Software Engineer Test Case

Introduction

The test case solution should be in a separate GitHub repo. In test case please do you best to show your software engineering skills including:

- Architecture and design
- Code comments and docstrings
- Unit-tests where applicable (Especially for Task 2).

Create **docker-compose** with 2 images:

- Docker container to create **PostgreSQL** database with tables `bars_1` and **bars_2 and fill them** with data from **bars_1.csv** and **bars_2.csv** respectively. A candidate should decide on tables' structures and column types based on the data in CSV files.
    - Also create `error_log` table with columns: **launch_timestamp(datetime), date(datetime), symbol(str), message(str)**.
- Docker container with microservice from Task 1.2.

Task 1.1

For this section, please send us not only the document (word, Jupyter, doesn't matter) with answers, but also the code used to generate them. The task should be solved using SQL only (no Pandas).

1. What is the % of symbols from table **bars_1** for which the following condition is satisfied:
   `Adj.Close(t) > Average(Adj_close{t-40:t}) for at least one t in 2019`?
2. Write a SQL query to calculate the average **dollar volume** (dollar volume = Adj Close * Volume) in February 2019. What is the average dollar volume in February 2019?

3. Rank stocks in 2015 by **Positive Volume** in ascending order.

   `Positive Volume(t) = Volume(t) if Adj.Close(t) >= Adj.Close(t-1) else 0.`

4. For each stock, calculate `Average Absolute Daily Percent Change`

   `Average Absolute Daily Percent Change = Mean[Abs(Daily % changes for a stock)]`

Task 1.2

Task 2 should be implemented and delivered as a container which is a part of Docker-compose from Task 1. You can use **Pandas**, **SQL**, **bash**, and **cron** to execute this task.

Create a **cron** job inside of a Docker container which does the following:

Every 30 seconds, it takes next batch of **20 000** rows from **bars_2**. For each row among these pulled 20 000 rows:

- If record's `<SYMBOL>` is not present in **bars_1** → put error message to **error_log** table with
  - **launch_timestamp** equal to the current timestamp,
  - **date** equal to record's `<DATE>`,
  - **symbol** equal to record's `<SYMBOL>`
  - **message** `'<SYMBOL> not present in tables_bars_1 on <DATE>'`

  where `<SYMBOL>`, `<DATE>` are the record's Symbol and Date values.
- If record's `<CLOSE>` is bigger than **Minimum of** `<SYMBOL>` **Close prices over last 10 days for a** `<SYMBOL>` from **bars_1** -> append the record to **bars_1**, else put error message to **error_log** table with
  - **launch_timestamp** equal to the current timestamp,
  - **date** equal to record `<DATE>`,
  - **symbol** equal to record `<SYMBOL>` ,
  - **message** `'<SYMBOL> close price is not bigger than the minimum over the past 10 days on <DATE>'` ,

  where `<SYMBOL>`, `<DATE>`, `<CLOSE>` are record's Symbol, Date and Close values respectively.

- When these 20 000 rows are processed, they should be deleted from **bars_2**.
- If no records are present in **bars_2** → put error message to **error_log** with
  - **launch_timestamp** equal to the current timestamp,
  - **date** equal to `NaN`,
  - **symbol** equal to `NaN`,
  - **message** `'No values available in bars_2'`.

Our team will run your docker-compose and analyze the correctness of the results of the cron job.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/42e2a53f-bff5-450e-af33-bf81f1f33c1a/db_data.zip

Task 2

Imagine that we have a parquet file with trade data. We need to emulate a W**ebSocket** which asynchronously replays trade data in chronological order from this file. In this task you need to implement:

1. Websocket which replays historical trades from parquet files (we provide test parquet files).
2. If several trades occur **on the same timestamp** - they should be sent simultaneously.
3. Websocket can't send a trade with timestamp **t** earlier than a trade with timestamp **t-N**.

The code should implement a runnable service (server-side) such that external client can connect to websocket to receive the data.

Websocket should send json-format messages.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/09ecc995-aa07-40eb-b723-7819022a6558/trades_sample.parquet