Stock Broker DBMS Project

Deliverable 2

Thomas Ciha

Jacob Burke

Ethan Murphy

Question #1

Entities:

- Order
- Account
- Stock
- Option
- User

Relationships With Tables:

- ContainsStock
- ContainsOption

1)
Order(Order Number, **AssetClass, Ticker, Account Number, UserID,** OrderType, Term, Order Action, TimeStamp, Size, Status)

Integrity Constraints (Domain) for Order attributes:

- Order Number - Uniquely Identifies orders, **Domain: Positive Real Numbers**
- AssetClass - foreign key which specifies the asset class the order is for. Domain: Stock, Option
- Ticker - foreign key, indicative of the particular asset the order is for
- AccountNumber - foreign key which indicates which account placed the order. Domain: All possible combinations of 8-10 Positive Real Numbers.
- UserID - foreign key which specifies the user that placed the order
- Order Type - specifies parameters for the order's execution.
    - Domain: {Limit, Market, Market on Close, Stop on Quote, Stop Limit On Quote, Hidden Stop, Trailing Stop in $, Trailing stop in %}
- Term - timing parameters which specify instructions as to under what conditions the order should be filled or cancelled.
    - Domain: { AON (All or none), GTC (Good til canceled), IOC (Immediate or canceled), FOK (fill or kill), Day }
- Order Action - The "Order Action" specifies whether the order is a buy, sell, short or cover order for stocks.
    - Order Action (Options) {Buy to open, sell to open, buy to close, sell to close}

    - Order Action (Stocks) {Buy, Sell, Short, Cover}
    - I'm assuming we can enforce constraints on our system so that the set of feasible values for *OrderAction* is contingent on the AssetID.

- Timestamp - the date and time the order has been placed
- Qty- the quantity of the stock or option the order is for
- Desired Price - the price you would like to purchase the security at. This will have a NULL value for market orders. Domain: Positive Real Numbers.
- Status - the order's status. Domain: {Cancelled, Filled (Completed), Pending}

2)
Stock(Ticker, AssetClass, Price)

Integrity Constraints (Domain) for Stock attributes:

- Price: is a historical metric as it is the price the stock last traded for. In liquid markets this often accurately reflects the current price as many trades are completed every minute. **Domain: Positive Real Numbers**
- Ticker: 8 characters, distinguishes each stock from all other stocks
- Asset Class: The asset class of a stock. Domain: Fixed value of "Stock". This is needed because the ticker for Apple stock is the same as the ticker for an Apple option.

3)

Option(AssetClass, Ticker, Price, Option Type, Strike, Expiration Date)

Integrity Constraints (Domain) for Option attributes:

- Asset Class: Fixed value of "Option". Needed to distinguish the type of financial asset.
- Ticker: 8 characters, uniquely identifies each option
- Price: also referred to as the option's premium, this is the amount paid to the seller for ownership of an option. **Domain: Positive Real Numbers**
- Option Type = {Call, Put}
    - (call is the right to buy, put is the right to sell)
- Strike - this is the price at which the option can be exercised at. That is, it's the price the owner has the option to sell at if it's a put or the price the owner can buy at if it's a call. **Domain: Positive Real Numbers**

- Account number: All possible combinations of 8-10 positive real numbers
- Ticker: 8 characters, distinguishes each stock from all other stocks
- Purchase Price: Price at which asset is purchased; Any positive real numbers
- Quantity: The amount owned by a particular account; Any positive real numbers

ContainsOption(**AccountNum**, **Ticker**, Option Type, Purchase Price, Quantity, Strike, Expiration Date)

Integrity Constraints (Domain) for ContainsOption Attributes:

- Account number: All possible combinations of 8-10 **positive real numbers**
- Ticker: **Domain: 8 characters**, distinguishes each stock from all other stocks
- Option Type: Indicative of the option's type. **Domain: call / put**
- Purchase Price: Price at which asset is purchased; **Any positive real numbers**
- Quantity: The amount owned by a particular account; **Any positive real numbers**
- Strike - this is the price at which the option can be exercised at. That is, it's the price the owner has the option to sell at if it's a put or the price the owner can buy at if it's a call. **Domain: Positive Real Numbers**
- Expiration date: the final day for the option holder to exercise the option. **Domain: Any Future Date**

Question #2

There are not any foreseeable opportunities to combine relations because each entity has a distinct purpose. The Stock and Bond entities are the most similar of the entities in our model, however, these cannot be combined as they have different attributes.

1)

Original: Account(Account Number, **UserID**, Balance, Account Type, Account Status, Account Activity Status, Account Username, Account Password)

- No repeating groups
- Each intersection or "cell" of the relation contains exactly one value
- Every non-primary key is fully functionally dependent on the primary key
- No apparent transitive dependencies among attributes

Typical Database Queries:

- Getting account information to display current positions
- Changing Account Balance
- Changing "inventory" of accounts when a stock or option is bought or sold

2)

Original: User(UserID, **Account Number,** Name, Address, SIN, DriversLicenseMasterNo, Phone, City)

User(UserID**,** Name, Address, SIN, DriversLicenseMasterNo, Phone, City)

- Account Number is a repeating group because a user may have numerous accounts. We will determine which accounts correspond to a UserID by looking up the UserID attribute in the Account relation instead of creating a separate relation with just the primary key, UserID, and Account Number.
- Each intersection or "cell" of the relation contains exactly one value subsequent to the change above.
- Every non-primary key attribute is fully functionally dependent on the primary key
- No transitive dependencies discovered

Typical Database Queries:

- Accessing user account information
- Updating any changes in user information

3)

Original: Stock(Ticker, AssetClass, Price)

- No repeating groups
- Each column and row contains one value. However, the price will be updated periodically as the market value of the stock fluctuates.
- Assuming the price updates continuously with the market, the non-primary key attribute price will be fully functionally dependent on the primary key (Ticker, Asset Class). Both Ticker and Asset Class are necessary as prices will differ between options and stocks with the same ticker.

Typical Database Queries:

- Updating / retrieving latest stock price
- Looking up Ticker

4)

Original: Order(Order Number, **AssetClass, Ticker, Account Number, UserID,** OrderType, Term, Order Action, TimeStamp, Size)

Order(Order Number, **AssetClass, Ticker, Account number,** OrderType, Term, Order Action, TimeStamp, Size)

- We can eliminate the UserID foreign key from this relation because the account number functionally determines UserID.
- No repeating groups
- Each cell contains only one value
- Every non-primary key attribute is fully functionally dependent on the primary key Order Number in this relation. Although Asset Class functionally determines the domain of values for Order Action, it does not determine the value for Order Action.
- No transitive dependencies.

Typical Database Queries:

- Purchasing or selling a security
- Evaluating order type

5)

Original: Option(AssetClass, Ticker, Price, Option Type, Strike, Expiration Date)

- No repeating groups
- Each intersection only contains one value
- The primary key does not determine the Option Type as there are put and call options for the same ticker. However, there does not exist a subset of the primary key that functionally determines Option Type either.
- No transitive dependencies exist.

Typical Database Queries:

- Evaluating option Type
- Looking up Ticker
- Updating / retrieving option price

6)

ContainsOption(**AccountNum**, **Ticker**, Option Type, Purchase Price, Quantity, Strike, Expiration Date)

- No repeating groups
- Each intersection only contains one value, no transitive dependencies and

7)

ContainsStock(**AccountNum**, **Ticker**, Purchase Price, Quantity)

- We initially thought about creating a unified “contains” relationship table that would map every account to the stocks and options they own, however, we decided to diverge this into two separate table because a unified table would need an “Option Type” attribute. This would be feasible if we set the “Option Type” value to NULL for every stock in the table, but we feel splitting the tables provides a more elegant solution that will provide faster lookup times in the event of exclusively looking up an account’s stocks or options.
- No repeating groups
- Each intersection contains one value and no transitive dependencies

<u>Work Delegation:</u>

Thomas:

- Found relevant level 2 historical data for the project
- Worked on number 1:
    - Order
    - Account
    - Stock
    - Option
- Redeveloped normalization solutions for number 2
- Drew final ERD

Ethan:

- Did number 2 initially
- Listed typical database queries.

Jacob:

- Drew rough draft of revised ERD
- Outlined Relational schema - generated from converting revised ERD
- Worked on number 1:
    - User
    - Account

QUESTIONS:

- What is the best way to store historical time-series data for multiple stocks in a relational database? Should we create a relation that maps every stock to another table containing the respective time-series data?

Application Data (OPTION & STOCK DATA)

StockData(Ticker, Timestamp, Change, Shares, Price, MPID, MCID)

- THIS IS DATA THAT WILL BE DEALT WITH BY THE APPLICATION, NOT STORED ON THE DBMS. HOWEVER, HISTORICAL DATA WILL BE STORED ON THE DBMS IF WE ARE GOING TO ACCESS IT LATER.

Integrity Constraints (Domain) for **StockData** attributes:

Timestamp - Number of milliseconds after the midnight.

Ticker - Equity symbol (up to 8 characters)

Change - Message type. Allowed values:
"B" -- Add Bid -- Add Bid order
"S" -- Add Ask -- Add Ask order
"E" -- Execute Bid | Execute Ask -- Execute outstanding order in part "C" -- Cancel Bid | Cancel Ask -- Cancel outstanding order in part
"F" -- Fill Bid | Fill Ask -- Execute outstanding order in full
"D" -- Delete Ask | Delete Bid -- Delete outstanding order in full
"X" -- Bulk volume for the cross event
"T" – Trade Bid | Trade Ask -- Execute non-displayed order

Shares - Quantity of shares in order, available for the "B", "S", "E", "X", "C", "T" messages. Zero for "F" and "D" messages.

Price - Order price, available for the "B", "S", "X", and "T" order messages. Zero for cancellations and executions. For the binary version: The last 4 digits are decimal digits. The decimal portion is padded on the right with zeros. The decimal

point is implied by position; it does not appear inside the price field. Divide by 10000 to convert into currency value.

MPID - Market Participant ID associated with the transaction (4 characters).

MCID - Market Center Code (originating exchange – 1 character)

OptionData(Ticker, Time, Type, Info, PutCall, Expiration, Strike, Quantity, Premium, Exchange)

Integrity Constraints (Domain) for **OptionData** attributes:

- "Time" – milliseconds after midnight
- "Type" – event type (see below for more information)
  Domain:
  - "I" – National Best Trading Interest, "i" is a non-BBO Trading Interest
  - "r" – Rotation (not BBO-eligible)
  - "T" – Trade
  - "C" – Previous trade cancellation (reduce total traded volume by this amount). Trade corrections are constructed as a cancellation of the previous trade, with the corrected trade following the cancellation.
- "Info" – additional information on the trade/quote transaction. **Domain:**
- "Ticker – equity symbol or index (up to 8 characters)
- "PutCall" – PUT ("P") or CALL ("C") option type
- "Expiration" – Expiration Date in YYYYMMDD format
- "Strike" – strike price (4 implicit decimal digits, divide by 10,000) **Domain: Positive Real Numbers**
- "Quantity" – number of contracts. **Domain: Positive Real Numbers**
- "Premium" – premium price (4 implicit decimal digits, divide by 10,000)
- "Exchange" – marketplace ID where the event originated from
  - Domain:
    - A – NYSE AMEX
    - B – NASDAQ OMX BX
    - C – National Stock Exchange

- D – FINRA
- I – International Securities Exchange
- J – Direct Edge A
- K – Direct Edge X
- M – Chicago Stock Exchange
- N – New York Stock Exchange
- O–OPRA
- P – NYSE Arca
- TorQ–NASDAQOMX
- W – CBOE Stock Exchange
- X – NASDAQ OMX PSX
- Y – BATS Y-Exchange Inc.
- Z – BATS Exchange, Inc.
- 2 - C2 Exchange
- H - Boston Exchange
- G - Miami Stock Exchange

DO NOT PRINT

- Addressing how to keep track of which stocks and options each account currently owns.
    - Method 1: create a table with columns Quantity, Account Number, Option / Stock. When account XYZ acquires N units of option or stock ABC, this information is compiled into a new row of the table. Then, to determine all of the stocks or options that account XYZ owns, we would have to parse the entire table (which would include every asset (stocks/options) owned by every account in the system) looking for the account number XYZ.

    - Method 2: we create two attributes of account called StockTableID and OptionTableID. These IDs are associated with a table of stocks and options that each account currently owns. Then when we want to lookup which stocks account XYZ owns, we use the XYZ's StockTableID to go to the table of stocks. This circumvents having to search through every account's assets.

Future(Ticker, Unit of Measurement, Settlement Type, Quantity, Quote Price, Denominated Price, Quality Specification)

Integrity Constraints for **Futures**:

- Quantity: **Domain: Positive Real Numbers**
- Quote Price:
- Denominated Price:
- Settlement Type: how the contract will be settled. **Domain = {Delivery, Cash}**
- Quality Specification: This attribute specifies the quality of the commodity in the contract (e.g. the grade of gasoline). Hence, there are a myriad of feasible values for this attribute, including NULL if its not applicable (e.g. no quality specification with bitcoin futures).

Note: FIX is the protocol used in the real world to transmit orders

Links:
Provides a good walkthrough about how hidden orders are handled:
https://www.quora.com/In-stock-trading-investing-how-do-hidden-orders-work

Option order Types:
https://optionalpha.com/mastering-the-4-different-types-of-option-orders-15097.html

DATA SOURCE:

https://www.algoseek.com/samples/

- Information regarding various order types, Option Order Types

ContainsStock(**AccountNum**, **Ticker**, PurchasePrice, Quantity, AssetClass)

ContainsOption(**AccountNum**, **Ticker**, Option Type, Purchase Price, Quantity, Strike, Expiration Date, AssetClass)

OPTN(AssetClass, Ticker, Price, Option Type, Strike, Expiration Date)

Account(Account NO, **UserID**, Balance, Account Type, Account Status, ActivityStatus, AccountUsername, AccountPassword)

UserTable(UserID, **Account Number,** FName, LName, Address, SIN, Phone, City)

StockOrder(Order Number, **AssetClass, Ticker, Account number, UserID,** OrderType, Term, Order Action, TimeStamp, SOSize, Status, Price)

- We might be able to eliminate user ID from stock order and option order

Optionorder(Order Number, **AssetClass, Ticker, Account number, UserID,** OrderType, Term, Order Action, TimeStamp, OOSize, Status, Price, ExpDate, Strike)

StockTable(Ticker, AssetClass, Price)