Many financial analysts and data engineers require **large volumes of historical stock data** for backtesting, risk models, or real-time analytics. Traditionally, collecting 10+ years of data for hundreds of tickers could take hours. By harnessing **asynchronous I/O** in Python (asyncio + asyncio.to\_thread), combined with **yfinance** for data retrieval and **PostgreSQL** for robust storage, the same job can finish in **mere seconds** or under a minute, even for over **one million rows** of data.

*(You can include a diagram—“Figure 1”—showing how an asyncio event loop hands off multiple I/O tasks to a thread pool, and how data eventually gets persisted in PostgreSQL.)*

**1. Setting Up the PostgreSQL Database**

Before we dive into the Python code, let’s ensure we have a **PostgreSQL** database ready. You can create it via the psql terminal (or a GUI like pgAdmin). Below is an **example** of how to create a database and a user:

Now you have a **finance\_db** database and a **finance\_user** who can access it. This sets the stage for our data ingestion pipeline.

**2. Extracting and Validating S&P 500 Tickers**

**2.1 Fetching Symbols from Wikipedia**

The S&P 500 constituents can be scraped directly from Wikipedia using pandas.read\_html. The snippet below retrieves the first table on the page and extracts the “Symbol” column:

sp500\_tickers = get\_sp500\_tickers()

print(f"S&P 500 Ticker Count: {len(sp500\_tickers)}")

**2.2 Fixing Ticker Symbols**

Some symbols—like BRK.B—must be changed to BRK-B to match Yahoo Finance’s format:

**2.3 Asynchronous Validation**

We validate each ticker asynchronously to confirm it returns data on Yahoo Finance:

* **asyncio.to\_thread**: Offloads the blocking .history() call to a background thread, leaving the main coroutine free to schedule other tasks.
* We also use a **semaphore** to limit concurrency (defaulting to **50** tasks at once).

**3. Filtering for 10+ Years of Data**

We next assess **how many years** of data each ticker provides on Yahoo Finance. Only those with at least **10 years** of history will be fetched in bulk.

**3.1 Collecting Date Ranges**

**3.2 Determining Which Tickers Qualify**

We store the results in a DataFrame, then calculate each ticker’s total **Years Available**:

**4. Setting Up PostgreSQL in Python**

With the **finance\_db** database created, we can connect using SQLAlchemy. You can adapt the credentials to your environment.

**4.1 Creating the Table**

This schema holds **ticker**, **date**, **OHLC** prices, adjusted close, and volume, with a composite primary key on **(Ticker, Date)**.

**5. Asynchronous Data Retrieval and Persistence**

**5.1 Fetching the Daily Records**

For each ticker, we perform an **I/O-bound** request to Yahoo Finance via yfinance. We offload the .history() call to a thread so our event loop can schedule **multiple** fetches simultaneously:

**5.2 Writing to PostgreSQL**

pandas.to\_sql is typically a blocking operation, so we also run it in a thread:

**5.3 Orchestrating All Tickers**

We combine the fetching and storing steps into one **task** per ticker:

By leveraging concurrency (max\_concurrent=50), we effectively create **50 parallel** I/O requests at any given moment, drastically cutting down the total runtime.

**6. Bringing It All Together**

Finally, here is a conceptual main() function that stitches everything together. We target **10 years** of daily data by subtracting 10 \* 365 days from today’s date:

**Conclusion**

By integrating **asyncio** and **yfinance** with a robust **PostgreSQL** backend, you can rapidly ingest millions of data points from Yahoo Finance. The secret sauce lies in:

1. **Asynchronous I/O**: Multiple .history() calls execute concurrently, minimizing idle time.
2. **Thread Offloading**: Blocking operations—data fetches and database inserts—are dispatched to threads, keeping the main event loop unblocked.
3. **PostgreSQL**: Provides a powerful, scalable store for subsequent queries, analysis, or modeling tasks.