Database Systems

Instructors: Winston Hsu, Hao-Hua Chu

Fall Semester, 2009

Assignment 6: Sort-Merge Join

Deadline: 23:59 Jan 6 (Wednesday), 2010

This is a group assignment, and at most 2 students per group are allowed.

Cheating Policy: If you are caught cheating, your grade is 0.

1. Introduction

In this assignment, you will implement the sort-merge join algorithm.

2. Available Documentation

You should begin by reading the chapter on Evaluating Relational Operations, in particular, the section on **14.4.2 Sort-Merge Join (algorithm)**.

```
proc smjoin(R, B,‘$R_i$ = $S'_j$)

if R not sorted on attribute i, sort it;
if B not sorted on attribute j, sort it;

Tr = first tuple in R;                                  // ranges over R
Ts = first tuple in B;                                  // ranges over S
Gs = first tuple in S;                      // start of current S-partition

while Tr ≠ eo! and Gs ≠ eo! do {

    while Tri < GSj do
        Tr = next tuple in Rafter Tr;                // continue scan of R

    while Tri > $GS_j$ do
        Gs = next tuple in S after Gs                // continue scan of B

    Ts = Gs;                                  // Needed in case Tri ≠ $GS_j$
    while Tri == $GS_j$ do {                    // process current R partition
        Ts = Gs;                                     // reset S partition scan
        while $TS_j$ == Tri do {                    // process current R tuple
            add (Tr, Ts) to result;                  // output joined tuples
            Ts = next tuple in S after Ts;}   // advance S partition scan
        Tr = next tuple in Rafter Tr;                  // advance scan of R
        }                                  // done with current R partition

    Gs = Ts;                        // initialize search for next S partition

    }
```

3. What You Have to Implement

```
class sortMerge
{
          public:
          sortMerge(
                    char        *filename1,      // Name of heapfile for relation R.
                    int         len_in1,         // # of columns in R.
                    AttrType    in1[],           // Array containing field types of R.
                    short       t1_str_sizes[],  // Array containing size of columns in R.
                    int         join_col_in1,    // The join column number of R.
                    char        *filename2,      // Name of heapfile for relation S
                    int         len_in2,         // # of columns in S.
                    AttrType    in2[],           // Array containing field types of S.
                    short       t2_str_sizes[],  // Array containing size of columns in S.
                    int         join_col_in2,    // The join column number of S.
                    char*       filename3,       // Name of heapfile for merged results
                    int         amt_of_mem,      // Number of pages available for sorting
                    TupleOrder order,            // Sorting order: Ascending or Descending
                    Status&     s                // Status of constructor
);

~sortMerge();
}
```

The sortMerge constructor joins two relations R and S, represented by the heapfiles filename1 and filename2, respectively, using the sort-merge join algorithms. Note that the columns for relation R (S) are numbered from 0 to len_in1 - 1 (len_in2 - 1). You are to concatenate each matching pair of records and write it into the **heapfile** filename3. The error layer for the sortMerge class in JOINS, that is, you may use *MINIBASE_CHAIN_ERROR(JOINS, status)* to append an error information to the global error queue. *MINIBASE_SHOW_ERRORS()* can show the error messages.

You will need to use the following classes which are given: ***Sort, HeapFile,*** and ***Scan***. You will call the **Sort constructor** to sort the input heapfiles (which means your primary responsibility will be to implement the merging phase of the algorithm). Use **openScan** to open the sorted heapfiles and get the first tuples. To compare the join columns of two tuples, you will call the function **tupleCmp** (declared in sort.h). Once a scan is opened on a heapfile, the scan cursor can be positioned to any record within the heapfile calling the Scan method **position** with an RID argument. The next call to the Scan method **getNext** will proceed from the new cursor position. Finally, insert the merge results into the output heapfile.

4. Compiling Your Code and Running the Tests

Please copy all the files from web site into your own local directory. The files are:

- *Makefile*: A sample Makefile for you to compile your project. Set up any dependencies (as needed) by editing this file.
- *sortMerge.h*: Specifications for the class sortMerge. You have to implement these specifications as part of the assignment.
- *SMJTester.C*: sort-merge test driver program.

You will also find in the project directory the implementation of the external sort algorithm in the files ***sort.C*** and ***sort.h*** and the corresponding files about ***Scan*** and ***HeapFile*** classes in the directory include.

5. How to hand-in

Submit your file “sortMerge.C” and a report to mingkuang.tsai@gmail.com. The email subject must be “[DB09] hw6 ID1 ID2 version”, (Ex.”[DB09] hw6 r98944005 b94902050 v1”). You can submit your program many times, and TA will use that latest one to grade.

If you modify any other files, submit it together and specify it in your report clearly.