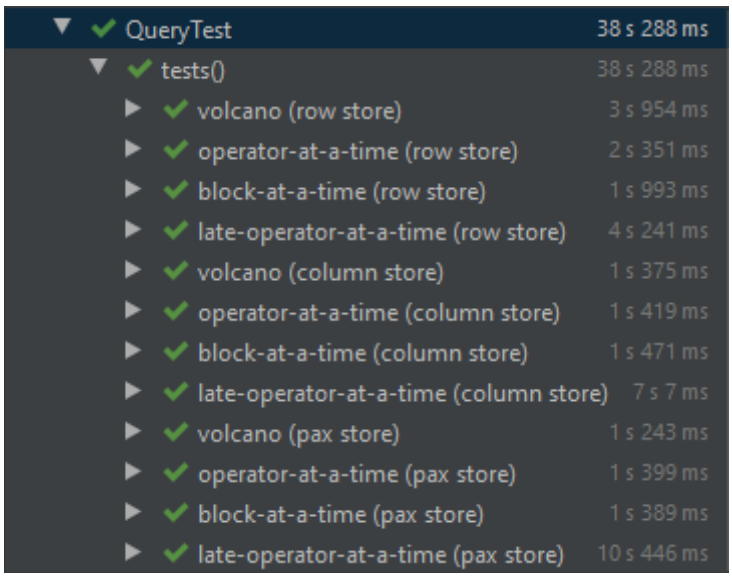
Task 2

Once implemented the different execution models and run queries to test them, it is possible to compare their relative performance, based on the implementation. In the following document, a few observation will be pointed out to explain the difference in performance. Finally, images will be presented to show the relative speed of the different models and layouts.



In the Volcano execution model, one tuple at a time is returned by every method. Despite expecting to have a faster Row Store implementation, as no conversion of the data format happens in the Scan method, Column and PAX store are actually more efficient. A possible explanation for this is the lower number of accesses to the database for large amount of data (nRows vs nCols/PAX).

In the Operator at a time execution model, results of operations are returned at once as a sequence of columns. This favors the PAX and Column storage, as no conversion is needed in the Scan operation. In addition, higher speed of these two storage layouts is to be attributed to the same reason listed in the previous paragraph.

The Block at a time execution model shares its relationships among the storage layouts with the Volcano, since data is returned as an array of (blockSize) rows. Conversion to and from blocks of tuples, made in order to perform the needed operations, hurts the efficiency of the model.

Late materialization operations are visibly less efficient than any of their counterpart. This may be due to the cost of the materialization operations, that is paid in most of the functions. The code might be optimizable by using evaluators to only materialize the needed data in methods such as Sort and Join.

Looking at single queries in detail, it is possible to point out how the different operations called lead to different performances among the execution models. We are going to analyse the queries run in the Row Store layout: this is not going to hinder the value of the experiment, as all operations but Scan are implemented in the same way for different layouts.

In the Volcano model, queries with an abundance of Sort, Aggregate or Join operation see their performance reduced, as misses on the instruction cache are not counterbalanced by possibility of pipelining. This can be clearly observed, for example, by the execution of the query q02.sql, that contains a large number of Sort operations. The following results have been obtained for Volcano, Operator at a time and Block at a time, respectively:

As we can see, Volcano has by far the longest execution time, as virtually no benefit is obtained from pipelining. Operator at a time, however, drastically improves the performance, proving to be a better model for this type of queries. Finally, block at a time provides an even better performance than operator at a time, which can be explained by the possibility of pipelining the few pipelinable operations without losing the advantages of the operator at a time execution.

On the other hand, queries with little to none non pipelinable operations lead to better performances for the Volcano model, compared to the operator at a time one. The following results have been obtained for Volcano, Operator at a time and Block at a time, respectively:

In this case, we can see that the Operator at a time model proves to be the least performing, followed by Volcano. Once again, Block at a time proves to be the fastest solution, adopting the best of both Volcano's and Operator at a time's features. The model proves in fact to be very adapt to the given query, that performs two Scans, two Projects and one Filter (all pipelinable and pipelined for the current implementation, for the Row Store layout we are analysing), as well as one Aggregate and one Join, that are not pipelined.

In conclusion, we can observe that different models and layouts, as well as their implementation, lead to drastic changes in performance.