Debdas Ghosh & Connor Wilson

INFSCI 1540 Data Engineering

22 April 2021

Data Engineering Behind Stock Analysis

**Overview**

The project enables data-driven decision making and market analysis of stock trends for companies listed in the S&P 500. Utilizing data drawn from Wikipedia, Yahoo Finance, Stocktwits, and Statista, the repository combines trade information, quarterly revenues, and investor and analyst sentiments to provide a more comprehensive view of stocks within the index. From day to week to quarter, headquarters location to industry to sub-sector, the warehouse provides an aggregated view of the data along multiple dimensions, able to drill-down along a variety of paths for more granular data analysis. Using this data, the project seeks to answer the following questions:

1. Which city/state has the highest volume of stock trades in the S&P 500 per day/week/month/quarter/year?
2. What is the combined quarterly/yearly revenue for companies in the same sector/industry?
3. What are the market sentiments for different companies on the S&P 500?

The project can be found on GitHub at <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis>.

**Project Structure (Docker)**

Utilizing the Docker platform to run components of the data pipeline in separate containers, the project is comprised of several key technologies. At the highest level, an Apache web server is used to support two instances of phpMyAdmin, each corresponding to a MySQL database: an instance for the operational database (ODB) containing raw data and another for the aggregated data warehouse (DW). A Kafka broker container is utilized for streaming data from the ODB to the DW, while a ZooKeeper container manages storing the streaming data. The docker-compose file for the project can be found at: <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/docker-compose.yml>.

**Docker Compose File:**

version: '2'

services:

web-server:

image: php:7.4.3-apache

volumes:

- "./html/:/var/www/html/"

ports:

- "8080:80"

mysql-odb-server:

image: mysql:8.0.19

environment:

MYSQL\_ROOT\_PASSWORD: secret

volumes:

- mysql-data:/var/lib/mysql\_odb

ports:

- "13306:3306"

mysql-dw-server:

image: mysql:8.0.19

environment:

MYSQL\_ROOT\_PASSWORD: secret

volumes:

- mysql-data:/var/lib/mysql\_dw

ports:

- "23306:3306"

phpmyadmin-odb:

image: phpmyadmin/phpmyadmin:5.0.1

environment:

PMA\_HOST: mysql-odb-server

PMA\_USER: root

PMA\_PASSWORD: secret

ports:

- "15000:80"

phpmyadmin-dw:

image: phpmyadmin/phpmyadmin:5.0.1

environment:

PMA\_HOST: mysql-dw-server

PMA\_USER: root

PMA\_PASSWORD: secret

ports:

- "25000:80"

broker:

image: confluentinc/cp-kafka:5.5.1

hostname: broker

container\_name: broker

depends\_on:

- zookeeper

ports:

- "29092:29092"

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: 'zookeeper:2181'

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:9092,PLAINTEXT\_HOST://192.168.1.227:29092

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

zookeeper:

image: confluentinc/cp-zookeeper:5.5.1

hostname: zookeeper

container\_name: zookeeper

ports:

- "2181:2181"

environment:

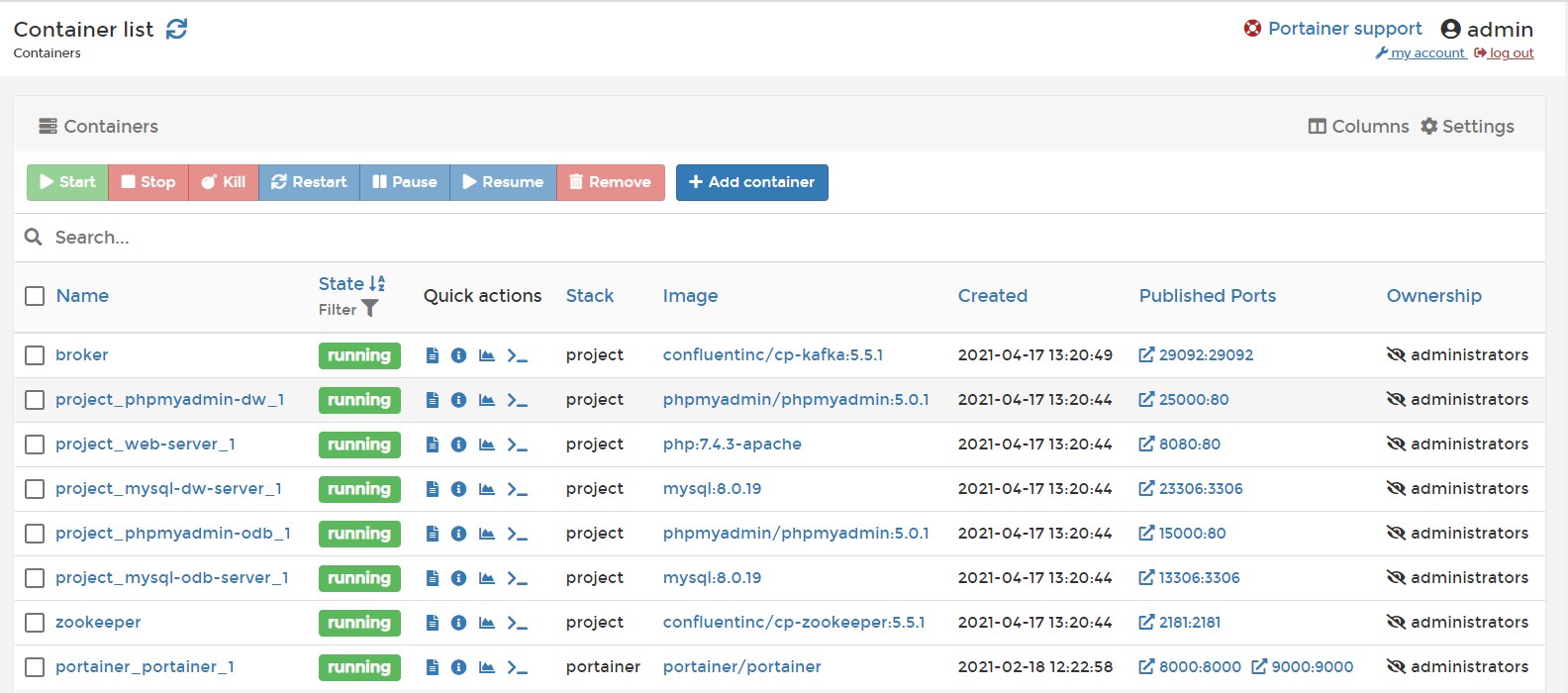
ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

volumes:

mysql-data:

List of Containers get created after running the docker-compose:



**Data ETL**

|  |  |  |
| --- | --- | --- |
| Source | Type | Description |
| Wikipedia | Semi-Structured | Information about companies on the S&P 500 stock index |
| Yahoo Finance | Structured | Daily stock market data for S&P 500 companies, ranging from 1 January 2017 to 8 April 2021 |
| StockTwits | Unstructured | Tweets about the stock market (“stock twits”), fetched from the website’s API |
| Statista | Structured | Quarterly revenues of S&P 500 companies |

Beginning with S&P 500 company information from Wikipedia, data for each company’s stock symbol, security (company name), sector, sub-industry, and headquarters location is pulled from the page. While each company’s name, associated stock symbol, sector, and sub-industry are used as-is, the headquarters location is separated into state and country, filtering out companies not headquartered in the United States. This information is then outputted to a CSV spreadsheet using a script written in R and later loaded into the ODB and DW by a Python script.

Next, the quarterly revenues for select companies are downloaded from the statistics site Statista based on available information. From the Microsoft Excel files provided by the website, the data for quarterly revenue (in billions of dollars) is extracted along with financial quarter, fiscal year, and stock symbol and outputted to a combined CSV file through another script written in R. In a similar manner to company information, the data is inserted into both databases using the aforementioned Python script, relegated to a separate table in the ODB and appearing in a derivative format in an aggregated FACT table in the DW.

Subsequently, daily stock information provided by Yahoo Finance is queried from the beginning of the 2017 fiscal year to April 2021 for each company included in the S&P 500 index (per Wikipedia), with information on the date, opening stock price, closing stock price, highest and lowest sale prices, and volume of stocks sold aggregated into a single, combined CSV spreadsheet using an R script. In combination with the previous spreadsheets, this is the last data source loaded into each repository by the previous Python files.

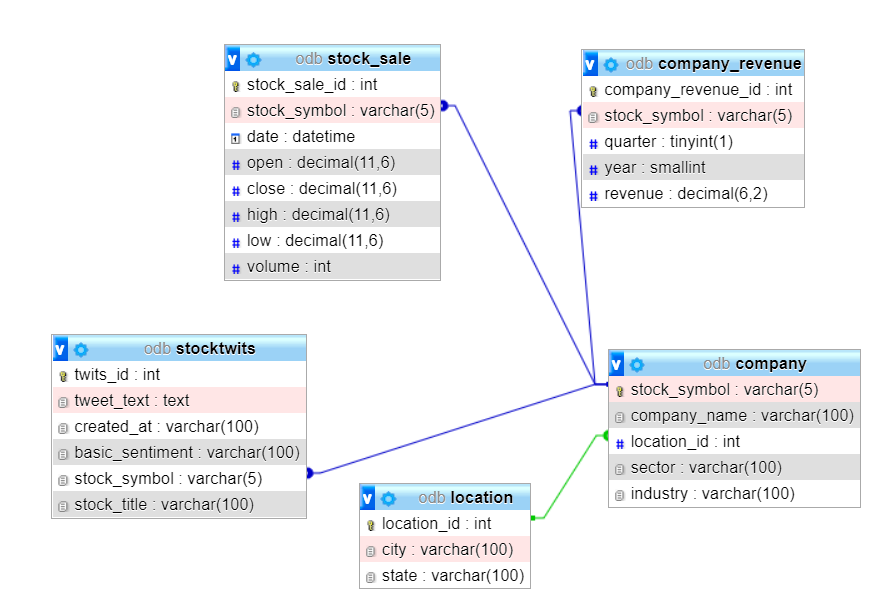
Finally, the Stocktwits API is used to collect information about the latest tweets and sentiments in relation to supported stocks (i.e., Apple, inc.), which are collected into a series of JSON files from an R script. (Due to limitations of the website’s API, the data collected for the project includes 6000 “twits” each about Apple and Amazon.) The data in the JSON files is then streamed from an Apache Kafka producer into a consumer for inserting the raw data into the ODB, with updates subsequently streamed to another Kafka consumer used for aggregating information this information and inserting it into the DW. Utilizing sentiment data from gathered “twits,” a sentiment score considering the “bearishness,” “bullishness,” or “unemotionality” of each is aggregated for each company’s stock.

**Operational Database**

We have 5 tables in our ODB, which are collecting operational data.

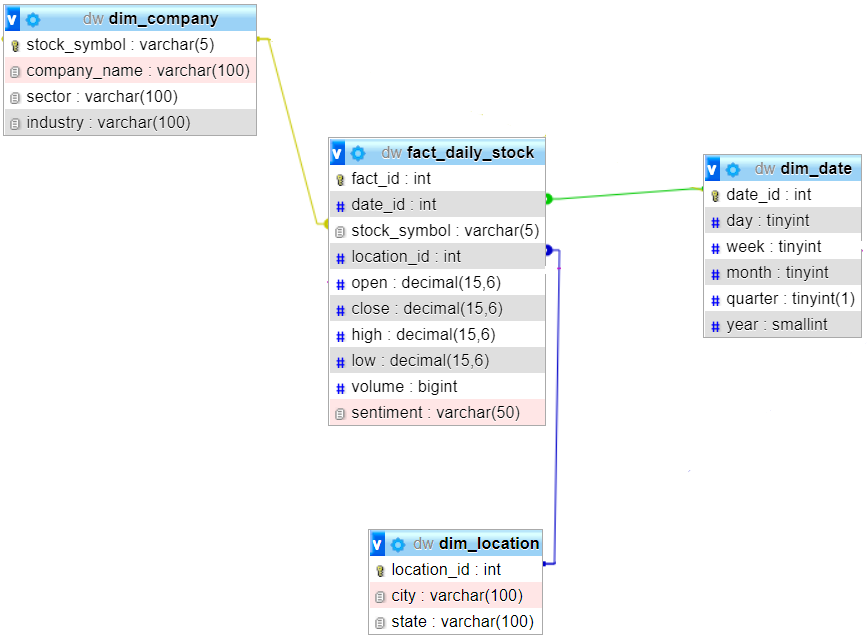
1. Company – In this table we are recording all S&P 500 companies' basic data, like Company name, Stock Symbol, Sector, Industry type and Location of that company.
2. Company revenue – It records quarter wise revenue (in billions of dollars) details for all those companies.
3. Location – It is a location master data table, consists of city and state pairs.
4. Stock Sales – This is the main transaction table, where we are recording daily stock trading for each company. Notable columns are stock volume, opening rate, closing rate, highest rate achieved and lowest rate achieved.
5. Stocktwits – It records company wise twits and basic sentiment of that twit fetched from Stocktwits API. We populate this table through Kafka streaming.

ODB Design:

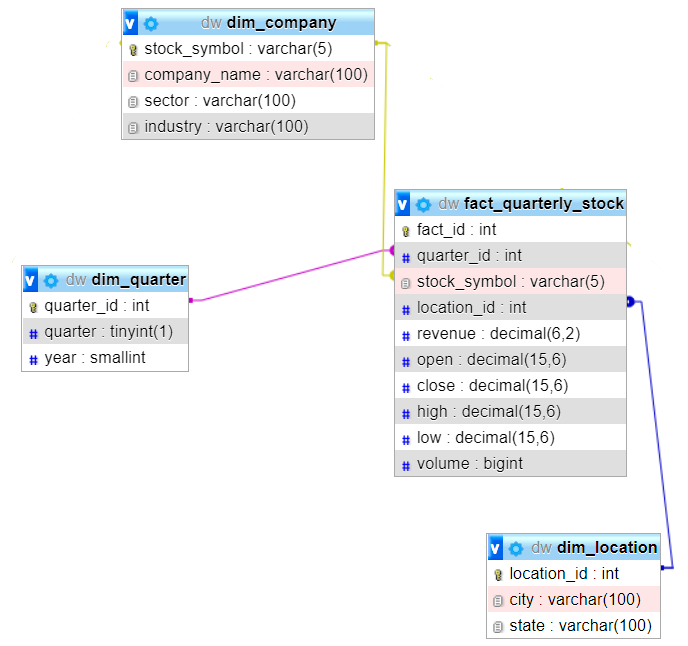


**STAR Schema**

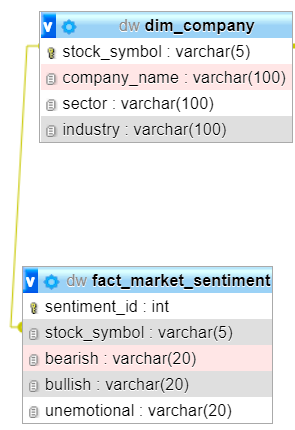
The STAR schema of the DW centers around three FACT tables. The first, the “fact\_daily\_stock,” table includes three dimensions: date, company (stock), and (headquarters) location, describing several measures: the opening and closing prices of a stock, its highest and lowest sale prices, and the volume of trades.



The second FACT table, “fact\_quarterly\_stock,” similarly describes these same measures with the addition of the corresponding quarterly revenue of the stock’s company along three similar dimensions: fiscal quarter, company (stock), and (headquarters) location.

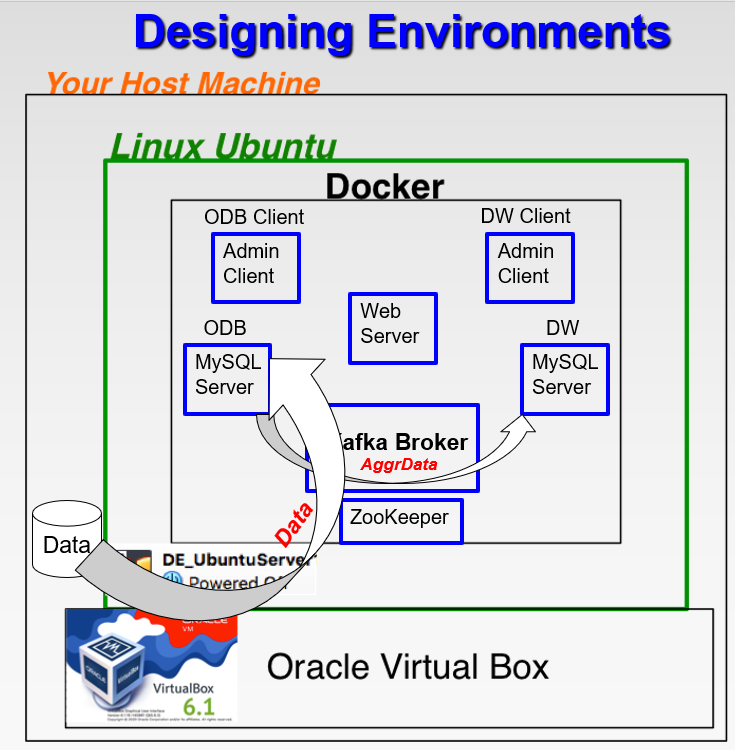


Finally, the third “fact\_market\_sentiment” table describes the “bearishness,” “bullishness,” and “unemotionality” of sentiments expressed about a stock along the single dimension of (stock) company.



In total, the schema includes four unique dimension tables. The table “dim\_quarter” represents the unique fiscal quarters for data contained in the warehouse. The “dim\_date” table describes all unique dates of stock trade data stored in the warehouse, split into component days, weeks, months, fiscal quarters, and years. The “dim\_location” table describes the city and state of various company’s headquarters, and “dim\_company” records information about S&P 500 companies, including their name, stock symbol, sector, and industry (sub-sector).

The DDL statements for both the ODB and DW can be found at <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/create_odb.sql> and <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/create_dw.sql>, respectively. The combined SQL statements and programming logic for populating the ODB can be found at <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/load_odb.py> and for populating the DW at <https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/load_dw.py>.



**DDL for ODB:**

DROP SCHEMA IF EXISTS `odb` ;

CREATE SCHEMA IF NOT EXISTS `odb` DEFAULT CHARACTER SET utf8 ;

USE `odb` ;

DROP TABLE IF EXISTS `odb`.`location` ;

CREATE TABLE IF NOT EXISTS `odb`.`location` ( `location\_id` INT NOT NULL AUTO\_INCREMENT, `city` VARCHAR(100) NOT NULL, `state` VARCHAR(100) NOT NULL, PRIMARY KEY (`location\_id`))ENGINE = InnoDB;

DROP TABLE IF EXISTS `odb`.`company` ;

CREATE TABLE IF NOT EXISTS `odb`.`company` ( `stock\_symbol` VARCHAR(5) NOT NULL, `company\_name` VARCHAR(100) NOT NULL, `location\_id` INT NOT NULL, `sector` VARCHAR(100) NOT NULL, `industry` VARCHAR(100) NOT NULL, PRIMARY KEY (`stock\_symbol`), INDEX `fk\_location\_id\_idx` (`location\_id` ASC) VISIBLE, CONSTRAINT `fk\_location\_id` FOREIGN KEY (`location\_id`) REFERENCES `odb`.`location` (`location\_id`) ON DELETE NO ACTION ON UPDATE NO ACTION)ENGINE = InnoDB;

DROP TABLE IF EXISTS `odb`.`stock\_sale` ;

CREATE TABLE IF NOT EXISTS `odb`.`stock\_sale` ( `stock\_sale\_id` INT NOT NULL AUTO\_INCREMENT, `stock\_symbol` VARCHAR(5) NOT NULL, `date` DATETIME NOT NULL, `open` DECIMAL(11,6) NOT NULL, `close` DECIMAL(11,6) NOT NULL, `high` DECIMAL(11,6) NOT NULL, `low` DECIMAL(11,6) NOT NULL, `volume` INT NOT NULL, PRIMARY KEY (`stock\_sale\_id`), INDEX `fk\_s\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_s\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `odb`.`company` (`stock\_symbol`) ON DELETE NO ACTION ON UPDATE NO ACTION)ENGINE = InnoDB;

DROP TABLE IF EXISTS `odb`.`company\_revenue` ;

CREATE TABLE IF NOT EXISTS `odb`.`company\_revenue` ( `company\_revenue\_id` INT NOT NULL AUTO\_INCREMENT, `stock\_symbol` VARCHAR(5) NOT NULL, `quarter` TINYINT(1) NOT NULL, `year` SMALLINT(4) NOT NULL, `revenue` DECIMAL(6,2) NOT NULL, PRIMARY KEY (`company\_revenue\_id`), INDEX `fk\_r\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_r\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `odb`.`company` (`stock\_symbol`) ON DELETE CASCADE ON UPDATE CASCADE)ENGINE = InnoDB;

DROP TABLE IF EXISTS `odb`.`stocktwits` ;

CREATE TABLE IF NOT EXISTS `odb`.`stocktwits` ( `twits\_id` INT NOT NULL AUTO\_INCREMENT, `tweet\_text` text NOT NULL, `created\_at` varchar(100) NOT NULL, `basic\_sentiment` varchar(100) NOT NULL, `stock\_symbol` VARCHAR(5) NOT NULL, `stock\_title` varchar(100) NOT NULL, PRIMARY KEY (`twits\_id`), INDEX `fk\_t\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_t\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `odb`.`company` (`stock\_symbol`) ON DELETE CASCADE ON UPDATE CASCADE)ENGINE = InnoDB;

**DDL for DW:**

DROP SCHEMA IF EXISTS `dw` ;

CREATE SCHEMA IF NOT EXISTS `dw` ;

USE `dw` ;

DROP TABLE IF EXISTS `dw`.`dim\_date` ;

CREATE TABLE IF NOT EXISTS `dw`.`dim\_date` ( `date\_id` INT NOT NULL AUTO\_INCREMENT, `day` TINYINT(2) NOT NULL, `week` TINYINT(2) NOT NULL, `month` TINYINT(2) NOT NULL, `quarter` TINYINT(1) NOT NULL, `year` SMALLINT(4) NOT NULL, PRIMARY KEY (`date\_id`))ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`dim\_location` ;

CREATE TABLE IF NOT EXISTS `dw`.`dim\_location` ( `location\_id` INT NOT NULL, `city` VARCHAR(100) NOT NULL, `state` VARCHAR(100) NOT NULL, PRIMARY KEY (`location\_id`))ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`dim\_company` ;

CREATE TABLE IF NOT EXISTS `dw`.`dim\_company` ( `stock\_symbol` VARCHAR(5) NOT NULL, `company\_name` VARCHAR(100) NOT NULL, `sector` VARCHAR(100) NOT NULL, `industry` VARCHAR(100) NOT NULL, PRIMARY KEY (`stock\_symbol`))ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`fact\_daily\_stock` ;

CREATE TABLE IF NOT EXISTS `dw`.`fact\_daily\_stock` ( `fact\_id` INT NOT NULL AUTO\_INCREMENT, `date\_id` INT NOT NULL, `stock\_symbol` VARCHAR(5) NOT NULL, `location\_id` INT NOT NULL, `open` DECIMAL(15,6) NOT NULL, `close` DECIMAL(15,6) NOT NULL, `high` DECIMAL(15,6) NOT NULL, `low` DECIMAL(15,6) NOT NULL, `volume` BIGINT NOT NULL, `sentiment` VARCHAR(50) NULL, PRIMARY KEY (`fact\_id`), INDEX `fk\_date\_id\_idx` (`date\_id` ASC) VISIBLE, INDEX `fk\_location\_id\_idx` (`location\_id` ASC) VISIBLE, INDEX `fk\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_d\_date\_id` FOREIGN KEY (`date\_id`) REFERENCES `dw`.`dim\_date` (`date\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_d\_location\_id` FOREIGN KEY (`location\_id`) REFERENCES `dw`.`dim\_location` (`location\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_d\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `dw`.`dim\_company` (`stock\_symbol`) ON DELETE NO ACTION ON UPDATE NO ACTION)ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`dim\_quarter` ;

CREATE TABLE IF NOT EXISTS `dw`.`dim\_quarter` ( `quarter\_id` INT NOT NULL AUTO\_INCREMENT, `quarter` TINYINT(1) NOT NULL, `year` SMALLINT(4) NOT NULL, PRIMARY KEY (`quarter\_id`))ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`fact\_quarterly\_stock` ;

CREATE TABLE IF NOT EXISTS `dw`.`fact\_quarterly\_stock` ( `fact\_id` INT NOT NULL AUTO\_INCREMENT, `quarter\_id` INT NOT NULL, `stock\_symbol` VARCHAR(5) NOT NULL, `location\_id` INT NOT NULL, `revenue` DECIMAL(6,2) NOT NULL, `open` DECIMAL(15,6) NOT NULL, `close` DECIMAL(15,6) NOT NULL, `high` DECIMAL(15,6) NOT NULL, `low` DECIMAL(15,6) NOT NULL, `volume` BIGINT NOT NULL, PRIMARY KEY (`fact\_id`), INDEX `fk\_quarter\_id\_idx` (`quarter\_id` ASC) VISIBLE, INDEX `fk\_location\_id\_idx` (`location\_id` ASC) VISIBLE, INDEX `fk\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_q\_quarter\_id` FOREIGN KEY (`quarter\_id`) REFERENCES `dw`.`dim\_quarter` (`quarter\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_q\_location\_id` FOREIGN KEY (`location\_id`) REFERENCES `dw`.`dim\_location` (`location\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_q\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `dw`.`dim\_company` (`stock\_symbol`) ON DELETE NO ACTION ON UPDATE NO ACTION)ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`fact\_market\_sentiment` ;

CREATE TABLE IF NOT EXISTS `dw`.`fact\_market\_sentiment` ( `sentiment\_id` INT NOT NULL AUTO\_INCREMENT, `stock\_symbol` VARCHAR(5) NOT NULL, `bearish` varchar(20) NOT NULL, `bullish` varchar(20) NOT NULL, `unemotional` varchar(20) NOT NULL, PRIMARY KEY (`sentiment\_id`), INDEX `fk\_tw\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_qtw\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `dw`.`dim\_company` (`stock\_symbol`) ON DELETE NO ACTION ON UPDATE NO ACTION)ENGINE = InnoDB;

DROP TABLE IF EXISTS `dw`.`agg\_stock\_volume` ;

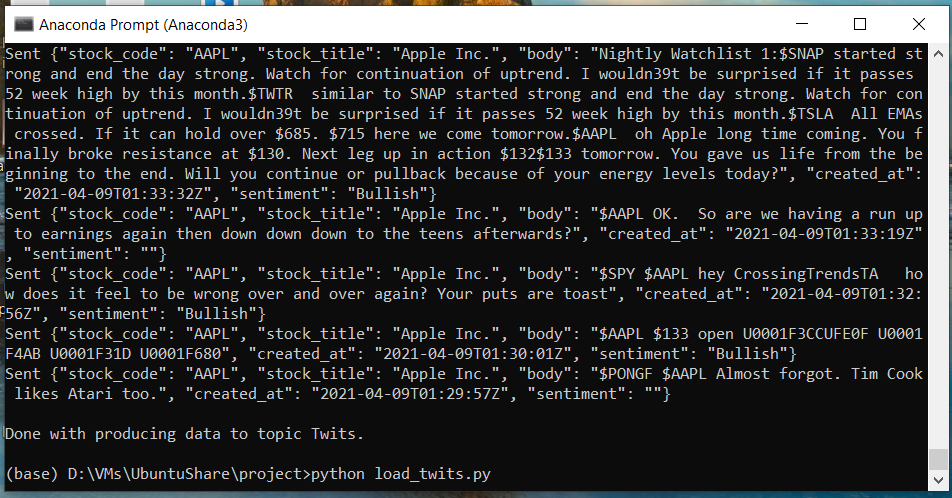
CREATE TABLE `dw`.`agg\_stock\_volume` (`agg\_id` int NOT NULL AUTO\_INCREMENT,`date\_id` int NOT NULL,`location\_id` int NOT NULL,`volume` bigint NOT NULL, PRIMARY KEY (`agg\_id`), INDEX `fk\_agg\_date\_id\_idx` (`date\_id` ASC) VISIBLE, INDEX `fk\_agg\_location\_id\_idx` (`location\_id` ASC) VISIBLE, CONSTRAINT `fk\_agg\_d\_date\_id` FOREIGN KEY (`date\_id`) REFERENCES `dw`.`dim\_date` (`date\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_agg\_d\_location\_id` FOREIGN KEY (`location\_id`) REFERENCES `dw`.`dim\_location` (`location\_id`) ON DELETE CASCADE ON UPDATE CASCADE ) ENGINE=InnoDB;

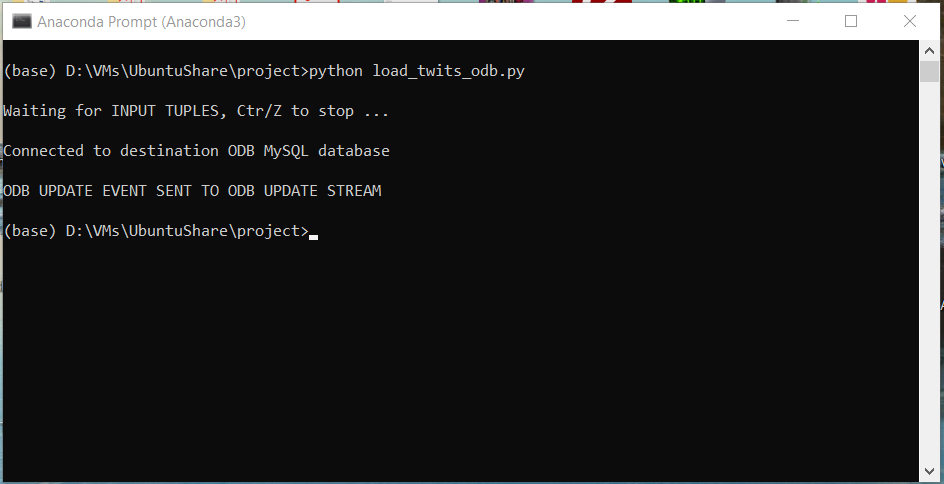
DROP TABLE IF EXISTS `dw`.`agg\_quaterly\_revenue` ;

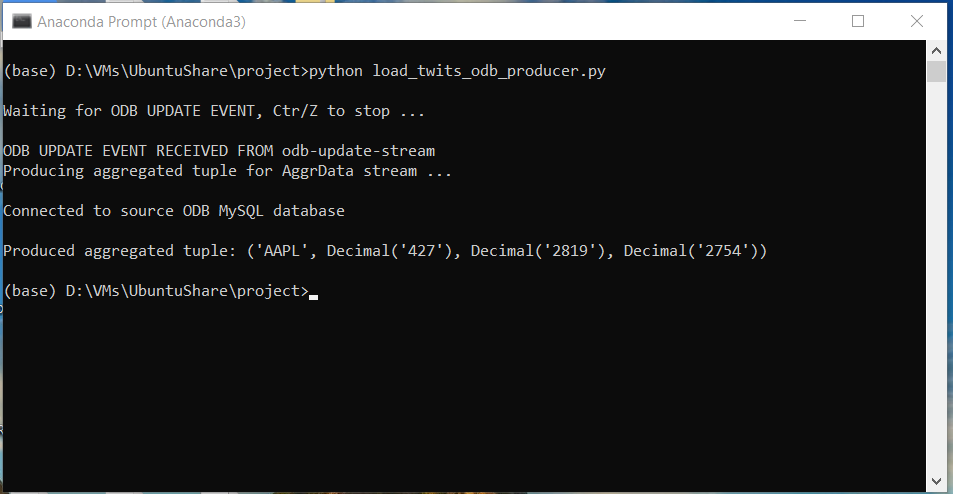
CREATE TABLE `dw`.`agg\_quaterly\_revenue` (`agg\_id` int NOT NULL AUTO\_INCREMENT,`quarter\_id` int NOT NULL,`stock\_symbol` varchar(5) NOT NULL,`revenue` decimal(6,2) NOT NULL, PRIMARY KEY (`agg\_id`), INDEX `fk\_agg\_quarter\_id\_idx` (`quarter\_id` ASC) VISIBLE, INDEX `fk\_agg\_stock\_symbol\_idx` (`stock\_symbol` ASC) VISIBLE, CONSTRAINT `fk\_agg\_q\_quarter\_id` FOREIGN KEY (`quarter\_id`) REFERENCES `dw`.`dim\_quarter` (`quarter\_id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `fk\_agg\_q\_stock\_symbol` FOREIGN KEY (`stock\_symbol`) REFERENCES `dw`.`dim\_company` (`stock\_symbol`) ON DELETE NO ACTION ON UPDATE NO ACTION) ENGINE=InnoDB;

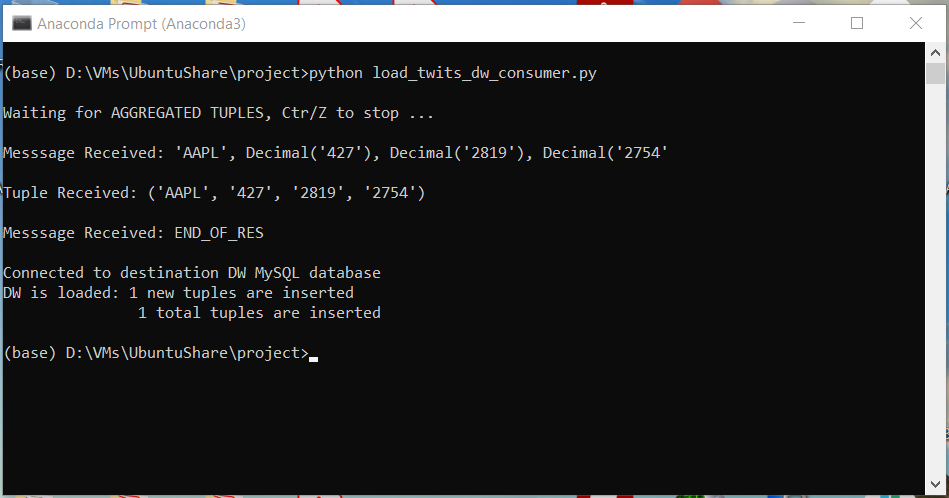
**Data Streaming**

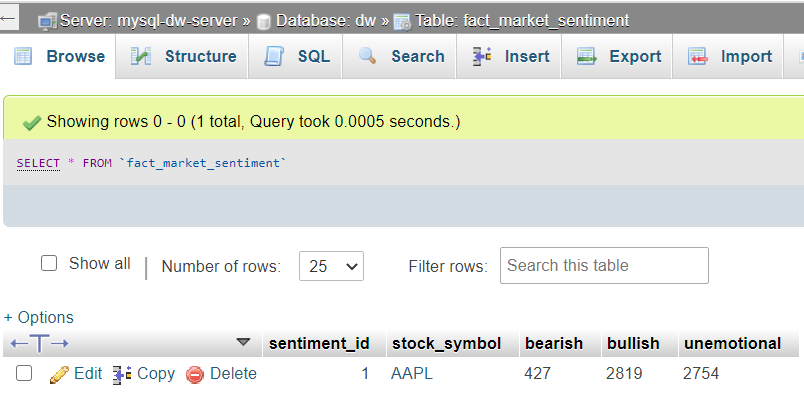
Data streaming using Apache Kafka was utilized for loading data from Stocktwits into the ODB and DW. The corresponding Python scripts include: load\_twits.py, load\_twits\_odb.py, load\_twits\_odb\_producer.py, and load\_twits\_dw\_consumer.py available on the provided GitHub repository.

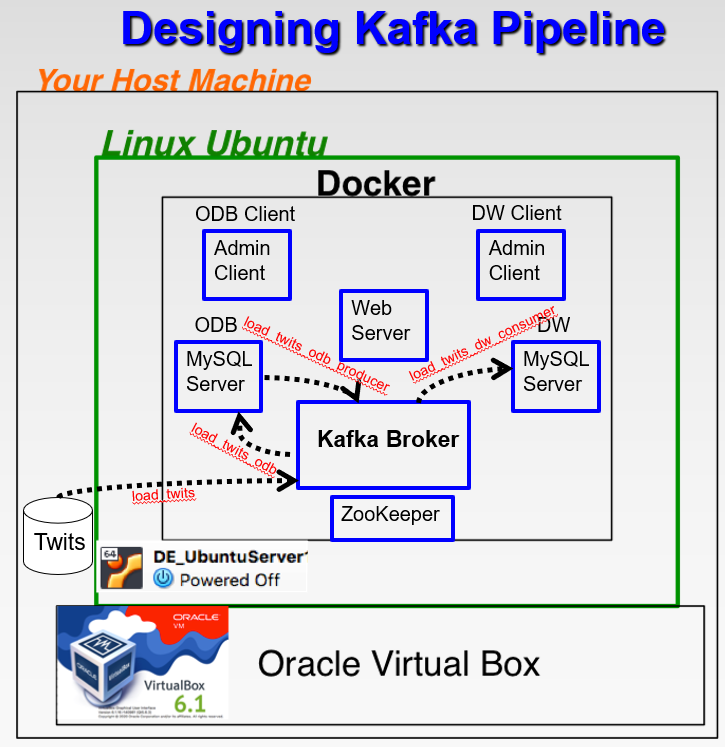












**Summary Tables**

We have created two summary tables.

* 1. agg\_stock\_volume: Created by aggregating the stock volume location wise and date wise from fact\_daily\_stock table. So that it will help us to speed up the execution time for our requirement 1 - Which state/city has the most number of volumes of stocks for the S&P 500 American companies on date/week/month/quarter/year basis?
  2. agg\_quaterly\_revenue: Created by aggregating the revenue quarter wise and company wise from fact\_daily\_stock table. So that it will help us to speed up the execution time for our requirement 2: Get revenue distribution of companies Sector/Industry wise and Quarter/Year wise

**Supported Queries**

-- Which state/city has the most number of volumes of stocks for the S&P 500 American companies on date/week/month/quarter/year basis?

-- Example: Which state has the most number of volumes of stocks for the S&P 500 American companies in a particular day?

SELECT dl.state, sum(rc.volume) AS tot\_vol

from agg\_stock\_volume rc

INNER JOIN dim\_location dl on dl.location\_id=rc.location\_id

INNER JOIN dim\_date d on d.date\_id = rc.date\_id

WHERE d.year = 2017 and d.month = 1 and d.day = 3

GROUP BY dl.state ORDER BY tot\_vol DESC;

-- Revenues distribution of companies Sector/Industry wise and Quarter/Year wise

-- Example: Revenues per Sector in a particular Quarter

SELECT sc.sector, sum(rc.revenue) AS revenue

from dim\_company sc

INNER JOIN agg\_quaterly\_revenue rc on rc.stock\_symbol=sc.stock\_symbol

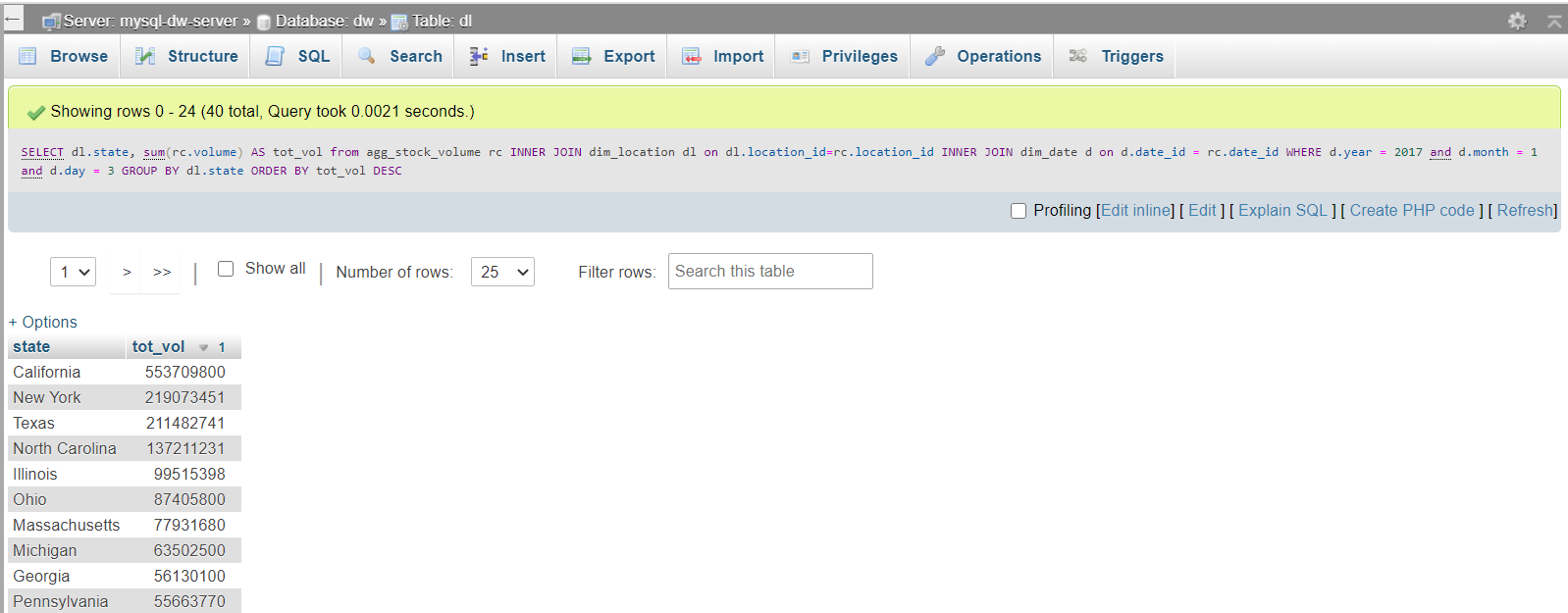
INNER JOIN dim\_quarter q on q.quarter\_id = rc.quarter\_id

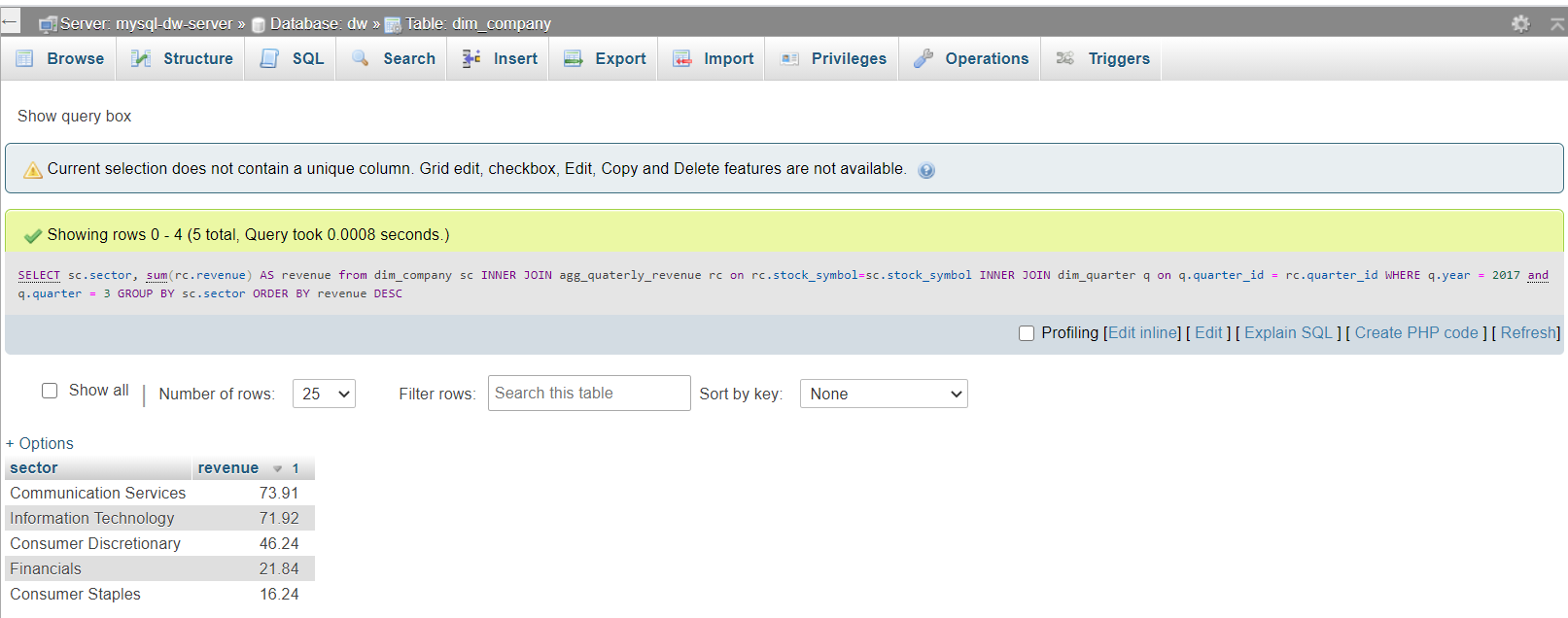
WHERE q.year = 2017 and q.quarter = 3

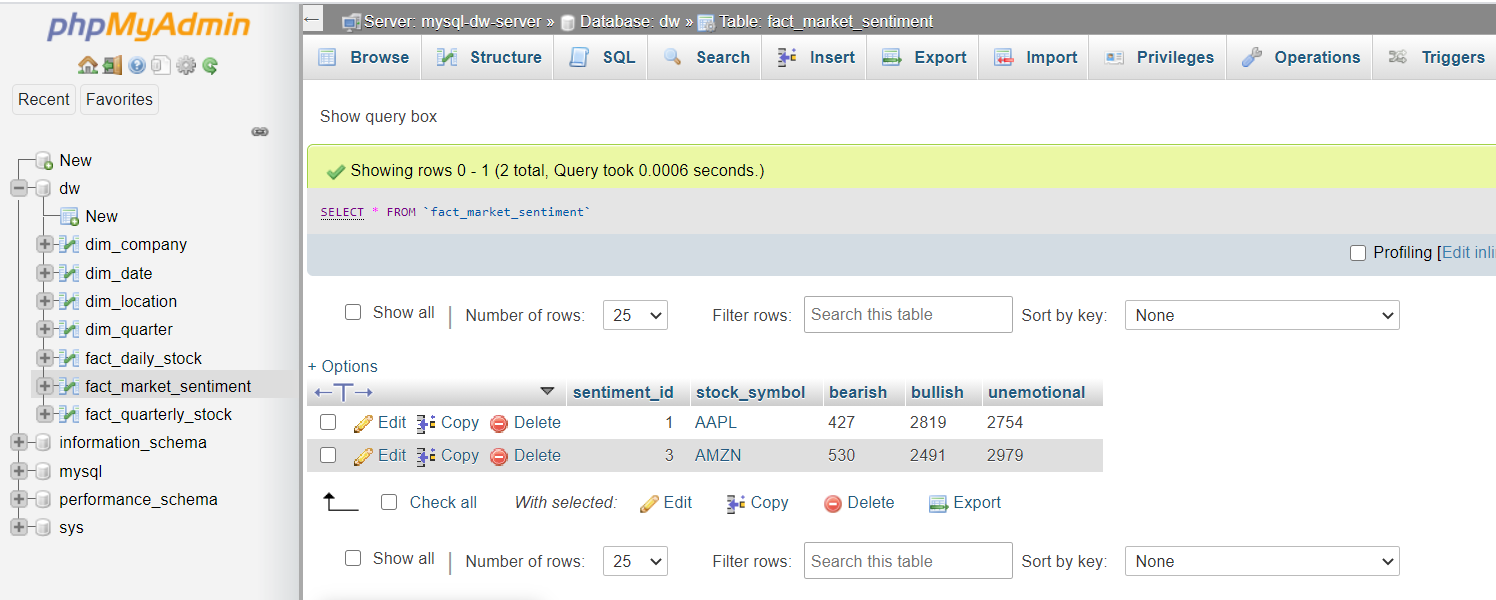
GROUP BY sc.sector ORDER BY revenue DESC;

-- Latest Sentiments about a Company

SELECT \* FROM `fact\_market\_sentiment`;







**Conclusions and Future Scope**

This project’s final data warehouse was developed to support business queries to analyze trends within the stock market, focused on American companies included in the S&P 500 index. While the supported queries enable the analysis of trends in stock trades by date (or fiscal quarter), company location, and sector/industry, the scope could further be expanded to incorporate additional companies and new dimensions. Most critically, the basic sentiment analysis utilized in the project could be further revised and expanded to help correlate market sentiments found on social media with actual trends seen in stock trades.