UNIVERSITY OF WISCONSIN–MADISON
COMPUTER SCIENCES DEPARTMENT

CS 564: DATABASE MANAGEMENT SYSTEMS

Assignment 5: Sort-Merge Join
Due: April 17, 1996 (5 pm)

Instructor: Raghu Ramakrishnan

1 Introduction

In this assignment, you will implement the sort-merge join algorithm. *You will carry out this assignment in teams of two with the same partner as for the previous assignments.*

2 Available Documentation

You should begin by reading the chapter on *Implementation of Relational Operations*, in particular, the section on *Sort-Merge Join*.

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 algorithm. 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 is JOINS.

You will need to use the following classes which are given: Sort, HeapFile, and Scan. You will call the Sort constructor to sort a heapfile. 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.

4 Improved Error Protocol

For this assignment, you will use an improved version of the error protocol. As in the previous error protocol, you will have to define an array of error messages and its corresponding array of error codes. Here is an example for the buffer manager component:

```
enum bufErrCodes { HASHTBLERROR, HASHNOTFOUND, BUFFEREXCEEDED, ... };

static const char* bufErrMsgs[] = {
"hash table error",
"hash entry not found",
"buffer pool full",
...
};
```

Note that the error message array is declared as "static". To register the error message array for the buffer manager component, you just need to create a static "error_string_table" object in the same file as the error message array as follows:

```
static error_string_table bufTable( BUFMGR, bufErrMsgs );
```

There are two macros defined that make it easy to report errors when you discover them. These macros also add the name of the file and the line number where the error happens, as a debugging aid.

To add a "first" error, use the MINIBASE_FIRST_ERROR macro. For example, if the buffer manager detects that the buffer pool is too full to complete an operation, it posts its BUFFEREXCEEDED error like this:

```
MINIBASE_FIRST_ERROR( BUFMGR, BUFFEREXCEEDED );
```

To add a "chained" error, use the MINIBASE_CHAIN_ERROR macro. For example, if the buffer manager calls on the database manager to write a page to disk, and that operation fails, the buffer manager adds an error that records the fact that the execution path that failed went through it:

```
status = MINIBASE_DB->write_page( ... );
if ( status != OK )
    return MINIBASE_CHAIN_ERROR( BUFMGR, status );
```

For this assignment, you can consider the sortMerge class as part of the JOINS layer.

5 Where to Find Makefiles, Code, etc.

Copy all the files from ~cs564-1/spring96/assign5/src into your own local directory. The files are:

- *Makefile*: A sample makefile to compile your project.
- *sortMerge.h*: Specifications for the class sortMerge. You have to implement these specifications as part of the assignment.
- *sortMerge_driver.C*: sort-merge test driver program.

You can find the interfaces for Sort, HeapFile, and Scan classes in ~cs564-1/spring96/assign5/include.

6 What to Turn In, and When

You should turn in copies of your code together with copies of the output produced by running the tests provided by the TAs. The assignment is due at 5PM on April 17. The solution will be made public after that time, and solutions turned in after that time will not receive any credit. So be sure to turn in whatever you have, even if it is not working fully, at that time.

I emphasize that late submissions will not receive any credit. Computers—and life!—being what they are, expect everything to take longer than you expect, even taking this expectation into account. So start early, and plan on getting things done well before the due date. Nothing short of a nuclear explosion (in the CS building, not the South Pacific) constitutes a valid reason for an extension.