查询执行器

姓名：王丁子睿　　学号：1183710211

1. 实验流程

下面依顺序介绍 NestedLoopJoinOperator::execute 方法的执行过程。

首先遍历两个关系，找到相同的属性（外键）。

```
    int leftForeignKeyId = -1;
    int rightForeignKeyId = -1;
    for (int i = 0; i < leftTableSchema.getAttrCount(); i++)
    {
      for (int j = 0; j < rightTableSchema.getAttrCount(); j++)
      {
        if ((leftTableSchema.getAttrName(i) == rightTableSchema.getAttr
Name(j)) && (leftTableSchema.getAttrType(i) == rightTableSchema.getAttr
Type(j)))
        {
          leftForeignKeyId = i;
          rightForeignKeyId = j;
          break;
        }
      }
      if (rightForeignKeyId != -1)
      {
        break;
      }
    }
```

然后声明块，设置块大小。

```
    vector<vector<string>> block;
    const int BLOCK_SIZE = 100;
```

遍历两个表中所有页，然后遍历每个页中的所有记录，注意只有在第一个关系放满块之后，才处理第二个关系。

```
    for (badgerdb::FileIterator leftPage = left.begin(); leftPage != le
ft.end(); leftPage++)
    {
      badgerdb::Page *bufferedLeftPage;
      bufMgr->readPage(&left, (*leftPage).page_number(), bufferedLeftPa
ge);
      numIOs += 1;

      for (badgerdb::FileIterator rightPage = right.begin(); rightPage
!= right.end(); rightPage++)
      {
        badgerdb::Page *bufferedRightPage;
```

```
        bufMgr->readPage(&right, (*rightPage).page_number(), bufferedRi
ghtPage);
        numIOs += 1;

        for (badgerdb::PageIterator leftRecord = bufferedLeftPage->begi
n(); leftRecord != bufferedLeftPage->end(); leftRecord++)
        {
          vector<string> leftInfo = analyze(*leftRecord, leftTableSchem
a);
          numUsedBufPages += 1;
          block.push_back(leftInfo);
          if (block.size() < BLOCK_SIZE)
          {
            continue;
          }

          for (badgerdb::PageIterator rightRecord = bufferedRightPage->
begin(); rightRecord != bufferedRightPage->end(); rightRecord++)
          {
            numUsedBufPages += 1;
```

处理的具体过程为，将当前记录对应的属性全都放到新生成的记录中，相同的属性只放一次。

```
            for (int i = 0; i < block.size(); i++)
            {
              vector<string> leftInfo = block[i];
              vector<string> rightInfo = analyze(*rightRecord, rightTab
leSchema);

              if (leftInfo[leftForeignKeyId] == rightInfo[rightForeignK
eyId])
              {
                string current_line = "INSERT INTO TEMP_TABLE VALUES ("
 + leftInfo[0];
                for (int i = 1; i < leftTableSchema.getAttrCount(); i++
)
                {
                  current_line = current_line + ", " + leftInfo[i];
                }
                for (int i = 0; i < rightTableSchema.getAttrCount(); i+
+)
                {
                  bool flag = true;
                  for (int j = 0; j < leftTableSchema.getAttrCount(); j
++)
```

```
                {
                  if (leftTableSchema.getAttrName(j) == rightTableSch
ema.getAttrName(i) && leftTableSchema.getAttrType(j) == rightTableSchem
a.getAttrType(i))
                  {
                    flag = false;
                    break;
                  }
                }
                if (flag)
                {
                  current_line = current_line + ", " + rightInfo[i];
                }
              }
              current_line = current_line + ");";

              string tuple = HeapFileManager::createTupleFromSQLState
ment(current_line, catalog);
              numResultTuples += 1;
              HeapFileManager::insertTuple(tuple, resultFile, bufMgr)
;
            }
          }
```

最后第一个关系的遍历结束的地方，需要额外处理块中残留的记录，处理方法与以上类似，故不再展示代码。

2. 实验结果

块大小为 100 时，输出为：

```
Test Nested-Loop Join ...
# Result Tuples: 554
# Used Buffer Pages: 1100
# I/Os: 4
```

3. 实验心得

- 加深了对自然连接算法，尤其是其分块优化的认识。
- 提高了 C++面向对象编程的能力。