Exercises for *Foundations in Data Engineering*, WiSe 23/24

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)
http://db.in.tum.de/teaching/ws2324/foundationsde

Sheet Nr. 05

Exercise 1 This exercise takes a closer look at which machine instructions are generated in different variants for the newline finding algorithm. To answer the following questions, you may find it helpful to use the Godbolt Compiler Explorer at https://godbolt.org/. Create a function in `C++` in the source window. Then, discover which instructions the compiler generates from your source code. (Note: Don't forget to choose appropriate optimization settings for the compiler at the top of the right-hand side window. E.g. use `-O3 -std=c++20`.)

1. Which instructions are necessary when searching for a newline character in a byte-array with a regular byte-wise for loop?
2. How many instructions are issued per non-newline character in the search loop?
3. How many instructions per byte are used in the bitwise-operation based solution presented in the lecture?
4. Why does the bitwise operation based solution execute faster than the byte-wise solution?

Exercise 2 Given a specific workload, which options are there to reduce the execution time?

Exercise 3

Please write SQL queries to answer the following questions on the TPC-H dataset, you can use the WebInterface to test your queries:

1. How many customers have an order whose comment contains the word packages?
2. What is the average number of digits in l_orderkey?
3. What are the names of all customers and suppliers?
4. Retrieve 10 customers and the corresponding nation. As output, create a JSON object for each customer with the custkey, name and the nation as embedded json object. The nation object shall contain the nationkey and name.

Exercise 4 Transform the following usage of coalesce into a usage of the case statement:

```
SELECT coalesce(description, 'None')
FROM sometable;
```

Exercise 5 Decorrelate this SQL query:

```
-- TPC-H Query 17
SELECT sum(l_extendedprice) / 7.0 as avg_yearly
FROM lineitem,
     part
WHERE p_partkey = l_partkey
  and p_brand = 'Brand#23'
  and p_container = 'MED BOX'
  and l_quantity < (
        SELECT 0.2 * avg(l_quantity)
        FROM lineitem
        WHERE l_partkey = p_partkey
      )
```

(Optional) Exercise 6 We continue with our `C++` exercise from the previous sheets. Again consider this simple query on the TPC-H dataset:

```
SELECT sum(l_extendedprice) FROM lineitem
```

Now, we want to improve our solution further by utilizing multiple processing cores. Clone the project from this repository and implement the missing code fragments in the section marked by `TODO`. Partition the input data into multiple chunks and compute the sum of each partition individually.

Once the implementation is done, measure the execution time and compare it with the single threaded approach. How well does your implementation scale, that means how much does it gain with each additional processing core? What new bottlenecks can you identify?

Hint: You can instantiate a `C++` template function by specifying parameters between angle brackets right after the function name e.g.`sum_extendedprice<true>(...)`. This page also provides an introduction to template functions.