CSCI 3901 Project

Vedant Patel (B00984592)

Overview

The InvestmentFirm class is a fund management service for the clients who owns individual portfolios. System monitors funds and the ones who invest in them, imposers, and their clients. It is the tool that can build a plan for those ideas that specify how many dollars should be placed in each investment sector. Financial advisors may use financial accounts they have set up , for particular clients based up on their programme .They will also specify whether or not to reinvest dividends. This system keeps up with the inventories and an ever-increasing market value based on current stock price. It enables both stock buying and stock trading and also manages the income balance. If a company is paying dividends referring full shares of stocks, the application can manage these exceptional situations, and it also performs bookkeeping on fractional shares. The system provides an example of the analysis of the holdings and ports that we stock. For instance, it can compare two investment portfolios according to the potential similarity based on the cosine similarity measure. This technique may initially be used for clients in order to suggest buying specific stocks or selling others by comparing them to other client holdings. It may also do the grouping of financial advisors by the type of investments they prefer which utilizes the k-means clustering technique.

Files and external data

In the context of the given assignment, files and external data plays an significant role,

There are 12 major file is used in this assignment

InvestmentFirm.java (or similar): This file likely contains the core functionalities of the investment firm system. It might define the InvestmentFirm class that offers methods for various actions like creating accounts, buying/selling stocks, calculating value, and performing analysis.

The main logic is defined in this files where all of the required methods are defined in this code like defining the sectors, stocks and profiles and all the trading are done in this files also used methods for analysis like advisor groups and recommending the stocks are buying including reporting methods like disbursing dividends and all.

Used files for data connections are handled in these files of DB config and DBconnection.

There shareTrader and shareManager class used in investment firm files, shareTrader class helps to buy and share shares based on the shares passed to it and shareManager helps to handles the operations related to shares like getting current price of shares and fetching the current shares for account and updating cash balance and checking if it has a sufficient balance and all.

FirmDividendManager this class helps to manage the dividends and help firm to manage how many shares it needs to buy to manage the fractional shares for particular stocks.

There is a other class used in my files which is AccountCheck that helps to handling the checks for the particular account if it is a valid or not.

Another class is CosignSimilarityManager that helps to count the cosign similarity for the 2 vectors given used in the analyzing the system like recommending and finding groups.

Final classes like AccountSectorWeights and StockVectorsCreator these class helps to create the vectors of account and sectors and another create the vectors of stock holdings for account. On the other hand, stockutils class sorts the StockVectors and StockTradingHelper class helps to manage the, finding all the sectors stocks, should I reinvest or not all of these methods are defined.

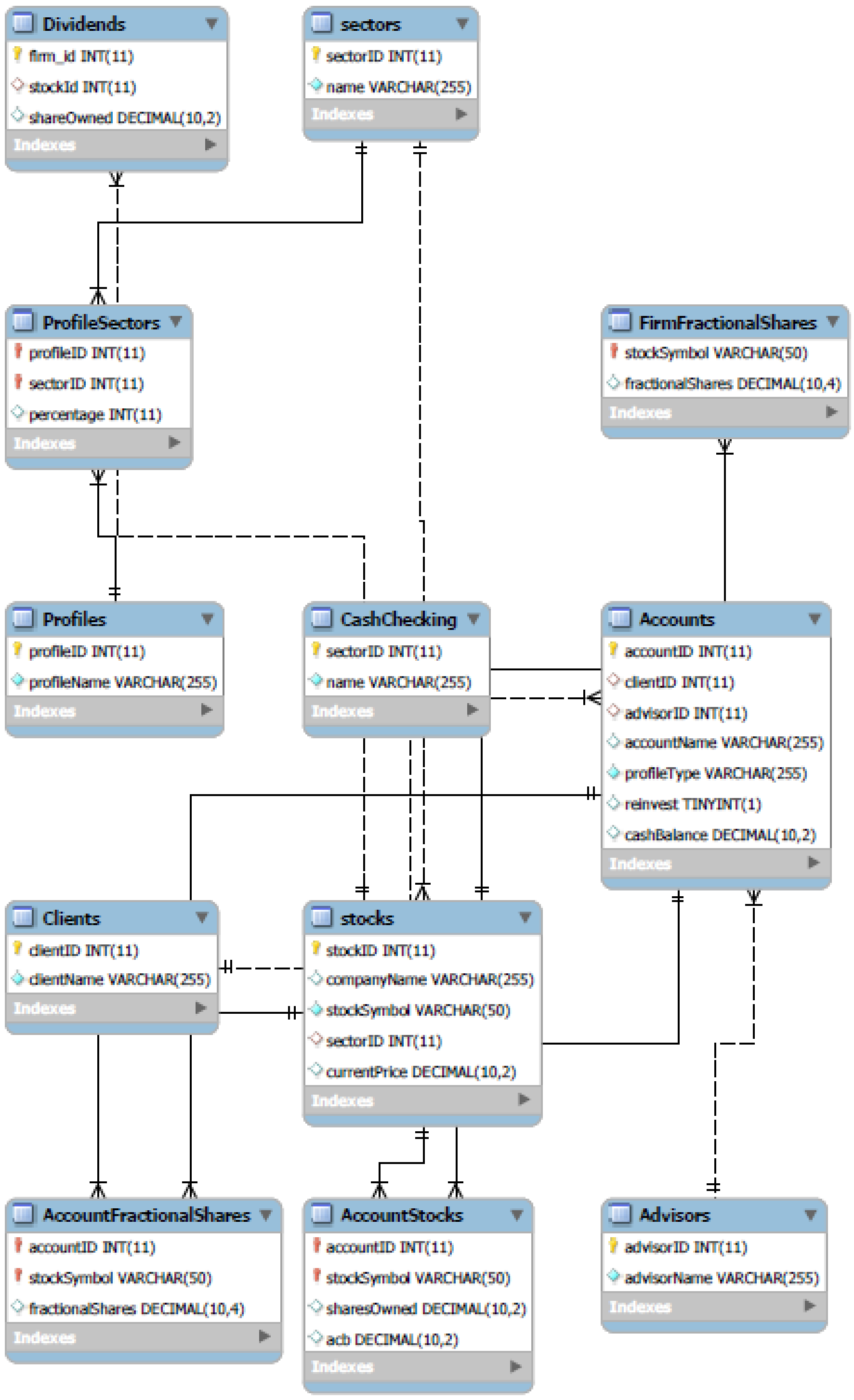

Figure : database and tables used in this projects.

Data structures and their relations to each other

Data structures used in this project explains how this code sample describes a financial management system, which consists of data structures organized within a relational database. At the table center are the beams and pillars of the system called Sectors and Stocks. The table of Sectors keeps detailed statistics regarding various investment sectors, while Stocks table carries the information about the individual stocks along with company name, symbol, sector, and price per each. Together, they form a linked meta table hierarchy through foreign key constraints. SectorID from the Stock table references Sector table, creating a foreign key constraint.

Besides, the module encodes the Profiles and ProfileSectors tables in order to address investment portfolios and their sector distribution. The Profiles table stores profile names as primary keys, along with the ProfileSectors table, which serves as a linking interface between profiles and sectors, enabling a diverse range of allocation approaches. FK Connections between Tables are designed in such a way that all tables can refer to the same sector data. This allows for an effective performance retrieving necessary sector information for profile management.

Furthermore, the database comprises four cards namely, the Advisors, Clients, and Accounts data. The data for financial advisers and clients are held in Advisors and Clients tables. This information handing is the management of relationships. The Storage of Accounts records the accounts of investments which are tied up together by the foreign key constraints of the clients and advisors. Besides this, the AccountStocks table maintains the stocks belonging to a single account and their respective quantities. Accounts table and AccountStocks tables cohesively interact with the Groups table, providing a powerful portfolio management tool.

And we go even further by adding other fields in the accounts table like cash balance to support determination of income to be declared and the adjusted cost nutation in the AccountStocks table for better financial management. The items compose of a financial management mechanism that is strong enough to be applied in different investment scenarios and to manage different portfolio arrangements.

The provided set of functions comprises a cohesive framework for financial analysis and management, each serving distinct yet interconnected purposes. Starting with advisorPortfolioValue(int advisorId), it initiates by fetching account IDs managed by a specified financial advisor and then calculates the total portfolio value by invoking accountValue(accountId) for each account. Moving to investorProfit(int clientId), it retrieves account IDs associated with a given client and proceeds to compute profits for each account based on their respective stock holdings and average cost basis. Meanwhile, profileSectorWeights(int accountId) focuses on determining the sector weights within a specific account, relying on StockTradingHelper.getAllSectorNames(connect) for retrieving all sector names. In parallel, divergentAccounts(int tolerance) scrutinizes account sector weights against target values with a defined tolerance, leveraging profileSectorWeights(accountId) to access sector weights. Further, disburseDividend(String stockSymbol, double dividendPerShare) facilitates dividend distribution to relevant accounts, utilizing StockTradingHelper.shouldReinvest(accountID, connect) to decide on dividend reinvestment. stockRecommendations(int accountId, int maxRecommendations, int numComparators) generates stock recommendations based on similar accounts, depending on StockVectorsCreator.createStockVectors(connect) for comparison. Lastly, advisorGroups(double tolerance, int maxGroups) forms advisor groups based on sector weight similarities, relying on AccountSectorWeights.getAccountSectorWeights(connect) for retrieving advisor sector weights. Together, these functions form an integrated system for comprehensive financial analysis and decision-making.

- Arrays/Lists/Maps: These data structures are used extensively throughout the functions. For example, in advisorPortfolioValue(int advisorId), a map is utilized to store account IDs and their corresponding values. Similarly, in investorProfit(int clientId), a map is used to store account IDs and their profits.

- HashMaps: The functions utilize HashMaps for efficient storage and retrieval of key-value pairs. For instance, in investorProfit(int clientId), a HashMap is employed to store profits by account ID.

- Sets: Sets are used to represent collections of unique elements. In divergentAccounts(int tolerance), a set is employed to store the IDs of divergent accounts.

- Lists: Lists are used to store ordered collections of elements. In advisorGroups(double tolerance, int maxGroups), a list of initial cluster representatives is randomly selected for clustering purposes.

- Nested Data Structures: Several functions use nested data structures to represent complex relationships. For instance, in stockRecommendations(int accountId, int maxRecommendations, int numComparators), nested maps are employed to represent stock symbols and their corresponding boolean values indicating whether to buy or sell.

- Object-Oriented Data Structures: While not explicitly mentioned, certain functions likely interact with custom-defined object-oriented data structures. For instance, methods like ShareManager.getCurrentSharePrice() and StockTradingHelper.shouldReinvest() may operate on objects representing shares, stocks, or accounts.

Assumptions

- program assumes that data coming as a input is a type of String being a case insensitive.
- Program assumes the while adding stocks in the stocks table it sets the by default price 1 for the stocks
- Program assumes in define profile method, I chose to store holdings for the profile only when all the percentage are 100 percent.
- In disburse dividend program assumes to buy the share for the account which needs the fraction part of the share without considering cash portion from firm's side.
- In advsorGroups program assumes the tolerance from -1 to 1

Choices

- The choice I made to add the cash as a default sector while defining the any sector name for the first time.
- In define profile method if all of the sector holdings are given of 100 percent then add the cash as a 0 percent.
- As a return type of the methods like define stock, profile, sectors I decided to return nothing if defined successfully else return a message shown as a exception.
- In disburseDividend method, I return a value of the zero if firm does not need to buy anything for fractions of the stocks for all of the accounts, else return the integer firm needs to buy for managing the fractions.
- I am maintaining the average value of the stock price in AccountStocks tables for evaluating the profits for each account and it will only be change when buying the stocks at different price.
- One more thing in disburse dividend method to maintain the fractional part of the stock I am storing it in double type

Key algorithms and design elements

- SQL Queries [2]: The functions connect with the database by SQL queries retrieving, proceeding, and changing accounting items. SQL queries including altitudes fetching account information, stock data, and sector details are the heart of the database queries.

- Error Handling: Exception handling is implemented in order to consider the possibility of , as well as access errors. This makes the system self-reliant and smart, so it can handle unexpected circumstances without losing its integrity.
- Iteration: On several instances iteration is used to realize data obtained from storage devices. For instance, in investorProfit(int clientId) a while loop is to be iterated across accounts and nested while loops to further iterate across stocks within each account.
- Mathematical Calculations: Calculations are performed using mathematical methods to assign value to portfolios, to calculate profits and sector weights also includes the delivery of dividends. These are procedures with fundamental realizations which are advanced in various mathematical branches such as arithmetic, and subdivision, multiplication, and division.
- Percentage Calculation: The calculation logic (%icount%)is included in the profileSectorWeights(int accountId) function, meaning that the weight of each sector is calculated proportionally in the portfolio. This is derived the division of the sector value by the whole portfolio value and truncation to get the percentage weight.
- Cosine Similarity [1]: Particularly, StockRecommendations(accountId, maxRecommendations, numComparators)which makes use of cosine similarity to determine the correlations between different stock holdings portfolios. This is being realised by using the cosine of the angle between two vectors, one of which indicating the stock ownership.
- Clustering: While the provided code does not explicitly indicate how K-means clustering is applied, this algorithm is amongst the options that may be employed to cluster financial advisors by extending the financialGroupAdvisor(double tolerance, int maxGroups) function so that it groups advisors who have similar portfolios.
- Data Structures: It is based on the fact that the functions carry out different operations on data structures such as maps, sets, lists and nested data structures (maps within maps) which help in the easy management and manipulation of financial data.
- Conditional Logic: Conditional statements are the part of the language that allows for the business logic and decision-making logic to be implemented. In such example, in dividendDistribution(String stockName, double dividendsPerShare), decision-making algorithms of whether dividends are reinvested or amount in cash is compared with account settings.
- Random Selection: An advisorGroups (double tolerance, int maxGroups) groups algorithm uses the random selection tool for selecting a set of initial

cluster representatives for clustering of financial advisors.

Limitations

- In recommending the stocks in analysis system I am storing stocks holdings with 0 for account as well for vector creating which might not be the best way.
- In advisorGroups I am using the random initialization of the clusters.
- For some cases it is not possible to change the data defined in the system like once we have added sector health care we might not able to change excepts for some cases like updating cash balance and all this is possible.

References

[1] GeeksforGeeks, "Angle between Two Vectors Formula," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/angle-between-two-vectors-formula/. [Accessed: April 07, 2024].
[2] "Statements and ResultSet Objects," MySQL :: MySQL Connector/J Developer Guide, [Online]. Available: https://dev.mysql.com/doc/connector-j/en/connector-j-usagenotes-statements.html. [Accessed: April 02,2024].