哈爾濱工業大學

数 据 库 实 验 三 报 告

实 验 三： 查 询 执 行 器

姓　　名 郭　　炼

学　　号 1 1 8 0 1 0 0 2 1 7

老　　师 邹　　兆　　年

专　　业 数据科学与大数据技术

日　　期 2020 年 05 月 25 日

目录

第1章 实验目的

在实验二实现的 BadgerDB 缓冲池管理器的基础上，本次实验继续实现 BadgerDB 的查询执行器，具体完成以下内容：

- 实现自然连接操作算法，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算法。

第2章 连接算法

在本次实验中，我们需要使用基于块的嵌套循环连接算法来实现对两个关系的自然连接。

基于块的嵌套循环连接算法就是一种朴素的暴力枚举算法。该算法会将两个关系中较小的关系（外关系）成批的读入内存中，之后将较大的关系（内关系）依次读入内存中。枚举两个块中的所有元组，之后尝试进行自然连接。如果连接成功，就将其插入到结果关系中。其如伪代码1所示。

第3章 代码实现

我们只需要编辑 executor.cpp 文件，实现 `NestedLoopJoinOperator` 类的 `NestedLoopJoinOperator::execute` 方法就可以完成本次实验。

Algorithm 1: 基于块的嵌套循环连接算法

Input: 两个待连接的关系 S 和 R，可使用的缓冲块数目 M

Output: S 和 R 自然连接的结果

```
if B(S) > B(R) then
    交换 S 和 R
end
for 外关系 S 中的每 M − 1 块 do
    将这 M − 1 块读入缓冲块
    用内存查找结构来组织 M − 1 块中的元组
    for 内关系 R 中的每一块 P do
        将 P 读入缓冲区 for P 中的每个元组 r do
            for 内存查找结构中能与 r 进行的元组 s do
                连接 r 和 s，将结果写入输出缓冲区
            end
        end
    end
end
```

3.1 tableSize 函数

我首先实现了一个 `tableSize` 函数，用于统计一个关系块的数目。该函数会遍历关系的所有块，用于统计数目。

该函数用于后续判断关系大小，确定外关系 S 和内关系 R。

3.2 parseTuple 函数

在本次实验中，元组是经过序列化的，所有的值均是以二进制字节码的方式存储的。为了便于后续的连接，我们需要实现一个函数通过关系的模式来将一个二进制码解析为一个二进制列表（`vector<string>`）来便于访问。

该函数仿照 `TableScanner` 的 `print` 方法，按照三种属性值来实现对二进制码的切片，将对应的部分存储在一个线性表中。这样就可以通过下标实现 $O(1)$ 的查找。

```
1 static void parseTuple(const TableSchema &tableSchema, const string &
    key, vector<string> &tuple) {
2   tuple.clear();
3   tuple.reserve(tableSchema.getAttrCount());
```

```
4   for (int i = 0, current_index = 0; i < tableSchema.getAttrCount(); ++
      i) {
5     switch (tableSchema.getAttrType(i)) {
6     case INT: {
7       tuple.push_back(string(key, current_index, 4));
8       current_index += 4;
9       break;
10    }
11    case CHAR: {
12      int max_len = tableSchema.getAttrMaxSize(i), last = current_index
      ;
13      current_index += max_len;
14      current_index += (4 - (max_len % 4)) % 4;
15      tuple.push_back(string(key, last, current_index - last));
16      break;
17    }
18    case VARCHAR: {
19      int actual_len = key[current_index], last = current_index;
20      current_index++;
21      current_index += actual_len;
22      current_index +=
23          (4 - ((actual_len + 1) % 4)) % 4;
24      tuple.push_back(string(key, last, current_index - last));
25      break;
26    }
27    }
28  }
29 }
```

代码 3.1: parseTuple 函数

3.3 NestedLoopJoinOperator::execute 方法

在本方法中，我们需要实现基于块的嵌套循环自然连接算法，大体上就是将伪代码1转换为 C++ 代码。

我们使用 `tableSize` 比较两个块的大小,判断外关系和内关系;`parseTuple` 将二进制码解析成二进制序列，便于后续比较；用一个数组存储 $M-1$ 块缓存对应的页的指针。之后根据结果的模式，来进行连接。对应结果模式的每一个属性：

- 如果 S 和 R 均含有该属性，那么判断其是否相等。如果不相等，则终止连接，否则连接到结果元组的二进制码中。
- 如果只有 S 或 R 均含有该属性，那么直接将该属性对应值连接到结果元组的二进制码中。

- 如果 S 和 R 均不含有该属性，那么报错。

如果连接成功，那么利用 `HeapFileManager` 的 `insertTuple` 方法将结果元组插入堆中。

需要注意的是，$B(S)$ 可以大于 $M-1$，所以可能需要多次读入 S 关系的块。此外每次读完对应的块后需要及时释放对应的缓冲块，防止缓冲池中块的数目不足。

第 4 章　实验总结

在本次实验中，我加深了对于基于块的嵌套循环连接算法的理解，初步了解了二进制序列的存储元组的具体方式。通过一些代码了解了获取关系中的属性、元组的内部表示方法、元组在文件中的存储方式的具体细节。