFieldName(this.gfield));
     }
     types.add(this.child.getTupleDesc().getFieldType(this.afield));
     names.add(this.child.getTupleDesc().getFieldName(this.afield));
     if (aop.equals(Aggregator.Op.SUM_COUNT)) {
         types.add(Type.INT_TYPE);
         names.add("COUNT");
     }
     assert (types.size() == names.size());
     this.td = new TupleDesc(types.toArray(new Type[types.size()]),
names.toArray(new String[names.size()]));
}
```

3. exercise3

3.1 设计思路

- HeapPage类用于实现元组的插入和删除，并标记该页是否被修改（dirty）。
- HeapFile类通过调用page的方法对page进行插入和删除元组操作。
- BufferPool类里的插入和删除元组的方法通过调用page的方法来实现。

3.2 重难点

注意同步修改RecordID，并且在实现HeapPage注意考虑周全，比如oage不存在等情况。

3.3 改动部分

私有成员变量的定义

```
private TransactionId dirtyId;
private boolean dirty;
```

部分方法的实现

```java
public void deleteTuple(Tuple t) throws DbException {
    // some code goes here
    // not necessary for lab1
    int tid = t.getRecordId().getTupleNumber();
    if(tuples[tid] == null)
    {
        throw new DbException("This tuple is not exist");
    }
    if(!isSlotUsed(tid))
    {
        throw new DbException("This slot is empty");
    }

   else if(tuples[tid] == null|||!t.getTupleDesc().equals(td) ||
!t.getRecordId().getPageId().equals(pid))
   {
       //System.out.println(tuples[tid]);
       //System.out.println(t);
        throw new DbException(String.format("tuple does not exits %d and
%d",t.getRecordId().hashCode(),tuples[tid].getRecordId().hashCode()));
    }

    else
    {
        this.markSlotUsed(tid, false);
        tuples[tid] = null;
    }

}
```

```java
public void insertTuple(Tuple t) throws DbException {
    // some code goes here
    // not necessary for lab1
    if(this.getNumEmptySlots() == 0 || !t.getTupleDesc().equals(td))
    {
        throw new DbException("the page is full  or tupledesc is mismatch");
    }
    for(int i = 0; i < numSlots; i++)
    {
        if(!this.isSlotUsed(i))
        {
            this.markSlotUsed(i, true);
            t.setRecordId(new RecordId(pid,i));
            tuples[i] = t;
            break;
        }
    }
}
```

```java
public void markDirty(boolean dirty, TransactionId tid) {
    // some code goes here
```

```
// not necessary for lab1
    this.dirty = dirty;
    this.dirtyId = tid;
}

/**
 * Returns the tid of the transaction that last dirtied this page, or null if the page is not dirty
 */
public TransactionId isDirty() {
    // some code goes here
// Not necessary for lab1
    if(this.dirty)
        return dirtyId;
    else
        return null;
}
```

```
private void markSlotUsed(int i, boolean value) {
    // some code goes here
    // not necessary for lab1
    int quot = i / 8;
    int remain = i % 8;
    byte b = header[quot];
    byte b2 = (byte)(1<<remain);
    if(value)
    {
        header[quot] = (byte)(b | b2);
    }
    else
    {
        header[quot] = (byte)(b & (~b2));
    }
}

/**
 * @return an iterator over all tuples on this page (calling remove on this iterator throws an UnsupportedOperationException)
 * (note that this iterator shouldn't return tuples in empty slots!)
 */
public Iterator<Tuple> iterator() {
    // some code goes here
    //所有已被占用的槽位的元组的迭代器
    ArrayList<Tuple> temp = new ArrayList<Tuple>();
    for(int i = 0; i < numSlots; i++)
        if(isSlotUsed(i))
            temp.add(tuples[i]);
        return temp.iterator();
    }
```

```
public ArrayList<Page> insertTuple(TransactionId tid, Tuple t)
```

```
        throws DbException, IOException, TransactionAbortedException {
    // some code goes here
    // not necessary for lab1
    ArrayList<Page> pageList= new ArrayList<Page>();
    for(int i=0;i<numPages();++i){
        // took care of getting new page
        HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
                new HeapPageId(this.getId(),i),Permissions.READ_WRITE);
        if(p.getNumEmptySlots() == 0)
            continue;
        p.insertTuple(t);
        pageList.add(p);
        return pageList;
    }
    // no new page
    BufferedOutputStream bw = new BufferedOutputStream(new
FileOutputStream(file,true));
    byte[] emptyData = HeapPage.createEmptyPageData();
    bw.write(emptyData);
    bw.close();
    // load into cache
    HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
            new HeapPageId(getId(),numPages()-1),Permissions.READ_WRITE);
    p.insertTuple(t);
    pageList.add(p);
    return pageList;
}
```

```
public ArrayList<Page> deleteTuple(TransactionId tid, Tuple t) throws
DbException,
        TransactionAbortedException {
    // some code goes here
     ArrayList<Page> pageList = new ArrayList<Page>();
     HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
             t.getRecordId().getPageId(),Permissions.READ_WRITE);
     p.deleteTuple(t);
     pageList.add(p);
     return pageList;

    // not necessary for lab1
}
```

```
public void insertTuple(TransactionId tid, int tableId, Tuple t)
    throws DbException, IOException, TransactionAbortedException {
    // some code goes here
    // not necessary for lab1
    DbFile f = Database.getCatalog().getDatabaseFile(tableId);
    updateBufferPool(f.insertTuple(tid,t),tid);
}
```

```
private void updateBufferPool(ArrayList<Page> pagelist,TransactionId tid) throws DbException{
    for(Page p:pagelist){
        p.markDirty(true,tid);
        // update bufferpool
        if(pageStore.size() > numPages)
            evictPage();
        pageStore.put(p.getId(),p);
    }
}
```

```
public  void deleteTuple(TransactionId tid, Tuple t)
    throws DbException, IOException, TransactionAbortedException {
    // some code goes here
    // not necessary for lab1
    DbFile f = Database.getCatalog().getDatabaseFile(t.getRecordId().getPageId().getTableId());
    updateBufferPool(f.deleteTuple(tid,t),tid);
}-
```

4. exercise4

4.1 设计思路

- Insert类用于将元组插入到tableId代表的表中，通过调用BufferPool.insertTuple()方法实现。
- Delete类用于删除指定元组，通过调用BufferPool.DeleteTuple()方法实现。

4.2 重难点

重写迭代器。

4.3 改动部分

私有成员变量的定义

```
private TransactionId tid;
private OpIterator child;
private int tableId;
private final TupleDesc td;

private int counter;
private boolean called;
```

```
private TransactionId tid;
private OpIterator child;
private final TupleDesc td;

private int counter;
private boolean called;
```

部分方法的实现

```
public Delete(TransactionId t, OpIterator child) {
    // some code goes here
    this.tid = t;
    this.child = child;
    this.td = new TupleDesc(new Type[] {Type.INT_TYPE}, new String[] {"number of deleted tuples"});
    this.counter = -1;
    this.called = false;
}

public TupleDesc getTupleDesc() {
    // some code goes here
    return td;
}

public void open() throws DbException, TransactionAbortedException {
    // some code goes here
    counter = 0;
    child.open();
    super.open();
}

public void close() {
    // some code goes here
    super.close();
    child.close();
    counter = -1;
    called = false;
}

public void rewind() throws DbException, TransactionAbortedException {
    // some code goes here
    child.rewind();
    counter = 0;
    called = false;
}

/**
 * Deletes tuples as they are read from the child operator. Deletes are
 * processed via the buffer pool (which can be accessed via the
 * Database.getBufferPool() method.
 *
```

```
 * @return A 1-field tuple containing the number of deleted records.
 * @see Database#getBufferPool
 * @see BufferPool#deleteTuple
 */
protected Tuple fetchNext() throws TransactionAbortedException, DbException {
    // some code goes here
    if (this.called)
        return null;

    this.called = true;
    while (this.child.hasNext()) {
        Tuple t = this.child.next();
        try {
            Database.getBufferPool().deleteTuple(this.tid, t);
            this.counter++;
        } catch (IOException e) {
            e.printStackTrace();
            break;
        }
    }
    Tuple tu = new Tuple(this.td);
    tu.setField(0, new IntField(this.counter));
    return tu;
}

@Override
public OpIterator[] getChildren() {
    // some code goes here
    return new OpIterator[] {child};
}

@Override
public void setChildren(OpIterator[] children) {
    // some code goes here
    child = children[0];
}
```

```
public Insert(TransactionId t, OpIterator child, int tableId)
        throws DbException {
    // some code goes here

 if(!child.getTupleDesc().equals(Database.getCatalog().getTupleDesc(tableId))){
        throw new DbException("TupleDesc does not match!");
    }
    this.tid = t;
    this.child = child;
    this.tableId = tableId;
    this.td = new TupleDesc(new Type[]{Type.INT_TYPE},new String[]{"number of
inserted tuples"});
    this.counter = -1;
    this.called = false;
```

```
}

public TupleDesc getTupleDesc() {
    // some code goes here
    return td;
}

public void open() throws DbException, TransactionAbortedException {
    // some code goes here
    counter = 0;
    child.open();
    super.open();
}

public void close() {
    // some code goes here
    super.close();
    child.close();
    counter = -1;
    called = false;
}

public void rewind() throws DbException, TransactionAbortedException {
    // some code goes here
    child.rewind();
    counter = 0;
    called = false;
}

/**
 * Inserts tuples read from child into the tableId specified by the
 * constructor. It returns a one field tuple containing the number of
 * inserted records. Inserts should be passed through BufferPool. An
 * instances of BufferPool is available via Database.getBufferPool(). Note
 * that insert DOES NOT need check to see if a particular tuple is a
 * duplicate before inserting it.
 *
 * @return A 1-field tuple containing the number of inserted records, or
 *         null if called more than once.
 * @see Database#getBufferPool
 * @see BufferPool#insertTuple
 */
protected Tuple fetchNext() throws TransactionAbortedException, DbException {
    // some code goes here
    if (this.called)
        return null;

    this.called = true;
    while (this.child.hasNext()) {
        Tuple t = this.child.next();
        try {
            Database.getBufferPool().insertTuple(this.tid, this.tableId, t);
```

```
            this.counter++;
        } catch (IOException e) {
            e.printStackTrace();
            break;
        }
    }
    Tuple tu = new Tuple(this.td);
    tu.setField(0, new IntField(this.counter));
    return tu;
}

@Override
public OpIterator[] getChildren() {
    // some code goes here
    return new OpIterator[] {child};

}

@Override
public void setChildren(OpIterator[] children) {
    // some code goes here
    child = children[0];
}
```

5. exercise5

5.1 设计思路

- 当bufferPool容纳的页数达到上限时，需要驱逐一些页。
- 如果该页面未被修改过，则该页面是最好的选择，因为它可以直接被驱逐出缓冲池，而无需写回到磁盘。
- 如果该页面已经被修改过，则它需要被写回到磁盘，因此，如果缓冲池中有多个被修改过的页面，优先选择最近未被访问的页面来驱逐，以最大程度地减少写回到磁盘的次数。

5.2 重难点

驱逐前判断该页是否被修改。

5.3 改动部分

部分方法的实现

```
    public synchronized void flushAllPages() throws IOException {
        // some code goes here
        // not necessary for lab1
        for(Page p : this.pageStore.values())
            flushPage(p.getId());
    }
    public synchronized void discardPage(PageId pid) {
        // some code goes here
```

```
        // not necessary for lab1
        pageStore.remove(pid);
    }

    private synchronized  void flushPage(PageId pid) throws IOException {
        // some code goes here
        // not necessary for lab1
        Page p = pageStore.get(pid);
        TransactionId tid = null;
        // flush it if it is dirty
        if((tid = p.isDirty())!= null){
            Database.getLogFile().logWrite(tid,p.getBeforeImage(),p);
            Database.getLogFile().force();
            // write to disk

Database.getCatalog().getDatabaseFile(pid.getTableId()).writePage(p);
            p.markDirty(false,null);
        }

    }
    private synchronized  void evictPage() throws DbException {
        // some code goes here
        // not necessary for lab1
         PageId pid = new ArrayList<>(pageStore.keySet()).get(0);
         try{
             flushPage(pid);
         }catch(IOException e){
             e.printStackTrace();
         }
         discardPage(pid);

    }
```

6. 实验提交记录