Name: Gensik A. Rubio Benitez
Student ID: 1002154893
Documentation File Name: gar4893_phase1.pdf
Title of the Project: Stock Database Management System

Abstract

The Stock Database Management System is designed to efficiently store and manage stock-related information, including stock prices, company details, and historical data. The system aims to provide users with a reliable and user-friendly interface for accessing and analyzing stock data.

Database I: Term Project STOCKS

Phase I [15%]: Project Initiation

Step 1: Consider what to do

My mini world in this project is a STOCK database. In the stock database, we will manage STOCK, PORTFOLIO, and TRANSACTION as entity types and their relationships, such as OWNS and TRADES.

Step 2: Problem definition, user requirements

The database is designed to track stock investments for individual investors. It will store information about stocks, portfolios, and transactions.

- Each STOCK has a symbol, name, price, volume, market cap, sector, and volatility.
- Each PORTFOLIO has a name, description, start date, and is owned by an INVESTOR.
- Each TRANSACTION has a date, type (buy/sell), quantity, price, and is associated with a PORTFOLIO.

Step 3: Formulate 10 realistic English queries

1. Retrieve the current price of a specific stock.
2. List all transactions (buy/sell) for a particular portfolio.
3. Calculate the total value of a portfolio based on current stock prices.
4. Find the highest and lowest performing stocks in a portfolio.
5. Identify stocks with the highest trading volume.
6. List all portfolios that contain a specific stock.
7. Calculate the total profit/loss for each transaction in a portfolio.
8. Identify stocks that have gained/lost the most value in a given time period.
9. List all portfolios with a total value above a specified amount.
10. Identify stocks with the highest dividend yield.

Step 4: Entity-Relationship with assumptions

STOCK entity type

Attributes:

- Symbol (Primary Key): String
- Name: String
- Price: Decimal
- Market Cap: Decimal
- Sector: String
- Historical Volatility: Decimal
- Implied Volatility: Decimal
- Beta: Decimal
- Dividend Yield: Decimal

PORTFOLIO entity type

Attributes:

- ID (Primary Key): Integer
- Name: String
- Description: String
- Start Date: Date
- Investor ID (Foreign Key): Integer (references INVESTOR entity)

TRANSACTION entity type

Attributes:

- ID (Primary Key): Integer
- Date: Date
- Type: String (Buy/Sell)
- Quantity: Integer
- Price: Decimal
- Portfolio ID (Foreign Key): Integer (references PORTFOLIO entity)
- Stock Symbol (Foreign Key): String (references STOCK entity)

INVESTOR entity type

Attributes:

- ID (Primary Key): Integer
- Name: String
- Email: String
- Phone Number: String

SECTOR entity type

Attributes:

- ID (Primary Key): Integer
- Name: String

Relationship Types

OWNS relationship type

- Portfolio OWNS Stock
- Cardinality is 1:N (one portfolio can own multiple stocks, but each stock is owned by one portfolio).

TRADES relationship type

- Portfolio TRADES Stock
- Cardinality is 1:N (one portfolio can trade multiple stocks, but each stock is traded by one portfolio).

Belongs_TO relationship type

- Stock Belongs_To Sector
- Cardinality is N:1 (multiple stocks belong to one sector, but each stock belongs to one sector).

Assumptions

- It is assumed that the data provided for each stock, portfolio, transaction, investor, and sector is accurate and up-to-date.
- Each portfolio is assumed to be owned by a single investor.
- Each stock is assumed to have a unique symbol as its primary key.
- Each transaction is associated with a single portfolio and a single stock.
- Each stock is associated with a single sector.
- Historical and implied volatility, as well as beta, are assumed to be accurately calculated based on historical stock price data.
- The total value of a portfolio is assumed to be calculated based on the current prices of stocks in the portfolio.
- Trades are assumed to be accurately recorded, including date, type (buy/sell), quantity, and price.

Phase II
Project Title: [Stocks] Database

Brief Description:

My mini world in this project is a STOCK database. In the stock database, we will manage STOCK, PORTFOLIO, and TRANSACTION as entity types and their relationships, such as OWNS and TRADES.

Phase I [Modified Steps]

Step 2: Problem definition, user requirements [modified]

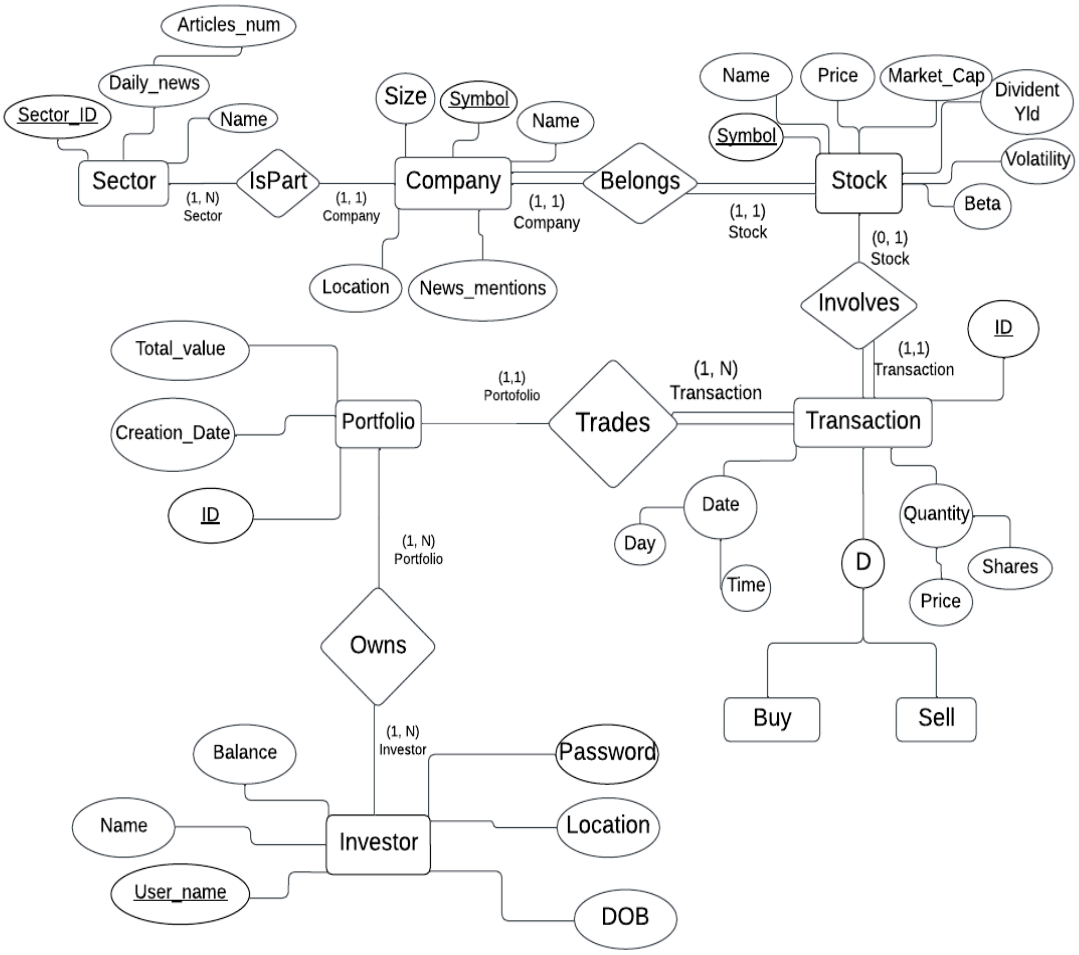
The database is designed to assist a company specializing in managing portfolios for individual investors in efficiently tracking stock metrics. It will store detailed information about stocks, portfolios, and transactions to facilitate informed trading decisions.

- Each STOCK entry has a symbol, name, price, volume, market cap, sector, and volatility.
- Each PORTFOLIO has a name, description, start date, and is owned by an investor.
- Each TRANSACTION will include a date, type (buy/sell), quantity, price, and be associated with a portfolio.

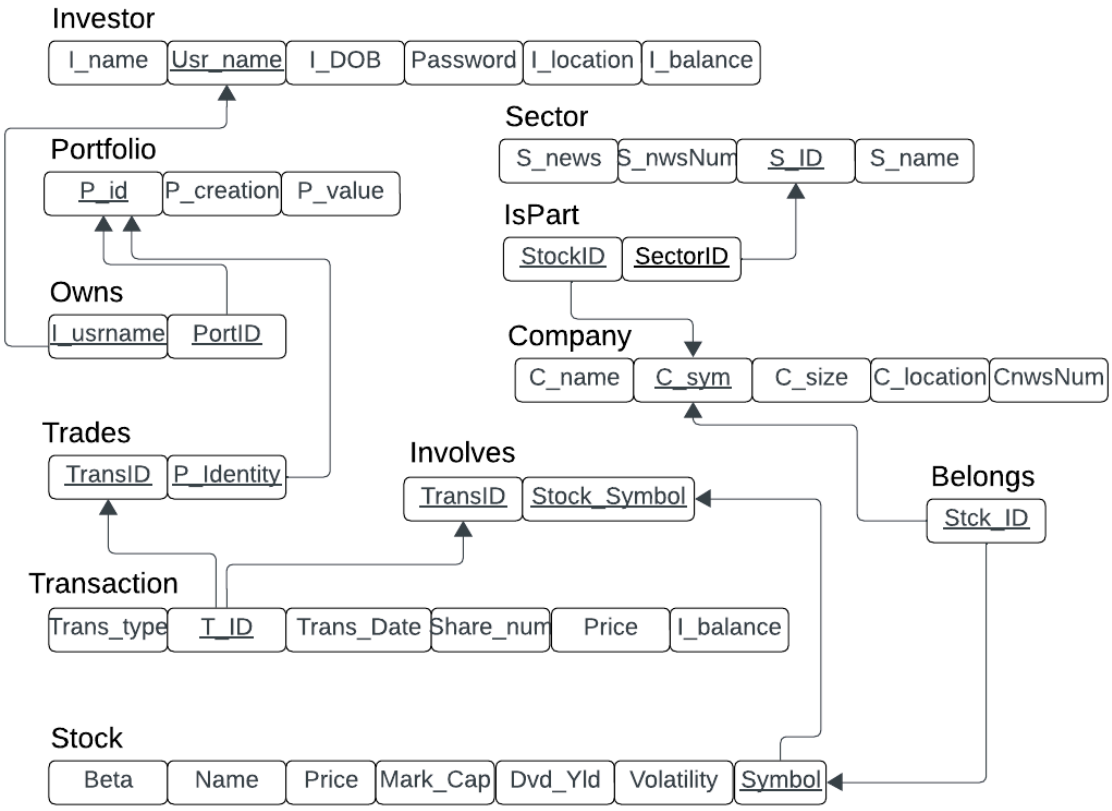
Step 3: 10 English queries [modified]

[No need to list queries here (only [modified] tag). Step 7 should have updated queries with relational algebra expressions]

Step 5: EER modeling



Step 6: Creating Relations



Step 7: Relational Algebra

1. Retrieve the current price of a stock and all portfolios with that specific stock.
   ➔ π StockName, CurrentPrice, PortfolioID (STOCK ⋈ PORTFOLIO)
2. Find the total value of each portfolio based on the current stock prices.
   ➔ π PortfolioID, SUM(CurrentPrice*Quantity) (STOCK ⋈ TRANSACTION)
3. Find the total value of all portfolios.
   ➔ π SUM(CurrentPrice*Quantity) (STOCK ⋈ TRANSACTION)
4. Find the highest, lowest stocks, stocks with the highest trading volume, and highest dividend stocks.
   ➔ π StockName, MAX(CurrentPrice), MIN(CurrentPrice), SUM(Quantity) AS TradingVolume, MAX(Dividend) (STOCK ⋈ TRANSACTION)
5. Find the total value of dividends received by each portfolio.
   ➔ π PortfolioID, SUM(Dividend*Quantity) (STOCK ⋈ TRANSACTION)
6. Retrieve the total investment value of each investor.
   ➔ π InvestorID, SUM(CurrentPrice*Quantity) (STOCK ⋈ TRANSACTION)
7. Find the total value of all investments.
   ➔ π SUM(CurrentPrice*Quantity) (STOCK ⋈ TRANSACTION)
8. Find the total value of dividends received by each investor.
   ➔ π InvestorID, SUM(Dividend*Quantity) (STOCK ⋈ TRANSACTION)
9. Retrieve the names of investors with a total investment value greater than a specified amount.
   ➔ π InvestorName (INVESTOR ⋈ (π InvestorID, SUM(CurrentPrice*Quantity) (STOCK ⋈ TRANSACTION)) σ SUM(CurrentPrice*Quantity) > X)
10. Retrieve the names of investors who have received dividends greater than a specified amount.
    ➔ π InvestorName (INVESTOR ⋈ (π InvestorID, SUM(Dividend*Quantity) (STOCK ⋈ TRANSACTION)) σ SUM(Dividend*Quantity) > X)

Conclusion

In conclusion, for the final version of our project, I have learned how to use Flask in Python for web development, how to make HTTP requests, and how to interact with databases. To improve the project further, I plan to add queries for buying and selling stocks to simplify database updates. Additionally, I aim to enhance the user interface, implement real-time data updates, and incorporate authentication and user management features. These improvements will make the application more user-friendly, efficient, and secure, providing a better experience for users.