Database

By analyzing all the 17 methods provided in the project document, I created the database in phpMyAdmin, and using the designer tab I generated the design of that database.

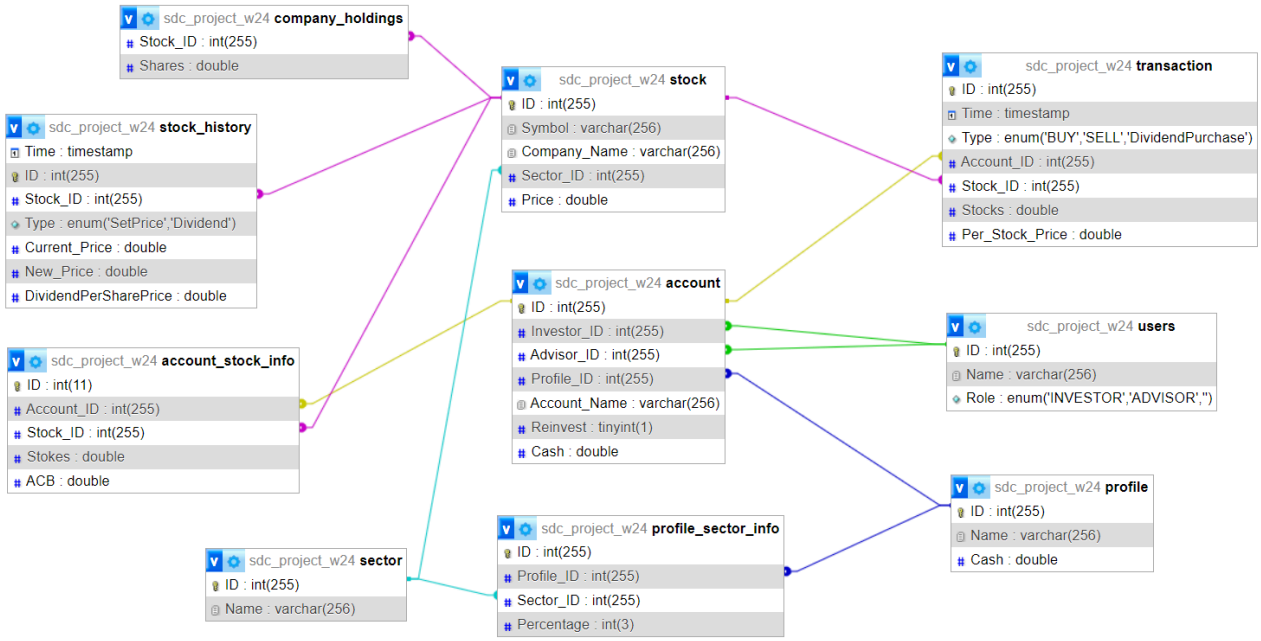

Figure 1 Database Design

Methods

int defineSector(String sectorName)

- This method will create a sector in the “sector” table and return the ID of that sector.

  Testing

  1. Create a sector with a null or empty string
  2. Create a sector with a valid name.
  3. Create a sector with the name already existing in the database.
  4. Create a sector with the name containing an integer as a string.
  5. Create a sector with a name containing special characters.

int defineStock(String companyName, String stockSymbol, String sector)

- We fetch the sector ID from the “sector” table of the given sector name.
- We set the initial stock price as $1.
- Now method will create a stock in the “stock” table with the given company name, stock symbol, fetched sector ID, and the decided stock price.
- Return the stock ID of the created stock.

Testing

1. Create a stock with a null or empty parameter.
2. Create a stock with the sector already existing in the database.
3. Create a stock with the sector not present in our database.
4. Create a stock that already exists in our database.

boolean setStockPrice(String stockSymbol, double perSharePrice)

- We fetch the stock ID from the “stock” table of the given stock symbol.
- Then we update the price in the “stock” table of this particular stock ID
- We also have to update in “stock_history” table where the type is SetPrice, the old price is the current price of this stock ID, and the new price is the given perSharePrice.
- Return true if the price is successfully set, else return false.

Testing

1. Set the price as negative.
2. Set the price below than the current price.
3. Set the price higher than the current price.
4. Set the price of the newly created stock.
5. Set the price as zero.
6. Set the price for the stock that is not present in our database.

int defineProfile(String profileName, Map<String, Integer> sectorHoldings)

- Create a profile in the “profile” table with the given profile name, and set the cash to the value present in the sector holdings.
- It is mandatory to have some percentage of the cash sector in the given sector holdings.
- Now enter the remaining sector other than cash into the “profile_sector_info” table.
- Return the ID of the created profile.

Testing

1. Create a profile with the given null or empty sector holdings
2. Create a profile with a null or empty profile name.
3. Create a profile where there is no sector named cash.
4. Create a profile with only two sectors, of which one is cash.
5. Create a profile which is already existing in our database.
6. Create a profile where an Integer as a string or special Character is present in the profile name.
7. Create a profile in which sector holdings are more than 100%.

int addAdvisor (String advisorName)
int addClient (String clientName)

- This both method have the same kind of code and affect the same table.
- Create a new user in the “user” table. If the user is an advisor than the type will be ADVISOR, else the type will be “CLIENT”.
- Return the ID of the created user.

Testing

1. Create a user with a null or empty string.
2. Create a user that already exists in our database.
3. Create a user which contains integers or special characters in the name.
4. Create a user with the name “SDC” and type “ADVISOR”.
5. Create a user with the name “3901” and type “CLIENT”

int createAccount(int clientId, int financialAdvisor, String accountName, String profileType, boolean reinvest)

- Fetch client ID, and advisor ID from the “users” table.
- Fetch the profile ID from the “profile” table.
- Create an account in the “account” table with the given fetched client ID, advisor ID, and profile ID, account name, and initially set reinvest as false.
- Return the created account ID.

Testing

1. Create an account with an invalid clientId, financialAdvisor. (Invalid means not exist in the system)

2. Create an account with the existing clientId or financialAdvisor in the “account” table.
3. Create an account with the account name as null or empty.
4. Create an account with the profileType which does not exist in the profile table.
5. Create an account where the client, advisor, and profile type are valid and the account name is also in the proper format.

boolean tradeShares(int account, String stockSymbol, int sharesExchanged)

- First, if the shares exchanged are positive then the user wants to buy the share then we have to check the amount (sharesExchanged * PricePerShare) must be present as cash in the user account. Moreover, ensure that the cash balance does not go to less than 0.
- Make the change in cash balance and enter details into the “transaction” table.
- If the user buys then the Type is “BUY”, and if the user wants to sell then the type is “SELL”, account ID is the account given in the parameter, and stock ID will be fetched from the “stock” table from the given stock symbol, stocks will be sharesExchanged, and the Per_Stock_Price will be the current price of this stock in “stock” table.
- Update “account_stock_info” for the current account, and also update the ACB value in that table.
- Return true if the trade of this stock is successfully done, else return false.

Testing

1. Call with the account ID does not exist in the account table.
2. Call with the stock symbol as null or empty.
3. Call with the stock symbol does not exist in the stock table.
4. Call with the sharesExchanged as a negative value.
5. Call with the sharesExchanged as a positive value.
6. Call with the sharesExchanged as zero.
7. Call with the sharesExchanged whose amount in cash is not present in that account.
8. Call with the account and stockSymbol both are valid and the value of sharesExchanged present in that account cash.

boolean changeAdvisor(int accountId, int newAdvisorId)

- Just change the Advisor_ID to newAdvisorId in the account table for the given accountId.

Testing

1. Call with the account not exist in the database.
2. Call with the newAdvisorId not present in our database.
3. Call with accountId or newAdvisorId as negative.
4. Call with those accountId, and newAdvisorId which exist in our database.

Reporting

double accountValue(int accountId)

Testing

1. Call with account value does not exist in the database.
2. Call with account value exists in database.
3. Call with an account that hasn't bought any shares till date.
4. Call with an account who have multiple transactions in the same stock.
5. Call with an account who have multiple transactions on multiple stocks.

double advisorPortfolioValue(int advisorId)

Testing

1. Call with an advisor who didn't exist in the database.
2. Call with an advisorId as a negative value.
3. Call with an advisorId as zero.
4. Call with an advisor exist in the database but didn't assign any client.
5. Call with an advisor who has multiple clients with the same profile.
6. Call with an advisor who has multiple clients with different profiles.
7. Call with advisor who had clients but they didn't perform any transaction on any stock.
8. Call with an advisor who has clients with different profiles and had many transactions with different stocks.

Map<Integer, Double> investorProfit(int clientId)

Testing

1. Call with a client who didn't exist in the database.
2. Call with a client as a negative value.
3. Call with a client as zero.

4. Call with a client who hasn't bought any shares till date.
5. Call with a client who has multiple transactions in the same stock.
6. Call with a client who has multiple transactions on multiple stocks.
7. Call with a client who had only money in cash, and has no stocks.
8. Call with the client who only buys stocks but never sells any stocks.
9. Call with the client who is at a profit.
10. Call with the client who is at a loss.
11. Call with the client who has multiple accounts in which some accounts are in profit and some are in loss.
12. Call with the client who had multiple accounts but did not hold a single stock in any account.

Map<String, Integer> profileSectorWeights(int accountId)

Testing

1. Call with an accountId as a negative value.
2. Call with an accountId set as zero.
3. Call with accountId does not exist in the database.
4. Call with accountId exists in database.
5. Call with an accountId that hasn't bought any shares till date.
6. Call with an accountId who only invests in one sector.
7. Call with an account that invests in multiple sectors.

Set<Integer> divergentAccounts(int tolerance)

Testing

1. Call with the tolerance as a negative value.
2. Call with the tolerance as zero
3. Call with the tolerance as a positive value less than 100.
4. Call with a tolerance greater than 100.
5. Call when no account exists in the system.
6. Call when all the accounts strictly follow the profile assigned by the advisor.
7. Call when some accounts had a tolerance greater than the given tolerance, some accounts had a smaller than the tolerance, and many had 0 tolerance from their assigned profile.
8. Call when all the clients have a single account in the system.
9. Call when all the clients have multiple accounts in the system some account tolerance is within the limit and some accounts exceed it.

10. Call when there is only one profile in the system and all the clients are assigned that same profile.

int disburseDividend(String stockSymbol, double dividendPerShare)

Testing

1. Call with the stockSymbol not exits in the database.
2. Call with the negative value of dividendPerShare.
3. Call with the dividendPerShare set as zero.
4. Call with the positive value of dividendPerShare, and the given stockSymbol exists in the database.
5. Call with the newly created stock.
6. Call when no account exists in the system.
7. Call when an account exists but not a single account holds that company shares.
8. Call when all accounts set the reinvest option as false.
9. Call when a few accounts set the reinvest options as true.
10. Call when a few accounts had to reinvest but he didn't have enough money to buy the entire 1 stock.

Map<String, Boolean> stockRecommendations(int accountId, int maxRecommendations, int numComparators)

Testing

1. Call with accountId not present in database.
2. Call When no account is present in the system.
3. Call with maxRecommendations set as zero.
4. Call with numComparators set as zero.
5. Call when all accounts are assigned the same profile.
6. Call when the account has a different profile and not a single stock is not between them.
7. Call with the newly created account.

Set<Set<Integer>> advisorGroups(double tolerance, int maxGroups)

Testing

1. Call with the negative value of tolerance or maxGroups.
2. Call with the tolerance or maxGroups set as zero.
3. Call with the positive value of tolerance or maxGroups.

4. Call when there is no advisor in the system.
5. Call when all the advisors give the same profile to all the clients.
6. Call when there are multiple advisors present and each one's advice is different from one another.
7. Call when there are multiple advisors and they are different but within the given tolerance.
8. Call when the client had multiple accounts but al the accounts had the same advisors.
9. Call when the client had multiple accounts and all the accounts had different advisors.

Planning

6 April - 7 April

Decide the software design and make a file structure according to that design.
Database Connectivity and Custom log feature, error handling.

8 April

defineSector
defineStock
defineProfile
addAdvisor
addClient

9 April

setStockPrice
createAccount
tradeShares
changeAdvisor

10 April - 11 April

accountValue
advisorPortfolioValue
investorProfit
profileSectorWeights
divergentAccounts

12 April
disburseDividend

13 April - 14 April
Map<String, Boolean> stockRecommendations(int accountId, int maxRecommendations, int numComparators)
Set<Set<Integer>> advisorGroups(double tolerance, int maxGroups)

15 April
Report