1/17/2019 Sam Judkis

Edmund Dea

**CS 340 Final Project Specs**

**Description**

Our goal is to create a website that displays stock market data. When a user logs in, the site will load the user’s portfolio. The user’s portfolio will contain one or more stock symbols, and they can enter stock purchase details. The website will display the price change and percentage price change for each stock in the user’s portfolio.

**Outline**

1. Project Theme
   1. Stock website
      1. Front-End Design Inspiration - <http://www.tradingview.com>
      2. Real-Time Stock Quotes using API
         1. <https://github.com/iexg/IEX-API/issues>
         2. <https://www.alphavantage.co/>
         3. <http://finance.google.com>
         4. <http://finance.yahoo.com>
         5. <https://stackoverflow.com/questions/10040954/alternative-to-google-finance-api>
2. Querying Stock Data
   1. Use API to query stock data via JSON/etc for most recent price
   2. INSERT INTO <table> VALUES <data>
   3. Then the current logged in user can query the primary stock data table and display stock data.
3. Watchlist
   1. Display a list of stocks in a portfolio.
   2. User can
      1. ADD various stocks to add to their portfolio
      2. DELETE stocks from their portfolio
      3. SORT stock data by ticker/percentage change
      4. UPDATE the portfolio’s stock tickers
      5. SEARCH for stocks within the portfolio by ticker ???
         1. Piazza -
            1. “Every table should be used in at least one SELECT query. For the SELECT queries, it is fine to just display the content of the tables, but your website needs to also have the ability to search using text or filter using a dynamically populated list of properties. This search/filter functionality should be present for at least one entity. It is generally not appropriate to have only a single query that joins all tables and displays them.”
            2. Do they want us to implement a SELECT statement for each entity and/or do they want us to to add search/filter functionality associated with each table?
   3. If a stock is removed from the stock exchange, then handle the case of removing the many-to-one relationship by settings all user portfolios containing that stock to NULL. Then, delete that stock from the appropriate table.
   4. In a many-to-many relationship, to remove a relationship one would need to delete a row from a table.
   5. Displays Data
      1. Stock ticker
      2. Current Price
      3. Percentage Change relative to the day’s opening price
4. Users
   1. User can login
   2. User’s portfolio loads after logging in
   3. (optional) Implement SSL connection
5. Optional Features
   1. Displays stock data in a chart
      1. Shows OHLC (Open-High-Low-Close)
   2. User can add purchase price and # of shared purchased. Then user can UPDATE their purchase price and # of shared purchased.

**Database Outline**

**Entities**

* User
  + Id (primary key) auto incremented
  + Username (varchar, 100)
  + Password hash (varchar, 255) (optional, nice-to-have feature)
* Stock
  + Id (primary key) auto incremented
  + Symbol (varchar, 4)
  + Company name (varchar, 100)
  + Industry (varchar, 100)
  + Any more attributes?
* Price
  + Id (Primary key) auto incremented
  + Stock.Id (int, foreign key)
  + Date/Time (datetime)
  + Price (dec)
* Purchase
  + Id (primary key) auto incremented
  + User.Id (int)
  + Ticker.Id (int)
  + Quantity (int)
  + Purchase price (dec)
  + Purchase date (datetime)
  + Any more attributes?
* Considering these changes but not sure if we want to add these changes yet. What happens if you later end up with a customer who has many portfolios? We need to make sure that our system has the flexibility to handle more relationships than we might have originally anticipated.
  + Portfolio (composite entity that associated users and stocks)
    - Id (primary key) auto incremented
    - User.Id (int)
    - Stock.Id (int)
  + OrderType
    - Id (primary key, int)
    - Type (varchar, 16)
      * market buy
      * market sell
      * limit buy
      * limit sell
  + Order (traditional entity that logs stock orders. This would replace the Purchase table.)
    - Id (primary key, int)
    - Portfolio.Id (foreign key, int)
    - OrderType.Id (foreign key, int)
    - Quantity (int)
    - Price (dec)
    - Date (datetime)

**Relationships**

* User and stock (many to many)
  + A user can follow 0 or more stocks
  + Stocks can be followed by 0 or more users
    - If a stock has been removed from watch list of all users that were following it, does it remain in database?
      * Yes. If a user re-adds the stock, they can query existing stock price data.
* User and Purchase (one to many)
  + A User can make many 0 or more Purchases
  + A Purchase can only be made by exactly 1 user
* Stock and Price (one to many)
  + A Price is related to exactly 1 Stock
  + A Stock is related to 0 or more Prices
* Purchase and Stock (one to many)
  + A Purchase is related to exactly 1 Stock
  + A Stock is related to 0 or more Purchases