Project-0 Report

This report presents an overview on the implementation of the volcano-style tuple-at-a-time query engine, as a part of the Project-0 for the Database Systems (CS-422) course at EPFL. For this project, the following six basic operators were designed by filling the open(), next() and close() methods:

1. Scan

- **Open() Block:** Initialize rowId to zero.
- **Next() Block:** Return nothing if we reached the end of the table. Else, increment the rowId and return the current row of the table as an instance of 'RowStore'.
- **Close() Block:** Empty.

2. Filter

- **Open() Block:** Create a list of inputs, and iterate over each entry to check which entries pass the 'predicate' function. Append the passed entries to the next() block via a list.
- **Next() Block:** Return nothing if we traversed the entire results list. Else, increment the count and return the current entry of the results list.
- **Close() Block:** Empty.

3. Project

- **Open() Block:** Initialize the input.
- **Next() Block:** For every input tuple - return nothing if there are no more available tuples, and return the 'evaluated' tuple otherwise.
- **Close() Block:** De-initialize the input.

4. Aggregate

- **Open() Block:** Initialize the input list. If it is empty, use aggEmptyValue for the result. Else, create a hashmap of the grouped tuples. Iterate over every group, over every aggCall and over every tuple in the group. Inside the loop, compute the aggregate value using aggReduce. Pass the results to the next() block via a list.
- **Next() Block:** Return nothing if we traversed the entire results list. Else, increment the count and return the current entry of the results list.
- **Close() Block:** Empty.

5. Sort

- **Comparator:** This function returns the order of two input tuples based on the given collation. Iterate over all collections (going in index order) and return true/false based on the given direction.
- **Open() Block:** Initialize the input as a list and 'sortWith' using the above-mentioned comparator function. Initialize the 'offset'.
- **Next() Block:** Initialize the end condition based on the 'fetch' limit. Return nothing if the end of the list or the fetch limit is reached. Else, increment the count and return the current entry of the results list.
- **Close() Block:** Empty.

6. Join

- **Open() Block:** Initialize the left and right inputs as lists. Iterate over all left tuples, over all right tuples and over all keys in the left/right key list. Inside the loop, check if the left-right tuple pair is not equal for any key. If they are equal for all keys, pass the concatenated tuple pair to the next() block via a list.
- **Next() Block:** Return nothing if we traversed the entire results list. Else, increment the count and return the current entry of the results list.
- **Close() Block:** Empty.
- **Additional Note:** The described implementation for join can be made more efficient using a hashmap. The hashmap's 'contains' function can perform comparisons much faster than iterative comparisons. The attempt to implement this method was not successful (some test cases failed).

Submitted by: Kushagra Shah (316002)

Dated: 10/03/2021