Programming Assignment 3

Brandeis University, COSI 127b, Spring 2019

Instructor: Ryan Marcus

In this assignment, you'll be creating an optimizer and execution engine for a relational database management system. Your system must take a restricted subset of SQL as an input, evaluate those queries against a dataset, and produce the correct results. You may write your system in any language you choose, although **five extra credit points** will be awarded to anyone who does not use Java.

Inputs

Your program will communicate through standard input and standard output. Each time your program is invoked, the following will occur:

1. A comma-separated line of paths to CSV files, each corrosponding to a relation (e.g., `A.csv` is the data for the relation `A`), will be sent to your program.
2. There will be a pause of a pre-determined amount of time, in which your program can prepare for query processing (e.g., convert each CSV file into a different format, build index structures).
3. After the pause, the next line of input will contain the number of queries you are to process.
4. The rest of the input will contain SQL queries which you should process.

After processing each SQL query, you should output the result to standard output (e.g., `System.out.println` in Java).

Relations

The first line of input sent to your program is a comma-separated list of paths to CSV files. For example, this line might look like:

```
data/xxxs/A.csv,data/xxxs/B.csv,data/xxxs/C.csv
```

This line would corrospond to a database with three relations (`A`, `B`, and `C`), each stored in the respective file. This line is *not* ensured to be in any particular order.

Each CSV file will contain data, for example:

```
123,43,45,31,809
54,-623,693,1942,123
```

... which corrosponds to a table with 5 columns and two rows. CSV files will never have a header row. All values in each CSV file are (potentially negative)

integers. All values are within the range of Java's `int` primitive (i.e., a 32-bit signed integer).

All relation names will be a single uppercase English ASCII character (i.e., `A` through `Z`). There are thus a maximum of 26 relations.

Indexing time

After the first line of input, your program will recieve no input for a fixed period of time, called indexing time. During this time, you can read each CSV file and write files to disk. You should use this time to prepare for query processing.

Query processing

After the indexing time delay, you will recieve the total number of queries to process, followed by the queries themselves. For example:

```
3
SELECT SUM(D.c0), SUM(D.c4), SUM(C.c1)
FROM A, B, C, D
WHERE A.c1 = B.c0 AND A.c3 = D.c0 AND C.c2 = D.c2
AND D.c3 > -9496;

SELECT SUM(A.c1)
FROM A, C, D
WHERE A.c2 = C.c0 AND A.c3 = D.c0 AND C.c2 = D.c2
AND C.c2 < 2247;

SELECT SUM(A.c6), SUM(A.c2)
FROM A, B, C, D
WHERE A.c2 = C.c0 AND A.c1 = B.c0 AND C.c2 = D.c2
AND A.c4 = -9868;
```

Each SQL query will occupy four lines.

1. The first line will contain several `SUM` aggregates, specifying a number of columns that should be totaled for each query. You may assume that `SUM` is the only aggregate used, and that all columns in the `SELECT` clause are aggregated. In other words, each SQL query will output only a single row, and each column of that output row will be the sum of some column. There will never be a `GROUP BY` clause.
2. The second line will contain the `FROM` clause, which will always contain at least one relation.
3. The third line will contain the first part of the `WHERE` clause. Each predicate on this line will be separated by an `AND`, and will represent a simple equijoin predicate.
4. The final line will contain all non-join predicates. Each predicate on this line will contain a column on the left hand side and a constant value on

the right hand side. Only the operators `=`. `>`, and `<` will be used. There may be zero or more of these predicates. This line will always end in a semicolon.

Each query will be separated by a blank line.

Your program must output the correct result for each query, in the order the queries are given. Each query result should be formatted as a comma-separated list, with no spacing. For example:

```
132,5432191
```

... may be the output for a particular query. Each query result should end in a newline.

Data

Several datasets are provided, ranging in size from `XXXS` (small enough to compute by hand), and `L` (multiple gigabytes). These datasets are provided in the `data/` directory. Each dataset also comes with a `queries.sql` file containing the queries that will be sent to your program, and a `answers.txt` file containing the correct results for each query.

You *may not* read the `answers.txt` file from your program. Assume the `answers.txt` file is not available during query processing.

Program setup

Your code should include two shell scripts:

- `compile.sh`: this script should invoke all the commands required to compile your code, e.g. call `javac`. If your code does not need to be compiled, you may leave this file blank. The `compile.sh` script *may not* access any data in the `data` directory.
- `run.sh`: this script should run your program, accepting input and producing query results as output. For example, this script may contain `java MyProgram` or `python3 main.py`.

To run your program while you are developing, you may use our `run_with_data.py` script. For convenience, this script waits only 1 second before sending the query workload. This script will call `run.sh` and feed your program the proper input, printing out the results produced by your program. For example, to run using the `XXXS` dataset:

```
python3 run_with_data.py ../data/xxxs
```

To check the correctness of your program, you may use our `grade.py` script. This script will call `run.sh` in much the same way as `run_with_data.py`, but will also enforce a particular amount of indexing time, a timeout, and will automatically check your query results against the correct answer. For example,

to run your code with 5 seconds of indexing time and a 30 second timeout on the `XXXS` dataset:

```
python3 grade.py 5 30 ../data/xxxs
```

The `grade.py` script will record your program's standard error and standard out into `proc_stderr` and `proc_stdout`.

Finally, the `package.sh` script will pack up everything in your working directory into a single `dist.tar.gz` package, which you can submit for grading. Please note the maximum file size limit for this package will be 5MB (and thus you should ensure not to keep the `data` directory in the same folder).

Data

The dataset is too large to distribute over LATTE. You can download it like this:

```
wget http://cs.brandeis.edu/~rcmarcus/pa3_data.tar.gz
tar xzvf pa3_data.tar.gz
```

Uncompressed, the data is a little over 5GB.

Environment

The performance of your code will be measured in the same environment as every other student. Your code will execute on a machine with `1GB` of RAM and approximately `50GB` of hard disk space. *You should pay special attention to how your program uses memory. If you use over 1GB of memory, your program will crash.* This machine will *not* be connected to the Internet. You may safely write data to the working directory your program is executed in.

Several compilers and programming languages are installed, including `gcc`, `g++`, `javac`, `rust`, `cargo`, `python3`. For `python3`, `numpy` and `pandas` are installed.

You can create a virtual machine of the exact execution environment we will test your code with using `vagrant`. To SSH into such a machine, install Vagrant, create a new directory, and execute:

```
vagrant init RyanMarcus/arch-dev
vagrant up
vagrant ssh
```

You can edit the resulting `Vagrantfile` to limit the amount of memory the VM gets (set it to 1024 for 1GB).

Restrictions and requirements

There are many ways to "cheat" on this assignment, including, but not limited to, using the `sqlite3` Python module, downloading PostgreSQL and submitting it (or another database system), or using other high-level libraries that do query

optimization or execution for you (e.g. Calcite, Spark). If you have any doubts at all about if something is allowed, it is best to ask Ryan. "Cheaters" will be given zeros and reported to the academic honesty board, if appropriate.

It is additionally possible to cheat by special-casing the answers. For example, one could hard code print statements to simply emit the contents of the `answers.txt` file, and `grade.py` would say that all was well. Obviously, this is not permitted. When we grade your code, in addition to the provided datasets, we will also test your code on new datasets that will not be made available. It is therefore in your best interest not to hard code *anything* about each dataset (e.g., do not hard-code a query result or even some fact about a particular table column).

In general, with the exception of `numpy` and `pandas` for Python, external libraries or code you find online is *not* permitted for this assignment. If you feel like a particular library would be helpful, and think it should be allowed, ask Ryan. They will be considered on a case-by-case basis.

Additionally, your `compile.sh` script must actually compile your program. You may not submit compiled code, the code must be built by us in order for it to count for your grade.

Submitting to the server

Within the first two weeks of the assignment being out, we will provide an evaluation server. You will be able to submit your code to the evaluation server, which will execute and time your performance. This performance will then be recorded, and your submitted code will be saved.

At the final due date, we will verify your best submission by manually inspecting it for cheating.

Checkpoints & Due Dates

While the final due date for this assignment is the last day of instruction, there will be several checkpoints. Late policies *do not apply* to these checkpoints. Missing either checkpoint will result in a 15% deduction to your final assignment grade.

- April 12th, midnight: you must have a successful run logged on the evaluation server. This means you have submitted code that produced the correct answer within the timeout for all datasets.
- April 26th, midnight: you must have a successful run logged on the evaluation server that beats the baseline.
- May 2nd, midnight: final due date.

Grading

TBD – based on class poll.