Report Project 1

Introduction

The content may change, but the way I approach any Scala project remains the same. Also known as the Lego brick analogy, most of the time, looking at the input and the expected output of a function is enough to implement it without understanding the finer details…

Volcano-style: tuple-at-a-time

Once I discovered that most of the structures could be cast from one type to another (the Store to RowStore was actually quite tricky to understand), most of the functions were pretty straightforward to implement and a few well placed map() calls were enough to do the job. Although, as mentioned earlier, my grasp of Scala is a bit lacking despite all the courses we took during the Bachelor, meaning that I had to print every single map() call I made to be sure it fit the expected output.

Late materialization

To implement late materialization, I re-used most of my code from the first part and returned LateTuples (i.e a Tuples with a vid) instead of Tuples.

Query optimisation

This was probably the hardest part to implement. But the TAs were very helpful. For the LazyFetchRule, I just replaced the LogicalStitch with a LogicalFetch, so basically giving all the inputs of the LogicalStitch to the LogicalFetch (i.e the child RelNode, the RowType as well as the ColumnType). From there the LazyFetchProjectRule was very similar in terms of implementation, where, instead of instantiating a LogicalFetch with a None as arguments for projects, I gave it the projects from the LogicalProject node. What I still don't understand is why this project only applies to the column in the Fetch (i.e using the evaluator on column.getColumn(vid)) and not the whole input tuple. But according to the Moodle Forum this seemed like the right thing to do (also a couple more tests passed). Finally, for the LazyFetchFilterRule, we have that the LogicalFetch replaces the LogicalStitch, and the LogicalFilter that was an input of the Stitch, now takes as input the newly created LogicalFetch. In the end, one test for this part still fails but would have required too much time to actually debug.

Execution Models

The way I did this part is probably not the way you intended us to do it, but despite not being efficient at all, the tests pass with no time outs. I basically took the columns and transposed them, to retrieve back the tuples. Once I had Tuples, it was simply a matter of copy/pasting my code from the first task, tweaking it a bit, and then transposing back the result into columns... I chose to make the Join and Aggregate only generate active tuples. Although this technique is not full-proof, as two of the queries do not pass in the tests. I believe the Sort to be wrong but was not able to find out why.