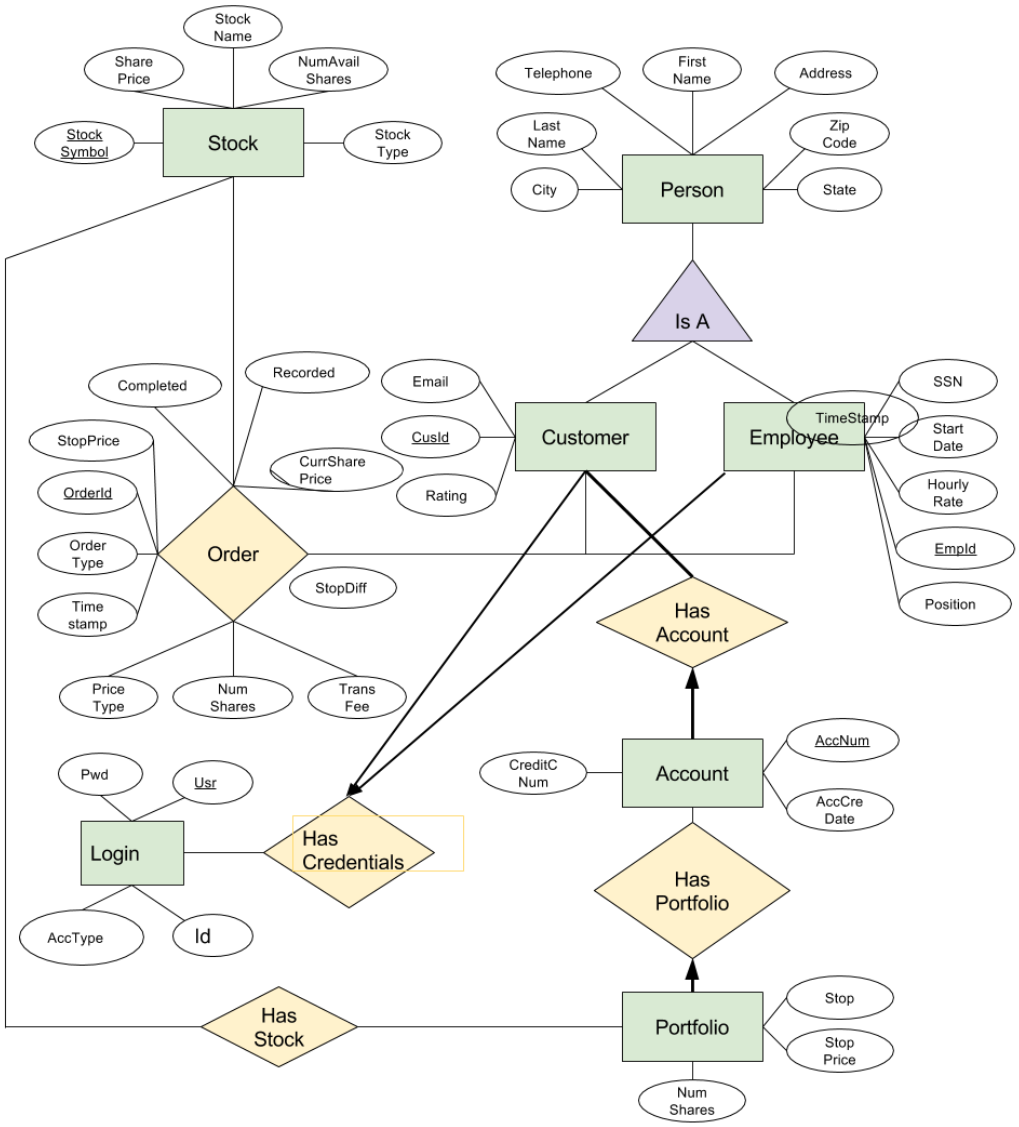
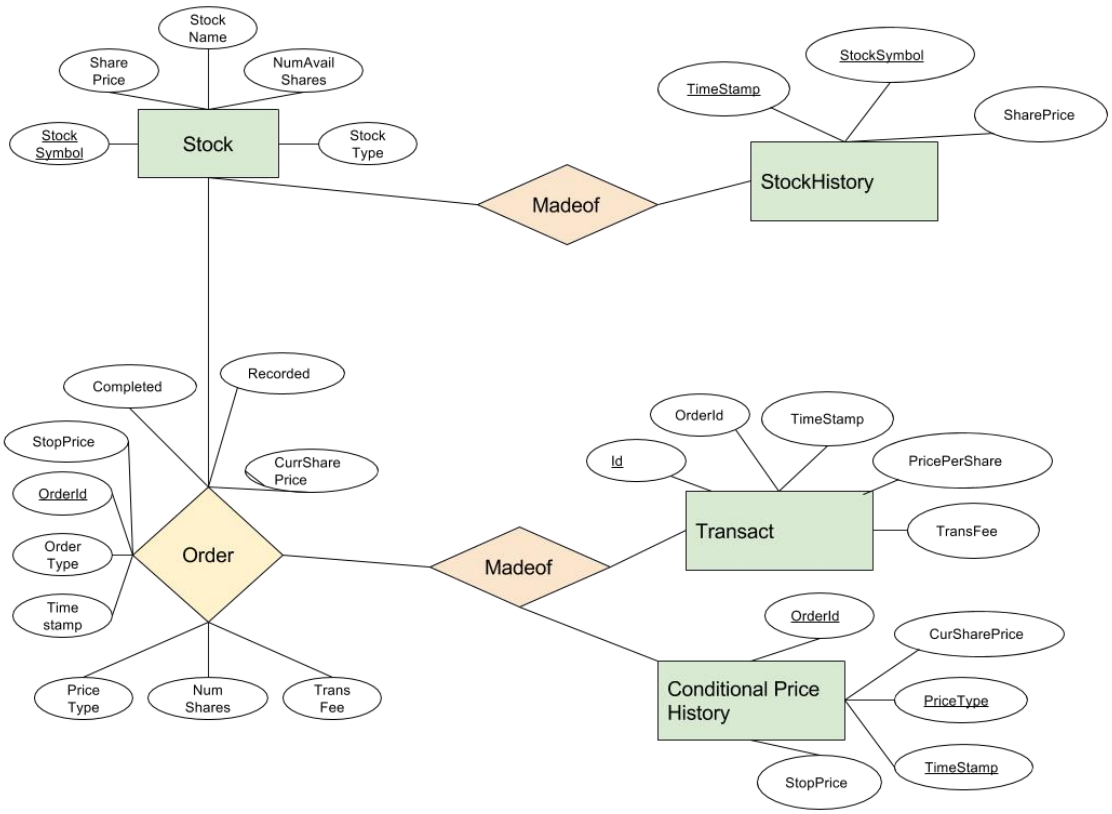
LIVESTOCK

CSE 305: Final

Wendy Zheng: wendy.zheng@stonybrook.edu
Ryan Goepfert: ryan.goepfert@stonybrook.edu
Yuliya Astapova: yuliya.astapova@stonybrook.edu

E-R Diagrams


Stock Name
Share Price
NumAvail Shares
Stock Symbol
Stock
Stock Type
Telephone
First Name
Address
Last Name
Zip Code
City
Person
State
Is A
Completed
Recorded
Email
SSN
TimeStamp
Start Date
StopPrice
CusId
Customer
Employee
CurrShare Price
Rating
Hourly Rate
OrderId
Order Type
Order
EmpId
Time stamp
StopDiff
Has Account
Position
Price Type
Num Shares
Trans Fee
AccNum
CreditC Num
Account
Pwd
Usr
AccCre Date
Has Credentials
Login
Has Portfolio
AccType
Id
Stop
Has Stock
Portfolio
Stop Price
Num Shares



E-R Diagram Rationale

We created an entity titled Person, which is a superclass of both Customer and Employee. Person contains the attributes common to both of these classes, helping to prevent the duplication of data. Customer and Employee have attributes unique to themselves. CusId and EmpId are the primary keys to their respective entities, as each customer and employee have a unique identification number.

A customer can have any number of accounts, but must have at least one, while each account must be linked to only one customer. Therefore, Customer to HasAccount has a bold line indicating a participation constraint to Account, while Account to HasAccount has a bolded arrow indicating a participation and key constraint to Customer.

Each account is represented by a unique account number, which is the primary key. Each account has a stock portfolio, so we created an entity Portfolio that has stocks. An account with no stocks does not have a stock portfolio, so there are no constraints. A portfolio is linked to

exactly one account, so there is a participation and key constraint from Portfolio to Account, indicated by a bolded arrow to HasPortfolio.

Portfolio may contain any number of stocks or none at all, so there are no constraints from Portfolio to entity Stock. A stock can belong to any number of portfolios or none at all, so there are no constraints from Stock to Portfolio. Each stock has a unique stock symbol, which is Stock's primary key.

Both customers and employees can place an order for a stock, but may also never place one. We created relationship Order to represent the customers and employees making orders for stocks. Each Order has a unique Id number, so OrderId is its primary key.

Customer and Employees have exactly one set of credentials, as denoted by the bolded participation constraint arrow. These credentials consists of the username, password, the id, and the Account type. The username is the primary key of Login because Employees and Customers must have unique usernames.

Entities Transact and ConditionalPriceHistory can be made up of any number of Orders. They have referential integrity constraints that are foreign keys because OrderId is the primary key of Order. Transact has a primary key of Id because all transactions are unique. On the other hand, Conditional Price History has primary keys of OrderId, PriceType, and Timestamp because unique entries are denoted by the order id, its price type, whether trailing stop or hidden stop, and the unique time at which the Order was recorded into the table.

StockHistory is an entity that is made of Stocks. An entity of StockHistory references a unique StockSymbol, hence it is part of the primary key, and has attributes timestamp and Share price, which keep track of when the stock price changed and what the new price is. StockHistory can be made of any number of Stocks.

Relational Model

```
CREATE TABLE Stock (
        StockSymbol         VARCHAR(5) NOT NULL,
        StockName           VARCHAR(20) NOT NULL,
        StockType           VARCHAR(20),
        SharePrice          FLOAT(2) NOT NULL,
        NumAvailShares      INTEGER NOT NULL,
        PRIMARY KEY         (StockSymbol),
        UNIQUE              (StockName)
);

CREATE TABLE Employee (
        SSN                 CHAR(9) NOT NULL,
        LastName            VARCHAR(20),
        FirstName           VARCHAR(20),
        Address             VARCHAR(50),
        City                VARCHAR(20),
        State               VARCHAR(20),
        ZipCode             CHAR(5),
        Telephone           CHAR(10),
        StartDate           DATETIME,
```

```
        HourlyRate          FLOAT(2),
        EmpId               INTEGER AUTO_INCREMENT NOT NULL,

        Position_           VARCHAR(7) NOT NULL,
        PRIMARY KEY         (EmpId),
        UNIQUE              (SSN)

);

CREATE TABLE Customer (
        LastName            VARCHAR(20) NOT NULL,
        FirstName           VARCHAR(20) NOT NULL,
        Address             VARCHAR(50),
        City                VARCHAR(20),
        State               VARCHAR(20),
        ZipCode             CHAR(5),
        Telephone           CHAR(10),
        Email               VARCHAR(50),
        Rating              INTEGER NOT NULL,
        CusId               INTEGER AUTO_INCREMENT NOT NULL,
        PRIMARY KEY (CusId)

);

CREATE TABLE Login (
        Usr                 VARCHAR(20) NOT NULL,
        Pwd                 VARCHAR(20) NOT NULL,
        AccType             INTEGER NOT NULL,
        Id                  INTEGER NOT NULL,
        PRIMARY KEY         (Usr)

);



CREATE TABLE Account (
        AccNum              INTEGER AUTO_INCREMENT NOT NULL,
        AccCreDate          DATETIME,
        CreditCNum          VARCHAR(16) NOT NULL,
        CusId               INTEGER NOT NULL,
        PRIMARY KEY         (AccNum),
        FOREIGN KEY         (CusId) REFERENCES Customer (CusId)
                ON DELETE NO ACTION
                ON UPDATE CASCADE

);
```

```
CREATE TABLE Order (
        OrderId                 INTEGER AUTO_INCREMENT,
        StockSymbol             VARCHAR(5),
        OrderType               VARCHAR(4) NOT NULL,
        NumShares               INTEGER NOT NULL,
        CusAccNum               INTEGER DEFAULT 0,
        Timestamp_              DATETIME DEFAULT NOW() NOT NULL,
        PriceType               VARCHAR(15) NOT NULL,
        StopPrice               FLOAT(2) DEFAULT 0,
        StopDiff                FLOAT(2),
        CurSharePrice           FLOAT(2),
        EmpId                   INTEGER DEFAULT 0,
        Recorded                BOOLEAN DEFAULT 0,
        Completed               BOOLEAN DEFAULT 0,
        PRIMARY KEY             (OrderId),
        UNIQUE                  (StockSymbol, Timestamp, CusAccNum, EmpId),
        FOREIGN KEY             (StockSymbol) REFERENCES Stock (StockSymbol)
                ON DELETE SET NULL
                ON UPDATE CASCADE,
        FOREIGN KEY             (CusAccNum) REFERENCES Account (AccNum)
                ON DELETE SET NULL
                ON UPDATE CASCADE,
        FOREIGN KEY             (EmpId) REFERENCES Employee (EmpId)
                ON DELETE SET NULL
                ON UPDATE CASCADE

);

CREATE TABLE Transact (
        Id                      INTEGER AUTO_INCREMENT,
        OrderId                 INTEGER,
        TransFee                FLOAT(2),
        TimeStamp_              DATETIME DEFAULT NOW() NOT NULL,
        PricePerShare           FLOAT(2),
        PRIMARY KEY             (Id),
        FOREIGN KEY             (OrderId) REFERENCES Order (OrderId)
                ON DELETE SET NULL
                ON UPDATE CASCADE

 );

CREATE TABLE Portfolio (
        AccNum                  INTEGER,
        StockSymbol             CHAR(5),
```

```
        NumShares       INTEGER,
        Stop_           VARCHAR(8) NOT NULL,
        StopPrice       FLOAT(2),
        PRIMARY KEY     (AccNum, StockSymbol),
        FOREIGN KEY     AccNum) REFERENCES Account_ (AccNum)
                ON DELETE NO ACTION
                ON UPDATE CASCADE,
        FOREIGN KEY     (StockSymbol) REFERENCES Stock (StockSymbol)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);

CREATE TABLE ConditionalPriceHistory (
        OrderId         INTEGER,
        CurSharePrice   FLOAT(2),
        PriceType       VARCHAR(15) NOT NULL,
        StopPrice       FLOAT(2),
        Timestamp_      DATETIME DEFAULT NOW(),
        PRIMARY KEY     (OrderId, PriceType, Timestamp_),
        FOREIGN KEY     (OrderId) REFERENCES Order_ (OrderId)
                ON DELETE CASCADE
                ON UPDATE CASCADE

);

CREATE TABLE StockPriceHistory (
        StockSymbol     VARCHAR(5),
        SharePrice      FLOAT(2),
        Timestamp_      DATETIME DEFAULT NOW(),
        PRIMARY KEY(StockSymbol, Timestamp_),
        FOREIGN KEY(StockSymbol) REFERENCES Stock (StockSymbol)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Relational Model Rationale

In our relational model, we decided to create a total of ten tables: Order, Stock, Customer, Employee, Account, and Portfolio, StockPriceHistory, ConditonalPriceHistory, Transact and Login. The primary key of Order is the OrderId, which we set to auto-increment each time an order is created. One candidate key of Order is the tuple containing the attributes StockSymbol, Timestamp, CusAccNum, and EmpId, because it would not make sense for someone to order the same stock multiple times, but at the exact same time. We set the default value of EmpId to 0 to account for the cases in which an employee is not part of the transaction. The foreign keys of Order are StockSymbol (from Stock), CusAccNum (AccNum from Account), and EmpId (from Employee). We chose the trigger in StockSymbol to be ON

DELETE SET NULL, because setting it to a default would not work reliably, since a new stock may accidentally have the default name, and no action would not work because a Stock must be able to be deleted in the case that a company goes private or bankrupt. We chose ON DELETE SET DEFAULT for CusAccNum instead of no action because since customers can close accounts, accounts must therefore be deletable, but set default allows us to easily refer to all closed accounts. We chose ON DELETE SET NULL for EmpId because no action would not work in case an employee leaves the company and must be deleted, and set default would not work because we want to differentiate between when an Employee is deleted and when they are not present for a transaction. The OrderType attribute has a domain associated with it, which only includes "Buy" and "Sell" as values. PriceTypes has a domain with four possible price types in it. We have also created an assertion to ensure that the NumShares of an Order does not exceed the number of available shares for that Stock.

In the Stock table, we have StockSymbol as the primary key and StockName as the candidate key. We chose this candidate key because a company cannot have two different stocks. Also, we have decided that the number of shares for a stock should refer to the number of shares that are currently available for purchase, reflected in the NumAvailShares attribute.

The Customer table is very straightforward; it includes all of a customer's information, along with an auto-incrementing CusId as the primary key. We did not include a candidate key because customers can have the same first and last name or the same contact information, but still be different people. The domain included for the Rating simply ensures that it is a number between 1 and 10.

The Employee table is also straightforward; it includes all of an employee's information, with an auto-incrementing EmpId as the primary key. We have also made SSN a candidate key, because no two people can have the same social security number. We did not create other candidate keys, because people can have the same names but be different people, as with Customer. For both Employee and Customer, we assume that all users have a United States format phone number.

Our Portfolio table represents the ownership of shares of a certain Stock by a certain customer's Account. The primary key of Portfolio is the tuple of AccNum and StockSymbol, because each account only has one possible entry for each Stock, though the number of shares it owns can change. AccNum and StockSymbol are also both foreign keys. We chose the trigger ON DELETE NO ACTION for AccNum, because we have decided that an Account cannot be deleted if it still has Portfolios attached to it. For StockSymbol, we chose the trigger ON DELETE CASCADE because if a Stock is taken off the market, that should be reflected in Portfolios. We have also included a domain for Stop, which includes "Trailing" and "Hidden" as values. We created a trigger, DeleteOnZeroShares, for the Portfolio table to delete a portfolio when the number of shares it contains hits zero.

Our Account table, which includes CusId as a foreign key in order to refer to the customer that owns the account. We chose ON DELETE NO ACTION for this foreign key because we decided that a Customer cannot be deleted if he/she still has open accounts, because this means that he/she still has stocks in the portfolios associated with it. The primary key of Account is the auto-incrementing AccNum, which counts all accounts in our database, not just those of a particular Customer.

Our ConditionalPriceHistory table is composed of orders that are not either trailing stops or hidden stops. It has primary keys OrderId, PriceType, and TimeStamp. It has a foreign key OrderId because all entries in ConditionalPriceHistory are orders. The built in trigger for this

foreign key is ON DELETE CASCADE and ON UPDATE CASCADE because the entries in ConditonalPriceHistory cannot exist without the base order. Therefore if the base order that it is referencing is deleted or updated, then the entry in ConditionalPriceHistory should follow the same route.

Our StockPriceHistory table is composed of stocks that are currently on the market. It has primary keys of Stock Symbol and Timestamp, where Stock Symbol is also a foreign key that references the Stock table. The time stamp attribute is need to make an entry in Stock Price History unique because entries may have the same stock symbol but not both the same stock symbol and timestamp. The timestamp field is used to keep track of the date and time the stock price for a particular stock was updated or set. Similar to the reasoning given for the built in trigger given for Conditional Price History, once a stock is deleted, then there is no need to keep an entry in the StockPriceHistory table because the stock is no longer available anyway.

Our Tranasct table keeps track of Orders that have been recorded and completed. In consequence, it has a foreign key constraint on OrderId even though its primary key is Id. We decided not to make OrderId the primary key of Transact because of the auto increment feature. Our Transact entries have TimeStamps, and by sorting the Ids in ascending order, we can get sorted times as well whereas such a sort on OrderId would not work.

Finally, our Login table holds all the credentials of the Customers and Employees of LiveStock. Since all the users have a unique username, the primary key of the Login table is Usr. It does not have any integrity constraints because usernames and passwords are independent of whether the user is a customer or employee.

Throughout our relational model, we have made all CHAR attributes into VARCHARs because they preserve space when not filled to capacity. Also, in many cases, we chose VARCHAR rather than INTEGER for an attribute, such as with credit card numbers, IDs, and phone numbers, because they need to have a certain number of digits. In those cases, it is our intent to check whether or not correct values are entered through the form in which users add data. We also choose to use BOOLEAN for columns such as Record and Completed for efficiency because for such columns, only two values are allowed, 0 and 1.

Functional Dependencies

Account

- AccNum → AccCreDate, CreditNum, CusId

ConditionalPriceHistory

- OrderId, PriceType, TimeStamp → CurSharePrice, StopPrice

Customer

- CusId → LastName, FirstName, Address, City, State, ZipCode, Telephone, Email, Rating, Email

Employee

- EmpId → SSN, LastName, FirstName, Address, City, State, ZipCode, Telephone, Start Date, HourlyRate, Position
- SSN → LastName, FirstName, Address, City, State, ZipCode, Telephone, StartDate, HourlyRate, Position, EmpId

Login

- User → Pwd, AccType, Id

Order

- OrderId → StockSymbol, OrderType, NumShares, CusAccNum, TimeStamp, PriceType, StopPrice, StopDiff, CurSharePrice, EmpId, Recorded, Completed
- StockSymbol, TimeStamp, CusAccNum, EmpId → OrderType, NumShares, PriceType, StopPrice, StopDiff, CurSharePrice, Recorded, Completed

Portfolio

- AccNum, StockSymbol → NumShares, Stop, StopPrice

Stock

- StockSymbol → StockName, StockType, SharePrice, NumAvailShares
- StockName → StockType, SharePrice, NumAvailShares

StockPriceHistory

- StockSymbol, TimeStamp → SharePrice

Transact

- Id → OrderId, Transfee, TimeStamp