实验三: 查询执行器

主讲教师: 邹兆年(znzou@hit.edu.cn)

姓名：刘璟烁 学号： 日期：2021年5月23日

1. 实验目的

在实验2实现的BadgerDB缓冲池管理器(buffffer pool manager)的基础上，本次实验继续实现BadgerDB的查询执⾏器(query executor)，具体完成以下内容：

* 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。还可以根据兴趣实现⼀趟连接(One-Pass Join)算法和/或Grace哈希连接(Grace Hash Join)算法。

1. 实现过程
2. **分析底层方法实现：**

为实现基于块的嵌套循环连接(Block-based Nested Loop Join)算法，实验提供了关系模式定义(relation schema defifinition)、系统⽬录(system catalog)、存储管理器 (storage manager)、查询执⾏器(query executor)的大部分实现，这些组件的具体声明和定义如下：

* **关系模式定义(schema.cpp中声明和定义):**

1. DataType数据类型:

BadgerDB⽀持三种SQL标准数据类型 INT整型 、CHAR(n)定长字符串、VARCHAR(n)变长字符串型；

1. Atrribute类：

⽤Attribute类的对象记录关系的属性(attribute)的定义，其包含以下属性：

·attrName: 属性名；

·attrType: 属性类型。例如，如果属性为VARCHAR(5)类型，则attrType = VARCHAR；maxSize: 属性的最⼤⻓度。例如，如果属性为VARCHAR(5)类型，则maxSize = 5；

·isNotNull: 如果属性不能为空，则为true；否则，为false；

·isUnique: 如果属性取值唯⼀，则为true；否则，为false；

1. TableSchema类：

TableSchema类的对象来记录关系的模式，属性如下：

·tableName: 表名；

·attrs: 全部属性的定义(每个属性的定义⽤Attribute类的对象来记录)；

·isTemp: 是否为临时表；

注意，TableSchema类中有以下三个常用方法

·const string& getAttrName(int num)：通过属性的编号获取属性名；

·const DataType& getAttrType(int num)：通过属性编号获取属性的数据类型；

· int getAttrMaxSize(int num)：通过属性编号获得属性的最大宽度；

* **系统目录(Catalog类)**

Catalog类的对象用来存储数据库的模式信息，其包含属性如下：

·dbName: 数据库名；

·tableIds: 表名到表号的映；

·tableSchemas: 表号到表的模式定义的映射；

·tableFilenames: 表号到表的数据⽂件名的映射；

·nextTableId: 下⼀个可⽤的表的编号，nextTableId单调递增。

注意，Catalog类中有如下三个常用方法：

·const TableId& getTableId(const string& tableName) ：通过表名获取表在数据库中的id;

·const TableSchema& getTableSchema(const TableId& id)： 通过表的id获得表的关系模式；

· const string& getTableFilename(const TableId& id) ：通过表的id获得表名；

* **存储管理器(Storage Manager)**

存储管理器(storage manager)的功能是向表中插⼊或删除元组。在BadgerDB中使⽤堆⽂件 (heap fifile)的形式存储表，并提供了HeapFileManager类，⽤于实现对向表中插⼊和删除元组的功能，HeapFileManager中有三个静态方法，如下所示：

·static RecordId insertTuple(const string& tuple, File& file, BufMgr\* bufMgr)：向关系中插⼊⼀条元组。该⽅法有3个参数： tuple为元组的值，fifile为关系的数据⽂件的句柄 ，bufMgr为指向缓冲池管理器对象的指针；它按照堆⽂件组织⽅式插⼊新元组tuple。如果元组插⼊成功，返回该元组的记录号(RecordId类型)。

·static void deleteTuple(const RecordId& rid, File& file, BufMgr\* bugMgr): 从关系中删除⼀条元组。该⽅法有3个参数，后2个参数fifile和bufMgr与insertTuple⽅法的同名参数相同，deleteTuple⽅法的rid参数是待删除的元组的记录号(RecordId类型)。

·static string createTupleFromSQLStatement(const string& sql, const Catalog\* catalog)：该⽅法根据输⼊的SQL INSERT语句和插⼊关系的模式，创建⼀条元组，并将该元组返回。

* **查询执⾏器(Query Executor)**

实验要求编写⾃然连接执⾏器(natural join operation executor)来实现⾃然连接(natural join)操作， 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)。实验定义了JoinOperator类作为各种连接操作执⾏器的基类(base class)。JoinOperator类有如下属性：

·leftTableFile: 左关系⽂件的句柄；

·rightTableFile: 右关系⽂件的句柄；

·leftTableSchema: 左关系的模式；

·rightTableSchema: 右关系的模式；

·resultTableSchema: 结果关系的模式；catalog: 系统⽬录对象的指针；

·bufMgr: 缓冲池管理器对象的指针；

·isComplete: 算法执⾏是否结束；

·numResultTuples: 结果元组数量；

·numUsedBufPages: 算法执⾏过程中实际使⽤的缓冲⻚⾯数；

·numbs: 算法执⾏过程中实际执⾏的I/O数；

在该类中需要实现NestedLoopJoinOperator::execute方法；

* **表扫描器(Table Scanner)**

TableScanner类⽤于对表进⾏扫描和打印，它具有如下属性：

·const File& tableFile：表名；

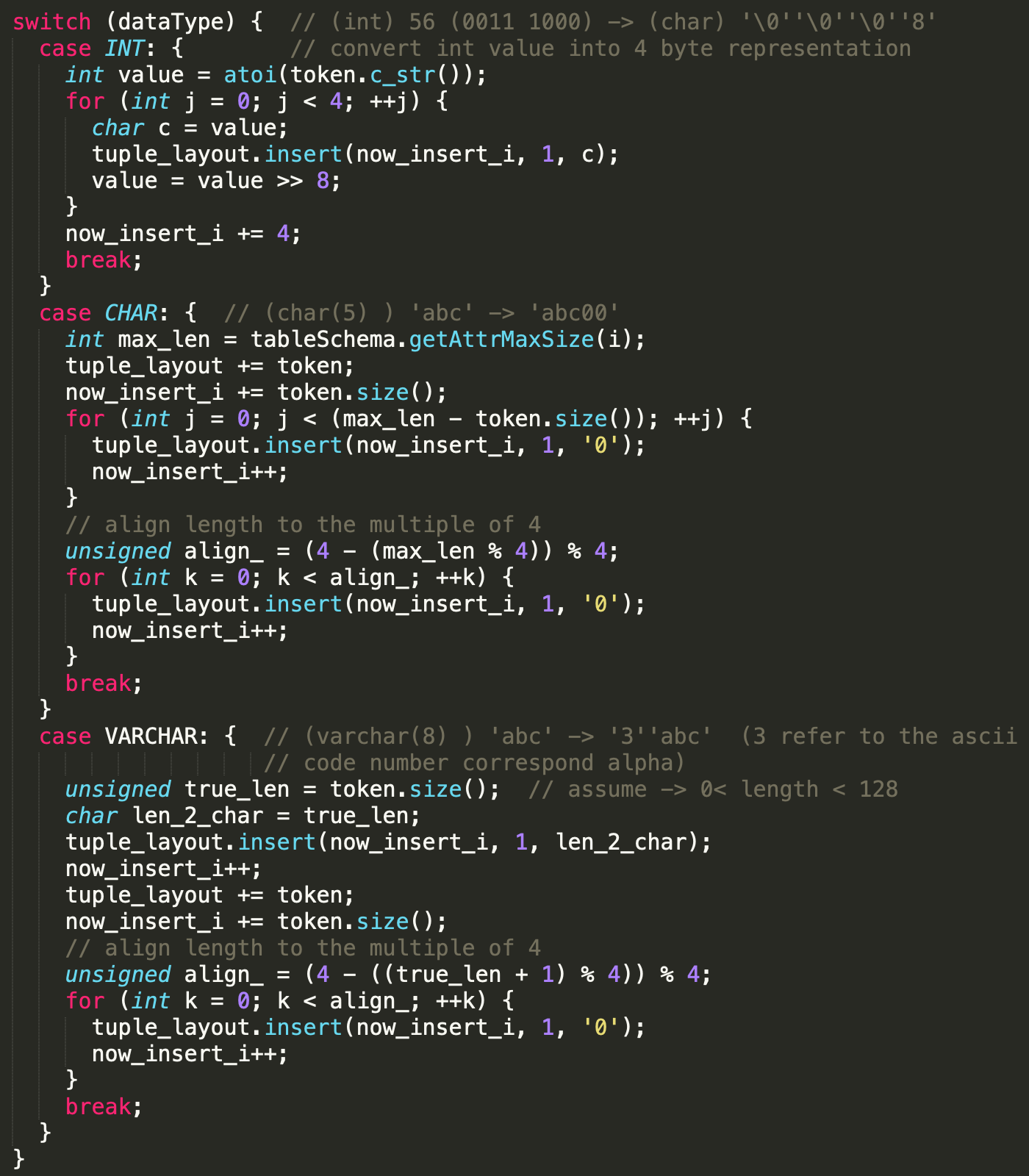
·const TableSchema& tableSchema：表的关系模式；

·BufMgr\* bufMgr：缓冲区管理器；

1. **NestedLoopJoinOperator::execute()实现**

基于块的嵌套连接算法应用内外两层循环，外层循环每次从右关系S中读取M-1个块进入缓冲区（其中M为缓冲区可用的块数量），并对这M-1个块中的元组构建内存存储结构(本次实验我们使用hashTable作为内存存储结构，并将关系R和S的共有属性的属性值作为key)，随后内循环遍历关系R中的每一个块并对其中的元组利用内存存储结构快速的进行连接操作；

由于元组在数据库中的存储形式与输入的SQL语句内容不同，我们首先观察其内部存储形式，根据课程中所讲的tuple layout可知元组不同属性值在存储时需要考虑到对齐的问题，另外本次实验不考虑tuple的头部，结合storage.cpp中对三种数值类型的存储：

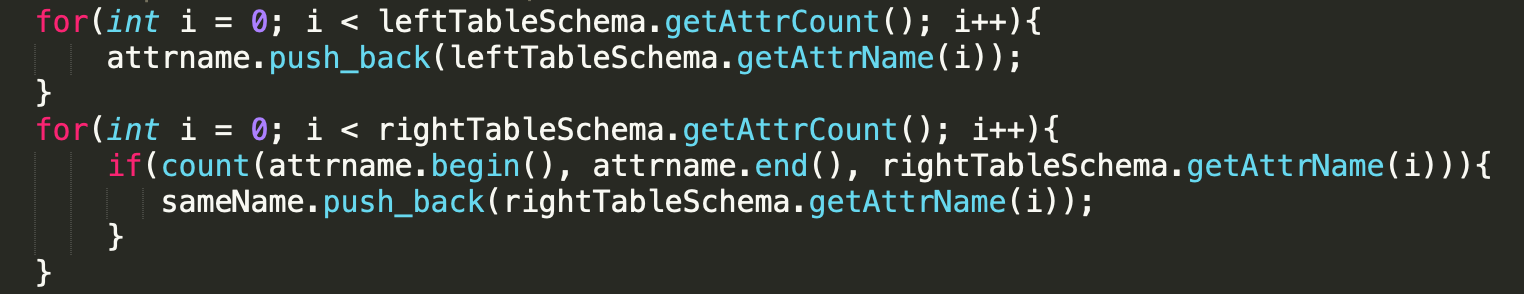


我们可以实现后续需要用到的handle\_tuple()方法：

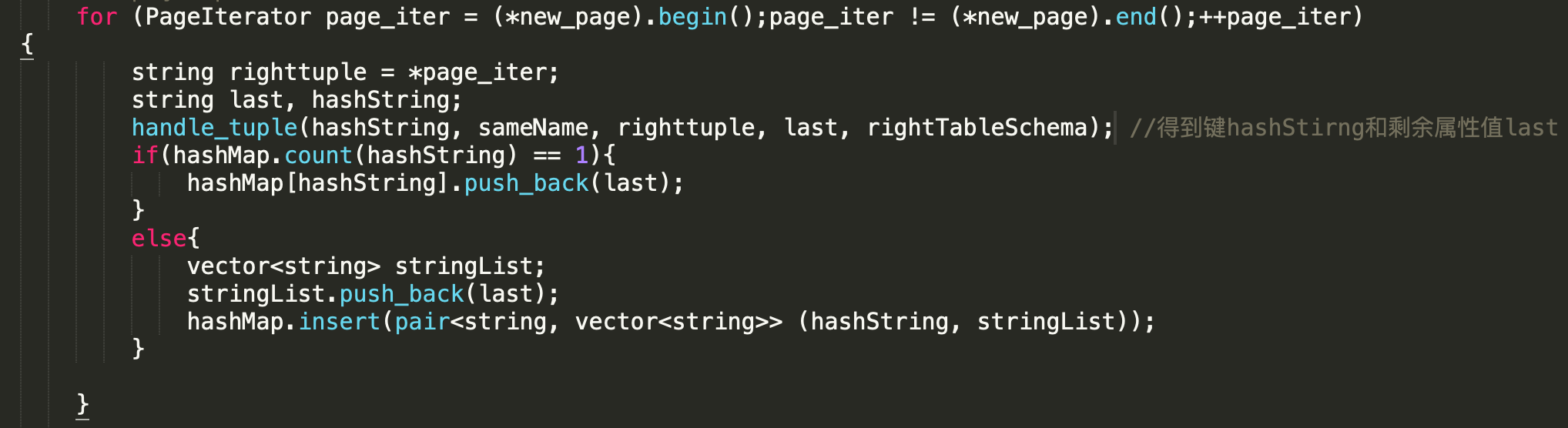
void handle\_tuple(string& hashString, vector<string> sameName, string &tup, string& last ,const TableSchema& tableSchema)：该方法将输入的元组从形参tup传入，并循环处理其中的每个属性，对于属于两表共有属性sameName中的属性直接将其加入到hashString中，作为hash表的键，而剩下的属性则放入到last中待用；

随后实现NestedLoopJoinOperator::execute(int numAvailableBufPages, File& resultFile)方法：

首先通过tableSchema中的两个方法获取两个关系的共有属性序列，并将其存入到sameName中：

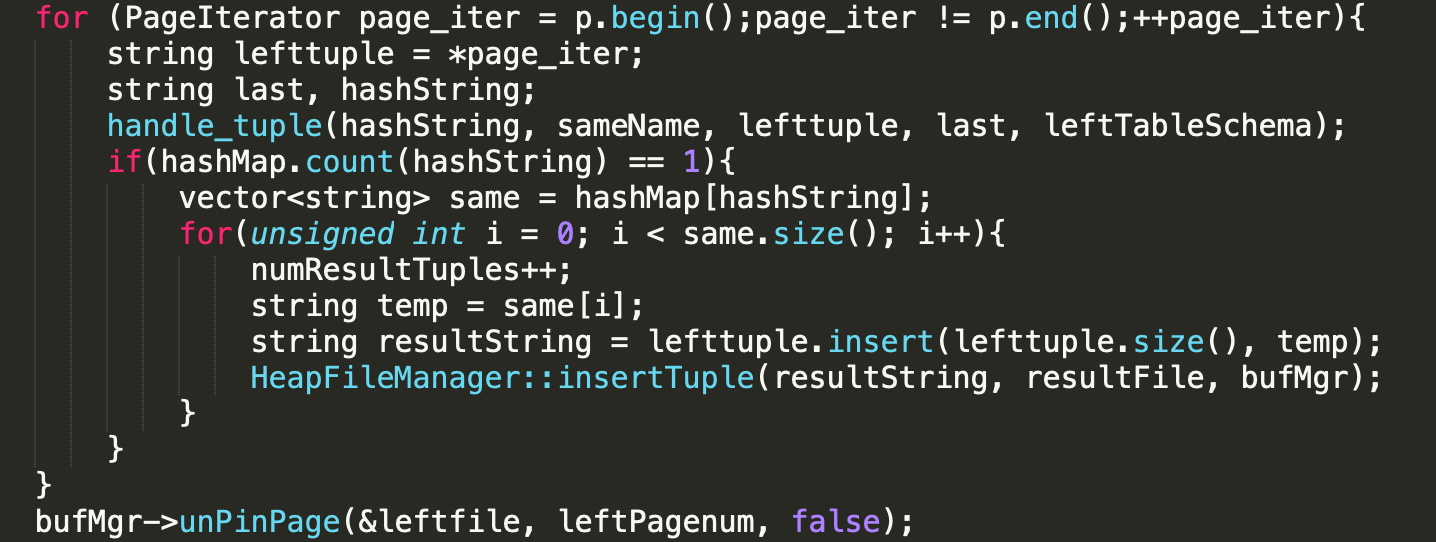


随后进入外层循环，调用File中的iterator，每次读取M-1个页进入到缓冲区，对于每一个块，用page的iterator读取元组，并调用上述函数handle\_tuple（）处理元组，插入hashTable中：



过程中需要记录I/O和使用缓冲区页数；

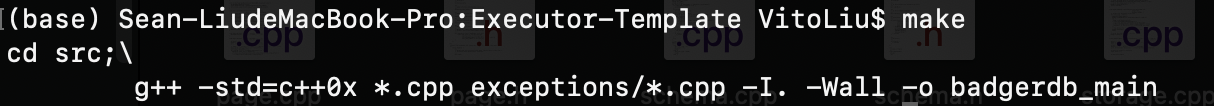
随后基于该内存存储结构（hahsTable），遍历关系R中的所有块：将其逐个读入缓冲区中，并类似上述部分逐条元组输入到hashTable中进行比对，对满足连接条件的结果进行连接：



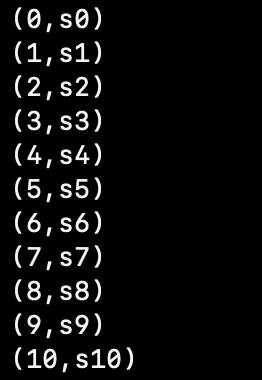
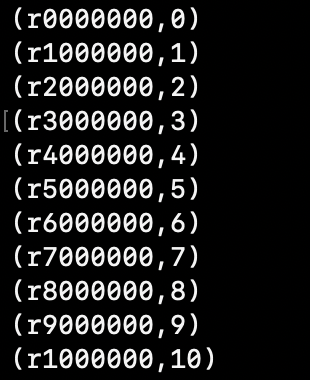
外循环结束后对相关数据结构进行释放即可。

1. 实验结果

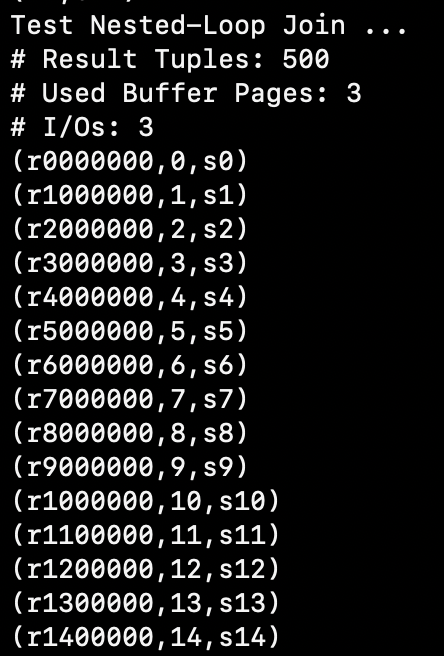
进入Executor-Template目录中，对项目进行编译：



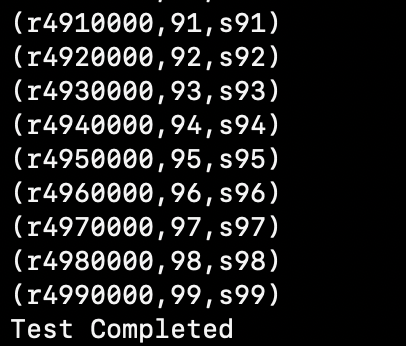
运行生成的可执行文件得到结果，首先结合main.cpp以及下图观察需要连接的两个关系R和S，可以发现前者的第二个属性和后者的第一个属性**均为INT类型并且属性名相同**（均为b），因此可以判断需要通过b属性进行连接；



进行基于块的嵌套循环连结后，可以看到使用的缓冲区页面数以及总的I/O数均为3，并生成了500个结果元组：



Test completed:



4. 总结与心得

本次实验实现了对于自然连接的基于块的嵌套连接算法，算法每次从右关系S中读取M-1个块进入缓冲区，并对这M-1个块中的元组构建内存存储结构，随后遍历关系R中的每一个块并对其中的元组利用内存存储结构快速的进行连接操作。在实现过程中，对于元组在数据库内部的存储结构有了更深一步的了解，包括各种类型的存储方式、存储空间中使用的的“对齐”技术等；同时通过分别处理进行连接的左右关系，对于存储执行、缓冲区管理的过程有了更深刻的领悟。