**一、实验目的**

本次实验三需要完成的内容为实现⾃然连接(natural join)操作算法，对两个关系进⾏⾃然连接，具体实现基于块的嵌套循环连接(Block-based Nested Loop Join)算法。

我们要实现的函数在executer.cpp文件中。

bool NestedLoopJoinOperator::execute(int numAvailableBufPages,  
 File &resultFile)

**二、实验内容**

首先，我们读取两个表头的信息

TableId leftTableId = catalog->getTableId("r");  
TableId rightTableId = catalog->getTableId("s");  
badgerdb::File left = File::open(catalog->getTableFilename(leftTableId));  
badgerdb::File right = File::open(catalog->getTableFilename(rightTableId));

运用两层循环寻找两个表中名称与类型完全相同的属性，将他们全部标记出来，用于之后的自然连接操作。

vector<int> leftForeignKeyId;  
vector<int> rightForeignKeyId;  
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.push\_back(i);  
 rightForeignKeyId.push\_back(j);  
 break;  
 }  
 }  
}

准备操作做完后，开始进行自然连接操作。

用循环从磁盘中读取两个页面的信息，记录io操作次数

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;

之后，从表中读取全部的元组的信息，进行对比。

读取的元组信息有特殊的格式，并不能直接利用，所以需要先了解元组在表中储存的格式，然后进行解读。元组的存储方式可以从storage.cpp中的createTupleFromSQLStatement函数中得知。

switch (dataType) { // (int) 56 (0011 1000) -> (char) '\0''\0''\0''8'  
 case *INT*: { // convert int value into 4 byte representation

case *CHAR*: { // (char(5) ) 'abc' -> 'abc00'

case *VARCHAR*: { // (varchar(8) ) 'abc' -> '3''abc' (3 refer to the ascii  
 // code number correspond alpha)

于是，我们根据注释的存储方式编写解析函数，该函数输入为文件中存储的元组，输出为数组表示的直观的元组内容

vector<string> analyze(string record, badgerdb::TableSchema schema)

先读取其中一个表的元组，用块来存储。

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;  
 }

然后读取另一个表的元组信息，

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

将两个元组当中的属性名相同的属性列信息进行对比，

bool f = true;  
for(int i = 0; i < leftForeignKeyId.size(); i++)  
{  
 if(leftInfo[leftForeignKeyId[i]] != rightInfo[rightForeignKeyId[i]])  
 {  
 f = false;  
 break;  
 }  
}

如果全部相同，则代表需要进行自然连接操作

if(f)  
{  
 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++)  
 {  
 current\_line = current\_line + ", " + rightInfo[i];  
 }  
 current\_line = current\_line + ");";  
  
 string tuple = HeapFileManager::createTupleFromSQLStatement(current\_line, catalog);  
 numResultTuples += 1;  
 HeapFileManager::insertTuple(tuple, resultFile, bufMgr);  
}

否则不进行任何操作。

在全部循环都结束之后，块中可能还会有剩余的信息没有进行处理，此时再单独对剩余信息进行处理，代码基本相同。

**三、实验结果**

代码运行结果如下：

