ECE 356 Project Report

Group 61, Topic : Stocks

Richmond Victorkumar, rvictork, 20780174

Daniel Arakcheev, darakche, 20778493

Umar Hussain, ufhussai, 20776073

# **Client Application:**

Our client application's main purpose is to provide the client with information about stocks currently listed on the stock exchange and allowing them to update the stored database by modifying, inserting, or deleting records within.

We believe a user of this application, perhaps your average individual who invests, will want to view general company information such as who the CEO is, what sector/industry it operates in, number of employees and their performance using yearly and daily stock prices. The user would also want to view news articles and other people’s comments on how the company is doing so they can get a feel for how others perceive the company. The user should also be able to modify certain contents within the database such as comments, articles, and companies along with their performance.

This is accomplished through a Python CLI app featuring a command line user interface that offers the following menu:

The interface operates through five different modes depending on what the user wishes to do. The current mode is indicated at the bottom left followed by the input space.

**Table 1:** Overview of the five operation modes and their corresponding submenus.

| **Menu** | **Overview** |
| --- | --- |
| **[main menu]** |  |
| **[lookup mode]**  User can retrieve general company information, or view yearly/daily performance, alongside articles and comments of the specified company |  |
| **[update mode]**  Updates can be made to pre-existing company records within the database to modify company name, CEO, employee count, or fiscal end date |  |
| **[insert mode]**  Inserting a new company, populates all related tables with the provided data. Users may also add daily/yearly data, articles or comments to pre-existing companies. |  |
| **[delete mode]**  When deleting a company, the changes are cascaded through all tables within the DB to ensure a thorough and deep deletion. |  |

Upon selecting an option within one of the above menus, in most cases you are asked to identify the company of interest by its ticker symbol or company name as shown:



There were certain features we were not able to incorporate into the client app such as Data Mining for getting deeper insights into our data such as “What factors (attributes) determine the likelihood of a stock price rise in excess of the market average?” or clustering the stocks based on certain attributes. Given more time these features would be a valuable asset for the application.

Further details of each mode's sub-menus are shown in the demonstration video.

Resolving issues such as fixing obvious data errors, removing any duplicate data, and modifying the design in certain cases are described in the ER Design and Relational Schema sections below.

# **ER Design:**

In order to accomplish this client-server application, we were required to create a database that efficiently stores all the information that would be accessible to the user using the application.

To model the database, we created an ER design that consisted of a few entities and relations that holds all the information relevant to companies listed on the stock exchange.

### **Entities:**

Company: This entity is responsible for holding general information about the company that the client may want to know. It holds attributes such as the name of the company, the year it was founded and their fiscal year end date. If the client wants to get information about the company that is not necessarily related to any stock metrics, we would simply query the Company table since it holds all the general information.

DailyData: This entity contains daily metrics recorded for a company's stock over a specific day of the year. This is relevant for users that want to know how a company does on a daily basis rather than yearly or if they want to know how the stock did on a specific day of the year. For each day listed, the table will specify the open/close price, the daily high and low as well as the volume on that specific day.

YearlyData: This entity contains yearly metrics for stocks. This will be beneficial for clients who want to know how a specific company has performed over a given year rather than on a daily basis. It includes important metrics such as revenue, eps, currentRatio and many more attributes that a client may use when deciding whether the company is a good fit for an investment. The data held in this table is useful for determining how well a company is progressing and can also be used to predict the company’s future growth.

Comments: This entity contains all the comments that users have posted about a stock. It is useful for users who may want to get other peoples opinions on a company's performance by reading through comments.

AnalystInfo: This entity contains article headers that discuss stocks and the specific stocks discussed in the article

### **Weak Entities:**

The following weak entities are all connected to the Company table through relations. These tables identify as weak entities since they depend on a company to exist and were initially attributes of the company table but were moved out to reduce redundancy within the Company table.

StateCountry: This weak entity contains the country/state of every company listed in the Company table. Creating a separate entity to hold this data removes duplication within our Company table since multiple companies can be based in the same country/state.

City: This weak entity contains the city of every company listed in the Company table. Creating a separate entity to hold this data removes duplication within our Company table since multiple companies can be based in the same city.

CEO: This weak entity contains all the distinct CEOs listed in our Company table. Creating a separate entity to hold this data improves the efficiency of our model since a CEO can run multiple companies.

Industry: This weak entity contains all the industries that are listed in our Company table. Creating a separate entity to hold this information reduces the amount of duplicate entries within our Company table since many companies work in the same industry.

Sector: This weak entity contains all the sectors that are listed in our Company table. Creating a separate entity to hold this information reduces the amount of duplicate entries within our Company table since many companies work in the same sector.

### **Relations:**

CompanyInCity: This relation is used to relate the Company and City tables. The Company table has a 1:\* cardinality since many companies can be located in the same city and the City table has a 1:1 cardinality. This relation maps a company to the city it's located in. The purpose of this relation was to efficiently store companies and their locations in the case where the user wants to filter stocks by location.

CityInCountry: This relation is used to relate the StateCountry and City tables. Both of these tables have a 1:1 cardinality constraint in this table since each city can only be in one state/country and vice versa. This relation is used to map a city to its state/country and allows the user to filter stocks by state/country.

CEOruns: This relation is used to relate the Company and CEO tables. The Company table has a 1:1 cardinality since it can only have one CEO and the CEO table has a 1:\* cardinality since one person can be the CEO of multiple tables. This relation maps a CEO to their companies. The purpose of this relation was to efficiently provide a lookup of CEOs and their companies in the case where the user wants to filter stocks by CEO.

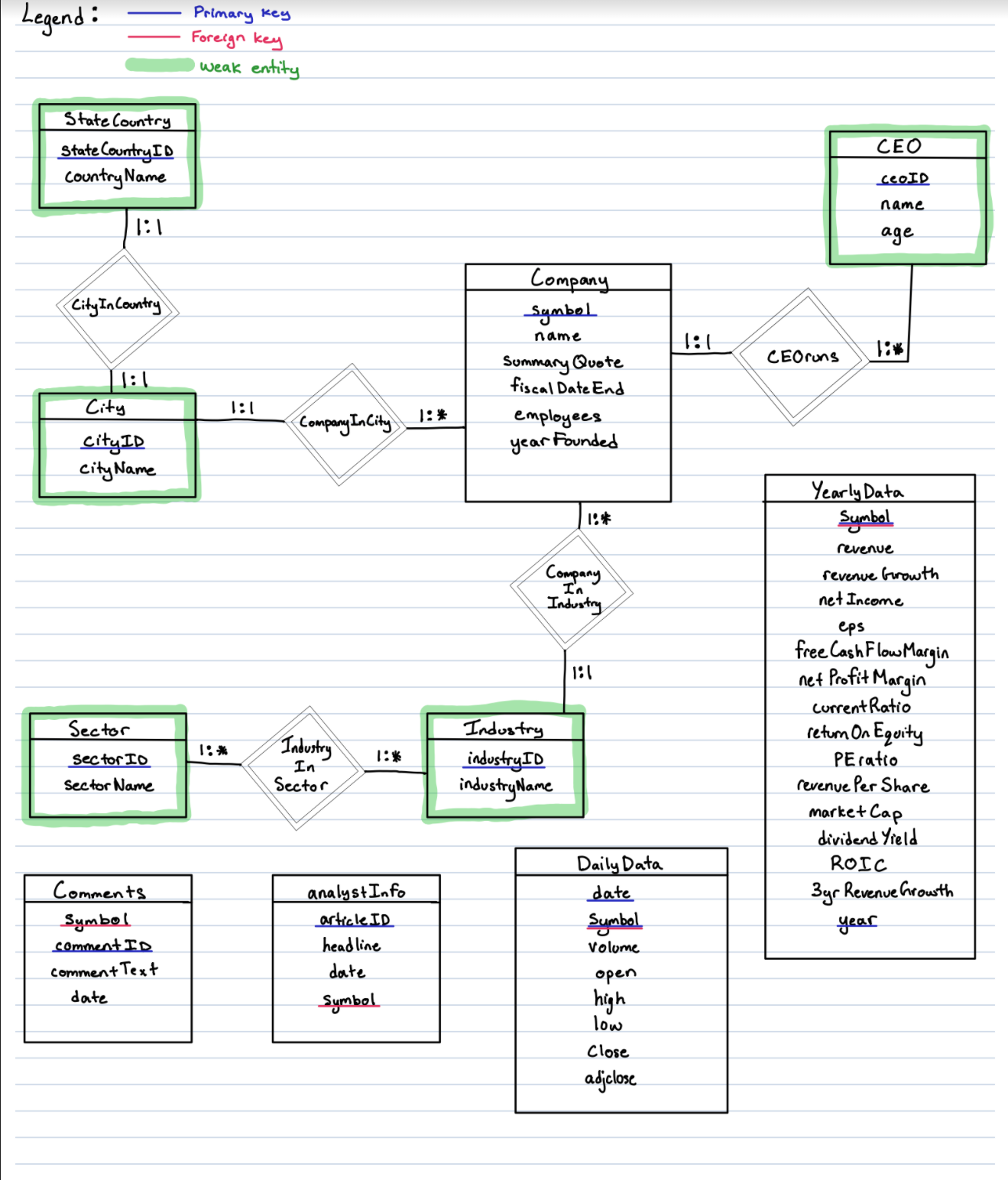
CompanyInIndustry: This relation is used to relate the Company and Industry tables. The Company table has a 1:\* cardinality since many companies can work in the same industry and the Industry table has a 1:1 cardinality. This relation maps a company to the industry they work in. The purpose of this relation was to efficiently provide a lookup of all the companies that work in a specific industry in the case that the user wants to filter companies by industry.

IndustryInSector: This relation is used to relate the Industry and Sector tables. Both of these tables have a 1:\* cardinality constraint in this table since industries can be associated with multiple sectors and vice versa. This relation is used to map an industry to a sector and allows the user to filter stocks by sector.

### **Alternative Design Options:**

An alternative design option we had thought of but decided not to implement was a relation for the Company-Comments table and Company-analystInfo table. We decided not to include this in our final ER design since it had no performance benefit. In fact it would have resulted in additional tables making our database less efficient since we have to use a join operation to query comments or analyst info related to a stock rather than a direct look up which is what is currently implemented.

## **Final ER Design:**



# **Relational Schema:**

To create our relational schema, we first created tables for all the entities/relations in our ER model and assigned each of the tables its appropriate primary/foreign key as specified in the model.

After creating all the necessary tables, we loaded in all the data from the csvs. For the DailyData and analyst info tables we loaded in the fh\_5yrs.csv and the analyst\_ratings\_processed.csv files respectively. For the Company and YearlyData tables we decided to only keep some of the columns since we decided that some of the data provided was irrelevant or redundant for our client-server application. To do this, we created a python script to extract the information we wanted and inserted it into a new csv which was then uploaded onto the server and loaded into our tables.

After creating the tables, the next thing we last needed to do for our relational schema was specify some check constraints for our tables to ensure that the user cannot insert invalid data.

The check constraints are as follows:

Company: The symbol and name attributes should never be empty and so to ensure this we added a NOT NULL constraint. We also added a check on the employees attribute to ensure that this value was greater than 0 since a company at least needs one employee.

Comments: The symbol and commentText attribute should never be empty and to ensure this we added a NOT NULL constraint.

AnalystInfo: The symbol and headline attribute should never be empty so we added a NOT NULL constraint.

DailyData: The symbol and date should not be empty so we added a NOT NULL constraint. The volume, open, high, low, close and adjclose attributes should not have any negative values since it is a price which was solved by a check to ensure the input values are greater than or equal to 0.

YearlyData: The symbol and year should not be empty so we added a NOT NULL constraint. Since the stock market only started in 1970, we also added a check to make sure the input value for the year was greater than or equal to 1970.

CEO: The age of a CEO should not be a negative value. To ensure this never happens we added a check to make sure the input value is greater than 0.

All the code for the relational schema is in stock\_schema.sql.

# **Test Case Plan:**

Case 1.1: Insert company

* Go into insert mode
* Select “Add a company”
* Insert company by providing required information
* Check if values appear in the tables in the backend by manually running ‘select’ queries

Case 1.2: Lookup company

* Go into lookup mode
* Select “Look up company information”
* Enter company symbol
* Check if values displayed match the values in the backend tables by manually running ‘select’ queries

Case 1.3: Delete company

* Go into delete mode
* Select “Delete company”
* Enter company symbol
* Check if values that were displayed during the previous lookup are now gone from the backend tables by manually running ‘select’ queries

Case 2.1: Update company info

* Go into update mode
* Select “Update company name”
* Enter company symbol
* Enter new company name
* Check if company name was updated in the database by manually running ‘select’ queries
* Run this test case for each choice in the Update mode choice menu.

GITHUB LINK : https://github.com/umarxfhu/ECE356\_Project\_Stocks