Assignment-1

In this assignment, we will refresh our basic of DBMS and use PostgresSQL for an OSS project that retrieves and downloads data from GitHub, called GHTorrent. GHTorrent works by following the Github event timeline and then retrieving all items linked from each event recursively and exhaustively. To make monitoring and debugging easier, the GHTorrent maintainers use extensive runtime logging for the downloader scripts.

Here is an extract of what the GHTorrent log looks like:

```
DEBUG, 2017-03-23T10:02:27+00:00, ghtorrent-40 -- ghtorrent.rb:
Repo EFForg/https-everywhere exists

DEBUG, 2017-03-24T12:06:23+00:00, ghtorrent-49 -- ghtorrent.rb:
Repo Shikanime/print exists

INFO, 2017-03-23T13:00:55+00:00, ghtorrent-42 -- api_client.rb:
Successful request. URL:
https://api.github.com/repos/CanonicalLtd/maas-
docs/issues/365/events?per_page=100, Remaining: 4943, Total: 88
ms

WARN, 2017-03-23T20:04:28+00:00, ghtorrent-13 -- api_client.rb:
Failed request. URL:
https://api.github.com/repos/greatfakeman/Tabchi/commits?sha=Tab
chi&per_page=100, Status code: 404, Status: Not Found, Access:
ac6168f8776, IP: 0.0.0.0, Remaining: 3031

DEBUG, 2017-03-23T09:06:09+00:00, ghtorrent-2 -- ghtorrent.rb:
Transaction committed (11 ms)
```

Each log line comprises of a standard part (up to `.rb:`) and an operation-specific part. The standard part fields are like so:

1. Logging level, one of `DEBUG`, `INFO`, `WARN`, `ERROR` (separated by `,`)
2. A timestamp (separated by `,`)
3. The downloader id, denoting the downloader instance (separated by `--`)
4. The retrieval stage, denoted by the Ruby class name, one of:
    - `event_processing`
    - `ght_data_retrieval`
    - `api_client`
    - `retriever`
    - `ghtorrent`

Loading and parsing the file

For the remaining of the assignment, you need to use the shared file “datafile”.

1. Download the log file and write a function to load it in PostgreSQL. Create your own schema for importing the data.
2. How many records does the table contain?

Basic counting and filtering

3. Count the number of WARNing messages.
4. How many repositories were processed in total? Use the `api_client.` (Focus on WARN messages and use the suffix part of the URL after repos till ?.)

Analytics

5. Which 10 clients did the highest HTTP requests?
6. Which 10 client did the highest FAILED HTTP requests? (Operation part starts with the string ”Failed”)
7. What is the most active hour of day?

8. What is the most active repository?
9. Which access keys are failing most often? *Hint*:: extract the `Access: ...` part from failing requests

Indexing

Typical operations on the database require grouping on a specific part of each record and then calculating specific counts given the groups. While this operation can be achieved with the `groupby` family of funcions, it can be useful to create indexes.

`I`ndexes enable us to quickly access the dataset. To see their effect on large datasets, create index on the downloader_id and run the following analytics queries.

10. Compute the number of different *repositories* accessed by the client `ghtorrent-22` . Note the time taken by your query.
11. Now drop your index and compute the number of different *repositories* accessed by the client `ghtorrent-22`. Note the time taken by your query now.

Joining

We now need to monitor the behaviour of interesting repositories. Use this link to download a list of repos into which we are interested to. This list was generated on Oct 10, 2017, more than 7 months after the log file was created. The format of the file is CSV, and the meaning of the fields can be found on the GHTorrent project web site documentation.

12. Read in the CSV file into another table call it *interesting*. How many records are there?
13. How many records in the log file refer to entries in the *interesting* file?

14. Which of the *interesting* repositories has the most failed API calls?