EPFL CS-422

Francesco Intoci

Project 0 Report

1 Abstract

This short report aims to briefly describe the implementation and design choices made for EPFL CS-422 course project 0.

2 Scan, Filter, Project

As far as the three, above mentioned, operators are concerned, implementation is straightforward: each operator takes a tuple at a time from the lower input operator, applies its function, and returns the tuple to the next level. Scan is implemented using the helper function 'getRow' defined in the skeleton code.

3 Aggregate

For the Aggregate operator, first I decided to implement the operator by consuming all the tuples from the input operator in the 'open' function.Then, I distinguished two cases, whether the 'GROUP BY' clause (i.e. 'groupSet' field) is empty or not. In the former case, the only group will be the relation passed from the input operator, In the latter case, for each tuple of the relation, the hash of the field in the 'GROUP BY' clause is computed, and an HashTable mapping the hash to the group of tuple corresponding to a specific key of the 'GROUP BY', and one mapping the hash to the key(i.e. the fields), are formed. Ultimately, as described in the pseudo implementation, for each group we extract pairs of tuples and compute the aggregate until I have one value remaining. The tuple containing the key fields and the aggregate results are formed for each group.

4 Join

For the Join operator, I have decided to implement the 'Hash Join' algorithm. First all the tuples from the left input operator are consumed. For each tuple, an Hash Map mapping the hash of the fields used in the join condition(i.e. 'getLeftKeys') to the group of tuples having those field values is formed. For each tuple of the right input operator(accessed in a streaming fashion in the 'next()' method), if the Hash of the field values (i.e. 'getRightKeys') is contained in the Hash Map, then the tuple is joined with all the tuples in the entry of the map and added to a queue. The head of the queue is returned in 'next()'.

5 Sort

For the Sort operator,in my first implementation, I had simply implemented the 'sortWith' native function of Scala, invoked with a 'customCompare' function which, given two tuples, iterates on the keys of the 'ORDER BY' clause while the 'compareTo' method of the field of the tuples to compare returns 0, comparing them according to the direction requested. In order to improve the performance of the engine, I have decided to switch the implementation to a priority queue defined with a 'customOrdering', in the same spirit of the 'customCompare'.