CSE177/EECS277 – DATABASE SYSTEMS IMPLEMENTATION
Project 5: Sort-Merge Join Operator
Due date: April 26 & 27 (in the lab)

This project requires the implementation of the **two-pass sort-merge join** operator. As the name implies, there are two phases in this operator, both governed by the amount of available memory, given in number of pages. In the sort phase, the input relations are split into fragments that fit in the available memory. Each fragment is sorted on the join attribute(s) and written to disk. A sorted fragment is also called a *run*. In the merge phase, a page is read from each run across the two relations, the minimum value for the join attribute is extracted from each relation, and a join tuple is generated, if the minimum values are identical. If that is not the case, the minimum value is discarded. Whenever a page from a run is emptied, a subsequent page from the same run is read in its place. The procedure finishes when all the runs from one of the two relations are exhausted. This algorithm has been discussed extensively in class and is also well-explained in the textbook.

Implementation

The implementation is confined to the `GetNext` method in the `Join` operator. The current in-memory implementation (as of Phase 4 of the project) has to be extended to sort-merge join, when the number of pages allocated to the operator cannot hold any of the input operands. The modified `Join` operator works as follows. It starts to read records from the smallest of its input operands. The smallest operand is determined based on information from the query optimizer. If all the records are read without filling the available memory, the in-memory operator can be executed. This is the `Join` implemented in Phase 4. If the available memory is exhausted, the current run is sorted and written to disk. The process is repeated until all the records in the small operand are produced and sorted. Then, the large operand is processed similarly. In the merge phase, all the tuples having the same join attributes are generated at the same time, but passed to the parent operator one-at-a-time. Remember that no tuple can be produced until the merge phase. When writing runs to disk, use the same `DBFile` objects. The simplest solution is to have one `DBFile` for every run. In merge, you have to open all the files simultaneously. To get the records with the minimum value(s) for the join attribute(s), you have to extract all records with the minimum value from each run. The minimum across runs is obtained by using a search data structure, e.g., heap or binary search tree, in which all the minimum records from each run are inserted. You do the same for both relations, each with its own data structure.

Requirements

1. Implement the extended `GetNext` method for the `Join` operator to include sort-merge join. The only modification to `main.cc` is to set the number of pages available to each `Join` operator. The value can be hard-coded. However, try different values, such that you can observe when in-memory or sort-merge join are executed.
2. Execute the queries provided with this stage of the project over the `TPC-H` data you generated and loaded in Phase 3 of the project.
3. For correctness and performance analysis, compare the results you obtain with the results generated by some other database server, e.g., `SQLite`.

Resources

- `http://www.tpc.org/tpch/`