Exercises for *Foundations in Data Engineering*, WiSe 22/23

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)
http://db.in.tum.de/teaching/ws2223/foundationsde

Sheet Nr. 02

Exercise 1 For the following SQL keywords, which GNU tools are useful to implement a similar functionality?

- FROM
- LIMIT
- SELECT
- WHERE
- ORDER BY
- GROUP BY

Solution:

- FROM: `cat`, most utils support input files, `join`
- LIMIT: `tail`, `head`
- SELECT: `cut`, `awk` (can perform computation on input columns)
- WHERE: `grep`, `awk` (supports line-wise grep style filters)
- ORDER BY: `sort`
- GROUP BY: `sort` + `awk`

Exercise 2

Answer the following queries for the *Linux Kernel Mailing List* data set using GNU utils and bash functionality:

1. Of how many lines does the longest mail consist?

   Solution:

   ```
   egrep "^Lines: [[:digit:]]+" gmane.linux.kernel | cut -d" "
      -f2 | sort -n -r | head -n1
   ```

2. Which mail received most (direct) replies?

   Solution:

   ```
   grep "^In-Reply-To:" gmane.linux.kernel | cut -d" " -f2-1000 | sort
      | uniq -c | sort -n -r | head -n1
   ```

Exercise 3

Answer the following queries for the TPC-H data set using GNU utils and bash functionality. You can download the data set using the links provided in the Moodle course and this link contains the schema.

1. According to the data set, which nations are in Europe?
   Solution:

   ```
   join -t'|' -1 3 -2 1 <(sort -t'|' -k3  nation.tbl) <(grep
      EUROPE region.tbl) | cut -d'|' -f3
   ```

   The purpose of this question is twofold: first, point out that join needs sorted input; second, show that it is beneficial to first do selection in region, then join the alternative would be:

   ```
   join -t'|' -1 3 -2 1 <(sort -t'|' -k3  nation.tbl) region.
      tbl | grep EUROPE | cut -d'|' -f3
   ```

   But in that case much more data would be put out from join and be processed by grep
2. How many lines are in lineitem.tbl?
   Solution:

   ```
   wc -l lineitem.tbl
   ```

3. Determine for each part how often it has been bought. (Hint: Sum up the quantities in lineitem.tbl, grouped by partkey)
   Solution:

   ```
   sort -n -t'|' -k2 lineitem.tbl | \
   awk -F'|' 'BEGIN{partkey=0;sum_quantity=0}\
   {if (partkey!=$2){print partkey, sum_quantity;\
           partkey=$2; sum_quantity=0}; \
     sum_quantity+=$5}\
   END{print partkey, sum_quantity}'
   ```

4. Now use the command `time` to determine how long it takes to compute query 3.

   `time` takes the command to measure as argument. For example `time echo hi` runs the command `echo hi`, which writes hi to the standard out stream, and then prints resource usage information to the standard error stream.
5. The measured command produces a lot of output. In this case, execution time will most likely be dominated by printing to the terminal. Of course we are not interested in measuring how long it takes to print to the terminal. Rather we want to measure how long the computation takes. Therefore, please redirect the output to /dev/null. The latter is a device which discards all data written to it. Luckily for us, it is very fast. Usage looks like this `time echo hi > /dev/null`.
6. Now, pipe the command's output through `head` (by removing > /dev/null and appending `| head`) and perform the time measurement on this modified variant again. Why does the modified command take much shorter to execute?

Exercise 4

In this exercise, we will compare a GNU utils based query on the TPC-H dataset against a simple `C++` implementation. You will gain some experience in setting up a `C++` toolchain, in the language itself as well as in performance analysis tools such as `perf` in this exercise. Please note that `perf` is a command-line tool only available on Linux.

1. Start by answering the following query for the TPC-H dataset using GNU utils and bash functionality (hint: have a look at `awk`):

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

Solution:

```
LC_NUMERIC="C" awk -F'|' 'BEGIN {sum=0.0} {sum += $6} END {
    printf "%f", sum}' lineitem.tbl
```

`LC_NUMERIC="C"` to ensure floats use '.' as decimal separator.

2. Now that you have a working `awk` based solution it is time to clone the project at https://gitlab.db.in.tum.de/fde/tpchjoin and implement the missing code fragments in the section marked by `"TODO"`. You will find further instructions on how to build and run this project in the provided `README` file.

Once the implementation is done, do simple performance measurements:

- Compare this to `awk` execution time.
- Estimate how fast the program can possibly be on your local machine. Assume that the input file is read from memory.
- Find out where most of the time is spent and why this program is not as fast of your estimate. Tools that can help here: `perf stat`, `perf record`, `perf report`. Which part would be a good candidate for optimization? (This exercise will be continued on the next exercise sheet.)

`perf` provides useful performance analysis tools for Linux. With `perf stat` you can obtain an analysis of certain kernel and CPU relevant statistics like the ratio between CPU cycles spent within your program and the kernel itself. `perf record` and `perf report` should be used pairwise. The former allows you to generate a profile of your program by issuing `perf record <your program>`. The latter is used to analyse the generated profile. You will find an estimate of how much time your program spent within a certain function down to the individual CPU instructions among each function call/instruction. It is advisable to compile your program with debug information (add the `-g` argument for `g++`) in order to obtain the original code lines together with the corresponding symbols.

Solution:

```
int64_t sum_extendedprice(const std::string& file_name) {
    std::ifstream f(file_name);
    CsvParser lineitem = CsvParser(f).delimiter('|');

    int64_t price_sum = 0;
    // For every line
    for (auto row : lineitem)
        // Take the sixth column, convert it to integer and
        // add it to the accumulator
        price_sum += ToInt(row[5]);

    return price_sum;
}
```

Estimate for runtime:
The file size is 750MB. Assuming it resides in the file cache, we can read it from memory. Thus $0.75GB/(20GB/s) = 0.0375s$