CSCD 427 Project #3 (50 points)

Due: 11:59pm on May 11, 2016

PROJECT DESCRIPTION

The purpose of this project is to implement the sort-merge join and hash-join algorithms for a relational database management system (RDBMS). You will simulate the algorithms based on the given configurations, i.e., database file size, block size, the number of available memory blocks, etc.

GETTING STARTED

You should at least implement the following three classes:

- **`sortMergeJoin`** class which implements the sort-merge join algorithm. You need to implement at least three methods in this class:
  - **`sort`** method which takes an unsorted database file as input, divides the file into a number of partitions, and sorts each partition in memory (You can use Java build-in `sort()` method directly). Remember that a partition is a list of blocks, and a block stores a list of database records. In other words, you should implement a partition as a two-dimensional array. This method should output the sorted partitions to files.
  - **`merge`** method which takes the sorted partitions as input, and merge-sorts them into a completely sorted list. You must implement this method by following the algorithm introduced in class. You CANNOT use Java build-in `sort()` method to implement this method. This method should output the completely sorted list to a file.
  - **`join`** method which takes two sorted database files as input and joins these two files together to implement the natural join operator. The final joining result should be written to an output file.
- **`hashJoin`** class which implements the hash join algorithm. You need to implement at least two methods in this class:
  - **`hash`** method which takes an unsorted database file as input, uses a hash function to hash the data records into a number of buckets, and outputs these buckets to files.
  - **`join`** method which takes two buckets (one from the build relation, the other from the probe relation) as input, uses the build relation to build an in-memory hash table, and joins the probe relation with the build relation. The final joining result should be written to an output file.
- **`joinTester`** class which holds your **`main`** method.

Your program will take two command line arguments:

```
% java joinTester block_size memory_size
```

The first argument `block_size` determines the capacity of a block, i.e., the number of records can be stored in one block. The second argument `memory_size` determines the number of available memory blocks. The two database files `student.txt` (244 records) and `takes.txt` (3694 records) can be downloaded from Canvas. The schemas of these two files are:

```
Student (ID, name, dept_name, credits)
Takes (ID, course_id, sec_id, semester, year, grade)
```

Your program should generate the following output files:

1) Sorted `student` and `takes` partitions generated from `sortMergeJoin.sort()` method. Note: use one file to store one partition, and name the file as `smj_student_i.txt` and `smj_takes_i.txt` (here *i* is the index of the partition).
2) Two completely sorted database files generated from `sortMergeJoin.merge()` method. Name them as `sorted_student.txt` and `sorted_takes.txt`, respectively.
3) The joining result generated from `sortMergeJoin.join()` method. Name the file as `smj.txt`.
4) Hashed `student` and `takes` buckets generated from `hashJoin.hash()` method. Note: use one file to store one bucket, and name the file as `hj_student_i.txt` and `hj_takes_i.txt` (here *i* is the index of the bucket).
5) The joining result generated from `hashJoin.join()` method. Name the file as `hj.txt`.

One last note: Remember that you are simulating two joining algorithms for an RDBMS, and the smallest unit of operation in an RDBMS is block. Therefore, all of your operations should be block-based. For example, when you read in a sorted partition into memory to implement `sortMergeJoin.join()` method, you can only read in one block of data from that partition, because we assume the memory is constrained by a limited number of available blocks.

SUBMISSION

You need to submit a single zip file as your solution via the Canvas system. In this zip file, you need to include:

- All the java source files you have created.
- A readme file which includes all the information you would like to share with the instructor and/or the grader. If there's nothing to share, this file can be omitted.

GRADING

- (10 points) Compilable program files
- (40 points) Program Correctness