实验题目	查询执行器实现			实验日期	2022.5.8
班级	1903501	学号	1190202425	姓名	傅彦璋

CS33503 数据库系统实验

实验检查记录

实验结果的正确性(60%)		表达能力(10%)	
实验过程的规范性(10%)		实验报告(20%)	
加分(5%)		总成绩(100%)	

实验报告

一、实验目的

1.掌握连接操作的实现算法；
2.在实验 2 的缓冲区管理器的基础上，使用 C++面向对象程序设计方法实现查询执行器。

二、实验环境

cmake version 3.21.1 on macOS 12.3 (Intel)

三、实验过程

3.1 熟悉 BadgerDB

阅读代码，尤其是 file.h, file_iterator.h, page.h, page_iterator.h 中关于扫描文件和页面的迭代器方法。阅读 schema.h 了解了关系模式和属性的实现方法。阅读 storage.cpp 理解元组的存储机制以及元组字节存储和字符串表示的转换办法。

3.2 补完代码

为了使基于块的嵌套循环连接算法 IO 代价较小，需要调整内外关系。通过一个 FileIterator 类的迭代器分别扫描关系 R 和关系 S 两个文件，得到其大小，令较小的一个作为右关系(S)。这里实现了函数 getTableSize()来求得文件大小。

通过多重循环实现连接。第一步是将块读入缓冲区页框，第二步是在内重循环中枚举元组对进行连接。设共有 m 个缓冲区页面。

首先，将 S 中的块读到第 0 到第 m-2 个页框中——我们使用 FileIterator 类的迭代器扫描 S 的文件，但读入页框时是通过已经实现好的缓冲区管理器的 readPage()方法来读入。接着，我们将关系 R 的一个块读入到剩下的一个页框中——仍然是用 FileIterator 扫描，但通过 bufMgr 读入。

此时，各页面已经在合适的位置上，我们枚举每一对 R 和 S 的元组对。其中 R 的元组需要一重循环，利用 PageIterator 类的迭代器扫描第 m-1 个页框中的每条元组。S 的元组需要二重循环：外循环依次指示页框中的各个页，内重循环对每个页，扫描该页中的元组。

对于每个元组对，我们判断其能否连接成功，如果能则加入 resultFile 中。我们通过一重循环扫描连接结果关系模式的每一个属性，有三种可能：1)该属性来自 S，2)该属性来自 R，3)S 和 R 都拥有该属性。前两种容易处理，若是第三种，则判断目前枚举到的这两个元组的该属性值是否相等，如果不相等则遗弃这条潜在的结果元组。如果相等再和前两种情况一样将属性值加入到结果元组中。判断的过程中，需要将元组表示为一个 string 类型的向量以便对比属性值——这是通过 splitTuple 函数实现的。

实验题目	查询执行器实现			实验日期	2022. 5. 8
班级	1903501	学号	1190202425	姓名	傅彦璋

3.3 测试

用 make 编译并运行后发现能够通过测试。调整 main.cpp 中的测试元组数量，也没有问题。我认为我正确实现了查询执行器。

四、实验结论

1.

虽然循环嵌套连接查询的原理很朴素，但是代价其实不小。我在编写代码的过程中深切地体会到了这一点——多重循环、多种迭代器令我非常困扰，在运行的过程中代价一定也会很大。连接操作的确应当是查询优化的核心。

2.

事实上，我的代码可以进一步优化，只是为了不增加代码复杂度而完成实验，我没有进行更进一步的优化。

我们不应该对于每一对元组对，枚举属性。我们应当提前预扫描属性，将结果元组的属性分为三类：来自 R、来自 S、来自 R 和 S。并且在最内重循环优先对第三类进行判断，尽早丢弃那些失败的连接，进而提高效率。同时，因为已经有每个属性的来源的信息，最内重循环内也不需要有 if 判断，效率会更高。