

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

```
1 private final int field;  
2 private final Op op;  
3 private final Field operand;  
4
```

```
1 private final int field1;  
2 private final int field2;  
3 private final Predicate.Op op;
```

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

```
1 private final JoinPredicate p;  
2 private OpIterator child1;  
3 private OpIterator child2;  
4 private Tuple leftTuple = null;
```

部分方法的实现

```
1 public String toString() {  
2     // some code goes here  
3     String str = String.format("f = %s op = %s operand = %s",  
4     ((Integer)field).toString(), op.toString(), operand.toString() );  
5     return str;  
6 }
```

```

1 public boolean filter(Tuple t) {
2     // some code goes here
3     Field f = t.getField(field);
4     if(f.compare(op, operand))
5         return true;
6     return false;
7 }

```

```

1 public boolean filter(Tuple t1, Tuple t2) {
2     // some code goes here
3     if(t1.getField(field1).compare(op, t2.getField(field2)))
4         return true;
5     return false;
6 }

```

```

1 protected Tuple fetchNext() throws TransactionAbortedException, DbException {
2     // some code goes here
3
4     Tuple rightTuple = null;
5     while(child1.hasNext() || leftTuple != null)
6     {
7         if(child1.hasNext() && leftTuple == null)
8         {
9             leftTuple = child1.next();
10        }
11        while(child2.hasNext())
12        {
13            rightTuple = child2.next();
14            if(p.filter(leftTuple, rightTuple))
15            {
16                Tuple t = new Tuple(this.getTupleDesc());
17                int numFields = leftTuple.getTupleDesc().numFields() +
rightTuple.getTupleDesc().numFields();
18                t.setRecordId(leftTuple.getRecordId());
19                for (int i = 0; i < numFields; i++)
20                {
21                    if (i < leftTuple.getTupleDesc().numFields())
22                    {
23                        t.setField(i, leftTuple.getField(i));
24                    } else
25                    {
26                        t.setField(i, rightTuple.getField(i -
leftTuple.getTupleDesc().numFields()));
27                    }
28                }
29                //System.out.println(t.toString());

```

```

30         return t;
31     }
32 }
33 child2.rewind();
34 leftTuple = null;
35 }
36 return null;
37
38
39 }

```

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

```

1 private int gbfield;
2 private Type gbfieldtype;
3 private int afield;
4 private Op what;
5
6 // running SUM,MIN,MAX,COUNT
7 private Map<Field,Integer> groupMap;
8 private Map<Field,Integer> countMap;
9 private Map<Field,List<Integer>> avgMap;

```

```

1 private int gbfield;
2 private Type gbfieldtype;
3 private int afield;
4 private Op what;
5
6 private Map<Field, Integer> groupMap;

```

```

1 private OpIterator child;
2 private final int afield;
3 private final int gfield;
4 private final Aggregator.Op aop;
5
6 private Aggregator aggregator;
7 private OpIterator it;
8 private TupleDesc td;

```

部分方法的实现

```

1 public void mergeTupleIntoGroup(Tuple tup) {
2     // some code goes here
3     IntField afield = (IntField)tup.getField(this.afield);
4     Field gbfield = this.gbfield == NO_GROUPING ? null :
5     tup.getField(this.gbfield);
6     int newValue = afield.getValue();
7     if(gbfield != null && gbfield.getType() != this.gbfieldtype){
8         throw new IllegalArgumentException("Given tuple has wrong type");
9     }
10    // get number
11    switch(this.what){
12        case MIN:
13            if(!this.groupMap.containsKey(gbfield))
14                this.groupMap.put(gbfield, newValue);
15            else
16                this.groupMap.put(gbfield, Math.min(this.groupMap.get(gbfield), newValue));
17            break;
18        case MAX:
19            if (!this.groupMap.containsKey(gbfield))
20                this.groupMap.put(gbfield, newValue);
21            else
22                this.groupMap.put(gbfield, Math.max(this.groupMap.get(gbfield),
23                newValue));
24            break;
25        case SUM:
26            if (!this.groupMap.containsKey(gbfield))
27                this.groupMap.put(gbfield, newValue);
28            else
29                this.groupMap.put(gbfield, this.groupMap.get(gbfield) +
30                newValue);
31            break;
32        case COUNT:
33            if (!this.groupMap.containsKey(gbfield))
34                this.groupMap.put(gbfield, 1);
35            else
36                this.groupMap.put(gbfield, this.groupMap.get(gbfield) + 1);

```

```

37         break;
38
39     case SC_AVG:
40         IntField countField = null;
41         if (gbfield == null)
42             countField = (IntField)tup.getField(1);
43         else
44             countField = (IntField)tup.getField(2);
45         int countValue = countField.getValue();
46         if (!this.groupMap.containsKey(gbfield)) {
47             this.groupMap.put(gbfield, newValue);
48             this.countMap.put(gbfield, countValue);
49         } else {
50             this.groupMap.put(gbfield, this.groupMap.get(gbfield) +
newValue);
51             this.countMap.put(gbfield, this.countMap.get(gbfield) +
countValue);
52         }
53     case SUM_COUNT:
54
55     case AVG:
56         if (!this.avgMap.containsKey(gbfield)) {
57             List<Integer> l = new ArrayList<>();
58             l.add(newValue);
59             this.avgMap.put(gbfield, l);
60         } else {
61             // reference
62             List<Integer> l = this.avgMap.get(gbfield);
63             l.add(newValue);
64         }
65         break;
66     default:
67         throw new IllegalArgumentException("Aggregate not supported!");
68     }
69 }

```

```

1 public void mergeTupleIntoGroup(Tuple tup) {
2     // some code goes here
3     StringField afield = (StringField) tup.getField(this.afield);
4     Field gbfield = this.gbfield == NO_GROUPING ? null :
tup.getField(this.gbfield);
5     //String newValue = afield.getValue();
6     if (gbfield != null && gbfield.getType() != this.gbfieldtype) {
7         throw new IllegalArgumentException("Given tuple has wrong type");
8     }
9     if (!this.groupMap.containsKey(gbfield))
10         this.groupMap.put(gbfield, 1);
11     else
12         this.groupMap.put(gbfield, this.groupMap.get(gbfield) + 1);
13 }

```

```

1  class AggregateIterator implements OpIterator {
2
3      protected Iterator<Map.Entry<Field, Integer>> it;
4      TupleDesc td;
5
6      private Map<Field, Integer> groupMap;
7      protected Type itgbfieldtype;
8
9      public AggregateIterator(Map<Field, Integer> groupMap, Type gbfieldtype) {
10         this.groupMap = groupMap;
11         this.itgbfieldtype = gbfieldtype;
12         // no grouping
13         if (this.itgbfieldtype == null)
14             this.td = new TupleDesc(new Type[] {Type.INT_TYPE}, new String[]
{"aggregateVal"});
15         else
16             this.td = new TupleDesc(new Type[] {this.itgbfieldtype,
Type.INT_TYPE}, new String[] {"groupVal", "aggregateVal"});
17     }
18
19
20     @Override
21     public void open() throws DbException, TransactionAbortedException {
22         this.it = groupMap.entrySet().iterator();
23     }
24
25     @Override
26     public boolean hasNext() throws DbException, TransactionAbortedException {
27         return it.hasNext();
28     }
29
30     @Override
31     public Tuple next() throws DbException, TransactionAbortedException,
NoSuchElementException {
32         Map.Entry<Field, Integer> entry = this.it.next();
33         Field f = entry.getKey();
34         Tuple rtn = new Tuple(this.td);
35         this.setFields(rtn, entry.getValue(), f);
36         return rtn;
37     }
38
39     @Override
40     public void rewind() throws DbException, TransactionAbortedException {
41         this.it = groupMap.entrySet().iterator();
42     }
43
44     @Override
45     public TupleDesc getTupleDesc() {
46         return this.td;
47     }
48
49     @Override

```

```

50     public void close() {
51         this.it = null;
52         this.td = null;
53     }
54
55     void setFields(Tuple rtn, int value, Field f) {
56         if (f == null) {
57             rtn.setField(0, new IntField(value));
58         } else {
59             rtn.setField(0, f);
60             rtn.setField(1, new IntField(value));
61         }
62     }
63 }

```

```

1  public Aggregate(OpIterator child, int afield, int gfield, Aggregator.Op aop) {
2      // some code goes here
3      this.child = child;
4      this.afield = afield;
5      this.gfield = gfield;
6      this.aop = aop;
7
8
9      Type gfieldtype = gfield == -1 ? null :
this.child.getTupleDesc().getFieldType(this.gfield);
10
11      if(this.child.getTupleDesc().getFieldType(this.afield) ==
(Type.STRING_TYPE)){
12          this.aggregator = new
StringAggregator(this.gfield,gfieldtype,this.afield,this.aop);
13      }else{
14          this.aggregator = new
IntegerAggregator(this.gfield,gfieldtype,this.afield,this.aop);
15      }
16      this.it = this.aggregator.iterator();
17      // create tupleDesc for agg
18      List<Type> types = new ArrayList<>();
19      List<String> names = new ArrayList<>();
20      // group field
21      if (gfieldtype != null) {
22          types.add(gfieldtype);
23          names.add(this.child.getTupleDesc().getFieldName(this.gfield));
24      }
25      types.add(this.child.getTupleDesc().getFieldType(this.afield));
26      names.add(this.child.getTupleDesc().getFieldName(this.afield));
27      if (aop.equals(Aggregator.Op.SUM_COUNT)) {
28          types.add(Type.INT_TYPE);
29          names.add("COUNT");
30      }
31      assert (types.size() == names.size());

```

```

32     this.td = new TupleDesc(types.toArray(new Type[types.size()]),
names.toArray(new String[names.size()]));
33
34 }

```

```

1 public void setChildren(OpIterator[] children) {
2     // some code goes here
3     this.child = children[0];
4     List<Type> types = new ArrayList<>();
5     List<String> names = new ArrayList<>();
6     Type gfieldtype = gfield == -1 ? null :
this.child.getTupleDesc().getFieldType(this.gfield);
7     // group field
8     if (gfieldtype != null) {
9         types.add(gfieldtype);
10        names.add(this.child.getTupleDesc().getFieldName(this.gfield));
11    }
12    types.add(this.child.getTupleDesc().getFieldType(this.affield));
13    names.add(this.child.getTupleDesc().getFieldName(this.affield));
14    if (aop.equals(Aggregator.Op.SUM_COUNT)) {
15        types.add(Type.INT_TYPE);
16        names.add("COUNT");
17    }
18    assert (types.size() == names.size());
19    this.td = new TupleDesc(types.toArray(new Type[types.size()]),
names.toArray(new String[names.size()]));
20 }

```

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

```

1 private TransactionId dirtyId;
2 private boolean dirty;

```

部分方法的实现


```

1 public void deleteTuple(Tuple t) throws DbException {
2     // some code goes here
3     // not necessary for lab1
4     int tid = t.getRecordId().getTupleNumber();
5     if(tuples[tid] == null)
6     {
7         throw new DbException("This tuple is not exist");
8     }
9     if(!isSlotUsed(tid))
10    {
11        throw new DbException("This slot is empty");
12    }
13
14    else if(tuples[tid] == null || !t.getTupleDesc().equals(td) ||
!t.getRecordId().getPageId().equals(pid))
15    {
16        //System.out.println(tuples[tid]);
17        //System.out.println(t);
18        throw new DbException(String.format("tuple does not exists %d and
%d", t.getRecordId().hashCode(), tuples[tid].getRecordId().hashCode()));
19    }
20
21    else
22    {
23        this.markSlotUsed(tid, false);
24        tuples[tid] = null;
25    }
26
27 }

```

```

1 public void insertTuple(Tuple t) throws DbException {
2     // some code goes here
3     // not necessary for lab1
4     if(this.getNumEmptySlots() == 0 || !t.getTupleDesc().equals(td))
5     {
6         throw new DbException("the page is full or tupledesc is mismatch");
7     }
8     for(int i = 0; i < numSlots; i++)
9     {
10        if(!this.isSlotUsed(i))
11        {
12            this.markSlotUsed(i, true);
13            t.setRecordId(new RecordId(pid,i));
14            tuples[i] = t;
15            break;
16        }
17    }
18 }

```

```

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

```

```

3 // not necessary for lab1
4     this.dirty = dirty;
5     this.dirtyId = tid;
6 }
7
8 /**
9  * Returns the tid of the transaction that last dirtied this page, or null if
10  * the page is not dirty
11  */
12 public TransactionId isDirty() {
13     // some code goes here
14     // Not necessary for lab1
15     if(this.dirty)
16         return dirtyId;
17     else
18         return null;
19 }

```

```

1 private void markSlotUsed(int i, boolean value) {
2     // some code goes here
3     // not necessary for lab1
4     int quot = i / 8;
5     int remain = i % 8;
6     byte b = header[quot];
7     byte b2 = (byte)(1<<remain);
8     if(value)
9     {
10         header[quot] = (byte)(b | b2);
11     }
12     else
13     {
14         header[quot] = (byte)(b & (~b2));
15     }
16 }
17
18 /**
19  * @return an iterator over all tuples on this page (calling remove on this
20  * iterator throws an UnsupportedOperationException)
21  * (note that this iterator shouldn't return tuples in empty slots!)
22  */
23 public Iterator<Tuple> iterator() {
24     // some code goes here
25     //所有已被占用的槽位的元组的迭代器
26     ArrayList<Tuple> temp = new ArrayList<Tuple>();
27     for(int i = 0; i < numSlots; i++)
28         if(isSlotUsed(i))
29             temp.add(tuples[i]);
30     return temp.iterator();
31 }

```

```

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

```

```

2         throws DbException, IOException, TransactionAbortedException {
3         // some code goes here
4         // not necessary for lab1
5         ArrayList<Page> pageList= new ArrayList<Page>();
6         for(int i=0;i<numPages();++i){
7             // took care of getting new page
8             HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
9                 new HeapPageId(this.getId(),i),Permissions.READ_WRITE);
10            if(p.getNumEmptySlots() == 0)
11                continue;
12            p.insertTuple(t);
13            pageList.add(p);
14            return pageList;
15        }
16        // no new page
17        BufferedOutputStream bw = new BufferedOutputStream(new
FileOutputStream(file,true));
18        byte[] emptyData = HeapPage.createEmptyPageData();
19        bw.write(emptyData);
20        bw.close();
21        // load into cache
22        HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
23            new HeapPageId(getId(),numPages()-1),Permissions.READ_WRITE);
24        p.insertTuple(t);
25        pageList.add(p);
26        return pageList;
27    }

```

```

1    public ArrayList<Page> deleteTuple(TransactionId tid, Tuple t) throws
DbException,
2        TransactionAbortedException {
3        // some code goes here
4        ArrayList<Page> pageList = new ArrayList<Page>();
5        HeapPage p = (HeapPage) Database.getBufferPool().getPage(tid,
6            t.getRecordId().getPageId(),Permissions.READ_WRITE);
7        p.deleteTuple(t);
8        pageList.add(p);
9        return pageList;
10
11        // not necessary for lab1
12    }

```

```

1    public void insertTuple(TransactionId tid, int tableId, Tuple t)
2        throws DbException, IOException, TransactionAbortedException {
3        // some code goes here
4        // not necessary for lab1
5        DbFile f = Database.getCatalog().getDatabaseFile(tableId);
6        updateBufferPool(f.insertTuple(tid,t),tid);
7    }

```

```

1 private void updateBufferPool(ArrayList<Page> pagelist, TransactionId tid) throws
  DbException{
2     for(Page p:pagelist){
3         p.markDirty(true,tid);
4         // update bufferpool
5         if(pageStore.size() > numPages)
6             evictPage();
7         pageStore.put(p.getId(),p);
8     }
9 }

```

```

1 public void deleteTuple(TransactionId tid, Tuple t)
2     throws DbException, IOException, TransactionAbortedException {
3     // some code goes here
4     // not necessary for lab1
5     DbFile f =
6     Database.getCatalog().getDatabaseFile(t.getRecordId().getPageId().getTableId());
7     updateBufferPool(f.deleteTuple(tid,t),tid);
8 }-

```

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

```

1 private TransactionId tid;
2 private OpIterator child;
3 private int tableId;
4 private final TupleDesc td;
5
6 private int counter;
7 private boolean called;

```

```

1 private TransactionId tid;
2 private OpIterator child;
3 private final TupleDesc td;
4
5 private int counter;
6 private boolean called;

```

部分方法的实现

```

1 public Delete(TransactionId t, OpIterator child) {
2     // some code goes here
3     this.tid = t;
4     this.child = child;
5     this.td = new TupleDesc(new Type[] {Type.INT_TYPE}, new String[] {"number of
deleted tuples"});
6     this.counter = -1;
7     this.called = false;
8 }
9
10 public TupleDesc getTupleDesc() {
11     // some code goes here
12     return td;
13 }
14
15 public void open() throws DbException, TransactionAbortedException {
16     // some code goes here
17     counter = 0;
18     child.open();
19     super.open();
20 }
21
22 public void close() {
23     // some code goes here
24     super.close();
25     child.close();
26     counter = -1;
27     called = false;
28 }
29
30 public void rewind() throws DbException, TransactionAbortedException {
31     // some code goes here
32     child.rewind();
33     counter = 0;
34     called = false;
35 }
36
37 /**
38  * Deletes tuples as they are read from the child operator. Deletes are
39  * processed via the buffer pool (which can be accessed via the
40  * Database.getBufferPool() method.
41  *

```

```

42  * @return A 1-field tuple containing the number of deleted records.
43  * @see Database#getBufferPool
44  * @see BufferPool#deleteTuple
45  */
46  protected Tuple fetchNext() throws TransactionAbortedException, DbException {
47      // some code goes here
48      if (this.called)
49          return null;
50
51      this.called = true;
52      while (this.child.hasNext()) {
53          Tuple t = this.child.next();
54          try {
55              Database.getBufferPool().deleteTuple(this.tid, t);
56              this.counter++;
57          } catch (IOException e) {
58              e.printStackTrace();
59              break;
60          }
61      }
62      Tuple tu = new Tuple(this.tid);
63      tu.setField(0, new IntField(this.counter));
64      return tu;
65  }
66
67  @Override
68  public OpIterator[] getChildren() {
69      // some code goes here
70      return new OpIterator[] {child};
71  }
72
73  @Override
74  public void setChildren(OpIterator[] children) {
75      // some code goes here
76      child = children[0];
77  }

```

```

1  public Insert(TransactionId t, OpIterator child, int tableId)
2      throws DbException {
3      // some code goes here
4
5      if(!child.getTupleDesc().equals(Database.getCatalog().getTupleDesc(tableId))){
6          throw new DbException("TupleDesc does not match!");
7      }
8      this.tid = t;
9      this.child = child;
10     this.tableId = tableId;
11     this.td = new TupleDesc(new Type[]{Type.INT_TYPE},new String[]{"number of
12     inserted tuples"});
13     this.counter = -1;
14     this.called = false;

```

```

13 }
14
15 public TupleDesc getTupleDesc() {
16     // some code goes here
17     return td;
18 }
19
20 public void open() throws DbException, TransactionAbortedException {
21     // some code goes here
22     counter = 0;
23     child.open();
24     super.open();
25 }
26
27 public void close() {
28     // some code goes here
29     super.close();
30     child.close();
31     counter = -1;
32     called = false;
33 }
34
35 public void rewind() throws DbException, TransactionAbortedException {
36     // some code goes here
37     child.rewind();
38     counter = 0;
39     called = false;
40 }
41
42 /**
43  * Inserts tuples read from child into the tableId specified by the
44  * constructor. It returns a one field tuple containing the number of
45  * inserted records. Inserts should be passed through BufferPool. An
46  * instances of BufferPool is available via Database.getBufferPool(). Note
47  * that insert DOES NOT need check to see if a particular tuple is a
48  * duplicate before inserting it.
49  *
50  * @return A 1-field tuple containing the number of inserted records, or
51  *         null if called more than once.
52  * @see Database#getBufferPool
53  * @see BufferPool#insertTuple
54  */
55 protected Tuple fetchNext() throws TransactionAbortedException, DbException {
56     // some code goes here
57     if (this.called)
58         return null;
59
60     this.called = true;
61     while (this.child.hasNext()) {
62         Tuple t = this.child.next();
63         try {
64             Database.getBufferPool().insertTuple(this.tid, this.tableId, t);

```

```

65         this.counter++;
66     } catch (IOException e) {
67         e.printStackTrace();
68         break;
69     }
70 }
71 Tuple tu = new Tuple(this.td);
72 tu.setField(0, new IntField(this.counter));
73 return tu;
74 }
75
76 @Override
77 public OpIterator[] getChildren() {
78     // some code goes here
79     return new OpIterator[] {child};
80 }
81 }
82
83 @Override
84 public void setChildren(OpIterator[] children) {
85     // some code goes here
86     child = children[0];
87 }

```

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

```

1     public synchronized void flushAllPages() throws IOException {
2         // some code goes here
3         // not necessary for lab1
4         for(Page p : this.pageStore.values())
5             flushPage(p.getId());
6     }
7     public synchronized void discardPage(PageId pid) {
8         // some code goes here

```



```

9      // not necessary for lab1
10     pageStore.remove(pid);
11 }
12
13 private synchronized void flushPage(PageId pid) throws IOException {
14     // some code goes here
15     // not necessary for lab1
16     Page p = pageStore.get(pid);
17     TransactionId tid = null;
18     // flush it if it is dirty
19     if((tid = p.isDirty())!= null){
20         Database.getLogFile().logWrite(tid,p.getBeforeImage(),p);
21         Database.getLogFile().force();
22         // write to disk
23
24         Database.getCatalog().getDatabaseFile(pid.getTableId()).writePage(p);
25         p.markDirty(false,null);
26     }
27 }
28 private synchronized void evictPage() throws DbException {
29     // some code goes here
30     // not necessary for lab1
31     PageId pid = new ArrayList<>(pageStore.keySet()).get(0);
32     try{
33         flushPage(pid);
34     }catch(IOException e){
35         e.printStackTrace();
36     }
37     discardPage(pid);
38
39 }

```

6. 实验提交记录

 image-20230407194947435