**Task 1: Revising Design**

Use cases, structural database rules, and ERDs are provided here without changes from Iteration 5:

*Account Creation*

1. The user visits StockTracker website or app
2. The user creates an account
3. The user enters info needed in order to create an account

*Holdings Creation*

1. The user log into account
2. The user creates an investment portfolio
3. The user adds individual investment to the portfolio

*Stock Transaction Tracking*

1. The user buys/sells stock(s) on a broker’s website/app
2. The user transfer money to broker account if insufficient fund available when buying stocks
3. The user enter the transaction made into StockTracker website/app

*Search Stock Transaction:*

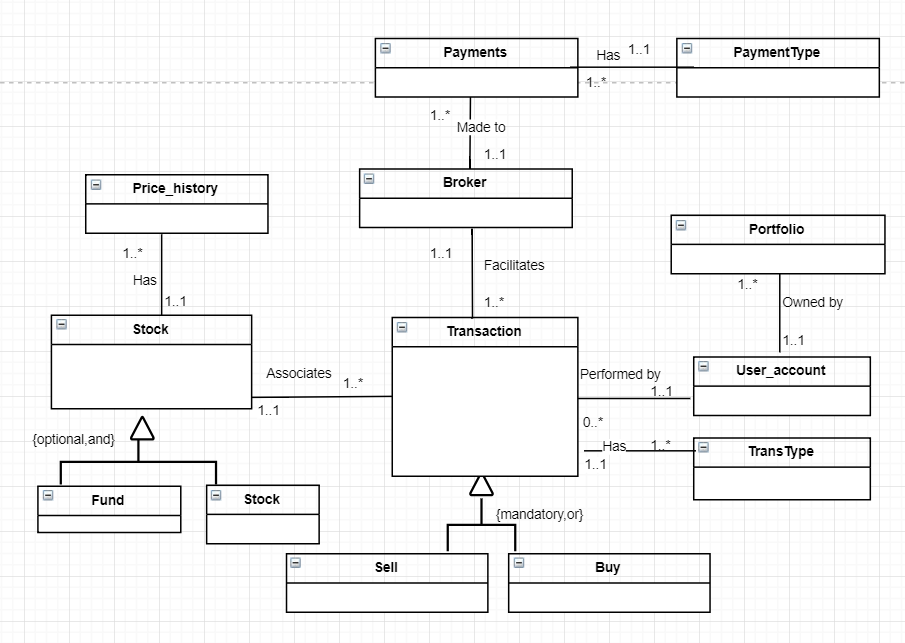
1. The user selects the option to search past transactions with a period of time
2. The user enter the stock symbol
3. The app pulls all transactions as requested by the user

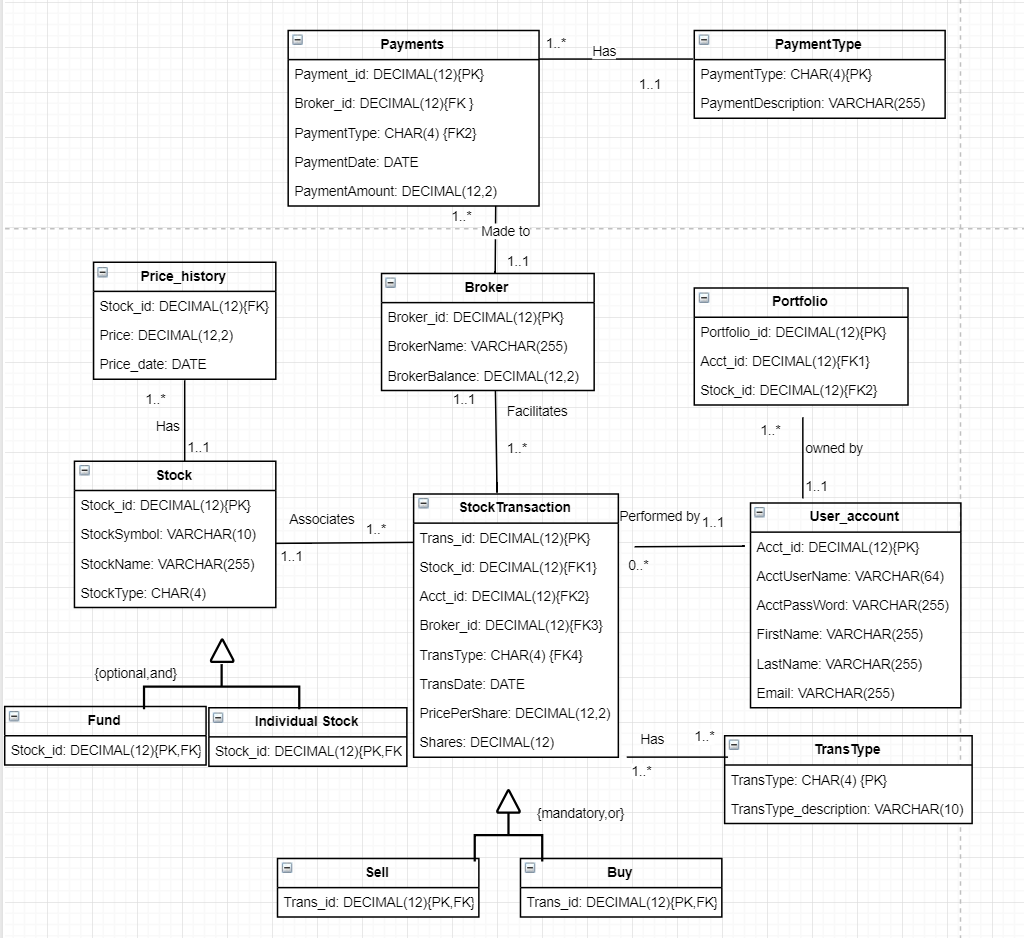
*Search price history of a stock*

1. The user enters the stock symbol
2. The user selects time period
3. The app pulls the prices in the time period selected by the user

**Structural database rules:**

1. Each transaction is performed by a user account; each user account may perform many transactions.
2. Each transaction is made to a stock; each stock has one to many transactions.
3. Each stock has one to many price history rows; each price history is associated to one stock.
4. Each broker facilitates one to many transactions; each transaction is facilitated by one broker.
5. Each portfolio holdings is owned by one user account; each user account owns zero to many holdings.
6. Each broker can receive one to several payments from user; each payment only goes to one broker.
7. A stock can be a fund, or an individual stock.
8. A transaction must be a sell or a buy.
9. A payment can be made via one to many payment types; a payment type is only associated with one payment.





**Task 2: Identifying Columns Needing Indexes for Database**

1. List of primary keys already indexed:

Payments.Payment\_id

PaymentTypes.PaymentType

Broker.Broker\_id

Portfolio.Portfolio\_id

Stock.Stock\_id

StockTransaction.Trans\_id

User\_account.Acct\_id

TransType.TransType

Fund.Stock\_id

Individual\_stock.Stock\_id

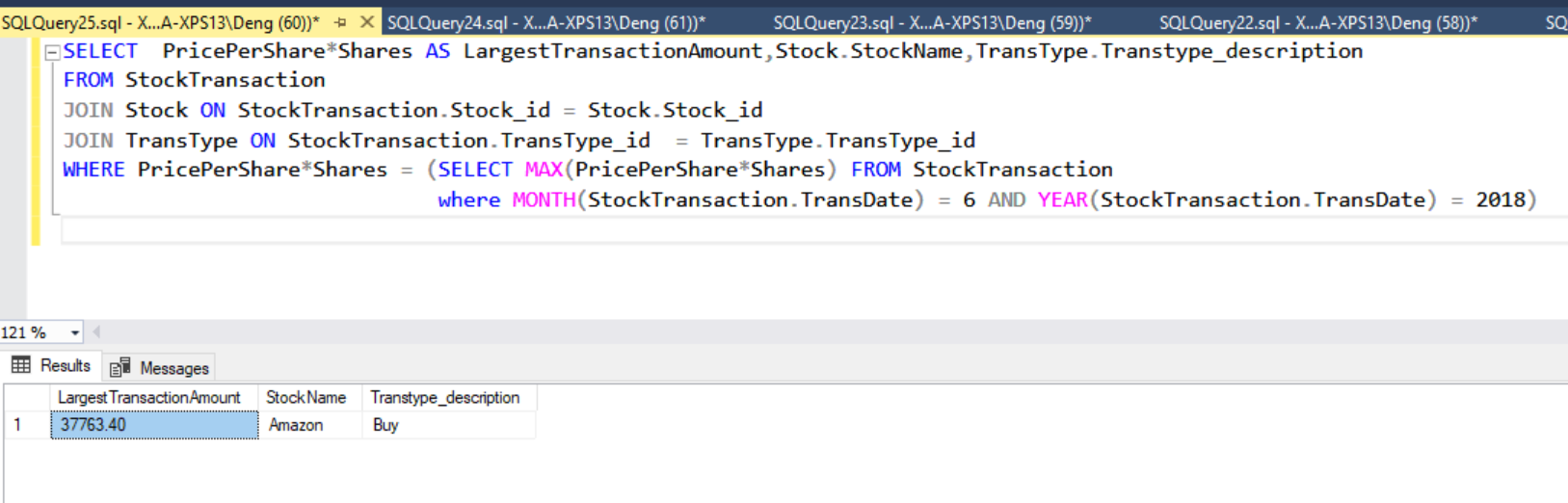
Sell.Trans\_id

Buy.Trans\_id

2. Identifying each foreign key column:

|  |  |  |
| --- | --- | --- |
| Column | Unique | Description |
| Payments.Broker\_id | Not unique | There can be many payments to the same broker |
| Payments.PaymentType | Not unique | There can be many payments with the same payment type |
| Price\_history.Stock\_id | Not unique | The same stock can have many rows of price history |
| Portfolio.Acct\_id | Not unique | The same account can have many portfolios |
| Portfolio.Stock\_id | Not unique | Many portfolios can have the same stock |
| StockTransaction.Stock\_id | Not unique | The same stock can be traded many times |
| StockTransaction.Acct\_id | Not unique | The same account can have many transactions |
| StockTransaction.Broker\_id | Not unique | There can be many transactions with the same broker |
| StockTransaction.TransType | Not unique | There can be many transactions with the same transaction type |

3. Three query driven indexes



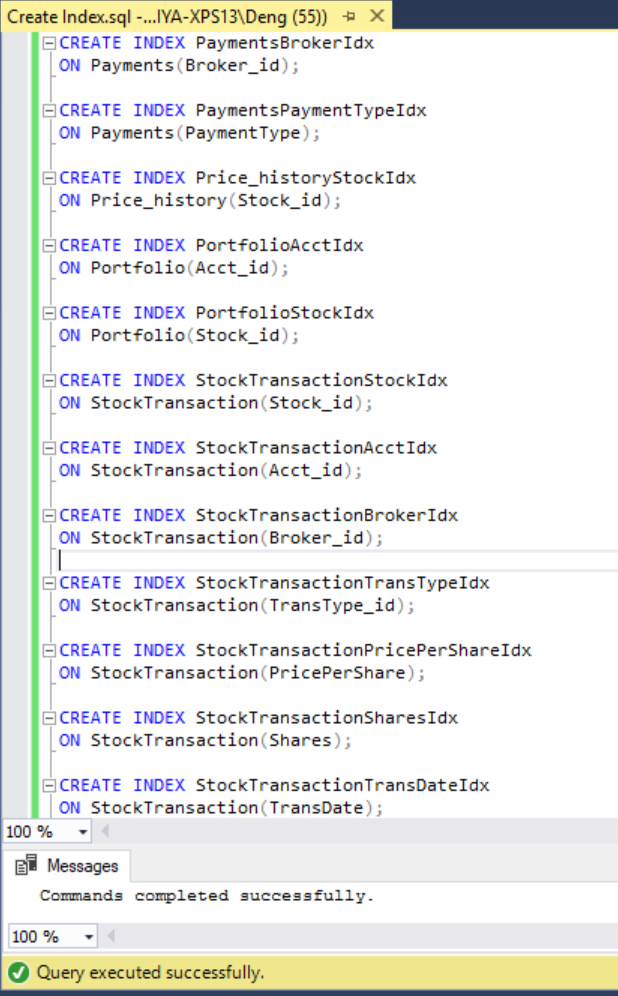
Reviewing the query question from Iteration 5:

‘What is the largest stock transaction in June? List the amount, stock name and transaction type.’

It’s clear that to support the WHERE clause, the followings should be indexed:

|  |  |  |
| --- | --- | --- |
| PricePerShare | Not Unique | Many stock transactions could have the same price per share |
| Shares | Not Unique | Many stock transactions could have the same amount of shares |
| TransDate | Not Unique | Many stock transactions could happen in the same day |

**Task 3: Creating Indexes in Database**

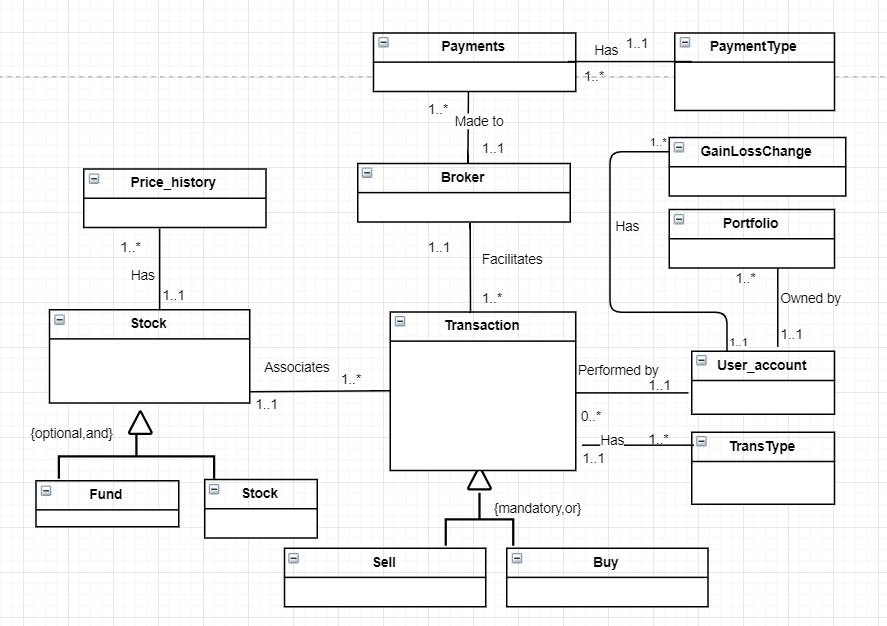


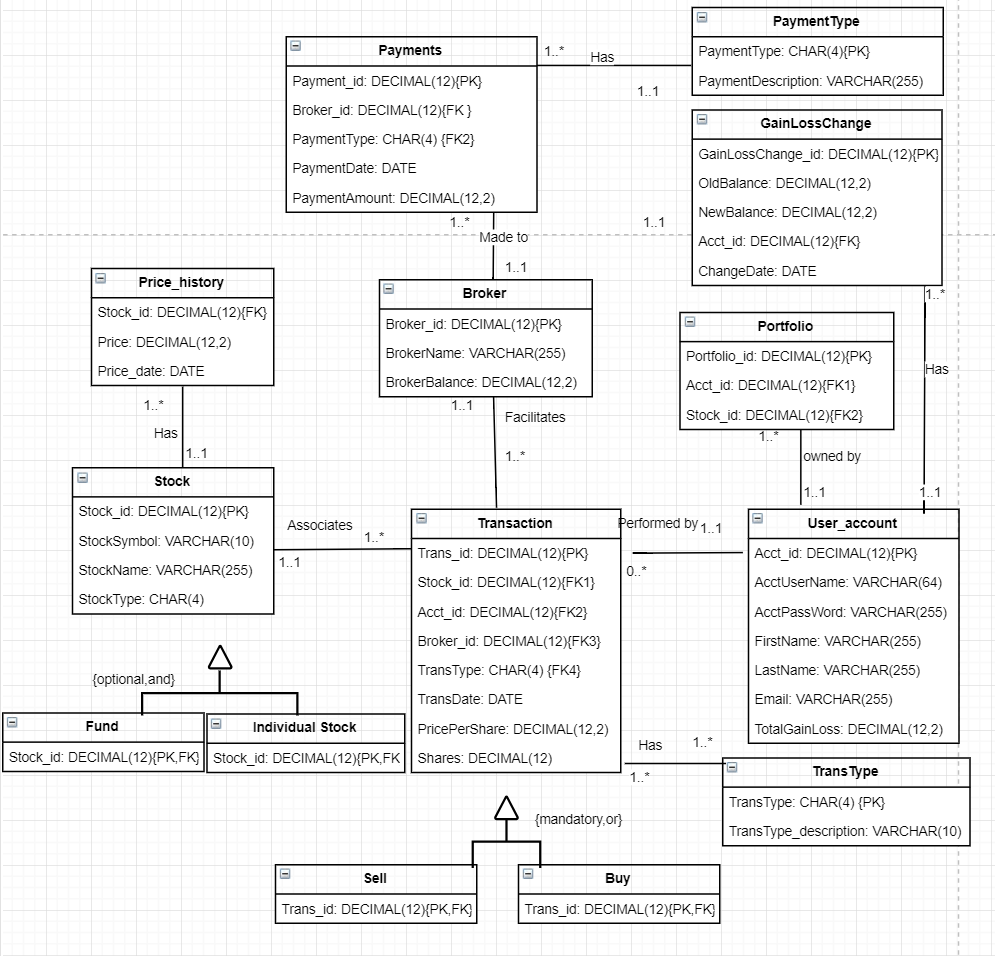
**Task 4: Capturing and Using History**

In order to measure the user’s trading performance, I would like to track the total gain and loss of a user account. Thus I have added a new attribute ‘TotalGainLoss’ in entity ‘User\_account’, and a new entity ‘GainLossChange’ related to ‘User\_account’ entity.

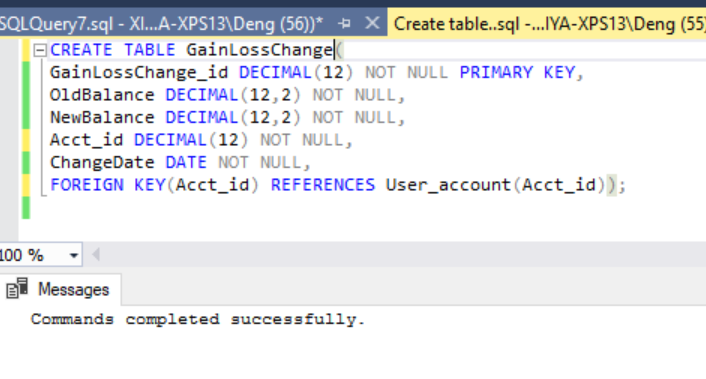
The new structural rule is: Each user account has many gain/loss changes, but each gain/loss change belongs to a user account.

The followings are the updated conceptual and physical ERD:

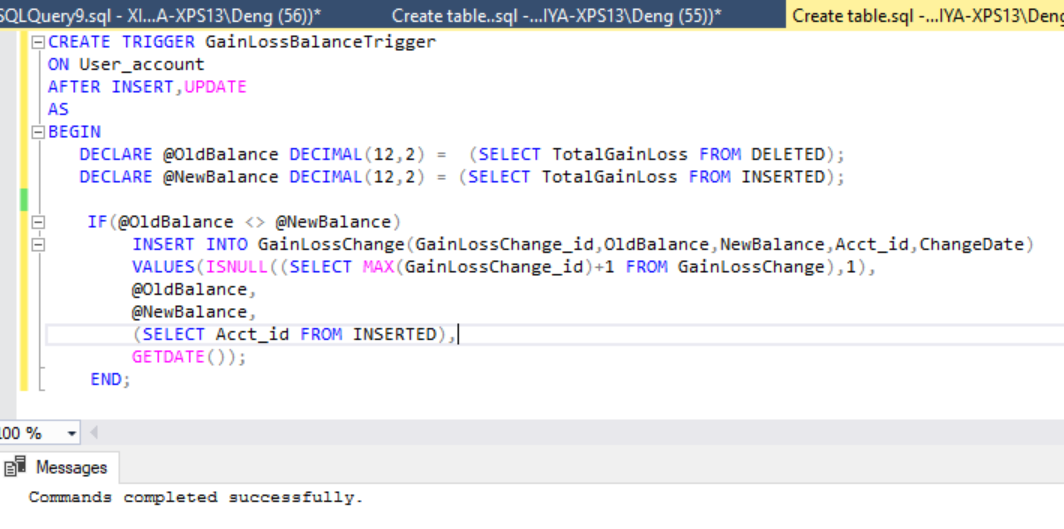




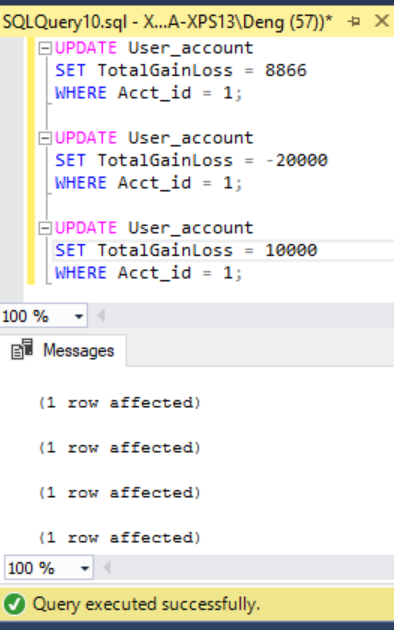
|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Data Type** |
| GainLossChange\_id | Primary key of the history table | DECIMAL(12) |
| OldBalance | Balance before the change | DECIMAL(12,2) |
| NewBalance | Balance after the change | DECIMAL(12,2) |
| Acct\_id | Foreign key referring to User\_account entity that has the change | DECIMAL(12) |
| ChangeDate | The date the change occurs | DATE |



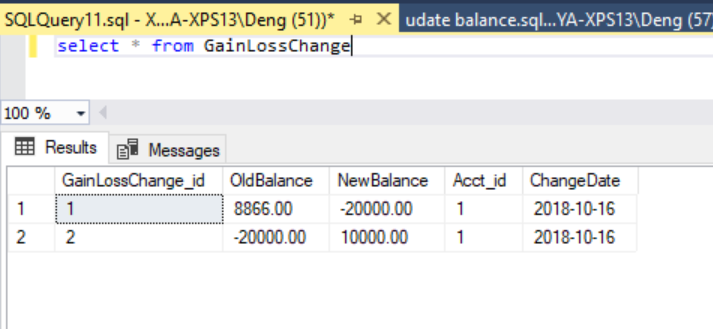
The following trigger will automatically insert values into GainLossChange table, if there is any update in the GainLossChange attribute of the User\_account table.



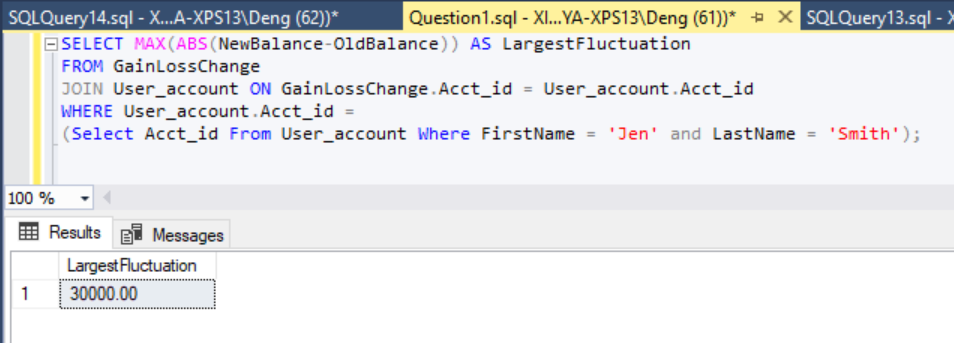
The TotalGainLoss in User\_account table has been updated several times:



The changes of TotalGainLoss has been verified:



Question: What is Jen Smith’s largest gain/loss fluctuation among all of her tradings?



**Task 5: Summarizing and Reflecting**

The most I like about database implementation is the logic in it. It is very interesting to learn how to correlate various entities and how to retrieve information. I wish I could add a lot more data into my database, and treat it like a real business tool. The data I added for the project purpose is more ideal, but in real life, I believe there will be a lot of error occurred or other design/implementation issues found in my database. I’m a Java programmer, so if I have more time and could do it over again, I would like to embed this database into my Java program, to make a real product.