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.getAttrName(j)) && (leftTableSchema.getAttrType(i) == rightTableSchema.getAttrType(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 != left.end(); leftPage++)

    {

      badgerdb::Page \*bufferedLeftPage;

      bufMgr->readPage(&left, (\*leftPage).page\_number(), bufferedLeftPage);

      numIOs += 1;

      for (badgerdb::FileIterator rightPage = right.begin(); rightPage != right.end(); rightPage++)

      {

        badgerdb::Page \*bufferedRightPage;

        bufMgr->readPage(&right, (\*rightPage).page\_number(), bufferedRightPage);

        numIOs += 1;

        for (badgerdb::PageIterator leftRecord = bufferedLeftPage->begin(); leftRecord != bufferedLeftPage->end(); leftRecord++)

        {

          vector<string> leftInfo = analyze(\*leftRecord, leftTableSchema);

          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, rightTableSchema);

              if (leftInfo[leftForeignKeyId] == rightInfo[rightForeignKeyId])

              {

                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) == rightTableSchema.getAttrName(i) && leftTableSchema.getAttrType(j) == rightTableSchema.getAttrType(i))

                    {

                      flag = false;

                      break;

                    }

                  }

                  if (flag)

                  {

                    current\_line = current\_line + ", " + rightInfo[i];

                  }

                }

                current\_line = current\_line + ");";

                string tuple = HeapFileManager::createTupleFromSQLStatement(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。

1. 实验结果

块大小为100时，输出为：

文本

描述已自动生成

1. 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。