CM3020 AI midterms
Database Report
Jaslyn Chan
Student ID: 240662387

1.0 Introduction

1.1 Purpose

This project focuses on analysing stock price and trading volume data for a group of large technology companies. By structuring and querying this data in a relational database, I used SQL queries to extract meaningful insights such as returns, volatility, and trading activity.

1.2 Dataset description

The dataset has daily stock price data (open, high, low, close prices, trading volume). Each record is one trading day for one company. Data was collected for seven publicly traded companies. Apple. Microsoft, Amazon, Google, Meta, Nvidia, and Tesla over approximately the last five years.

I cropped the data to remove older records, leaving only the last five years to ensure a consistent time window for all companies.

1.3 Queries

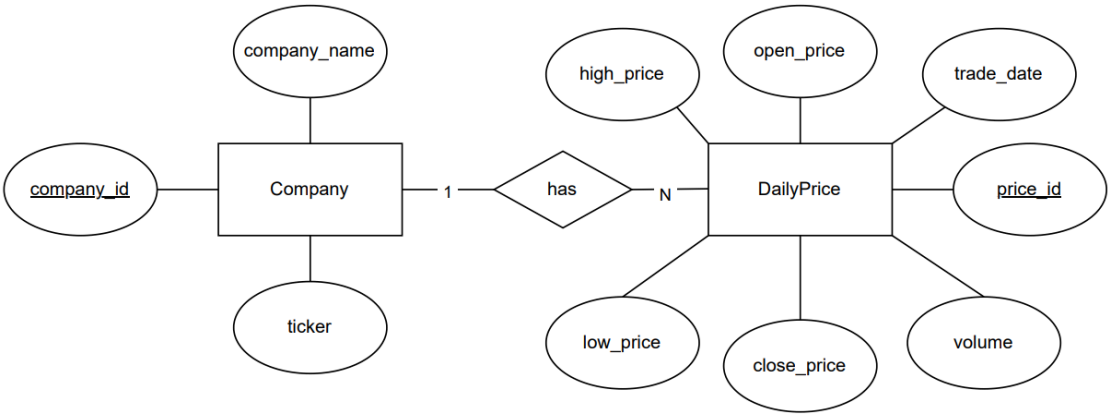
This project answers the questions:

- How many records exist for each company, and what date range is covered?
- Which company achieved the highest overall return over the dataset period?
- Which company exhibits the highest day-to-day price volatility?
- Which company has the highest average trading volume?
- How do average closing prices compare across companies?

2.0 Data Modelling & Design

2.1 ER Diagram

The diagram consists of two main entities, Company and DailyPrice. Each company can have many daily price records, forming a one-to-many relationship.



2.2 Schema

- Company
  - company_id (primary key)
  - ticker
  - company_name
- DailyPrice
  - price_id (primary key)
  - company_id (foreign key -> company)
  - trade_date
  - open_price
  - high_price
  - low_price
  - close_price
  - volume

2.3 Design Decisions

A unique constraint was applied on the combination of (company_id, trade_date) to ensure that each company has at most one record per trading day.

I used surrogate primary keys for both tables. I enforced referential integrity using foreign key constraints. I separated the company metadata from time-series price data to avoid redundancy. I index on company_id and trade date to have efficient querying.

3.0 Database Implementation & Queries

3.1 Database implementation

The database was implemented using MySQL. All tables were created according to the schema. A dedicated database user with read-only (SELECT) privileges was created for use by the web application.

3.2 Data loading and Queries

Data was imported from CSV files into a staging table then inserted into the main DailyPrice table. Duplicate records were prevented by the unique constraint. Referential integrity was validated and each company was confirmed to have the same number of records after cropping.

This query shows the summary of data for each company. (record count & date range)

```sql
-- Dataset summary: record count and date range per ticker
SELECT
  c.ticker,
  COUNT(*) AS row_count,
  MIN(d.trade_date) AS min_date,
  MAX(d.trade_date) AS max_date
FROM DailyPrice d
JOIN Company c
  ON c.company_id = d.company_id
GROUP BY c.ticker
ORDER BY c.ticker;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ticker	row_count	min_date	max_date
AAPL	1748	2019-01-02	2025-12-12
AMZN	1748	2019-01-02	2025-12-12
GOOGL	1748	2019-01-02	2025-12-12
META	1748	2019-01-02	2025-12-12
MSFT	1748	2019-01-02	2025-12-12
NVDA	1748	2019-01-02	2025-12-12
TSLA	1748	2019-01-02	2025-12-12

This query fetches the overall return values

```sql
-- Overall percentage return per company
WITH bounds AS (
  SELECT
    company_id,
    MIN(trade_date) AS first_date,
    MAX(trade_date) AS last_date
  FROM DailyPrice
  GROUP BY company_id
)
SELECT
  c.ticker,
  ROUND((MAX(d.close_price) / MIN(d.close_price) - 1) * 100, 2) AS pct_return
FROM DailyPrice d
JOIN bounds b
  ON d.company_id = b.company_id
JOIN Company c
  ON c.company_id = d.company_id
GROUP BY c.ticker
ORDER BY pct_return DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ticker	pct_return
NVDA	6423.27
TSLA	3921.86
META	788.54
AAPL	743.84
GOOGL	533.10
MSFT	490.46
AMZN	238.60

This query fetches the volatility values

```sql
-- Daily return volatility per company
WITH returns AS (
  SELECT
    company_id,
    close_price / LAG(close_price)
      OVER (PARTITION BY company_id ORDER BY trade_date) - 1 AS daily_return
  FROM DailyPrice
)
SELECT
  c.ticker,
  ROUND(STDDEV_SAMP(daily_return) * 100, 4) AS volatility
FROM returns r
JOIN Company c
  ON c.company_id = r.company_id
WHERE daily_return IS NOT NULL
GROUP BY c.ticker
ORDER BY volatility DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ticker	volatility
TSLA	4.0607
NVDA	3.2551
META	2.6467
AMZN	2.1578
GOOGL	1.9801
AAPL	1.9631
MSFT	1.7924

This is the query for the explore page, where users can choose parameters for the company and number of days

```sql
-- Explore: recent price history for a selected ticker
-- Parameters:
--   :ticker → selected stock ticker
--   :days   → number of recent trading days

SELECT
  d.trade_date,
  d.open_price,
  d.high_price,
  d.low_price,
  d.close_price,
  d.volume
FROM DailyPrice d
JOIN Company c
  ON c.company_id = d.company_id
WHERE c.ticker = 'AAPL'
ORDER BY d.trade_date DESC
LIMIT 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

trade_date	open_price	high_price	low_price	close_price	volume
2025-12-12	277.900000	279.220000	276.820000	278.280000	39532887
2025-12-11	279.095000	279.590000	273.810000	278.030000	33247986
2025-12-10	277.750000	279.750000	276.440000	278.780000	33038318
2025-12-09	278.160000	280.030000	276.920000	277.180000	32193256
2025-12-08	278.130000	279.669300	276.150000	277.890000	38211832
2025-12-05	280.540000	281.140000	278.050000	278.780000	47265845
2025-12-04	284.095000	284.730000	278.590000	280.700000	43989056
2025-12-03	286.200000	288.620000	283.300000	284.150000	43538687
2025-12-02	283.000000	287.400000	282.630100	286.190000	53669532
2025-12-01	278.010000	283.420000	276.140000	283.100000	46587722

These queries are directly used by the Node.js application to populate each analytical view

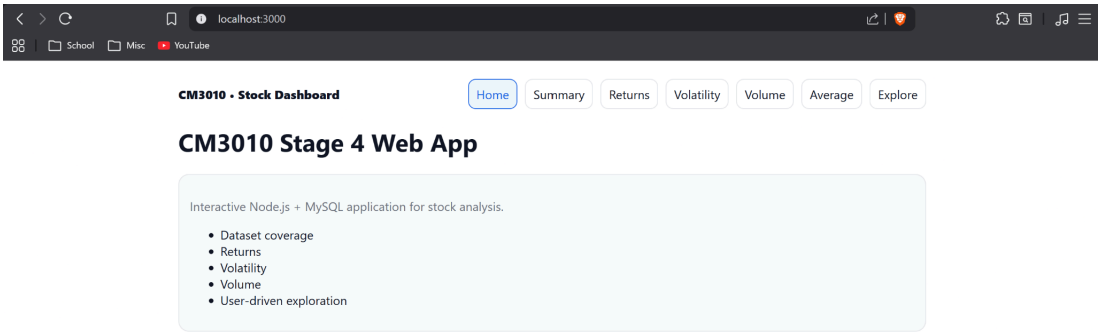
4.0 Web Application

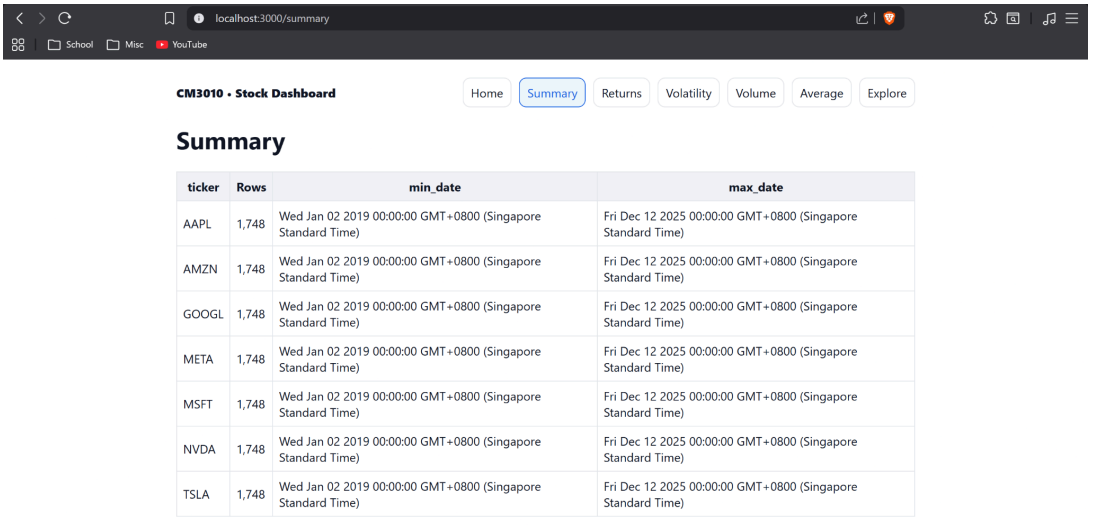
A simple web application was developed using Node.js and Express to provide interactive access to the stock database.

4.1 Architecture

The application connected to the MySQL database using the mysql2 library. Key features include use of environment variables for configuration, parameterised queries for user-selected inputs, and a dedicated read-only database user to enforce appropriate privileges.

Functions include:
Summary page, to display record counts and date ranges per company
Returns page, to rank companies by overall percentage return
Volatility page, to show daily return volatility per company
Average page, to compare average closing prices across companies.





CM3010 • Stock Dashboard

Home | Summary | Returns | Volatility | Volume | Average | Explore

Returns

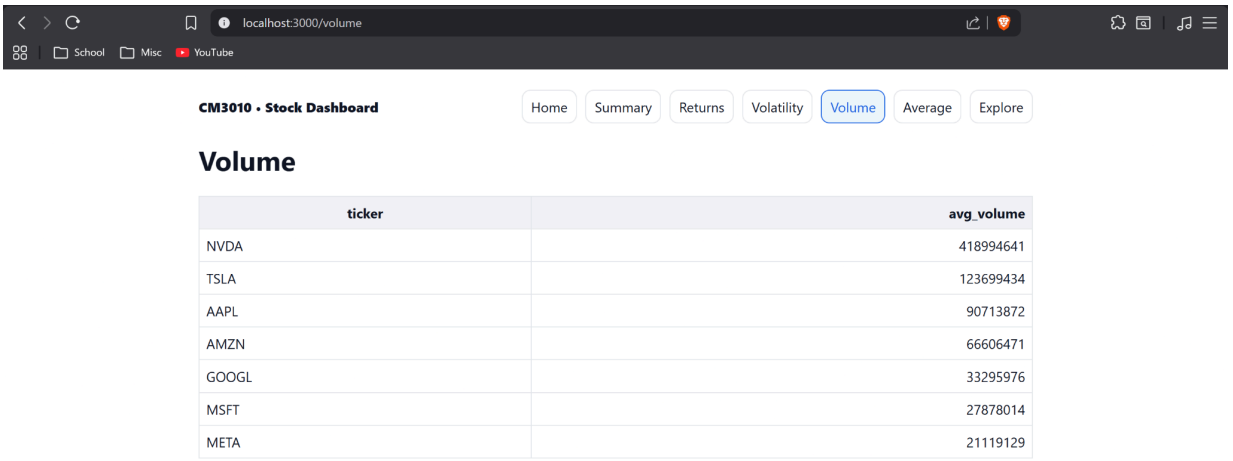
ticker	pct_return
NVDA	6423.27
TSLA	3921.86
META	788.54
AAPL	743.84
GOOGL	533.10
MSFT	490.46
AMZN	238.60

CM3010 • Stock Dashboard

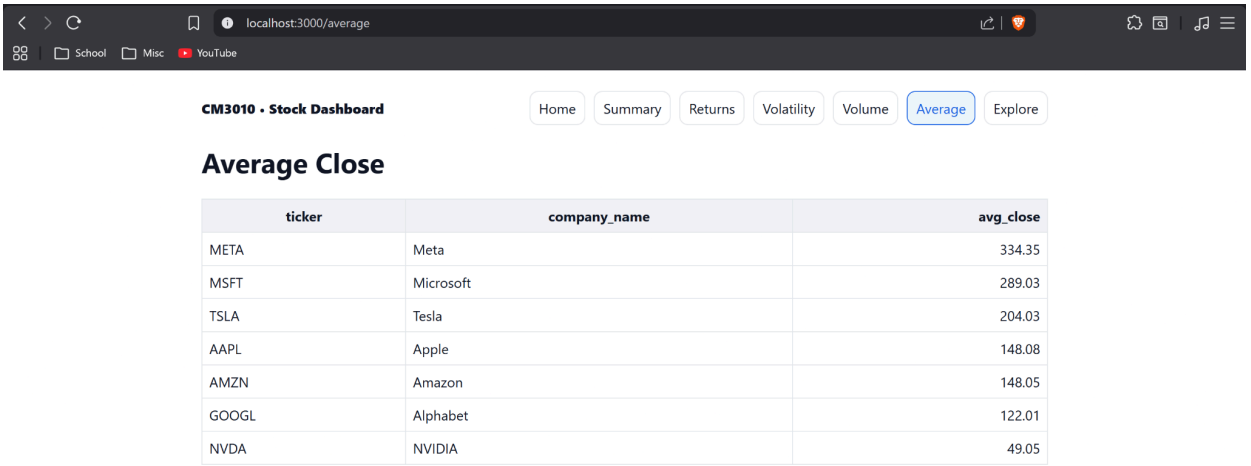
Home | Summary | Returns | Volatility | Volume | Average | Explore

Volatility

ticker	volatility
TSLA	4.0607
NVDA	3.2551
META	2.6467
AMZN	2.1578
GOOGL	1.9801
AAPL	1.9631
MSFT	1.7924



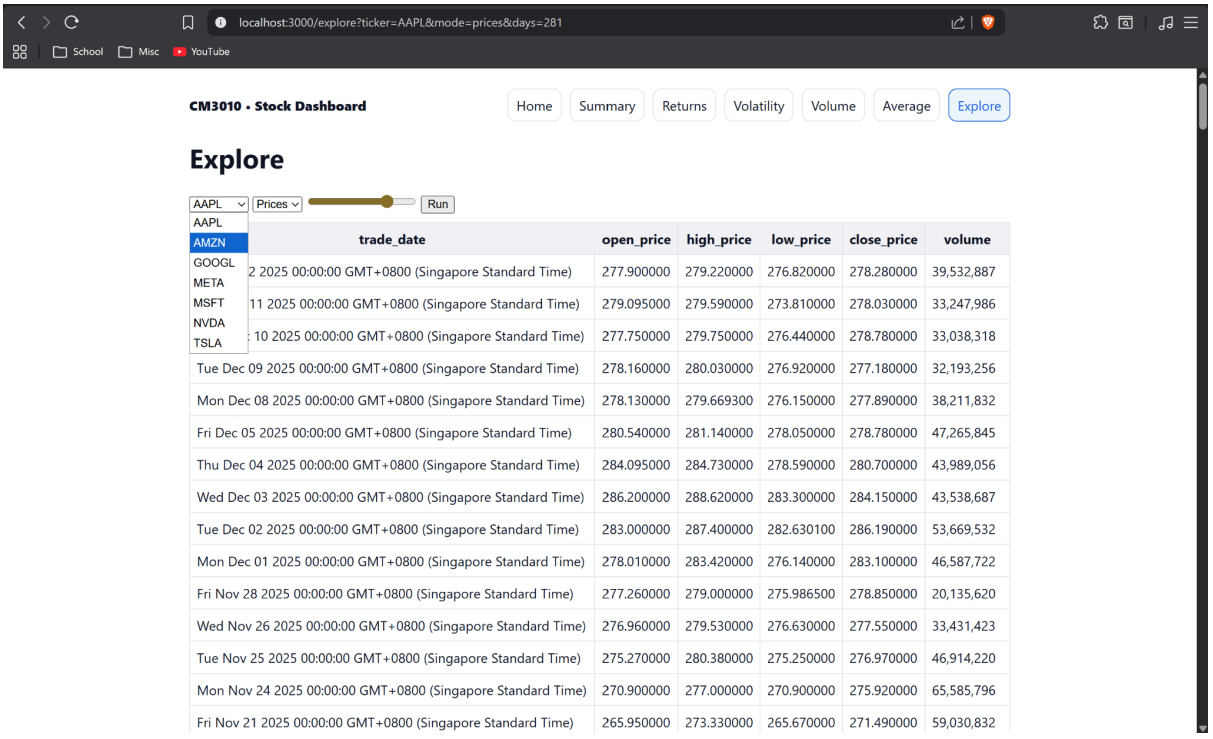
ticker	avg_volume
NVDA	418994641
TSLA	123699434
AAPL	90713872
AMZN	66606471
GOOGL	33295976
MSFT	27878014
META	21119129



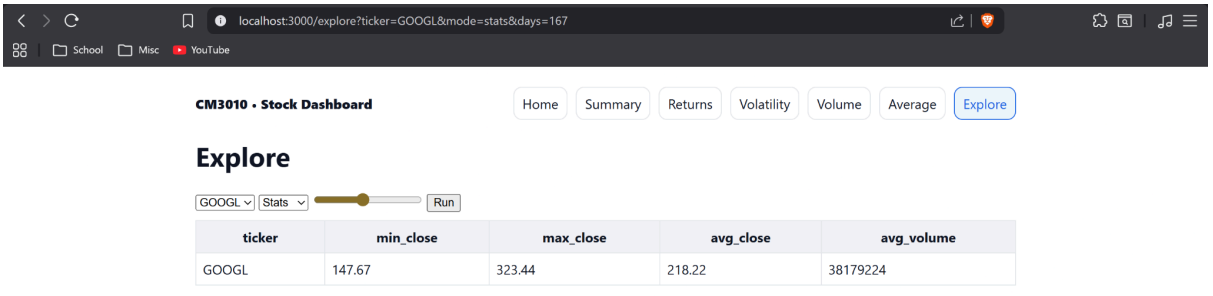
ticker	company_name	avg_close
META	Meta	334.35
MSFT	Microsoft	289.03
TSLA	Tesla	204.03
AAPL	Apple	148.08
AMZN	Amazon	148.05
GOOGL	Alphabet	122.01
NVDA	NVIDIA	49.05

4.2 User-Driven Querying

An Explore page was implemented to allow user-controlled querying of the dataset. Users can select a stock ticker, choose the number of trading days using a slider, and switch between price history and summary statistics. User inputs are validated before being passed into SQL queries, ensuring safe and meaningful interactions.



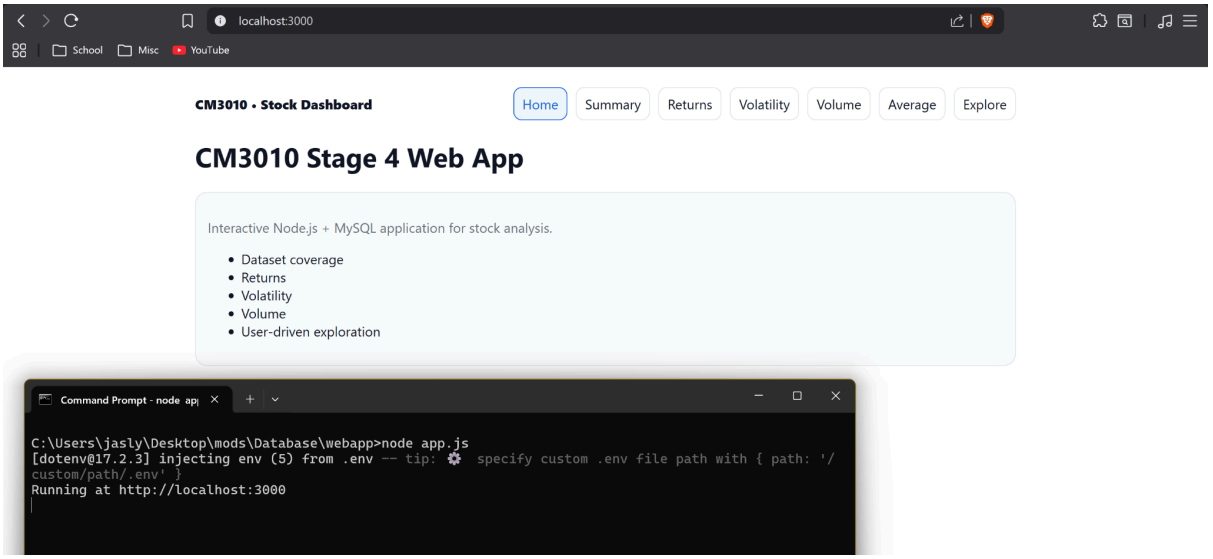
trade_date	open_price	high_price	low_price	close_price	volume
2 2025 00:00:00 GMT+0800 (Singapore Standard Time)	277.900000	279.220000	276.820000	278.280000	39,532,887
11 2025 00:00:00 GMT+0800 (Singapore Standard Time)	279.095000	279.590000	273.810000	278.030000	33,247,986
10 2025 00:00:00 GMT+0800 (Singapore Standard Time)	277.750000	279.750000	276.440000	278.780000	33,038,318
Tue Dec 09 2025 00:00:00 GMT+0800 (Singapore Standard Time)	278.160000	280.030000	276.920000	277.180000	32,193,256
Mon Dec 08 2025 00:00:00 GMT+0800 (Singapore Standard Time)	278.130000	279.669300	276.150000	277.890000	38,211,832
Fri Dec 05 2025 00:00:00 GMT+0800 (Singapore Standard Time)	280.540000	281.140000	278.050000	278.780000	47,265,845
Thu Dec 04 2025 00:00:00 GMT+0800 (Singapore Standard Time)	284.095000	284.730000	278.590000	280.700000	43,989,056
Wed Dec 03 2025 00:00:00 GMT+0800 (Singapore Standard Time)	286.200000	288.620000	283.300000	284.150000	43,538,687
Tue Dec 02 2025 00:00:00 GMT+0800 (Singapore Standard Time)	283.000000	287.400000	282.630100	286.190000	53,669,532
Mon Dec 01 2025 00:00:00 GMT+0800 (Singapore Standard Time)	278.010000	283.420000	276.140000	283.100000	46,587,722
Fri Nov 28 2025 00:00:00 GMT+0800 (Singapore Standard Time)	277.260000	279.000000	275.986500	278.850000	20,135,620
Wed Nov 26 2025 00:00:00 GMT+0800 (Singapore Standard Time)	276.960000	279.530000	276.630000	277.550000	33,431,423
Tue Nov 25 2025 00:00:00 GMT+0800 (Singapore Standard Time)	275.270000	280.380000	275.250000	276.970000	46,914,220
Mon Nov 24 2025 00:00:00 GMT+0800 (Singapore Standard Time)	270.900000	277.000000	270.900000	275.920000	65,585,796
Fri Nov 21 2025 00:00:00 GMT+0800 (Singapore Standard Time)	265.950000	273.330000	265.670000	271.490000	59,030,832



ticker	min_close	max_close	avg_close	avg_volume
GOOGL	147.67	323.44	218.22	38179224

4.3 Execution Evidence

The application was successfully executed using Node.js, confirming that the web application runs and interacts correctly with the database.



5.0 Conclusion

This project shows the entire process from designing a database and building a data model to interactively exploring data. Using relational database design, SQL querying and Node.js, insights into stock price behaviour was successfully extracted and shown in a meaningful manner