The DBMS Project Report
数据库课程设计实验报告

目录

Part 1: Group Member 小组成员

Name	Sid	Email
朱新强	12330441	sysu.ss.jeason@qq.com
庄泽帆	12330442	sysu.zhuang@gamil.com
钟盼盼	12330425	897269925@qq.com

Part 2: Environment 实验环境

CPU	Intel(R) Core™ i7-2760QM
OS	Mac OS X 10.9 Mavericks, Ubuntu 14.04(32bit)
Compiler	g++
Language	C++（不含 STL）

Part 3: Contents 实验内容

3.1: 不带压缩地导入数据

实现分页存储机制。对于每一列，将所含的定长数据如同数组一样存储在磁盘上的页中 。每一页的大小固定。存储 ORDERS 表中的 O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_SHIPPRIORITY。存储之后，提供接口，以便于查询记录。

3.2: 外排序和 RLE 压缩

对数据做外排序。并在排序之后，使用行程长度编码 (Run-Length Encoding) 方式压缩数据。

3.3: JOIN

将 ORDERS 表和 CUSTOMER 表以 CUSTKEY 进行 JOIN 操作。

Part 4: Design Ideas 实验思路和设计

// 将给出我们对于整个项目中各个环节、流程、实现的具体思考过程、重要数据结构、函数伪代码等；

4.1 实验思路

按照实验内容所给出的要求，C-STORE 实验思路比较明确，分为以下几个步骤：

① 导入：将数据导入并进行分页存储；

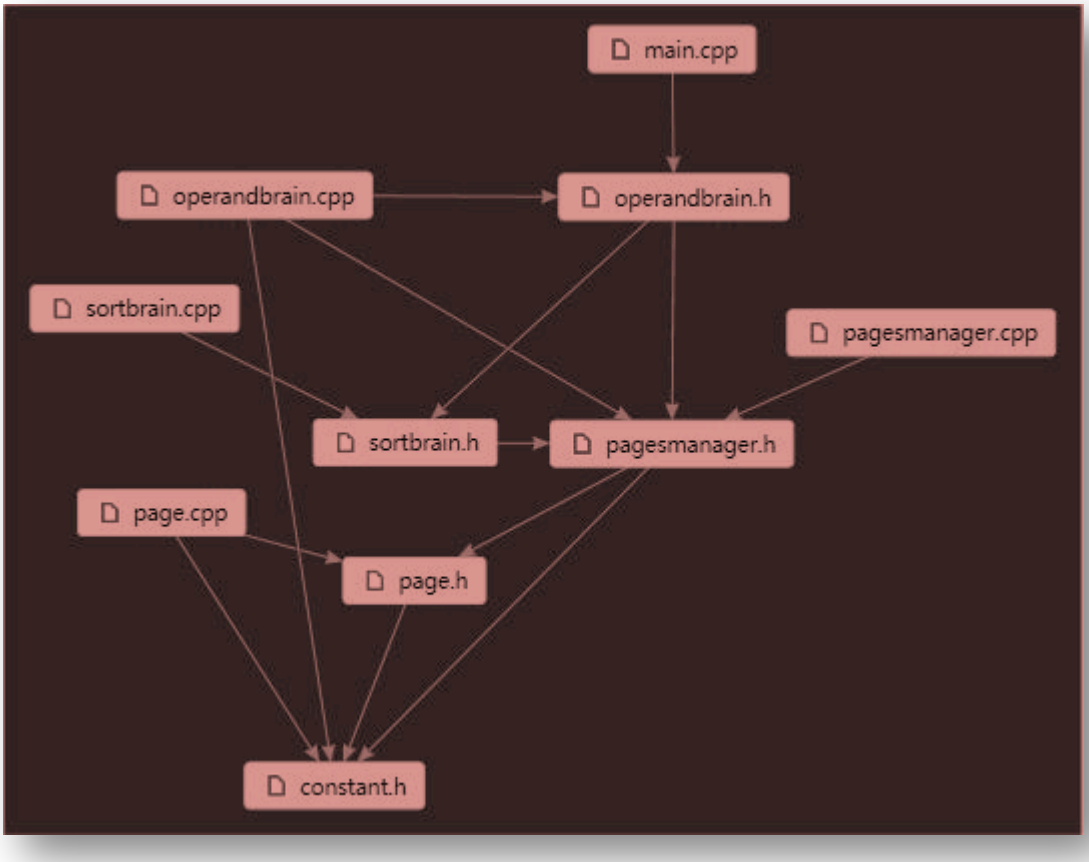
② 外排序：先对页面内进行 qsort，再对页面进行 mergesort；

③ 压缩：对于排序后的 CUSTKEY 进行压缩，并写入文件中；

④ Join：将 ORDERS 表和 CUSTOMER 表以 CUSTKEY 进行 JOIN 操作；

4.2 项目结构

UML 结构图



Part 5: Implementation 代码结构与实现

5.1 结构简述

这次整体采用面向对象的思路进行编程，对于每一个功能作为一个模块，对应建立不同的类来实现，并上下层次之间实现对接。这样的编程思想比较利于团队的协作，有便于分工，且在各自的代码出现 bug 需要修改的情况下，不会影响到其他人的代码的正确性。这样保证了统一性，加快了完成的效率。

5.2 整体结构

5.2.1 Main

在 Main 函数中接受命令行传入的字符串参数，并使用 switch 对接到 join，load，compress 等操作对应的函数实现中。

```
int main(int argc, const char * argv[])
{
    // insert code here...
    OperandBrain _operandBrain;
    switch (argc) {
        case 2:
            if ( strcmp( argv[1], JOIN_OPERATION_NAME) == 0 ) {
                //db join
                _operandBrain.join();
            } else if ( strcmp( argv[1], COUNT_OPERATION_NAME ) == 0 ){
                //db count
                printf( "Count: %ld\n", _operandBrain.count() );
            }
            break;

        case 3:
        ...
```

5.2.2 OperandBrain

与 Main 相对接，用于具体实现导入，查询，压缩等等操作。具体实现文件的开关，读写等。

```
class OperandBrain {
private:
    //memory management
    pagesManager *_pageManager;

    //extern sort helper
    SortBrain *_sortBrain;

public:
    OperandBrain();

    ~OperandBrain();

/* operands the system support */
    void queryInTable( const char *tableName );

    void loadData( const char *_fileName );

    void compressData( const char *tableName, const char* column );

    void join();

    long count();
};
```

5.2.3 PageManager

用于管理页面，除了一些直接调用某页的操作外，还可以支持多页面的操作，为要实现 OperandBrain 里的操作提供辅助接口。

```
class pagesManager {
private:
    Page *pages;
    /**
     *  sigletron instance
     *  private constructor and destructor
     */
    static pagesManager *_instance;
    pagesManager();
    ~pagesManager();

    //helper function for qsort
    static int compare( const void *a, const void *b );

public:
    static pagesManager *sharedManager();
    //operations on one page
    //just simple to call functions in Page
    void writePageAtIndexToFile( size_t index, FILE* fptr );

    void clearPageAtIndex( size_t index );

    void readFileToPageAtIndex( FILE *fptr, int index );

    bool searchKeyInPageAtIndex( const char* key, size_t keySize,
char *entryOutput, size_t entrySize, int index );

    bool readPropertyFromPageAtIndexWithOffSet( char *property,
size_t propertySize, int offset, int pageIndex );

    //operations on pages
    bool insertData( char *data, size_t _size, size_t pageCondition );

    int readFileToPages( FILE* fptr, int begin, int end );

    void clearPages( int begin, int end );

    //extern sort helper functions
    void sortPages( int begin, int end );

    void megrePagesToFile( int begin, int end, FILE* tempFptr );

    void megreFilesToFile( FILE **tempFileArray, int tempFileCount,
                           FILE *outputFptr );

    void compressPagesToFile( FILE *fptr, int begin, int end );
```

5.2.4 SortBrain

考虑到外排序这个问题比较复杂，定义 SortBrain 这个类专门用于实现外排序。

```
class SortBrain {
private:
    pagesManager *_pageManager;
    /**
     *  sort pages into temp files
     *
     *  @param inputFile file needed to be sortd
     *
     *  @return temp file count
     */
    int pagesMemorySort( const char* inputFile );
    /**
     *  megre temp files to a file
     *
     *  @param count     the number of the temp files
     *  @param outputFile the final sorted file
     */
    void mergeSortTempFiles( int count, const char* outputFile );

public:
    SortBrain();
    ~SortBrain();
    void externSort( const char *inputFileName, const char*
outputFileName );
};
```

5.2.5 Page

作为整个系统的底层数据结构，集合了“处理插入数据到页存储”，“删除页中的内容”，“在页与文件中的读写操作”，“对页进行查找，压缩”等操作。在最低层实现页面内操作，在上层调用中使结构更加清晰。

```
class Page {
private:
    char data[PAGE_SIZE];
    int offSet; //pointer to the current used data
public:
    Page();
    ~Page();
    /**
     *  copy data to page memory
     *
     *  @param data pointer to the data to insert
     *  @param size size of the data to insert
     *
     *  @return return false if the page has been full, otherwise,
return true
     */
    bool insertDataToPage( char *data_, size_t size );

    void clearPage();
    /**
     *  write and read content between file and page
     *
     *  @param fptr the file pointer to write or read
     */
    void writePageToFile( FILE *fptr );
    void readFileToPage( FILE *fptr );
    /**
     *  search in page with key
     *
     *  @param key the start address of the key to search in page
     *  @param _size the size of the key
     *
     *  @return true for found, false for unfound
     */
    bool searchInPage( const char *key, size_t keySize,
                      char* & outputEntry,  size_t entrySize );

    bool readPropertyWithOffSet( char *property, size_t proertySize,
int _offSet );
    inline bool isFull() { return offSet >= PAGE_SIZE; }
    inline bool isEmpty() { return offSet <= 0; }
    /**
     *  getters and setters for the private member
     */
    inline char *getData() { return data; }
    inline int getOffset() { return offSet; }
```

5.3 具体实现

5.3.1 导入

进行一系列的文件打开和读取操作后，对指定的KEY进行导入，以下代码为对ORDERKEY的导入操作，其他 KEY 同。

```
          while ( !feof( fptr ) ) {
              //read one line of file to memory
              fgets( strLine, ENTIRE_ENTRY_MAXSIZE, fptr );
              //split into properities
              char *currentProperity = strtok(strLine, "|");
              //current track variable
              size_t currentCondition = 0;
              size_t maxCondition = O_SHIPPRIORITY_CONDITION;

              while ( currentProperity != NULL &&
                     currentCondition <= maxCondition ) {
                 //while not the end of the line
                 switch ( currentCondition ) {                //
TODO: clear the switch code

                 case O_ORDERKEY_CONDITION:
                    primaryKey = atoi( currentProperity );
                    memcpy(tempArr, &primaryKey, sizeof(int));
                    while
( !_pageManager->insertData((char*)&primaryKey,
sizeof(int),pageIndexs[0] ) ) {  //if insert fail
                       //write the page to file

_pageManager->writePageAtIndexToFile( pageIndexs[0],orderKeyFptr
);
                       //clear page

_pageManager->clearPageAtIndex( pageIndexs[0] );
                       //re-insert
                    }
                    break;
```

导入到临时页面后再将页面写入文件中

5.3.2 查询

给定一个 ORDERKEY，在页面中查询

```
          while ( !feof( orderKeyFptr ) ) {
              //search key in page
              if
( !_pageManager->searchKeyInPageAtIndex( (char*)&orderKey,
sizeof(int), (char*)&orderKey, 0, 0 ) ) {
                  // if not found, read next page of data from file

_pageManager->readFileToPageAtIndex( orderKeyFptr, 0 );
              } else {
                  found = true;
                  break;
              }
          }
```

若查询到了，将其他三个 KEY 分别从文件中读取出来，下面代码为从 CUSTKEY 文件中读取 CUSTKEY 的部分

```
          if ( found ) {  //if find orderKey, it means the tuple can
be found by the orderKey
              //find relative custkey
              //read one page of data from custkey file
              _pageManager->readFileToPageAtIndex(custKeyFptr, 0);
              while ( !feof( custKeyFptr ) ) {
                  //search key in page
                  if
( !_pageManager->searchKeyInPageAtIndex((char*)&orderKey,
sizeof(int), (char*)&custKey, sizeof(int), 0) ) {
                      // if not found, read next page of data from file

_pageManager->readFileToPageAtIndex(custKeyFptr, 0);
                  } else {
                      break;
                  }
              }
……
```

若查询不到，错误提示

```
        } else {    //can not find orderKey, the query fail
            printf( "%d is not found!\n", orderKey );
        }
```

5.3.3 外排序

a.先读入 127 页

b.然后对每一页进行内部快速排序，直接用 qsort 完成

```
#pragma mark -
#pragma mark -extern sort helper functions
void pagesManager::sortPages( int begin, int end ) {
    for ( int i = begin; i <= end; i++ ) {
        // TODO: Implete a algorithm to sort general page with
comparable entries
        int entrySize = sizeof(int) + sizeof(int);
        qsort( pages[i].getData(), pages[i].getOffset() / entrySize,
entrySize, compare );
    }
}
```

c.接下来将 127 页的数据采用 127 路归并算法归并成一个有序的临时文件。主要在 pageManager 内的 megrePagesToFile 函数内完成.

c.1.先找到最小值

```
        int min = -1;    //find the minimize of the first entry of pages
        bool first = true;

        for ( int i = begin ; i <= end; i++ ) {
           if ( offsets[ i - begin ] < pages[i].getOffset() ) {   //if
this page is not been read at end
              if ( first ) {                //init the min
                 first = false;
                 min = i;
              } else if( compare(
                      pages[i].getData() + offsets[i-begin],
                      pages[min].getData() + offsets[min-begin] )
                    < 0 ) {
                 min = i;
              }
           }
        }   //for loop end, min save the value of the pageIndex of min
```

c.2. 将选出来的最小的值插入到 page 当中，若 page 已满，则写入文件。

```
        while ( !lastPage->insertDataToPage( pages[min].getData() +
offsets[min-begin], sizeof(int)+sizeof(int) ) ) {  //insert the min
value to the output page
           //if insert fail
           //write page to file
           lastPage->writePageToFile( tempFptr );
           lastPage->clearPage();
        }
        offsets[ min - begin ] += sizeof(int)+sizeof(int);    //next
entry
```

c.3. 检查是否已经全部写入，退出条件

```
        //exam if all pages has been read
        int i;
        for ( i = begin ; i <= end; i++ ) {
            if ( offsets[ i - begin ] < pages[i].getOffset() )
{   //if a page is not at the end
                break;
            }
        }

        if ( i == end + 1 ) {     //if all pages has been read,
break the loop
            break;
        }
```

d. 接下来将临时文件归并到一个文件里,思路与将 127 页归并到一个临时文件的思路相同，只是当某一页已经读完时，进行换页。

```
        //if pages[min] have been read
        if ( offsets[min] >= pages[min].getOffset() ) {
            //read next page
            readFileToPageAtIndex( tempFileArray[min], min );
            offsets[min] = 0;
             if ( feof( tempFileArray[min] ) ) {
             //if file have been read
                readCount++;
                fileHaveRead[min] = true;
            }
        }
```

5.3.4 压缩

a. 从排好序的文件中读取一页，并从中读出第一个 key

```
//read first page from sorted file
    _pageManager->readFileToPageAtIndex( sortedFptr, 0 );
//read first custKey from page
    _pageManager->readPropertyFromPageAtIndexWithOffSet(
(char*)&currentCustKey, sizeof(int), offset, 0 );

     offset += 8;
```

b. 读出下一个 key，与之前的 key 比较，若相同则 count 加一，若不同则将<key><count>的结构写入缓存页内，若缓存页已经满了则写入到文

```
              //if two custKey is same
              if ( currentCustKey == nextCustKey ) {
                  count++;
              } else {       //else insert <key><length> to output
                 bool success =
_pageManager->insertData( (char*)&currentCustKey, sizeof(int), 1 )
&& _pageManager->insertData( (char*)&count, sizeof(int), 1 );
                 if ( !success ) {   //if insert fail
                    _pageManager->writePageAtIndexToFile( 1,
outputFptr );
                  //write page
                    _pageManager->clearPageAtIndex( 1 );
                   //clear output page

                  _pageManager->insertData( (char*)&currentCustKe
                  y, sizeof(int), 1 ); //reinsert
                    _pageManager->insertData( (char*)&count,
sizeof(int), 1 );
                 }
                 currentCustKey = nextCustKey;   //find next
                 count = 1;
              }
```

c. 重复 b 直到排好序的文件已经读完。

d. 计算压缩率

```
        //compute compress rate
        fseek( sortedFptr , 0, SEEK_END );
        long beforeSize = ftell( sortedFptr );
        fclose( sortedFptr );

        fseek( outputFptr, 0, SEEK_END );
        long compressedSize = ftell( outputFptr );
        fclose( outputFptr );

        rate = compressedSize / ( beforeSize / 2.0 );
```

5.3.5 JOIN

a. 首先从排好序的 custkey，压缩好的 custkey，以及从 customer.tbl 中导入的 custkey 文件中各读出一页。

```
    //read first pages
    _pageManager->readFileToPageAtIndex( cCustKeyFptr, 0 );
    _pageManager->readFileToPageAtIndex( compressFptr, 1 );
    _pageManager->readFileToPageAtIndex( oCustKeyFptr, 2 );
.....
```

b. 从压缩好的页中读出一个 custkey，然后在 customer 表中的 custkey 页中查找。

```
            //read a custKey from compressed page
        while
( !_pageManager->readPropertyFromPageAtIndexWithOffSet( (char*)&o
_custKey, sizeof(int), offSets[1], 1 )

|| !_pageManager->readPropertyFromPageAtIndexWithOffSet( (char*)&
o_custKeyCount, sizeof(int), offSets[1] + sizeof(int), 1 ) ) {
            //read fail
            //read another page
            if ( feof( compressFptr ) ) {
                //end
                end =  true;
                break;
            } else {
                _pageManager->readFileToPageAtIndex( compressFptr,
1 );
                offSets[1] = 0;
            }
            //while loops, re-readProperty
        }
        if ( end ) break;

        offSets[1] += sizeof(int) * 2;
        //read a custKey from customer page
        while
( !_pageManager->readPropertyFromPageAtIndexWithOffSet( (char*)&c
_custKey, sizeof(int), offSets[0], 0 ) ) {
                //read fail
                //read another page
            if ( feof( cCustKeyFptr ) ) {
                break;
            } else {
                _pageManager->readFileToPageAtIndex( cCustKeyFptr,
0 );
                offSets[0] = 0;
            }
        }
        offSets[0] += sizeof(int);
```

d. 找到相同之后，读出相应的length，从order表中的排好序的custkey中打印出相应的orderkey和custkey.

```
    //while two custKeys is not same
        while ( c_custKey != o_custKey ) {
            //because all o_custKey can be find in c_custKey
            //just read next custKey from customer page
            while
( !_pageManager->readPropertyFromPageAtIndexWithOffSet( (char*)&c
_custKey, sizeof(int), offSets[0], 0 ) ) {
                //read fail
                //read another page
                if ( feof( cCustKeyFptr ) ) {
                    break;
                } else {

_pageManager->readFileToPageAtIndex( cCustKeyFptr, 0 );
                    offSets[0] = 0;
                }
            }
            offSets[0] += sizeof(int);
        }

            //now, c_custKey == o_custKey
        for ( int i = 0; i < o_custKeyCount; i++ ) {    //print result
            while
( !_pageManager->readPropertyFromPageAtIndexWithOffSet( (char*)&o
rderKey, sizeof(int), offSets[2], 2 ) ) {
                _pageManager->readFileToPageAtIndex( oCustKeyFptr,
2 );
                offSets[2] = 0;
            }
            offSets[2] += sizeof(int) * 2;
            printf( "|%-10d|%-10d|\n",o_custKey, orderKey );
        }
```

e. 若压缩好的文件已经读完了，结束。

5.3.6 COUNT

从压缩好的文件中读出一页，然后再在一页中读出各个 length 值相加即可。

5.4 Contant

用于定义各种变量，优化代码让代码更加可视化，运用宏定义对各种常用的特定字符串进行重命名。也用于处理一些可以部分代替全部的字符串，方便程序的运行，减少开销。

```
/* Operation name definition */
#define JOIN_OPERATION_NAME "join"
#define COUNT_OPERATION_NAME "count"
#define LOAD_DATA_OPERATION_NAME "load"
#define QUERY_OPERATION_NAME "retrieve"
#define COMPRESS_OPERATION_NAME "compress"

#define ENTIRE_ENTRY_MAXSIZE 300 /* Size of one line */

/* Columns of order table need to be loaded into system */
#define O_ORDERKEY_CONDITION 0
#define O_CUSTKEY_CONDITION 1
#define O_TOTAL_PRICE_CONDITION 3
#define O_SHIPPRIORITY_CONDITION 5
```

Part 6: Performance 实验结果与性能比较

将给出基于上述代码执行相关操作的结果和时间性能上分析；

6.1 Compile

在根目录下执行 make 指令，将把 src 中的 cpp 文件编译成.o 文件，并把.o 文件链接成可执行文件（db）

```
bash-3.2$ ls
LICENSE         README          report.docx
Makefile        bin             src
bash-3.2$ make
g++ -c -g -Wall -o bin/main.o src/main.cpp
g++ -c -g -Wall -o bin/OperandBrain.o src/OperandBrain.cpp
g++ -c -g -Wall -o bin/Page.o src/Page.cpp
g++ -c -g -Wall -o bin/pagesManager.o src/pagesManager.cpp
g++ -c -g -Wall -o bin/SortBrain.o src/SortBrain.cpp
g++ -o bin/db ./bin/main.o ./bin/OperandBrain.o ./bin/Page.o ./bin/pagesManager.
o ./bin/SortBrain.o
bash-3.2$
```

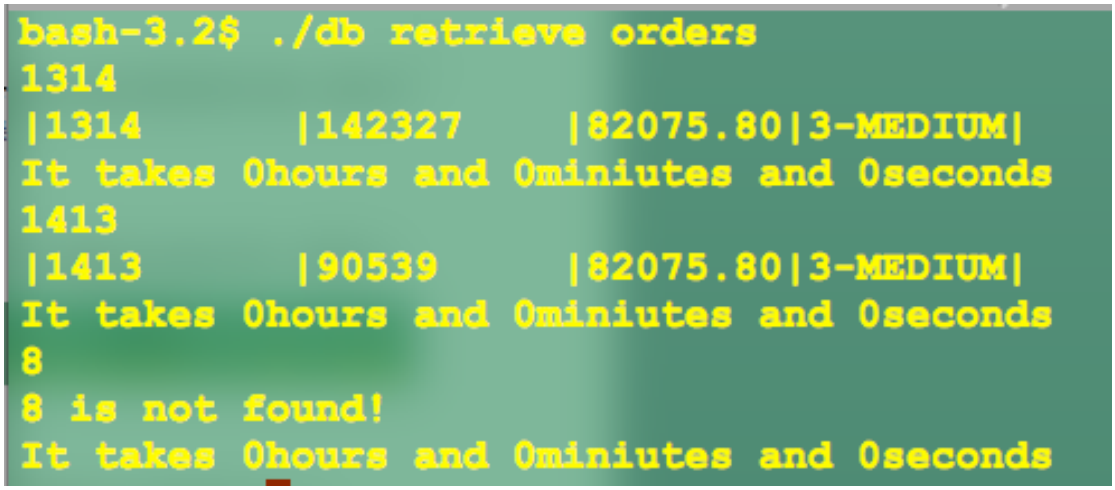
结果显示无编译警告（在 OS X 中，其他系统中#pragma mark 预处理指令无效，该指令只为标记便于在 Xcode 中查找代码），也无编译错误。

6.1 Load Orders

将 orders.tbl 拷贝入根目录中，CD 进入 bin 目录，运行./db load orders 指令，结果如下：

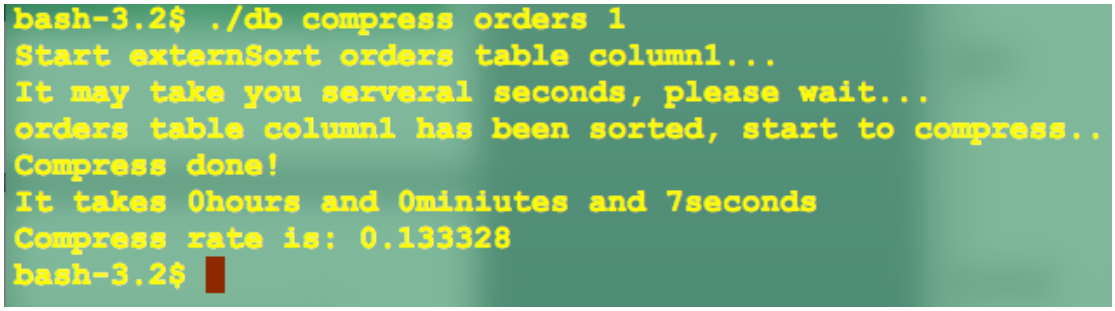
6.2 retrieve orders

运行./db retrieve orders 之后进入查询模式(输入 EOF 结束)，输入 orderkey 可以获得相应的一行.



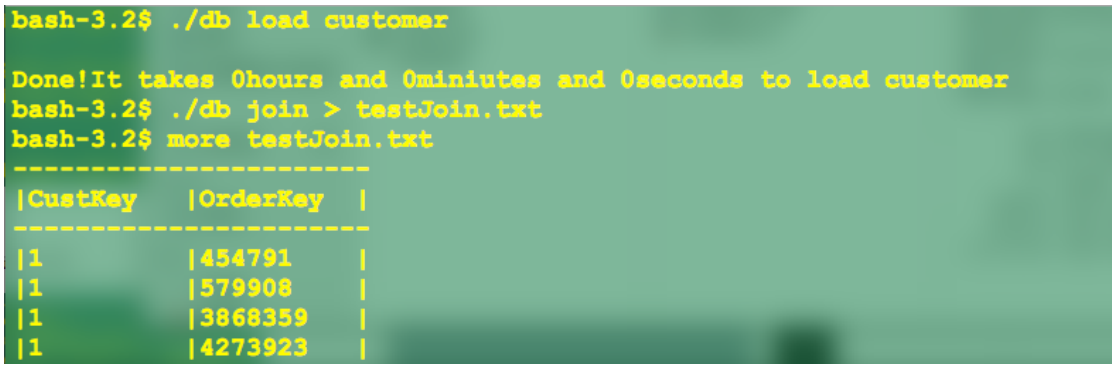
6.3 compress

运行./db compress orders 1 可以对 custkey 列进行排序压缩



6.4 join

在导入 customer 表之后，运行 ./db join 可以对 customer 表和 order 表进行 join 操作



6.5 Count

Part 7: Inspiration 心得与体会

这次 project 无疑是今年面对的最大的挑战，之前完全没有听说过 C-store 的概念，而查阅了相关的参考资料之后，我们小组经历了讨论，确定实验思路，确定实验分工，到代码重构及完善各个阶段。特别是代码重构这个环节，可谓是几经波折，改了又改，改了再改。可以说组员在这次实验中都收获颇多！

7.1 起始的准备

准备工作可谓是十分繁杂，由于之前翻译阶段每个人阅读参考的资料皆不尽相同，所以对于 C-store 的理解也建立在自己阅读的那一个部分上面。而两次讨论总是因为“知识背景不同”而讨论不出结果。最后才认识到了有统一的知识储备背景的重要性。于是我们小组成员多花了很多时间先把 C-store 介绍性的论文仔细地阅读，实际了解它一些具体实现的作用之后再来进行讨论，结论才慢慢有一些起色。

7.2 最初的想法

最初看到这个问题，我们小组经过讨论觉得这是一个相对简单的项目，因为涉及的导入其实就是一个文件的分段写入，而外排序则由于有老师课堂的讲解，基本可以拆分成一个内部的快排加上一个外部的归并操作。而压缩在查阅了一些资料之后，也是比较容易解决的问题。

可以说，项目的前期我们感觉思路是比较明确的，所以讨论觉得要在一开始就规范化我们的代码，把各种接口，类的层级结构都规划好，然后按照面向对象的理念，尽力把它做成一个真正实用的数据库，让它可以在其他 tbl 或者其他的数据表格导入后也能进行相应的操作，并且可以对于不同的列不同的表明实现操作。

这样当然要求我们的代码需要有很明确的分层以及设计得比较合理的接口，让调试可以比较轻松地完成。从而实现代码的可维护性。而且要求代码的组织结构比较统一，使得后面的扩充操作可以不断地添加进去完善它让它适用于更广泛的输入数据。

7.3 遇到的问题

一开始还是总体的结构不明确的问题。由于没有意识到外排等代码的复杂性，一开始都归在一个类中完成，结果由于实现的东西很多，代码写得十分冗长，可读性很差，但又已经临近第二阶段的提交时间，所以只能先完成了，再来重构代码。

于是完成了外排，压缩等函数之后，重新把他们归入不同的类中来实现，使代码的区分度更高，便于阅读和维护。

还有，一开始完成的代码由于考虑的东西比较少，后面重构的过程中就会发现其中的一些局限性，例如说函数虽然有传入参数，但是传入的参数却没有用到，这导致了代码中很多不安全和冗余。于是，在重构的过程其实也是重新修正各种函数的过程。

中间外排还出现了不小的问题，出现了比较大的 BUG。而且显得无厘头。因为排序完居然比原来的文件小了将近 50K。而这是之前的所有测试所没有遇到的问题。提出的几个假设也在接下来的调试中被否定掉。进制一度停滞不前。结果最后发现其实是测试的文件写错了。这也为我们敲响了警钟，不管是项目的代码，还是测试的代码都要保持严谨的态度，要不然因为测试代码的问题而不断地进行无谓的调试显然是十分不划算的！

7.4 感想与收获

其实我们做这个项目前期的想法我觉得是没有问题的，就是我们要先定好框架，然后不断地充实它，然后再慢慢地让它实现更多的功能最后实现题目的要求甚至是做出一些扩展。

但是在实验过程中我们发现，有时候为了实现一些功能，往往会先不顾组织架构的设计，先把它完成了，然后再进行调整。其实后来觉得这是一种新的思路。就是其实一个项目一开始并不用把框架描述得那么细致，因为有一些细节的东西处理起来也许并不是一开始想象的

那么简单。但代码的过程中往往是先把它实现出来，然后再去完善它，效率会高过直接就按照最后那种很高的标准去写。也许就是程序员经常说的“先做到在做好”吧。

还有在小组当中，及时的沟通显然是非常重要的。像我们小组中，虽然说比较主体的代码是由同一个人完成的，但是其他两个人也分别负责了一些小的模块。而因为代码都要与上一层的代码相对接，有时候由于阅读代码的偏差，有一些函数的功能理解不同，往往就会实现出另外一种效果。也许有时也能实现出来，但是无疑打乱了之前的思路而影响接下来的进展。而及时地沟通，及时地说出自己的理解，当意见发生碰撞时，也能及时地理解对方的思路，统一意见。有时思维的碰撞在沟通中还能得出一些更好的实现方法。

其实这次 project 也是对于编程能力的一次很好的训练，毕竟这学期也没有算法之类的课程，代码多少有一点忘记。特别是对于 C++里面文件流的输入输出这方面的内容，确实已经遗忘了很多。这次实验就采用 C++语言进行编写，debug 的过程其实也会出现很多之前学习 C++编程时遇到的低级错误等，就当是一次以实战为基础的很好的复习！

最后，在我们 project 进行的过程中也遇到了不少问题，感谢对我们提出的问题进行解答的同学，TA。也谢谢老师提供这样一个很好的机会让我们加深了对于数据库的理解与实战运用。这次 project 真的受益匪浅！