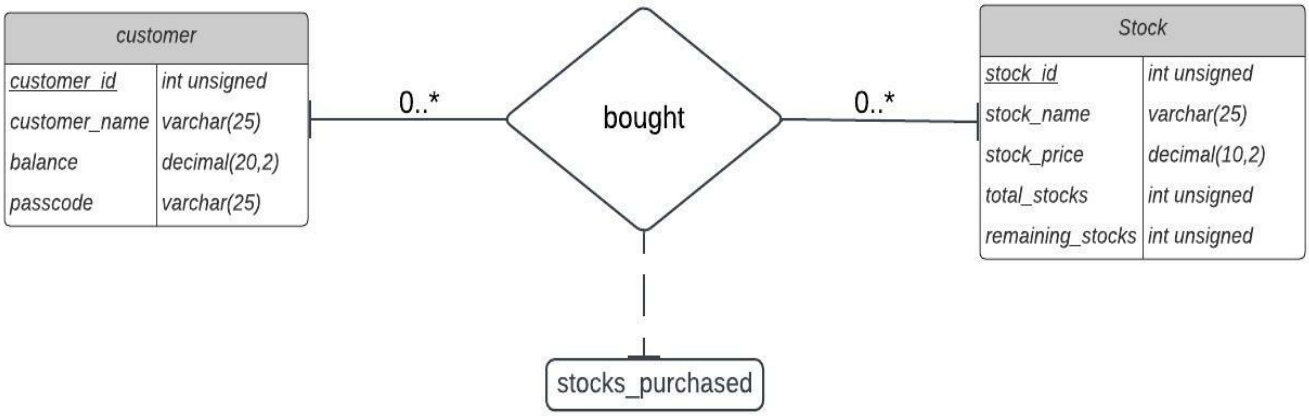
Trading Platform Documentation

- **Functional and Data Requirements**

  The trading platform we have created, basically allows users to buy and sell stocks. The users can create their account with a username, password and balance. Once logged in, the user can buy stocks at his/her stated price, if possible. The user can also sell stocks at the marked price. The user can also deposit more money into his/her account and spend accordingly. (S)he can retrieve information about different stocks and his/her own purchase history. The total stocks of a given firm are maintained to be constant. Hence, the user account data, company stock data and the data of stock purchases of the users need to be maintained.

  A single valued base price (called ‘stock_price’) for each stock (in the ‘stock’ relation) is taken into consideration, which remains fixed during the course of this project. Monetary transactions of buying and selling of stocks are made on the basis of this stock price.

- **ER Model**



- **Schema Design**
    1. customer(<u>customer_id</u>, customer_name, balance, passcode)
    2. stock(<u>stock_id</u>, stock_name, stock_price, total_stocks, remaining_stocks)
    3. bought(<u>stock_id</u>, <u>customer_id</u>, stocks_purchased)

- **Relations**

1. customer(<u>customer_id</u>, customer_name, balance, passcode)

    Description:

    customer_id(int unsigned): It is the primary key of the relation and it is set to auto_increment so that every time a new row in this relation is created, it is assigned a unique customer_id on its own.

customer_name(varchar(25)): name of the customer
balance(decimal(20,2)): amount of money in user's account
passcode (varchar(25)): user's password to facilitate login feature

2. stock(<u>stock_id</u>, stock_name, stock_price, total_stocks, remaining_stocks)

   Description:

   stock_id(int unsigned): It is the primary key of the relation and it is set to auto_increment so that every time a new row in this relation is created, it is assigned a unique stock_id on its own.
   stock_name (varchar(25)): name of the stock
   stock_price (decimal(10,2)): market price of a given stock
   total_stocks (int unsigned): total number of stocks issued by the firm
   remaining_stocks (int unsigned): number of stocks that are unsold

3. bought(<u>stock_id</u>, <u>customer_id</u>, stocks_purchased):

   Description:

   customer_id(int unsigned): ID of the customer who has purchased some stocks. It references the primary key 'customer_id' of the customer relation.
   stock_id(int unsigned): ID of the stock that is purchased by the customer with corresponding customer_id. It references the primary key 'stock_id' of the stock relation.
   stocks_purchased (int unsigned): number of that particular stock purchased by the customer
   Customer_id and stock_id together form the primary key of the relation.

- **Data Normalization:**

1. **customer**

   The relation ‘customer’ is flat, that is all attributes of the relation are atomic. So, the relation ‘customer’ is in 1NF. Since the relation has only a single attribute primary key, ‘customer_id’ so, it does not have any partial dependencies, which means that it is also normalized upto 2NF. None of the non-prime attributes ‘customer_name’, ‘balance’ and ‘passcode’ can determine each other (multiple customers can have the same customer_name). Hence, there are no transitive dependencies in the relation where a non-prime attribute determines another non-prime attribute. So, it is also normalized upto 3NF.

2. **stock**

   The relation ‘stock’ is flat, that is all attributes of the relation are atomic. So, the relation ‘stock’ is in 1NF. Since the relation has only a single attribute primary key, ‘stock_id’ so, it does not have any partial dependencies, which means that it is also normalized upto 2NF. There are no transitive dependencies in the relation where non-prime attributes stock_name, stock_price, total_stocks, remaining_stocks determine each other. So, it is also normalized upto 3NF. (here we have assumed that stock_name is not unique, as there is a possibility that the same company may have different stocks at different prices, so the unique identification is on the basis of the stock_id only)

3. **bought**

   The relation ‘stock’ is flat, that is all attributes of the relation are atomic. So, the relation ‘stock’ is in 1NF. The relation has no

partial dependencies that is, no attribute depends only on one of the prime attributes of the binary primary key. This means that it is also normalized upto 2NF. There are no transitive dependencies in the relation where a non-prime attribute determines another non-prime attribute. So, it is also normalized upto 3NF.

- **Procedures:**

1. **buy**
    a. Input Parameters
        i. c_id (int unsigned): ID of the customer who wishes to buy stocks
        ii. s_id (int unsigned): ID of the stock that the given customer wants to buy
        iii. price (decimal(10,2)): price at which the customer wishes to buy the given stock
        iv. stock_number (int unsigned): number of the given stocks that the customer wishes to buy
        v. passcode (varchar(25)) : password of the customer
    b. Description
        First, it is checked whether the inputted c_id and s_id correspond to existing values in the ‘customer’ table and the ‘stock’ table. If they do, then it is further checked if the customer has sufficient balance to buy the given number of stocks at the price specified by him/her. If (s)he does, and the price is greater than or equal to the corresponding stock_price in the ‘stock’ table and the number of stocks demanded by the customer is less than the corresponding remaining_stocks in the ‘stock’ table, then we move forward and let the customer purchase the given number of stocks at his/her price. As a result, the

balance is updated in the ‘customer’ table and remaining_stocks is updated in the ‘stock’ table. At this point, we check if the given customer has already purchased some of these same stocks before. If so, we simply update the stocks_purchased in the ‘bought’ relation, else we insert a new row corresponding to the given customer_id, stock_id and stock_purchased.

2. **sell**
    a. Input Parameters
        i. c_id (int unsigned): ID of the customer who wishes to sell stocks
        ii. s_id (int unsigned): ID of the stock that the given customer wants to sell
        iii. stock_number (int unsigned): number of the given stocks that the customer wishes to sell
    b. Description

        First, it is checked whether the inputted c_id and s_id correspond to existing values in the ‘customer’ table and the ‘stock’ table. If they do, then the given number of stocks are sold at the current price of the stock and the customer balance is updated accordingly, that is the amount of money is added to the customer’s account in the ‘customer’ relation. The number of remaining stocks are also increased accordingly in the ‘stock’ relation and the stocks purchased are decreased in the ‘bought’ relation.

3. **new_user**
    a. Input Parameters
        i. user_name (varchar(25)): the name user wants for his/her new account

ii. user_balance (decimal(20,2)): amount of money, the user has in his/her account
iii. new_password (varchar(25)): password set by the user

b. Description

This procedure simply inserts a new row in the ‘customer’ table with the customer_name and balance as the user_name and user_balance. It assigns a unique customer_id to each new customer, via the use of auto_increment. Then it displays the entire row with customer_id, customer_name and balance.

4. update_balance

a. Input Parameters
   i. c_id (int unsigned): id of the customer who wishes to update his/her account balance
   ii. add_balance (decimal(20,2)): the amount of money that the customer wishes to add to his/her existing balance

b. Description

This parameter allows the user to add money to his/her account.

5. stock_info_specific

a. Input Parameters
   i. s_id (int unsigned) : id of the stock whose information, the user wants

b. Description

This procedure displays the stock_id, stock_name, stock_price and the remaining number of stocks, based on the stock id the user specifies

6. stock_info_overall

a. No input Parameters

b. Description

This procedure displays the stock_id, stock_name, stock_price and the remaining number of stocks of all the stocks in the database.

7. purchase_history

a. Input Parameters
   i. c_id (int unsigned) : id of the customer who wishes to see the stock (s)he has purchased
   ii. passcode (varchar(25)) : customer's password

b. Description

This procedure allows customers to view the list of all the stocks that they have purchased along with the number purchased.

8. view_balance

a. Input Parameters
   i. c_id (int unsigned) : id of the customer who wishes to see his/her balance
   ii. passcode (varchar(25)) : customer's password

b. Description

This procedure allows customers to view the balance in their account.