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-industry, 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)

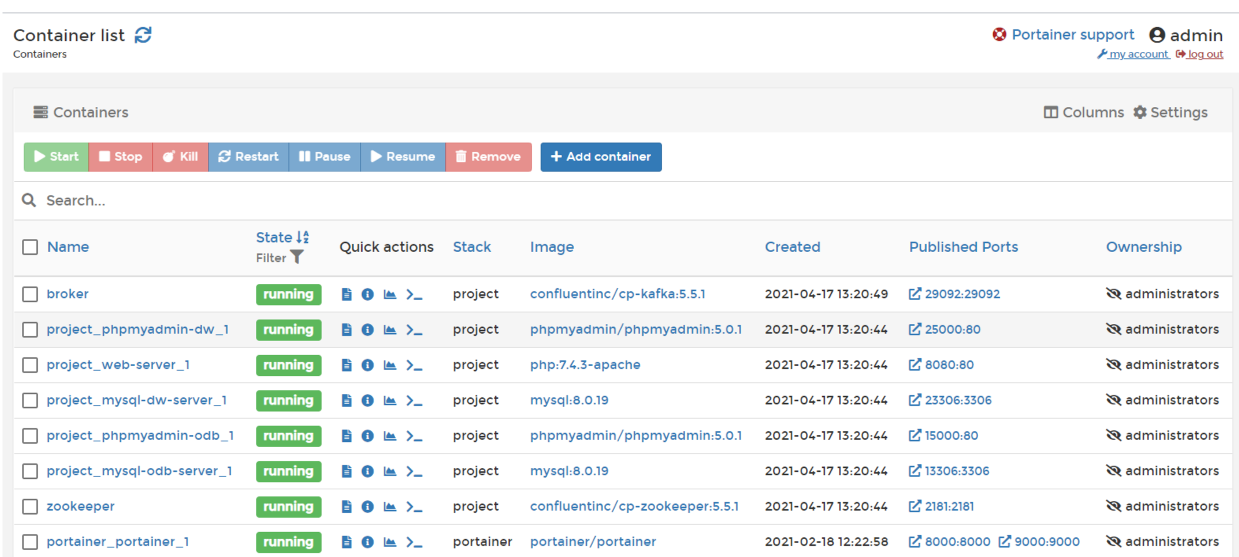
Utilizing the Docker platform to run components of the data pipeline in separate containers, the project is comprised of several key technologies. At the highest level, an Apache web server is used to support two instances of phpMyAdmin, each corresponding to a MySQL database: an instance for the operational 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 project's docker-compose file (contents listed below) can also be found at: https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/docker-compose.yml.

docker-compose.yml

```
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
```

```yaml
  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:
```



Name	State	Quick actions	Stack	Image	Created	Published Ports	Ownership
broker	running		project	confluentinc/cp-kafka:5.5.1	2021-04-17 13:20:49	29092:29092	administrators
project_phpmyadmin-dw_1	running		project	phpmyadmin/phpmyadmin:5.0.1	2021-04-17 13:20:44	25000:80	administrators
project_web-server_1	running		project	php:7.4.3-apache	2021-04-17 13:20:44	8080:80	administrators
project_mysql-dw-server_1	running		project	mysql:8.0.19	2021-04-17 13:20:44	23306:3306	administrators
project_phpmyadmin-odb_1	running		project	phpmyadmin/phpmyadmin:5.0.1	2021-04-17 13:20:44	15000:80	administrators
project_mysql-odb-server_1	running		project	mysql:8.0.19	2021-04-17 13:20:44	13306:3306	administrators
zookeeper	running		project	confluentinc/cp-zookeeper:5.5.1	2021-04-17 13:20:44	2181:2181	administrators
portainer_portainer_1	running		portainer	portainer/portainer	2021-02-18 12:22:58	8000:8000 9000:9000	administrators

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 website 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), 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 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.

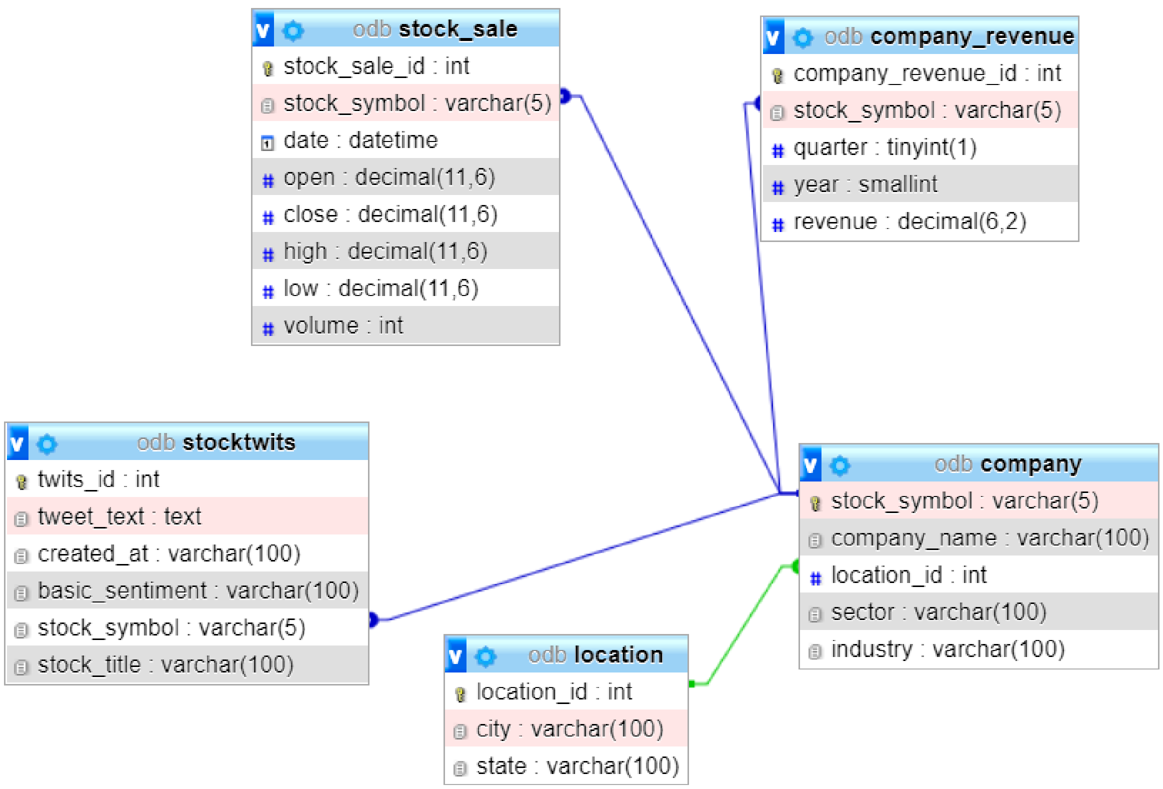
Operational Database

The schema of the project's ODB includes the following five tables:

1. company – contains basic information on S&P 500 companies, including stock symbol (used as the primary key), company name, sector, industry, and location (a foreign key referencing the "location" table)
2. location – contains all unique city-state combinations for headquarters of S&P 500 companies included in the data set
3. company_revenue – records quarterly revenues for S&P 500 companies (in billions of dollars), connecting to corresponding companies using stock symbols as foreign keys
4. stock_sale – the main transaction table of the ODB, comprised of daily stock trade information for each company, including the opening and closing price, highest and

lowest sale prices, and volume of stocks sold for a given day; connects stock trades to the corresponding company by the stock symbol foreign key

5. stocktwits – records stock “twits” about companies, including the content of the twit, its date of creation, and basic sentiment; connects to the company table with the stock symbol foreign key, also including a company’s name, mirroring the data retrieved from the Stocktwits API



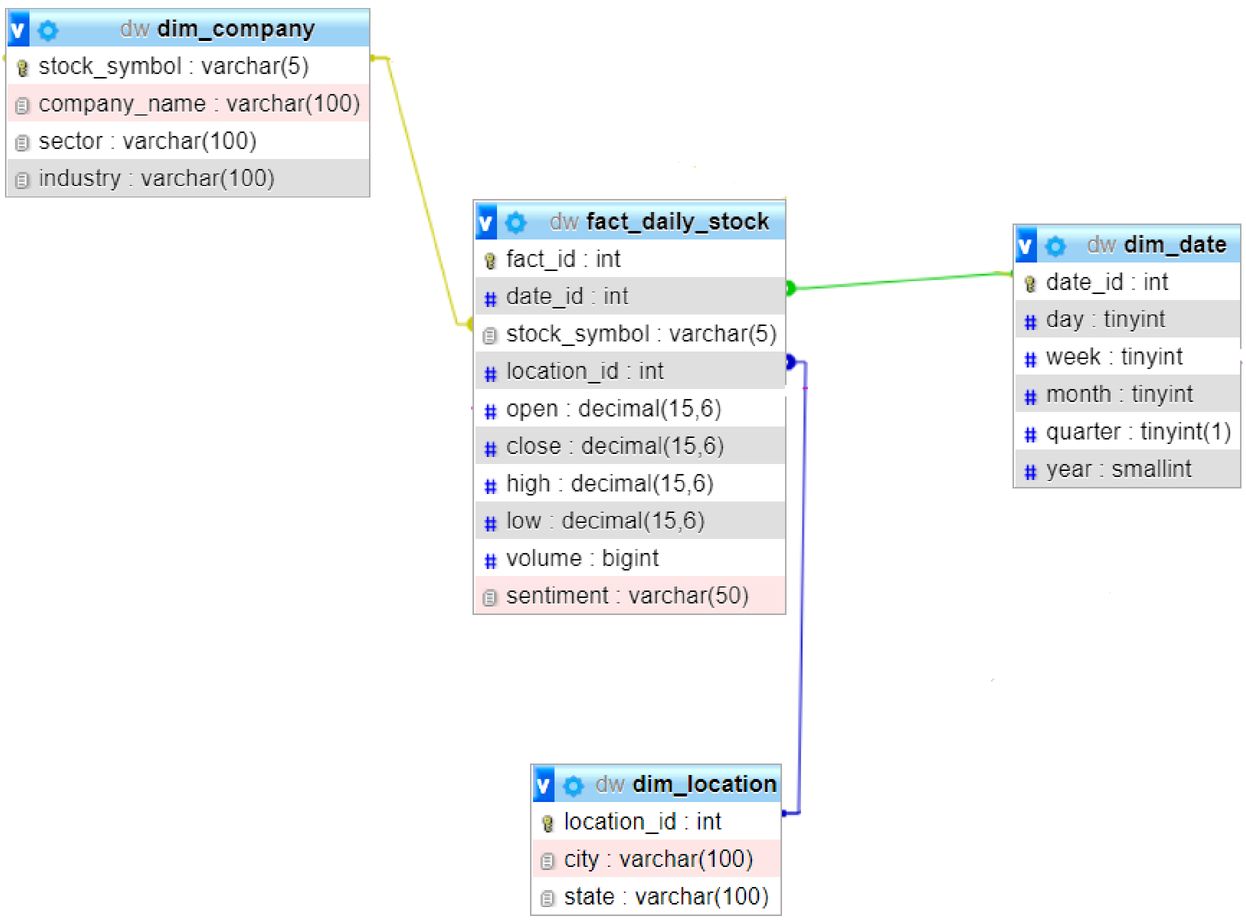
The DDL statements for the ODB (listed below) can also be found at https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/create_odb.sql. The combined SQL statements and programming logic for populating the ODB (sans stock twit data) can be found at https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/load_odb.py.

```
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
```

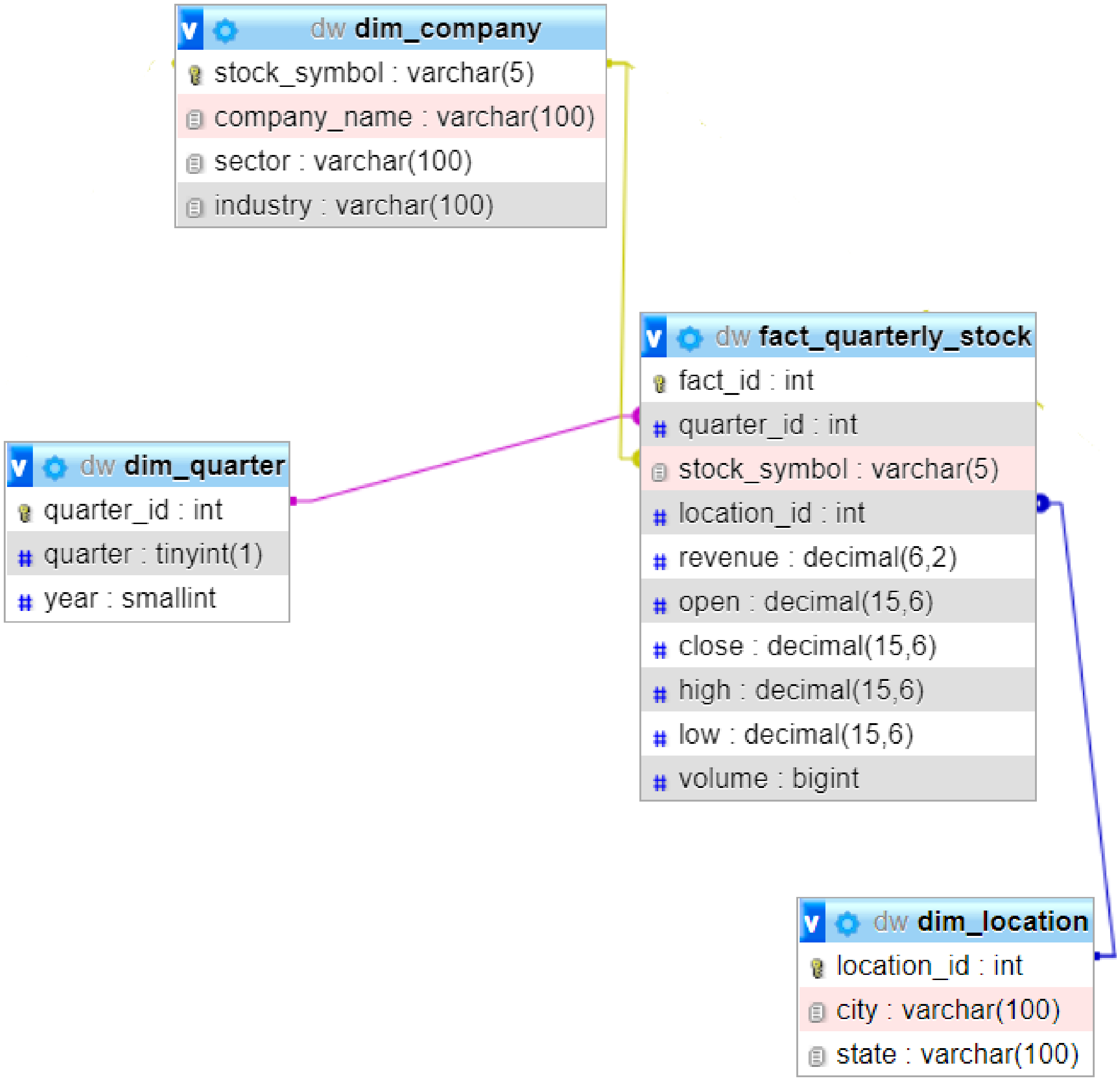
```
`fk_t_stock_symbol`   FOREIGN KEY (`stock_symbol`)   REFERENCES `odb`.`company`
(`stock_symbol`)   ON DELETE CASCADE   ON UPDATE CASCADE)ENGINE = InnoDB;
```

STAR Schema

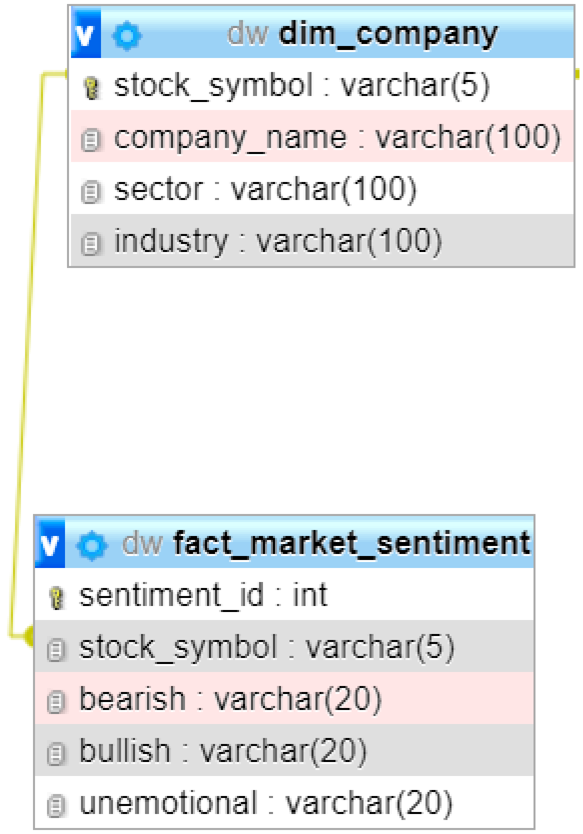
The STAR schema of the DW centers around three FACT tables. The first, the "fact_daily_stock," table includes three dimensions: date, company (stock), and (headquarters) location, describing several measures: the opening and closing prices of a stock, its highest and lowest sale prices, and the volume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

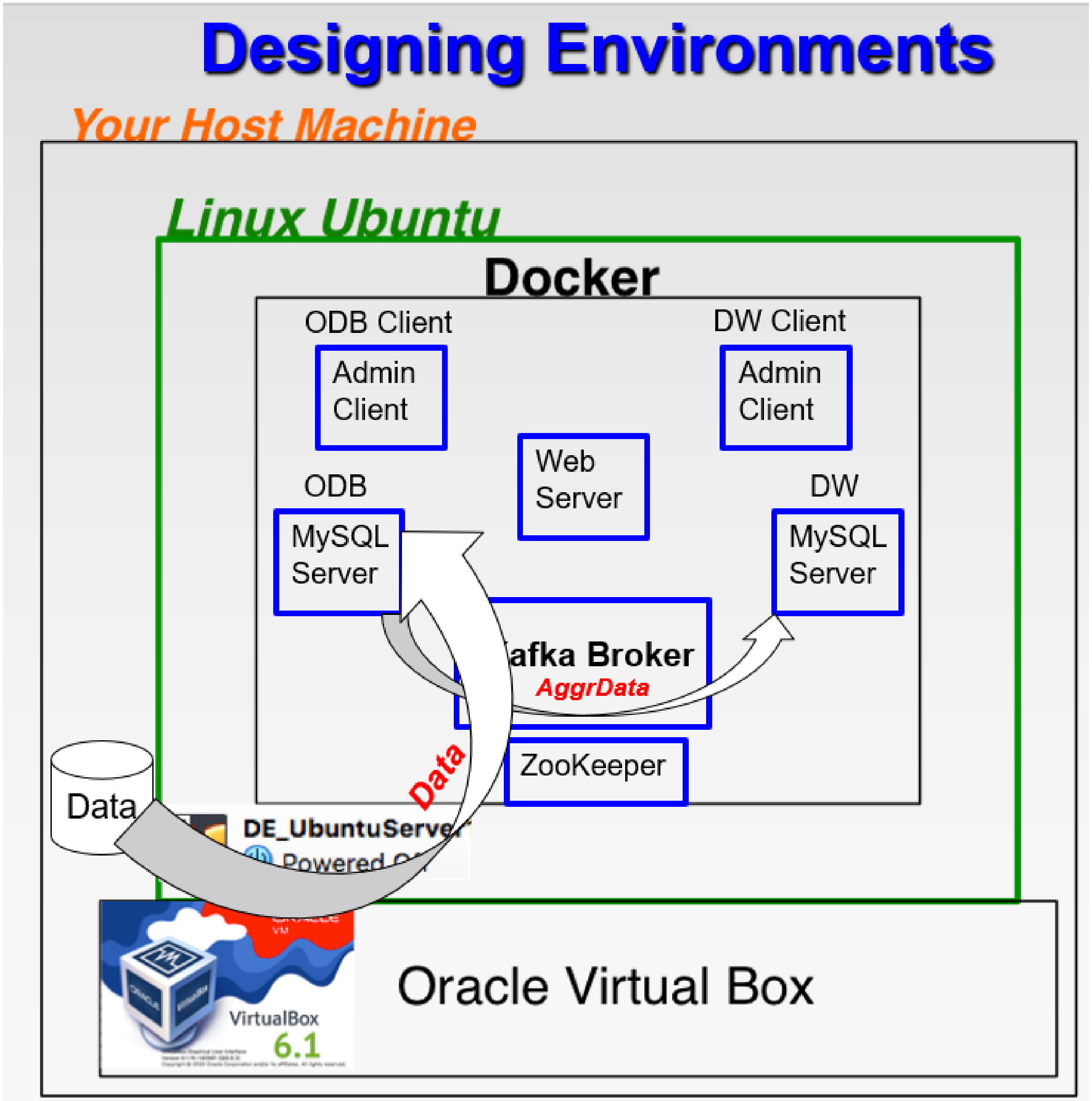


In total, the schema includes four unique dimension tables. The table “dim_quarter” represents the unique fiscal quarters for data contained in the warehouse. The “dim_date” table describes all unique dates of stock trade data stored in the warehouse, split into component days, weeks, months, fiscal 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 the DW (listed below) can also be found at https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/create_dw.sql. The combined SQL statements and programming logic for populating the DW can be found at https://github.com/debdasghosh/Data-Engineering-Behind-Stock-Analysis/blob/main/load_dw.py.

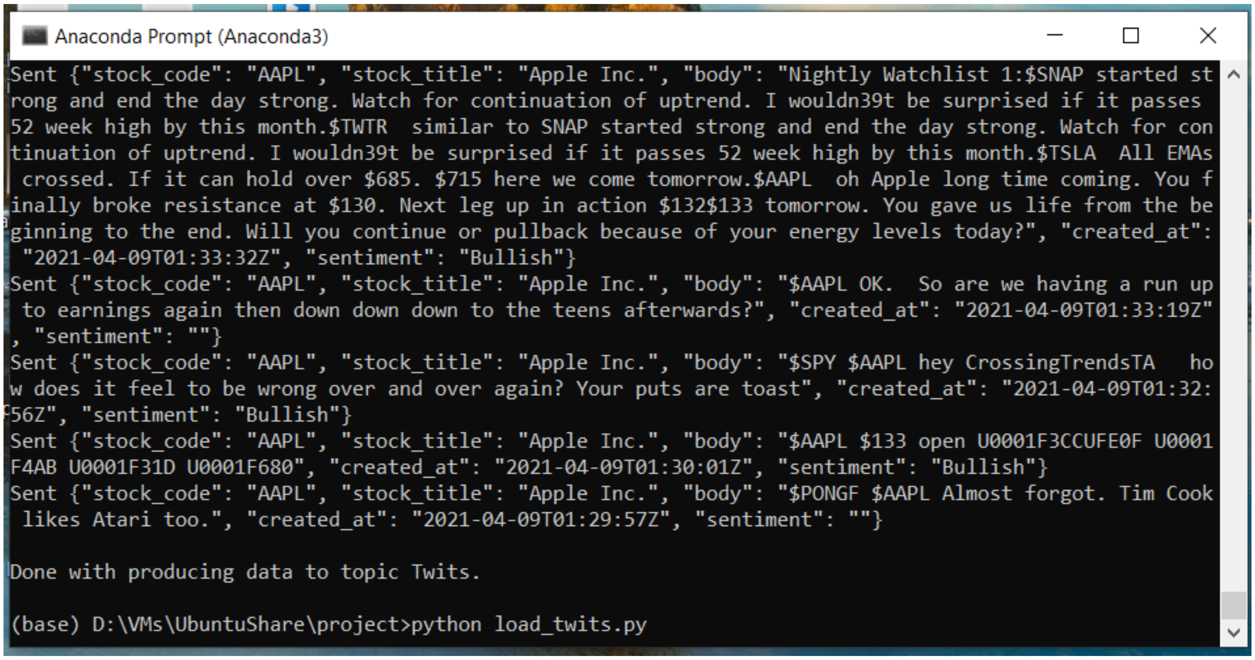
```sql
DROP SCHEMA IF EXISTS `dw` ;

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` ;
```

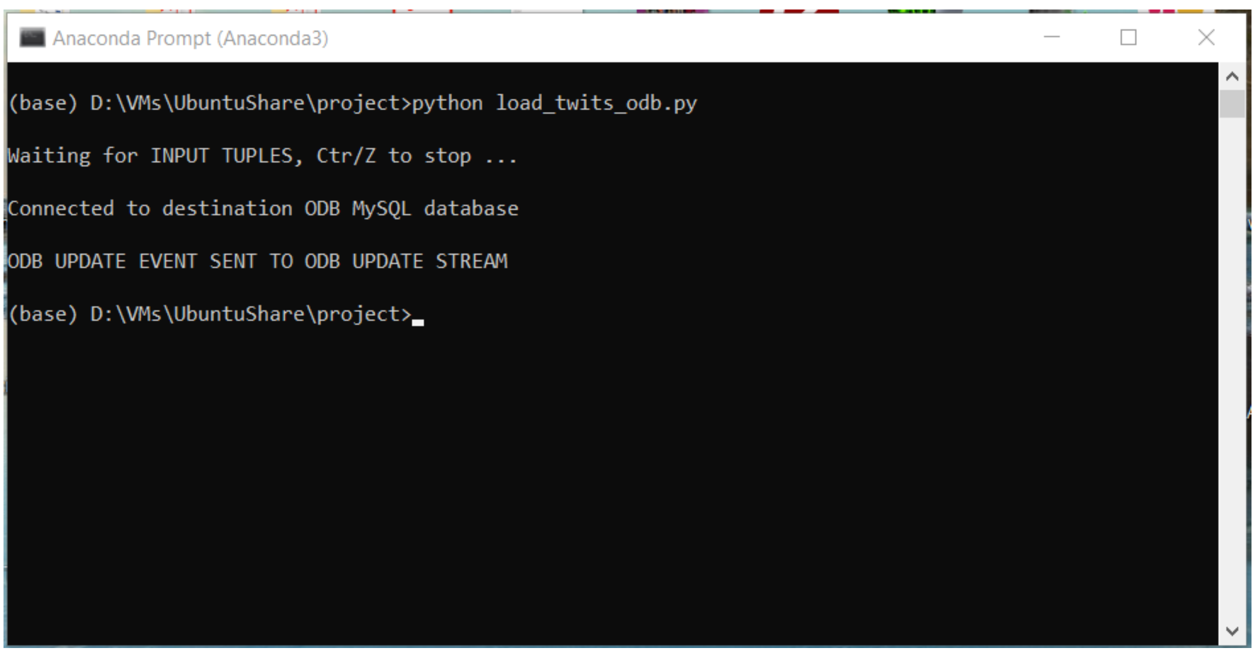
```
CREATE TABLE IF NOT EXISTS `dw`.`fact_quarterly_stock` (  `fact_id` INT NOT NULL
AUTO_INCREMENT,  `quarter_id` INT NOT NULL,  `stock_symbol` VARCHAR(5) NOT NULL,
`location_id` INT NOT NULL,  `revenue` DECIMAL(6,2) NOT NULL,  `open`
DECIMAL(15,6) NOT NULL,  `close` DECIMAL(15,6) NOT NULL,  `high` 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;
```

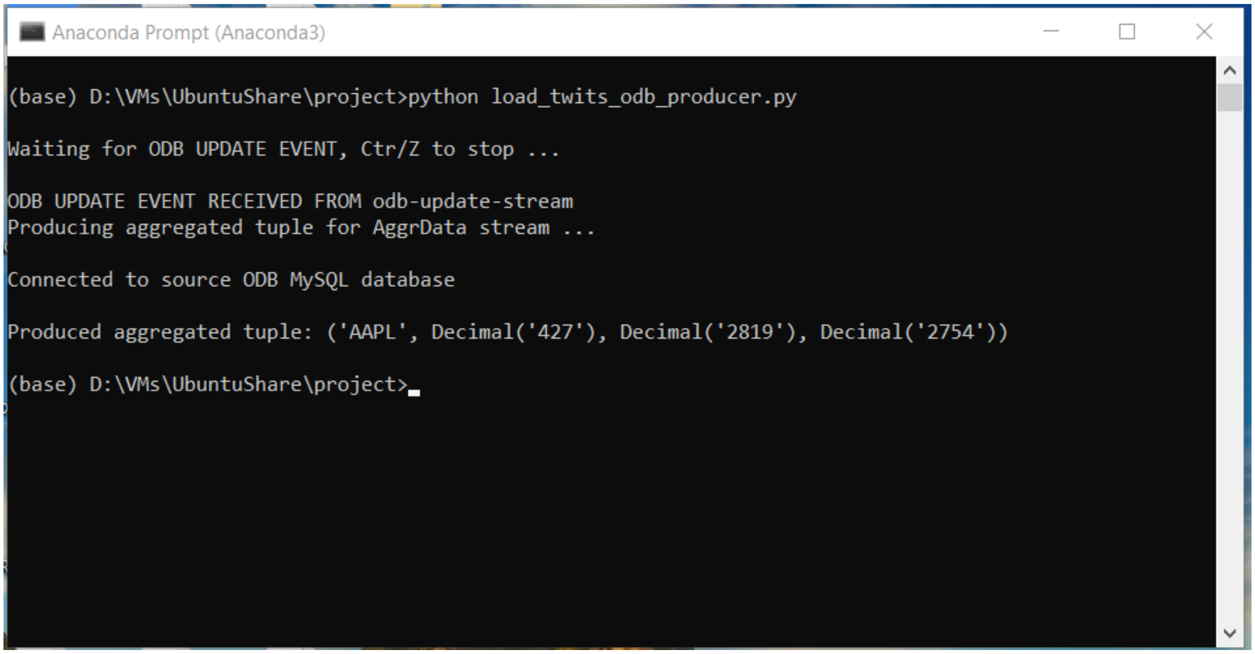


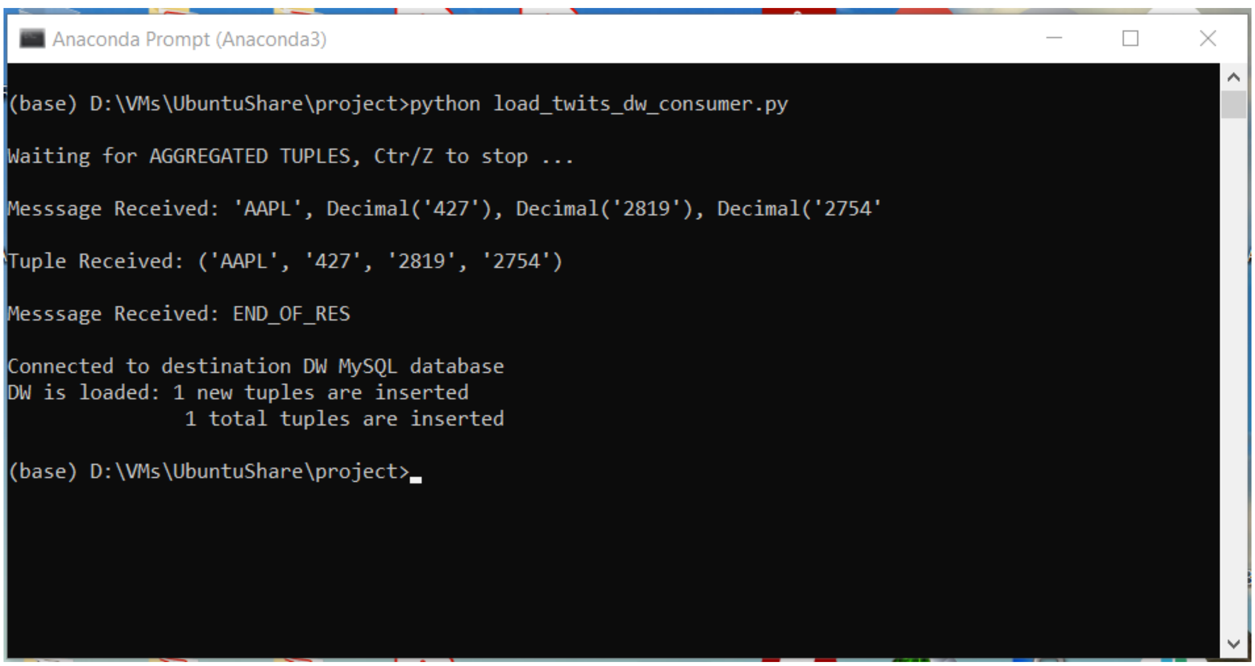
Data Streaming

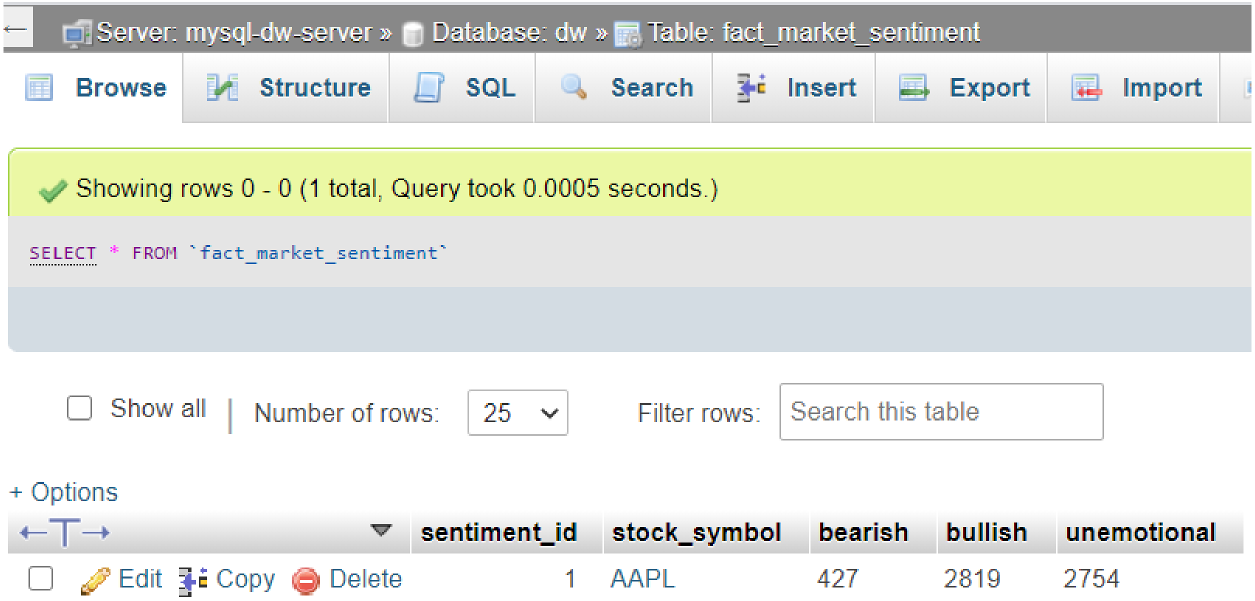
Data streaming using Apache Kafka was utilized for loading data from Stocktwits into the ODB and DW. The corresponding Python scripts include: load_twits.py, load_twits_odb.py, load_twits_odb_producer.py, and load_twits_dw_consumer.py, available on the provided GitHub repository.

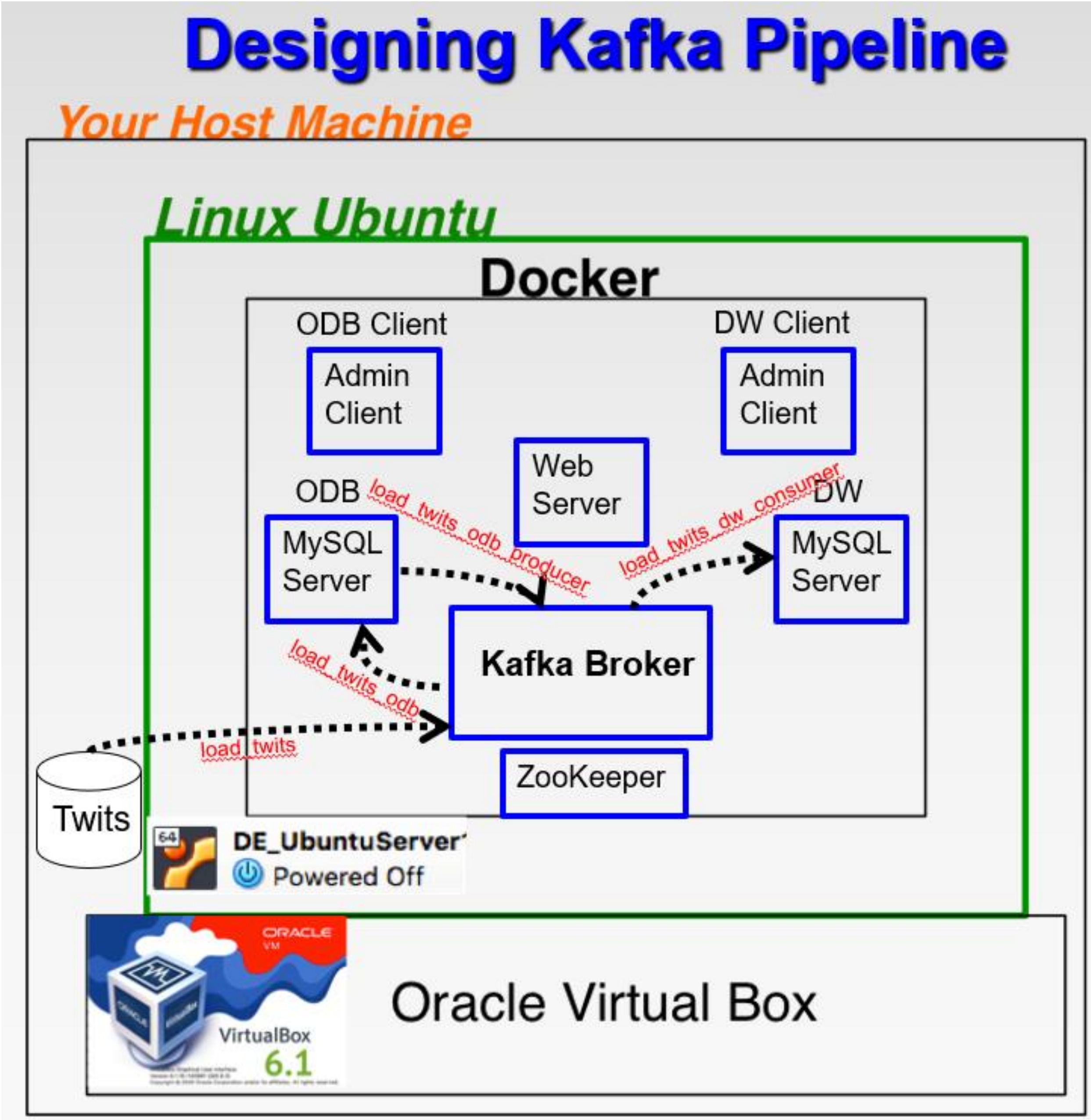

Anaconda Prompt (Anaconda3)
Sent {"stock_code": "AAPL", "stock_title": "Apple Inc.", "body": "Nightly Watchlist 1:$SNAP started st
rong and end the day strong. Watch for continuation of uptrend. I wouldn39t be surprised if it passes
52 week high by this month.$TWTR similar to SNAP started strong and end the day strong. Watch for con
tinuation of uptrend. I wouldn39t be surprised if it passes 52 week high by this month.$TSLA All EMAs
 crossed. If it can hold over $685. $715 here we come tomorrow.$AAPL oh Apple long time coming. You f
inally broke resistance at $130. Next leg up in action $132$133 tomorrow. You gave us life from the be
ginning to the end. Will you continue or pullback because of your energy levels today?", "created_at":
 "2021-04-09T01:33:32Z", "sentiment": "Bullish"}
Sent {"stock_code": "AAPL", "stock_title": "Apple Inc.", "body": "$AAPL OK. So are we having a run up
 to earnings again then down down down to the teens afterwards?", "created_at": "2021-04-09T01:33:19Z"
, "sentiment": ""}
Sent {"stock_code": "AAPL", "stock_title": "Apple Inc.", "body": "$SPY $AAPL hey CrossingTrendsTA   ho
w does it feel to be wrong over and over again? Your puts are toast", "created_at": "2021-04-09T01:32:
56Z", "sentiment": "Bullish"}
Sent {"stock_code": "AAPL", "stock_title": "Apple Inc.", "body": "$AAPL $133 open U0001F3CCUFE0F U0001
F4AB U0001F31D U0001F680", "created_at": "2021-04-09T01:30:01Z", "sentiment": "Bullish"}
Sent {"stock_code": "AAPL", "stock_title": "Apple Inc.", "body": "$PONGF $AAPL Almost forgot. Tim Cook
 likes Atari too.", "created_at": "2021-04-09T01:29:57Z", "sentiment": ""}
Done with producing data to topic Twits.
(base) D:\VMs\UbuntuShare\project>python load_twits.py


Anaconda Prompt (Anaconda3)
(base) D:\VMs\UbuntuShare\project>python load_twits_odb.py
Waiting for INPUT TUPLES, Ctr/Z to stop ...
Connected to destination ODB MySQL database
ODB UPDATE EVENT SENT TO ODB UPDATE STREAM
(base) D:\VMs\UbuntuShare\project>


Anaconda Prompt (Anaconda3)
(base) D:\VMs\UbuntuShare\project>python load_twits_odb_producer.py
Waiting for ODB UPDATE EVENT, Ctr/Z to stop ...
ODB UPDATE EVENT RECEIVED FROM odb-update-stream
Producing aggregated tuple for AggrData stream ...
Connected to source ODB MySQL database
Produced aggregated tuple: ('AAPL', Decimal('427'), Decimal('2819'), Decimal('2754'))
(base) D:\VMs\UbuntuShare\project>_


Anaconda Prompt (Anaconda3)
(base) D:\VMs\UbuntuShare\project>python load_twits_dw_consumer.py
Waiting for AGGREGATED TUPLES, Ctr/Z to stop ...
Messsage Received: 'AAPL', Decimal('427'), Decimal('2819'), Decimal('2754'
Tuple Received: ('AAPL', '427', '2819', '2754')
Messsage Received: END_OF_RES
Connected to destination DW MySQL database
DW is loaded: 1 new tuples are inserted
1 total tuples are inserted
(base) D:\VMs\UbuntuShare\project>_


Server: mysql-dw-server » Database: dw » Table: fact_market_sentiment
Browse
Structure
SQL
Search
Insert
Export
Import
Showing rows 0 - 0 (1 total, Query took 0.0005 seconds.)
SELECT * FROM `fact_market_sentiment`
Show all
Number of rows:
25
Filter rows:
Search this table
+ Options
sentiment_id
stock_symbol
bearish
bullish
unemotional
Edit
Copy
Delete
1
AAPL
427
2819
2754



Summary Tables

In addition to the specified FACT and dimension tables, the DW also includes the following pre-aggregated tables:

1. agg_stock_volume – aggregates the volume of stocks sold for date and (headquarters) location in the DW; supports queries of the type “Which city/state has the highest volume of stock trades in the S&P 500 per day/week/month/quarter/year?”
2. agg_quarterly_revenue – aggregates company quarterly revenues by company; supports queries of the type “What is the combined quarterly/yearly revenue for companies in the same sector/industry?”

The following DDL queries create the aforementioned summary tables:

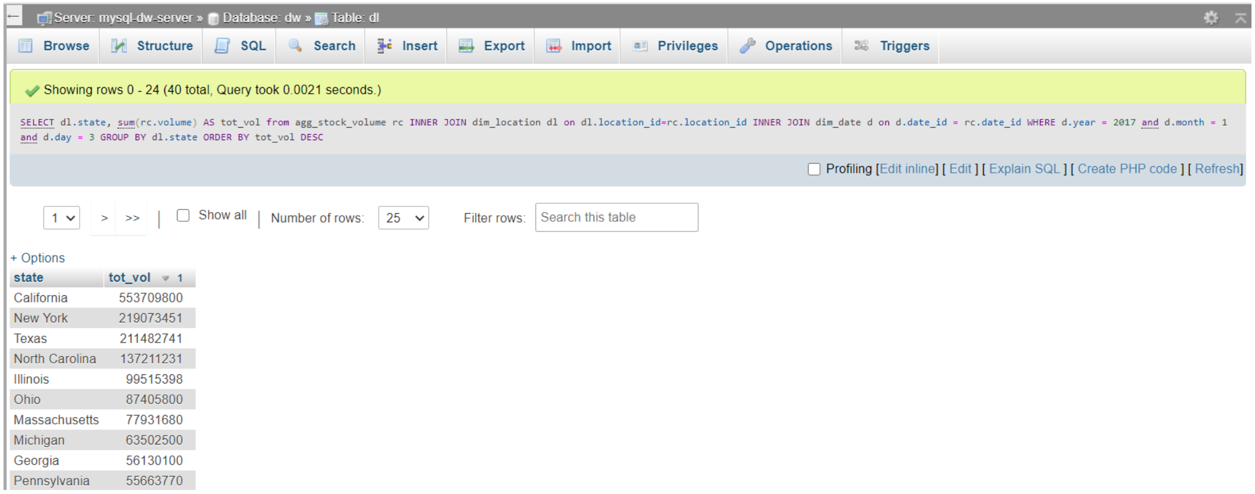
```
DROP TABLE IF EXISTS `dw`.`agg_stock_volume` ;

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_quarterly_revenue` ;

CREATE TABLE `dw`.`agg_quarterly_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;
```

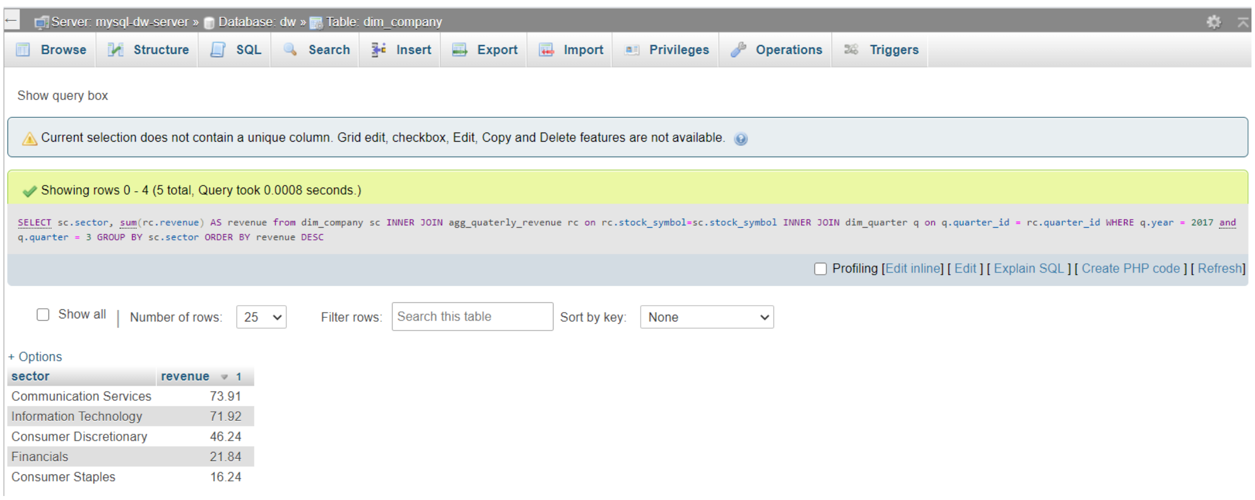
The following queries populate the previous summary tables with data:

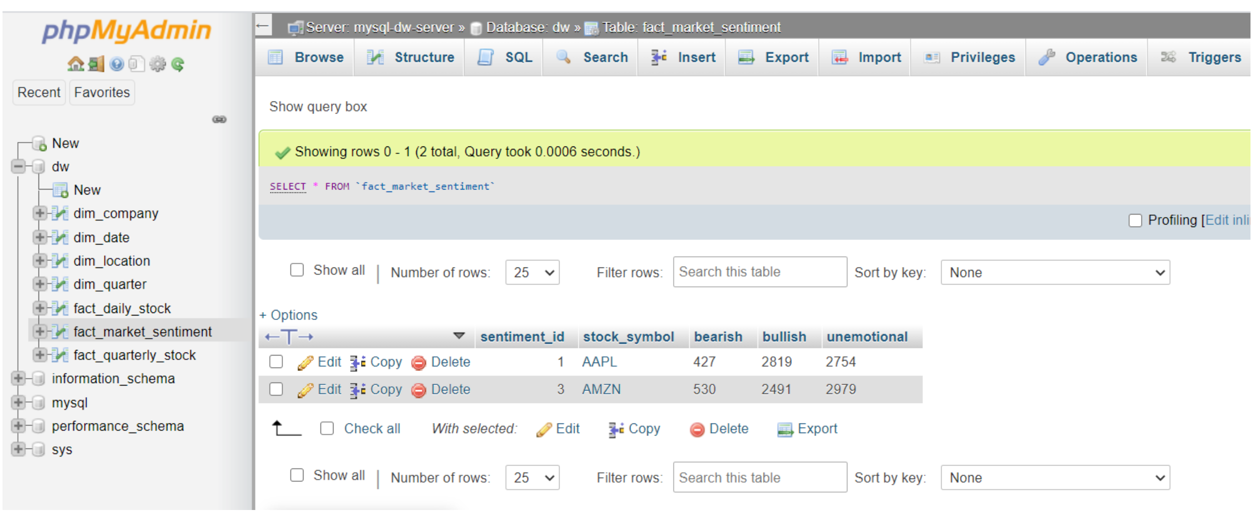
```
INSERT INTO `agg_stock_volume` SELECT NULL, `date_id`, `location_id`,
SUM(`volume`) FROM `fact_daily_stock` GROUP BY `location_id`,`date_id`;

INSERT INTO `agg_quaterly_revenue` SELECT NULL, `quarter_id`, `stock_symbol`,
SUM(`revenue`) FROM `fact_quarterly_stock` GROUP BY `stock_symbol`,`quarter_id`;
```

Supported Queries

```
-- Which city/state has the highest volume of stock trades in the S&P 500 per
day/week/month/quarter/year?

-- Example: Which state has the highest volume of stock trades on 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;


-- What is the combined quarterly/yearly revenue for companies in the same
sector/industry?

-- 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 an S&P 500 company


SELECT * FROM `fact_market_sentiment`;
```


Server: mysql-dw-server » Database: dw » Table: dl
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Showing rows 0 - 24 (40 total, Query took 0.0021 seconds.)
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
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
+ Options
state tot_vol 1
California 553709800
New York 219073451
Texas 211482741
North Carolina 137211231
Illinois 99515398
Ohio 87405800
Massachusetts 77931680
Michigan 63502500
Georgia 56130100
Pennsylvania 55663770


Server: mysql-dw-server » Database: dw » Table: dim_company
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Show query box
Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.
Showing rows 0 - 4 (5 total, Query took 0.0008 seconds.)
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
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
+ Options
sector revenue 1
Communication Services 73.91
Information Technology 71.92
Consumer Discretionary 46.24
Financials 21.84
Consumer Staples 16.24


phpMyAdmin
Recent Favorites
New
dw
New
dim_company
dim_date
dim_location
dim_quarter
fact_daily_stock
fact_market_sentiment
fact_quarterly_stock
information_schema
mysql
performance_schema
sys
Server: mysql-dw-server » Database: dw » Table: fact_market_sentiment
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Show query box
Showing rows 0 - 1 (2 total, Query took 0.0006 seconds.)
SELECT * FROM `fact_market_sentiment`
Profiling [Edit inli
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
+ Options
sentiment_id stock_symbol bearish bullish unemotional
Edit Copy Delete 1 AAPL 427 2819 2754
Edit Copy Delete 3 AMZN 530 2491 2979
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

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.