# TECH CHALLENGE

## Data Engineer - **Rather Labs**

Documentation of the setup, thought process and different inquiries about the project.

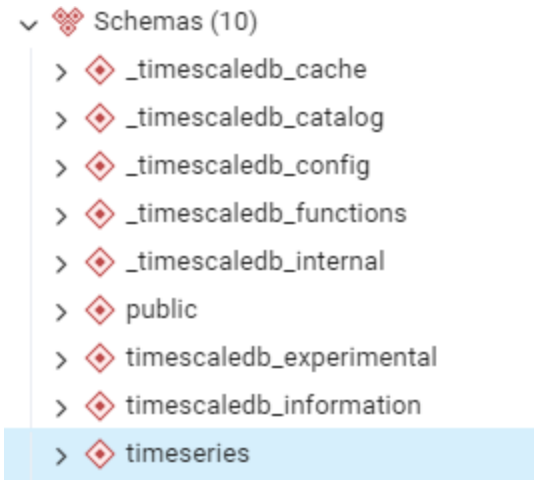
Mariano Tancredi

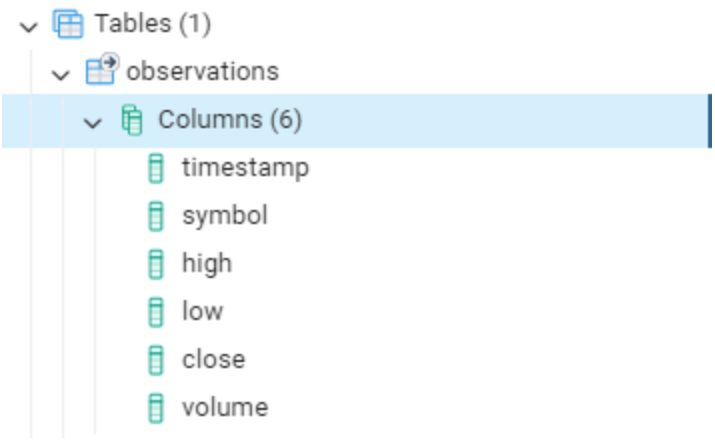
# Introduction

The project for this challenge required the creation and management of an ETL pipeline as well as developing an analytics system to check on the data. The challenge asked to use Python and a TimeScale DB to create a clean and useful solution.

# Database Schema/Setup

For this project, I used the 30 days free trial of TimeScale Cloud to create my own server. I connected to it through pgAdmin and created a **TimeSeries** Schema with a single table labelled **Observations** on it:




The columns on the observations table refer to the dataset I used for this challenge (Trades.csv). I configured the observations table to be a **Hypertable**, which led the table to use the timestamp column as the indexing column.

To run this project, I created a **constants** file which contains all the variables needed to run the script. This file can be modified depending on the user database information. The constants for the database connection are:

- HOST: Host of the database
- DATABASE: Name of the database
- USER: User for the database
- PASSWORD: Password for the database
- PORT: Port of the database
- DATABASE_URL: The url that is used in the script. It's actually created from the previous values, so it does not require any type of modification

# Data Parser/Loader Script

The script called **parser.py** is in charge of both transforming the data and inserting it into the database. Since the data required almost no cleaning, we could adjust all this logic to just a few lines of codes. The script is made to be easy to run with other types of csv, you just have to adjust the variables for it in the constants file.
The script uses pandas to parse the csv, transform the numeric and datetime columns, drop any null value that exists and insert the rest into the table. I used the Panda integrated function for this since it was faster than parsing the data manually and inserting it. The script also retrieves the latest date in the database and slices the data frame so that it only inserts new timestamps. This is useful since we are working with almost a million rows and we avoid checking the database for repeated values. This can be set to false by the **UPDATE_HISTORIC** variable in case we need to update previous values.
The constants used in this file, beside the ones listed previously, are:

- RETRIEVE_DATE: This is a query used to retrieve the latest date in the database
- PRE_ACTION_QUERY: This query creates the table for the time series in case it is not created
- TABLE_NAME: Name of the table we are going to use
- SCHEMA: Name of the schema we are going to use
- TIME_COLUMN: Name of the column containing the timestamp, in case you want to test the project with another type of csv.
- NUMERIC_COLUMNS: List of numeric columns for conversion, in case you want to test the project with another type of csv.
- FILE_SOURCE: Path to the CSV

# Queries Script

For part 2 and 3 of this project, I decided to create a simple menu that runs in a Python Shell to show some Data Analytics and Plotting points. This script also uses the constants for the database connection. The script uses a Pandas Dataframe for simple calculations but also retrieves some data through the use of TimeScale specific functions. The points available in this menu are:

- **Average closing for each symbol:** Shows a Dataframe with the average closing price of each symbol in the database
- **Filter Data based on a symbol:** Shows only data related to the selected symbol
- **Correlation between variables:** Shows a heatmap grid of the correlation between numeric variables

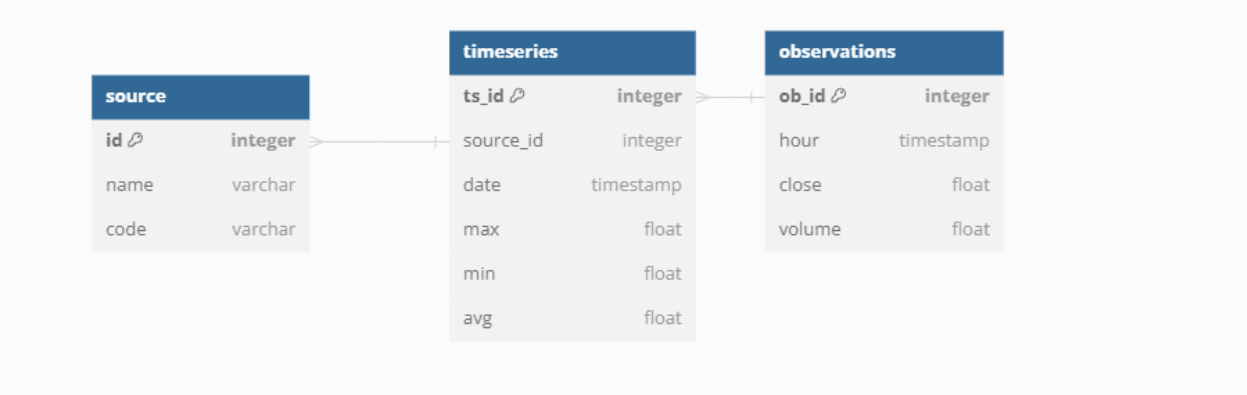
- **Average trading volume of a stock in the last quarter:** This was specified in the challenge document.
- **Average volume per hour of each symbol:** This graphic is shown in a Plot
- **Select time series with gap filled:** Shows a plot of a query in Timescale that uses the **time_bucket_gapfill** function to fill any time series value that is missing with an approximate value.
- **Calculate the volatility for each symbol:** Shows a plot with the volatility of each symbol.

# Optimization and Scaling

For this project, I ran some sort of trial and error. Respecting the Parser Script, I first tried doing it only by using Python Dicts and Python built-in functions (Besides psycopg2) but the result was too slow given the size of the dataset. From there, I tried using the Pandas library to parse and insert the data and found myself with a much faster approach. According to these benchmarks, Pandas works much faster than the CSV library when working with big Datasets.

As for the database, my initial approach was very different. First, I used the other two csv given to me for this challenge (es.csv and bz.csv). I designed a schema where each time series would contain an id, the date, the average for that day, the max and minimum values for that day, the source (Depending from where it was parsed **ES** for E-MINI S&P 500 (CME MINI) or **BZ** for BRENT

(NYMEX GLOBEX)) and the observations for each hour would be stored in another table associated to the timeseries id of that date. A diagram of this:




This was good because it allowed us to retrieve useful information such as average or max for a day in the fastest way as well as having the data correctly segmented. But it had some major flaws in it, the first one being that TimeScale uses the timestamp as ID, so this solution could not be transformed into a hypertable, and second, the logic for the code part was much more complicated than the final solution.

After this, I decided to switch datasets and to keep the current solution, which is much easier to read/use and also easier to scale.

To implement this in a Production Environment, I would integrate a cloud-computing service. For example, we could use AWS and develop a Workflow to parse, load and consume the data. We could create a Glue Job for the data parsing/loading. Deploy a cluster for Timescale since it has a plug-in created for AWS, and from there we could dump the data into a Dashboard or even develop a simple API with an EC2 instance so that Analyst can hit and endpoint and retrieve the necessary data.