Moogle

The Evil Geniuses

Ryan Despres, John Manley, Angel Mendoza,
Michael Onjack, Dennis Samuel

Phase 2

3-4-2016

Table of Contents

1.1 Introduction

Moogle is a startup company that wishes to pursue the exploding opportunities of online business. In an attempt to stand out from the multitude of other online retailers, Moogle hopes to draw from the successes of their competition by combining the unique aspects of these other e-businesses into a single full-fledged site. They have recognized Amazon.com as the primary market leader and have also noticed that eBay.com has found its own immense success through more of an auction-styled online buying experience. Therefore, Moogle has decided to combine the traditional style that is provided by Amazon with the bidding system of eBay to create a new hybrid that hopes to grab market shares from the current industry giants. Unfortunately, such a site cannot be built overnight; it takes careful preparation, planning, and execution and even then uncommon challenges will be faced along the way. However, if the proper precautions are taken, the rewards can be tremendous. Since the owners of Moogle are not technically inclined, they have contracted our group to prototype and validate these business aspirations. Therefore it is our duty to make Moogle a reality.

In the proceeding sections, we discuss the different features that Moogle must have in order to challenge the likes of Amazon. Some of these features include actions (such as searching and bidding) while others detail objects (such as users and sale items). As we explore each of these features, we determine what pieces (or attributes) they are comprised of as well as how how some may be related to others and what these relations could mean for the application. Once the essential features are discussed, we move on to two additional features that we feel are important for the success of Moogle, specifically watch lists and a “Herd Membership” which serves a similar purpose to Amazon’s Prime program. In the final section, we conclude our thoughts and summarize the format and structure of our report.

1.2 List of Figures

2.1 Sale Item

An item is sold either by listed price or by auction. The sources of the items may be a company (i.e., a supplier) or an individual (i.e., an online seller). A unique identifier is assigned to an item when it is in stock or put up for auction. A short description is associated with an item (provided by a supplier or a seller). The online seller may specify a reserve price (which is hidden from the buyers) for an item he posted for auction. A reserve price is the minimum price a seller is willing to accept for the item. At the end of the auction, if no bid is higher than the reserve price, the item will not be sold. The seller must also specify the location of the item.

Item (see Figure 1 on following page) is an entity with the attributes:

- *IID* – This is the item ID, which is used to distinguish between every item in the store
- *Quantity* – The number of items remaining
- *Category* – Which department the item falls under
- *Description* – A brief description of the item being sold

We have an ISA relationship with *Sale_Item* and *Auction_Item*.
Sale_Item is an entity with the attributes:

- *Price* – This is the fixed price of the item being sold

Auction_Item is an entity with the attributes:

- *Min Price* – The minimum bid the seller is willing to take on the item. if the bidder does not reach this price, then the item will not be sold
- *Current Bid* – The current bid placed on the item by the buyer
- *End date* – The date set by the seller for the completion of the auction for the item.

2.1 Continued

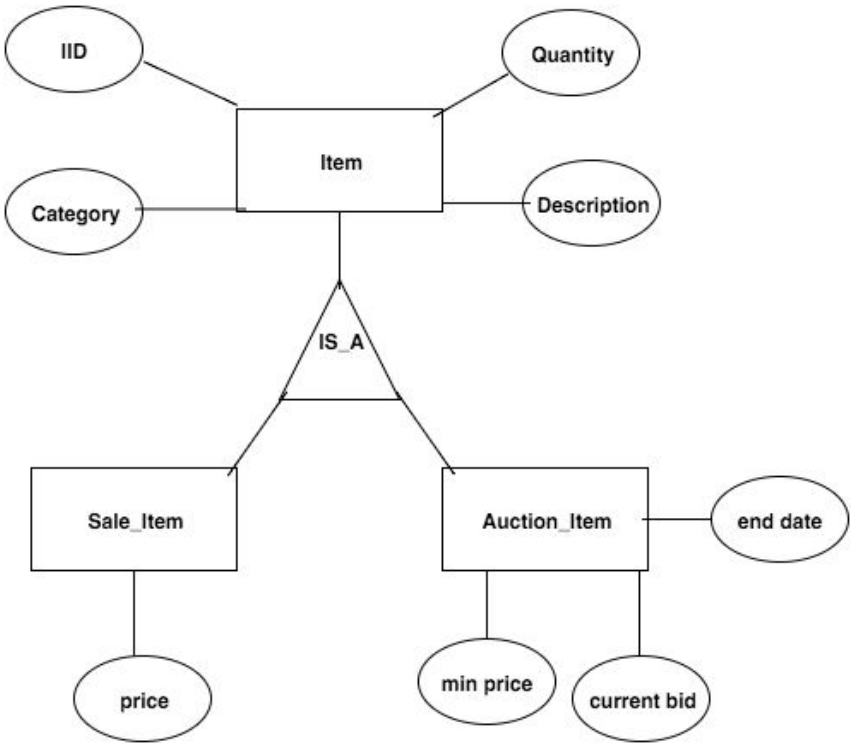

Figure 1: ER Model for Sale Item

We created 3 separate tables for this ISA relationship: Items, Sale_Items, and Auction_Items. We made a lot of the fields NOT NULL because we wanted exact numbers and values for items, prices, and times. The whole auctioning process is very dependent on the time and prices be- cause they can change all the time.

2.1 Continued

```
CREATE TABLE Items (
       IID CHAR(40),
       quantity INT NOT NULL,
       category CHAR(30),
       description TEXT,
       PRIMARY KEY (IID),
       FOREIGN KEY (category) REFERENCES Categories (name)
 );

CREATE TABLE Sale_Items (
         IID CHAR(40),
         price FLOAT NOT NULL,
         PRIMARY KEY (IID),
         FOREIGN KEY (IID) REFERENCES Items (IID)
                      ON DELETE CASCADE
);

CREATE TABLE Auction_Items (
         IID CHAR(40),
         end_date DATE NOT NULL,
         min_price FLOAT NOT NULL,
         current_bid FLOAT,
         PRIMARY KEY (IID),
         FOREIGN KEY (IID) REFERENCES Items (IID)
                      ON DELETE CASCADE
);
```

Figure 2: SQL for Sale Item Table

2.1 Continued

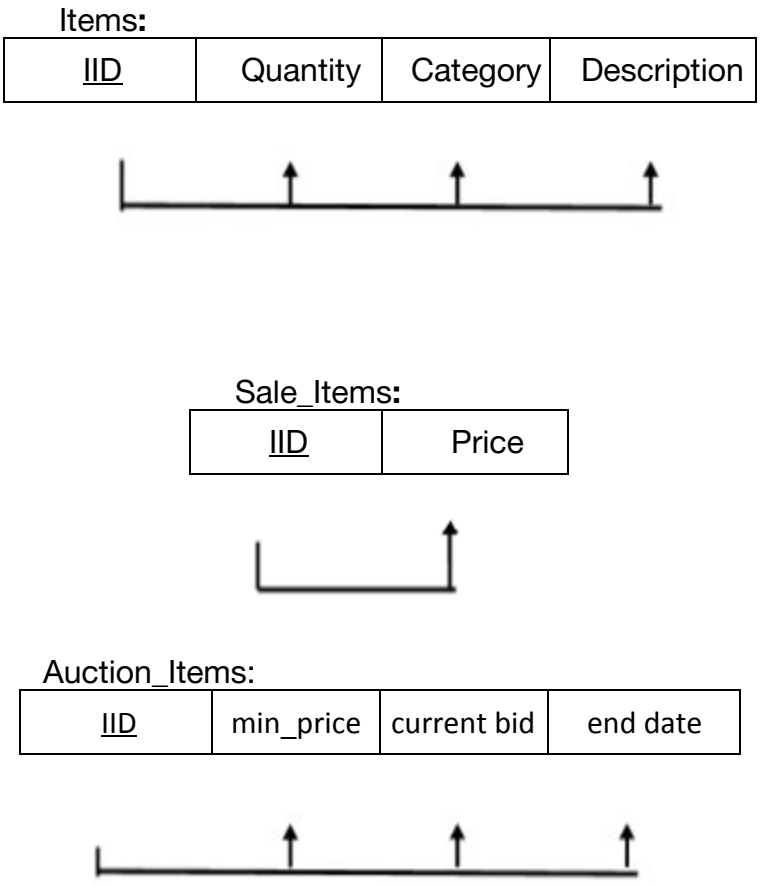

Figure 3: Dependency Diagram for Sale Item

2.2 Categories

In order to stay competitive with the other online retailers in the world today, Moogle will need to have a massive catalog of products for sale. However, users can easily become overwhelmed when faced with these products without any kind of organization or structure. To avoid this, we introduce categories. With categories, Moogle can better organize every sale item in the database by having each item belong to exactly one category. To add further detail, categories can have “subcategories”

2.2 Continued

which present a more specific way to view a subset of items that are in the parent category and are able to refine a user's search with even more accuracy. For example,

if we have a category "Movies" some of its subcategories may be "DVD", "Blu-ray", or "Digital". All categories will be stored under the "All" root category and a user can traverse through the category tree in order to more easily find a sale item they are looking for.

The next concern is: how exactly should we model categories in our database? A category will need to have a *Name* (which describes the type of items that belong to that category), a unique identifier (*CategoryID*), and a field for its parent's identifier (so subcategories can know who their parent is in the tree). Therefore, we model the categories as an entity set with attributes for each of these. We also know that every product will belong to exactly one category. Therefore we can set up a many-to-one relationship set between *Sale Item* and *Category* entities called "of_type" which features total participation from Sale Items entity set (because every Sale Item belong to exactly one category). See Figure 2 below.

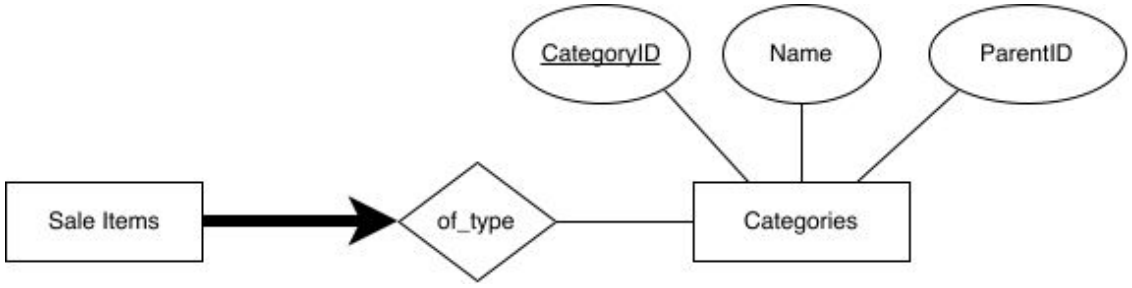

Figure 4: ER Model for Categories

2.2 Continued

```
     CREATE TABLE Categories (
          name CHAR(30),
          parent CHAR(30),
          PRIMARY KEY (name),
          FOREIGN KEY (parent) REFERENCES Categories (name)
     );

     CREATE TABLE Supplier_Category (
          supplier CHAR(20),
          category CHAR(30),
          PRIMARY KEY (supplier, category),
          FOREIGN KEY (supplier) REFERENCES Suppliers
               (supplierid)
               ON DELETE CASCADE,
          FOREIGN KEY (category) REFERENCES Categories (name)
               ON DELETE CASCADE
);
```

Figure 5: SQL for Categories

Categories:

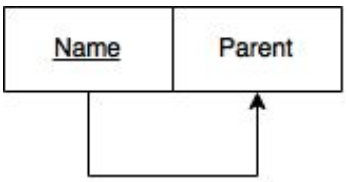

Figure 6: Dependency Diagram for Categories

2.3 Suppliers

Moogle does not actually make any products but rather serves as a marketplace where people can browse items that are for sale that someone else has posted. There is no limitation on who can post items for bidding or sale so a supplier can either be a large company or any individual looking to make a quick buck on an item they have. These suppliers are essential to the business model since without any suppliers there would be no items for sale or for bidding. Since the suppliers is a backbone to the business it is necessary to make sure that they are reliable and that their information is kept. Information about the supplier is necessary because if anything were to go wrong with the products or shipping you need to be able to contact them to straighten things out. There is also other reasons that certain information is needed in order for different aspects of the business to run smoothly so here they are listed out.

An important key piece of information that is needed about the supplier is their *point of contact*. It was covered earlier why this is important but it is because if something were to go wrong with the item it would be your job to contact them and straighten things out and we want that information to be available to you. Their *address* would be necessary as well in case you needed to check out the supplier for yourself or to estimate shipping times or cost from the location. Also within the database model along with the previous attributes mentioned would be *phone number*, *name* and *revenue*. Keeping *revenue* is important since we want to keep track of what suppliers are making a lot of money and will help see what customers want and help with estimating our own revenue and costs. Since the *suppliers* are supplying the items there needs to be a relationship between the suppliers and the *items* that they posted for sale or bidding. Along with a relationship to items we can have the suppliers have a relationship with the *categories* entity so that suppliers can be linked to different categories to help organize or narrow search results for different products on the site. See Figure 3 on the following page for more details.

2.3 Continued

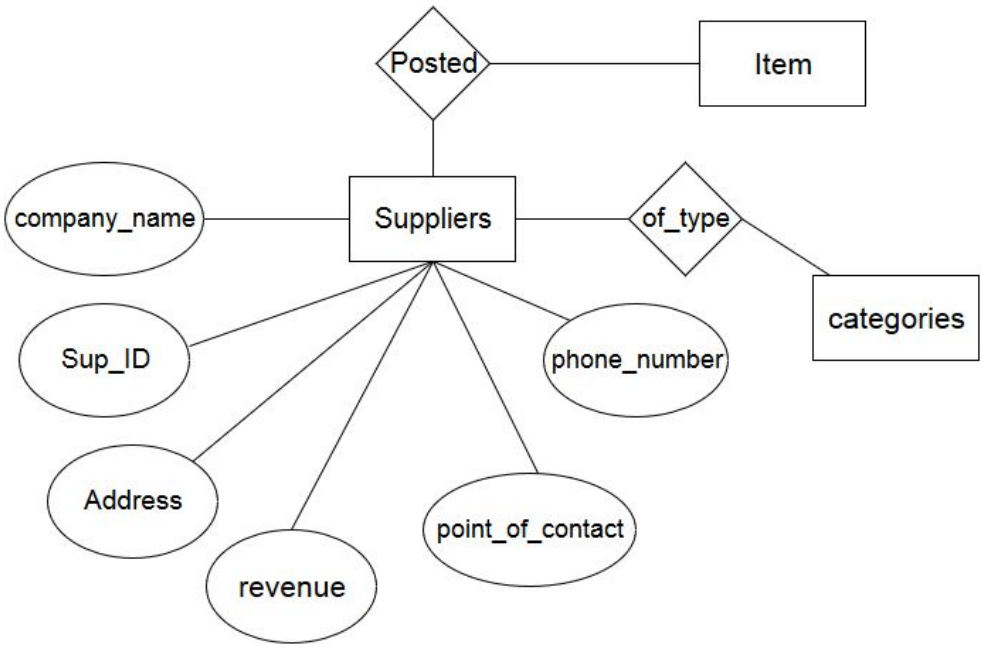

Figure 7: ER Model for Suppliers

```
CREATE TABLE Suppliers (
      supplierid CHAR(30),
      Name CHAR (50),
      Revenue FLOAT,
      Point_of_contact CHAR (20),
      Phone_Number CHAR (20),
      Address CHAR(50),
      PRIMARY KEY (supplierid)
);

CREATE TABLE Suppliers_Items (
      Supplier CHAR(30),
      Item CHAR (40),
      PRIMARY KEY (Supplier, Item),
```

```
      FOREIGN KEY (Supplier) REFERENCES Suppliers (supplierid)
            ON DELETE CASCADE,
      FOREIGN KEY (Item) REFERENCES Sale_Items (IID)
            ON DELETE CASCADE
);

CREATE TABLE User_Purchases (
      Purchase_ID CHAR(30),
      User CHAR(20),
      Item CHAR(40),
      PRIMARY KEY (Purchase_ID),
      FOREIGN KEY (User) REFERENCES Registered_Users (Username),
      FOREIGN KEY (Item) REFERENCES Sale_Items(IID)
);
```

Figure 8: SQL for Suppliers

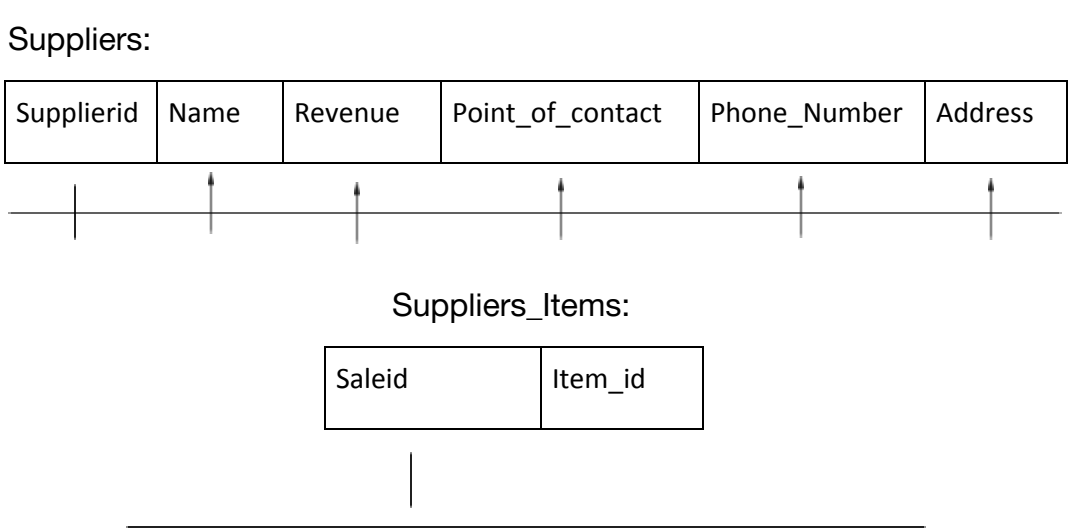

Figure 9: Dependency Diagram for Suppliers

2.4 Registered Users

In order for Moogle to be successful, it will need a lot of features, but the most important part of the business is to have users that are able to buy, sell, bid, and rate in the items that are being displayed on the website. Users will be able to have all the benefits of the website when they register in the website by creating an account name, providing their personal information such as *phone number*, *email*, *credit card* information for purchases, and some other basic information for transactions to be processed correctly and thoroughly. Next, we are interested to design a database model in which all this information would be stored and be linked to purchases where the current user has his/her information correctly extracted from the database. See Figure 4 on the following page for more details.

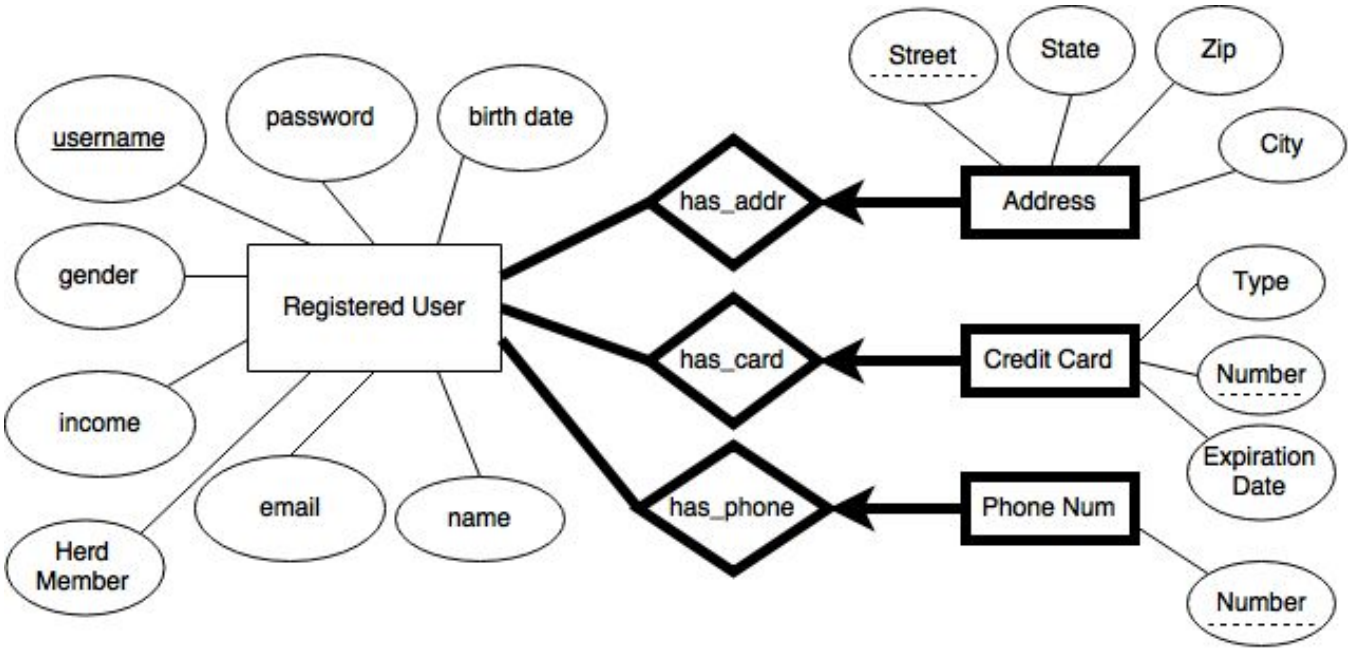

Figure 10: ER Model for Registered User

2.4 Continued

```
CREATE TABLE Registered_Users (
     Name CHAR(30) NOT NULL,
     Email CHAR(50) NOT NULL,
     Herd_Member BIT,
     Income INTEGER,
     Gender CHAR(6),
     Username CHAR(20) NOT NULL,
     Password CHAR (20) NOT NULL,
     Date_of_Birth DATE,
     UNIQUE (Email, Username),
     PRIMARY KEY (Username)
);

CREATE TABLE User_Address (
     Username CHAR(20),
     Street TEXT,
     Zipcode CHAR(10),
     PRIMARY KEY (Username, Street, Zipcode),
     FOREIGN KEY (Username) REFERENCES Registered_Users
          (Username)
          ON DELETE CASCADE,
     FOREIGN KEY (Zipcode) REFERENCES Zipcodes_Areas
          (Zipcode)
          ON DELETE CASCADE
);

CREATE TABLE Zipcodes_Areas(
     Zipcode CHAR(10),
     City CHAR(20) NOT NULL,
     State CHAR(20) NOT NULL,
     PRIMARY KEY(Zipcode)
);
CREATE TABLE User_Credit_Card (
     Username CHAR(20),
     Number CHAR(20),
     Type CHAR(20) NOT NULL,
     Expiration_Date DATE NOT NULL,
     UNIQUE (Number, Type),
     PRIMARY KEY (Username, Number),
     FOREIGN KEY (Username) REFERENCES Registered_Users
          (Username)
          ON DELETE CASCADE
);
```

2.4 Continued

```
CREATE TABLE Phone_Number(
     Username CHAR(20),
     Phone_Number CHAR(20),
     PRIMARY KEY (Username, Phone_Number),
     FOREIGN KEY (Username) REFERENCES Registered_Users
          (Username)
     ON DELETE CASCADE
);
```

Figure 11: SQL for Registered User

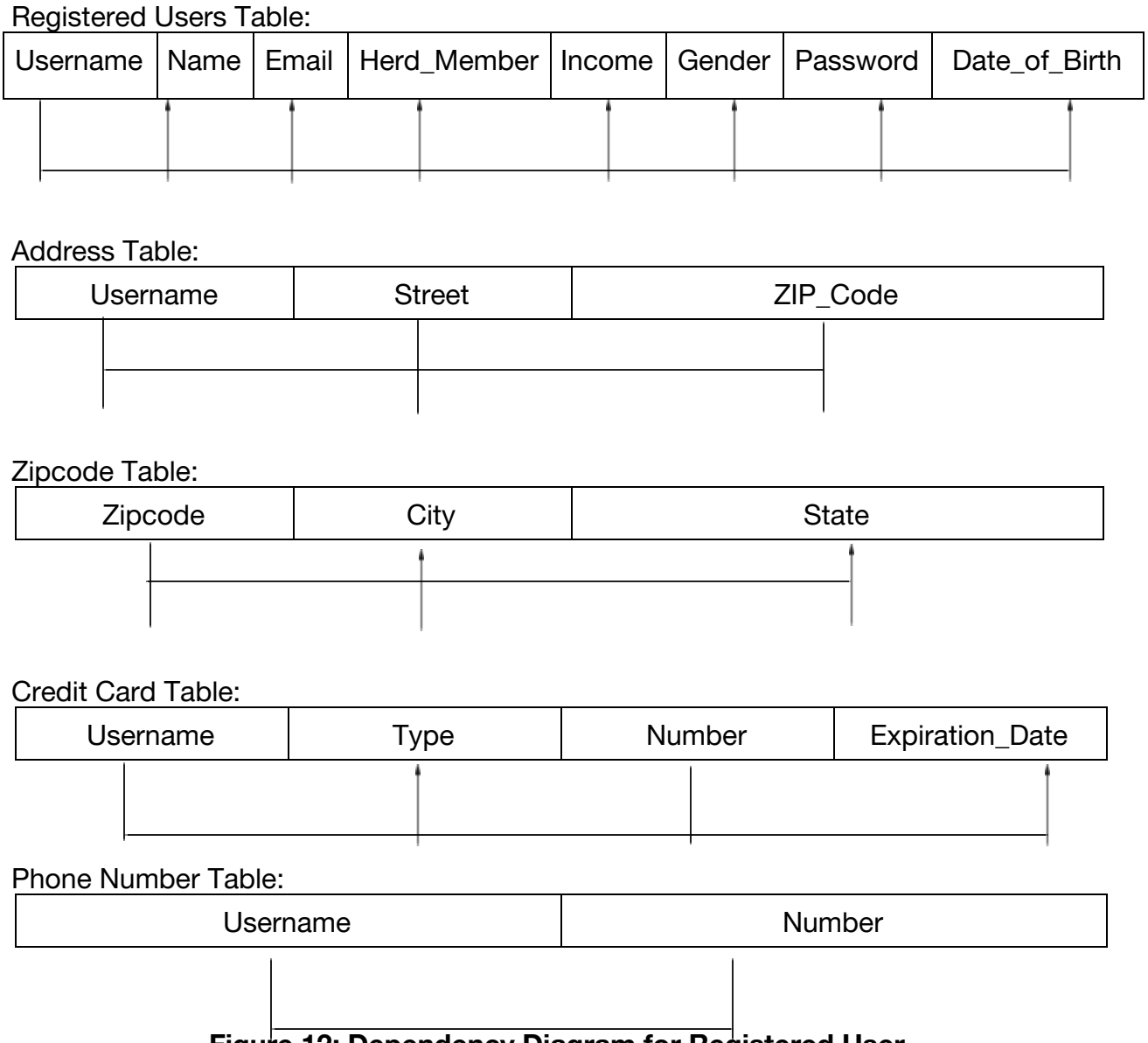

Figure 12: Dependency Diagram for Registered User

2.5 Ratings

We attempt to control fraud by allowing users to comment on the past behavior of other users. The rating systems also gives feedback to the seller on how they like the item being sold. We require buyers to give a *rating* from one to five *stars* at the minimum when giving a review. If the buyer also wants, they can give a written *review* of the item. The *date* of the review is recorded at the time the review is submitted.

Ratings is an entity with the attributes:

- *Date* – This is the date of when the review was written
- *Review* – What are the reviews given by previous buyers on the seller
- *Stars* – A visual representation of the rating, which ranges from 1 to 5 stars

This is a urinary relationship because only registered users can review other registered users. See Figure 5 below.

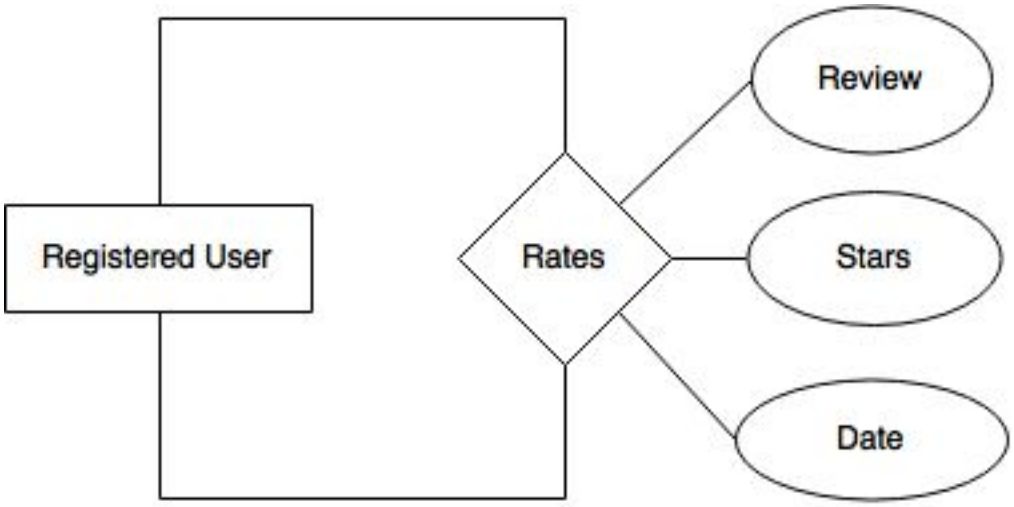

Figure 13: ER Model for Ratings

2.5 Continued

```
CREATE TABLE User_ratings (
          reviewer CHAR(20),
          reviewee CHAR(20),
          review TEXT,
          stars INT NOT NULL,
          date DATE NOT NULL,
          PRIMARY KEY (reviewer, reviewee),
          FOREIGN KEY (reviewer) REFERENCES RegisteredUsers
                (username),
          FOREIGN KEY (reviewee) REFERENCES RegisteredUsers
                (username),
                ON DELETE CASCADE
);
```

Figure 14: SQL for Ratings

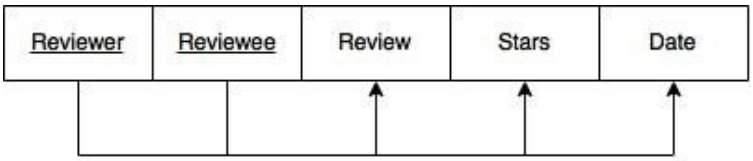

Figure 15: Dependency Diagram for Ratings

2.6 Browsing

The entire purpose of an online retailer is to get consumers to purchase their products. It's how they gain revenue, it's how they gain customers, and it's how they succeed. However, a user may not always know exactly what they want. They may have a general idea of what they are looking for but don't quite have an exact keyword or product name that they can explicitly search for. For this reason, it is crucial for Moogle to have a browsing feature. From the user's perspective, browsing will allow you to view different products, organized by their types, by choosing from a list of predefined categories and subcategories to further refine your search. Once a category is selected, the user is presented with a brief summary of the types of products that the selected category contains as well as the products themselves, laid out across multiple pages if necessary. From a user interface perspective, this category menu should be made immediately available to all users and therefore, should be located on the store's home page in order to encourage quick exploration of the store's items.

From the database's perspective, creating new entities will not be necessary for implementing the browse feature; we can simply rely on the entity sets that are already in place by retrieving the necessary data from these existing sets. For example, when a user selects a certain category they wish to examine, we can query the Categories entity (through a command similar to "select * from categories where name='selected category'). Then once the correct category is retrieved, we can get the summary associated with it and display this to the user. In addition to the category summary, Moogle will also need to provide the user with all the products that fall under the selected category. This can again be easily accomplished by querying the Sale Items entity set to retrieve all of the products whose category attribute match the currently selected category.

2.7 Searches

Users are able to search for a specific item or group of items by using keywords or conditions. As a search result, a list of items that satisfy the search criteria is returned to the users. There is also an option for users to use the categories section to find a specific type of sale or bid item they want to purchase. By searching by category, users will be given a wider range of items in the list of search results compared to when using a keyword.

2.8 Sale

Since we want Moogle to incorporate bidding and the sale of items we need to be able to adjust the database to account for two different types of buying an item on the site. This an important aspect of the business since some customers may not like bidding for items or do not want to wait for the time to be up on the bidding to get the item. Having items just for sale will help bring in those other customers that want to just buy out the item right then and there for a reasonable price. Implementation for this feature will not be hard to incorporate with the bidding system so it will be in the company's best interest to add this feature to items. Since users will have credit cards in our system it will be easy for them to click buy items right away raising the amount of items sold. Also items can be posted by suppliers for either sale or bidding at the discretion of the suppliers and will give them a chance to post their items so that they are sure they will get the money they want for an item. Instead of possibly selling the item at a lower price than they were hoping for through bidding. This will also encourage suppliers to post more items than if we just incorporated bidding in the business.

2.8 Continued

With respect to the database model the sale of an item will be incorporated in the item entity as described in section X.1. If the item is for sale and not bidding then the information will be null in the bidding attributes and the sale attribute will have the information needed such as the price. If the item is for bidding then the opposite will happen where the sale attributes will be null. This will also require a user to have a credit card on file so if there is no credit card or no valid credit card then the user will be forced to add one to their profile. Since we will want to keep the history of the sale items for at least 6 months after the sale we can set up a relationship between *Registered User* and *Sale Item* as shown below. The *SaleID* will be there so that even if the user or the item leaves our database there will still be record of the relationship and record of the sale. See Figure 6.

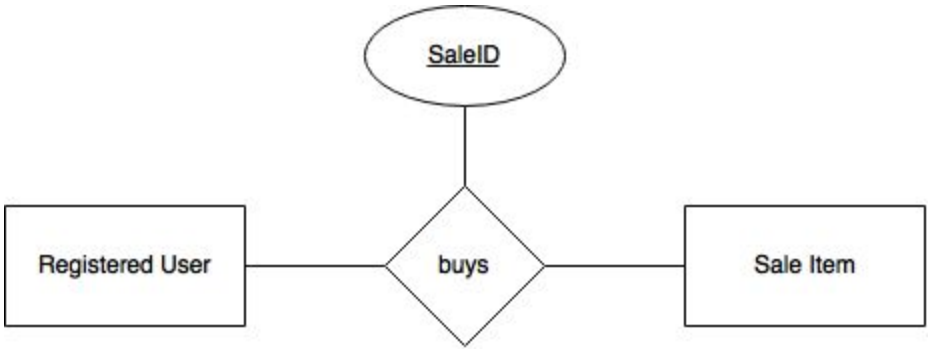

Figure 16: ER Model for Sales

2.9 Bidding

We know that many websites have either sales or bidding, our goal for Moogle is for it to be versatile and incorporate both, sales and bidding. We would like to give the option to our customers not only to buy and item and perhaps be disappointed by the fact that it may be out of stock and they could not do anything about it, but also give them the opportunity to bid for an item they want and be able to bid as much as they want. Our main goal by incorporating this feature is to have a higher amount of people being registered and interested in a site that will be able to incorporate both methods of purchase. When adding this feature, there are also some restrictions in which will be placed for the bidders such as the minimum amount for a bid or that the seller cannot bid for the item he/she is selling; however, all the restriction information is not necessary in the database because all the restrictions may be checked for within the application itself. In the ER diagram posted below, we have *bids* as a relationship between registered *users* and *sale* items because a registered user is able to bid on an item that is for sale. See Figure 7 below.

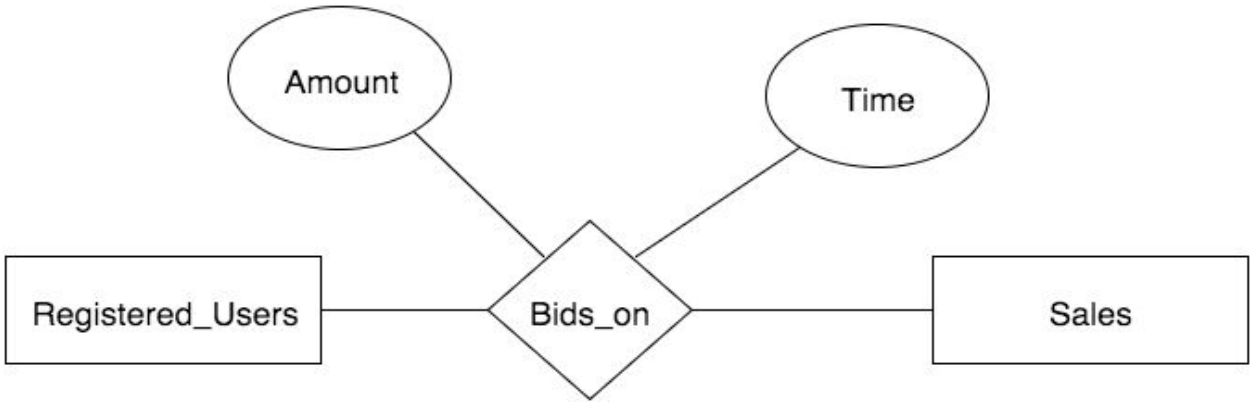

Figure 17: ER Model for Bidding

2.9 Continued

```
CREATE TABLE Bidding(
     Username CHAR(20),
     ItemID CHAR(40),
     Amount INTEGER,
     Time TIMESTAMP,
     PRIMARY KEY (Username, ItemID)
     FOREIGN KEY (Username) REFERENCES Registered_Users
          (Username)
          ON DELETE CASCADE,
     FOREIGN KEY (ItemID) REFERENCES Items (IID)
          ON DELETE CASCADE
);
```

Figure 18: SQL for Bidding

2.10 Orders and Sales Reports

Every company needs to have reports on their orders and sales to see how sales were for the week they are looking into. Moogle will generate weekly reports on orders and sales in order to see what categories and items are customers are buying the most. In order to generate these reports, we will not need to have data stored in the database for these specifics entities, instead, we are going to take information from the Sales entity and combine it into one report since the Sales Report and Sales are very closely related; therefore, we will not be needing an ER diagram for Order or Sales Reports. Orders would have the same format, information from the Sales entity would be combined into a report that will contain what the order was and all of the information needed for the company to have an idea on what is being ordered the most.

2.11 Delivery

Moogle takes the delivery needs of its clients very seriously. That is we have partnered with all of the major shipping companies to supply the fastest delivery system possible. Fortunately, our model already supports almost all we need to put that system into place already. A delivery request and report will be generated as soon as a sale is made. We already have the registered user's address on file, so we will simply mail to that address. Once our partners have started to ship the item, the users will receive a delivery confirmation number to track the shipment through their respective shipping companies site. We will also periodically pull our own shipping status using said confirmation number to keep our our customers updated, and keep that status for the 6 month sales history that is maintained.

```
CREATE TABLE Deliveries (
      DID CHAR(40),
      recipient CHAR(20),
      item CHAR(40),
      PRIMARY KEY (DID),
      FOREIGN KEY (recipient) REFERENCES Registered_Users
            (Username)
            ON DELETE NO ACTION,
      FOREIGN KEY (item) REFERENCES Items (IID)
            ON DELETE NO ACTION
);
```

Figure 19: SQL for Deliveries

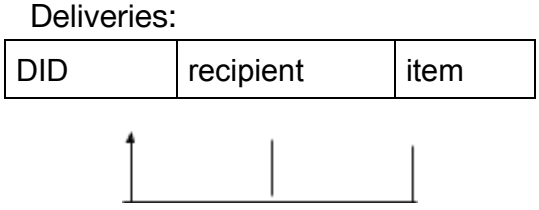

Figure 20: Dependency Diagram for Deliveries

2.12 Watchlists

It isn't unusual for a shopper to find an item in the store that they'd like to purchase, but for one reason or another, are not currently able to. This could be because the customer has an insufficient amount of funds or perhaps because they wish to save the item for a later date (such as a birthday or holiday). However, as the owner of the store, it is your responsibility to ensure that this customer doesn't forget about these items as it would result in the loss of a potential sale. For this reason, Moogle needs to utilize a "watch list' feature that would allow users to easily keep track of the items that they wish to buy at a future time. Registered users would then be able to create multiple watch lists for separate occasions and simply add items that they find throughout the store to one or more of these lists to serve as a reminder.

Now that we can see why watch lists are an important feature of any online store, we discuss how they can be modeled in the database. We know that each watch list has a name, belongs to a specific user (and thus should be deleted when its owning user is deleted), and consists of zero or more sale items. Therefore, we can model the watch lists as a weak entity set involved in an identifying relationship with the Registered Users set. Additionally, the watch lists entity set should also be involved in a relationship set with Sale Items in order to keep track of which items the user has added to a list, see Figure 8 below.

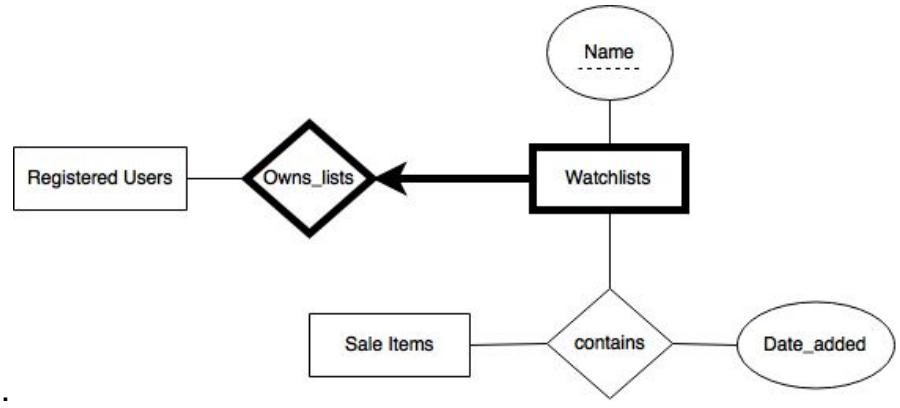

Figure 21: ER Model for Watchlists

2.12 Continued

```
CREATE TABLE Watchlists (
     name CHAR(30),
     owner CHAR(30),
     PRIMARY KEY (name, owner),
     FOREIGN KEY (owner) REFERENCES RegisteredUsers
          (username)
          ON DELETE CASCADE
);


CREATE TABLE Watchlist_Items (
     watchlist CHAR(30),
     item CHAR(40),
     added_on DATE,
     PRIMARY KEY (watchlist, item),
     FOREIGN KEY (watchlist) REFERENCES Watchlists (name)
          ON DELETE CASCADE,
     FOREIGN KEY (item) REFERENCES Items (IID)
          ON DELETE CASCADE
);
```

Figure 22: SQL for Watchlists

Watchlist_Items:

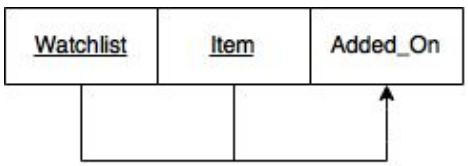

Figure 23: Dependency Diagram for Watchlists

2.13 Herd Membership

Our final feature is based on one of the most recent trends in online shopping applications, the featured user option. Services like “Amazon Prime” offer premium rewards at a price to the user. The most frequent users can take advantage of this system to hopefully save money in the end, and cut down on shipping time. Our “Herd Membership” will be available for a yearly fee yet to be determined. In return, members of the Herd will receive a 10% discount on all sale items. In addition, they will also receive free shipping on all of their orders. In order to implement this system, all we need to do is check if someone paid for the feature. If they did and it was less than a year from the sale date, they still qualify for the discounts. Once we set a fair price, loyal customers are sure to take advantage of our Herd Membership.

3.1 Technology Survey

Given all the requirements that have been previously detailed, Moogle will undoubtedly need to be an incredibly robust web application. It will need to handle user input that will come in multiple forms, store large amounts of data relating to both its products and its customers, and frequently communicate with the databases that store all this data in order to perform expected web store tasks and display accurate information.

Building such an application will require the use of software tools that are both powerful and efficient. The proper selection of which tools to use is an integral decision in the development of Moogle; choosing incorrectly could result in multiple delays as well as a potentially less-than-sufficient final result. Fortunately, after performing research on many available options, we as the developers of Moogle, feel that we have found the proper tools to begin creating the web store of the future.

3.2 Google App Engine

The first, and perhaps the most important, of these tools is Google App Engine. Google App Engine is a platform as a service cloud computing platform for developing and hosting web applications in Google-managed data centers. What this means is that the Moogle application will be built to run on top of Google's world-class infrastructure and therefore we will not need to be concerned with provisioning and managing a data center ourselves because it is completely taken care of by Google. This allows us to focus entirely on building the best possible version of Moogle for the users by worrying solely about its interface and how the application should perform. Google App Engine allows its users to code their applications with a variety of programming languages including Python, Java, and PHP. We have chosen to use the Python programming

3.2 Continued

language to create Moogle due to its intuitive syntax and its ability to perform very high level tasks in very few lines of code. This will allow us to create the final Moogle application in the shortest amount of time and with the greatest level of efficiency.

Perhaps the most important feature of App Engine from a business perspective is that it will be completely free to use. For very large scale applications, fees are charged for additional storage, bandwidth, or instance hours required by the application, but because Moogle will need time before it grows to such a large size, we will not need to concern ourselves with these fees at the current time.

Google App Engine also provides support for MySQL in the form of Google Cloud SQL. As Moogle will rely heavily on the use of databases, this is an incredibly useful feature of App Engine. A Cloud SQL instance is equivalent to a server and a single instance can contain multiple databases. Cloud SQL was designed to work well with Google App Engine and thus it is easily integrated into the code that will make Moogle work. Therefore we will be able to easily read, create, and update records within all of Moogle's databases all within our Python code. These changes/updates will be easily automated using different Python procedures instead of having to be manually entered by the developers which can be an incredibly time-consuming task.

3.3 Jinja

The other software tool that we will use to build Moogle is Jinja. Jinja is a template engine for the Python programming language that allows us to use Python-like syntax within our HTML files. Jinja offers various benefits such as a powerful automatic HTML escaping system for cross site scripting prevention, template inheritance which makes it possible to use the same or a similar layout for all templates, and a debug system that can quickly detect errors in the Jinja code. All these benefits streamline the

3.3 Continued

process of writing the HTML templates that will be used by Moogle to create a user-friendly interface.

One example of where Jinja will be useful in the creation of Moogle will be when it comes to our Herd Members. Since Herd Members receive a discount on all sale items in the Moogle store, we will need to display different web contents (such as sale price) to these users that aren't normally displayed to regular users. Therefore we can use the Jinja "if" statement in the HTML template to first determine whether the user viewing the page is a Herd Member and if so, then we display the discounted price to this user. Another good example would be on the home page of the Moogle store. On this page we will want to display multiple products to the user in order to entice them to make an impulse buy. However, we would need to write the same HTML code repeatedly in order to display each of these products. However, if we instead use the Jinja "for loop", we will be able to iterate through all the products we wish to display and pass each of these products to the same boiler plate HTML that displays these products on the web page. Because of this, we will be able to greatly reduce the amount of time spent writing HTML by reusing the same code with multiple parameters.

Through the combination of these software tools, we will be able to efficiently build Moogle into a powerful web application.

4.1 Entity Relationship Overview

On the following page in Figure 9 is our proposed complete diagram for all of our entities. The diagram revolves around entities with many attributes, relationships, and smaller entities surrounding them to implement all of our features listed in the previous pages. Our *registerd_user* entity contains all of the personal information needed in the application to successfully make purchases, bid on items at auctions, write reviews, and even create their own watchlists. Our *item* differentiates between *sale_items* and *auction_items*, and then contains all of the information the users need to successfully buy and receive said items. *Suppliers* then supply said items. All of the items and suppliers are then broken down into easily browsable and searchable *categories* to let the users find what they are looking for in a timely manner.

4.1 Continued

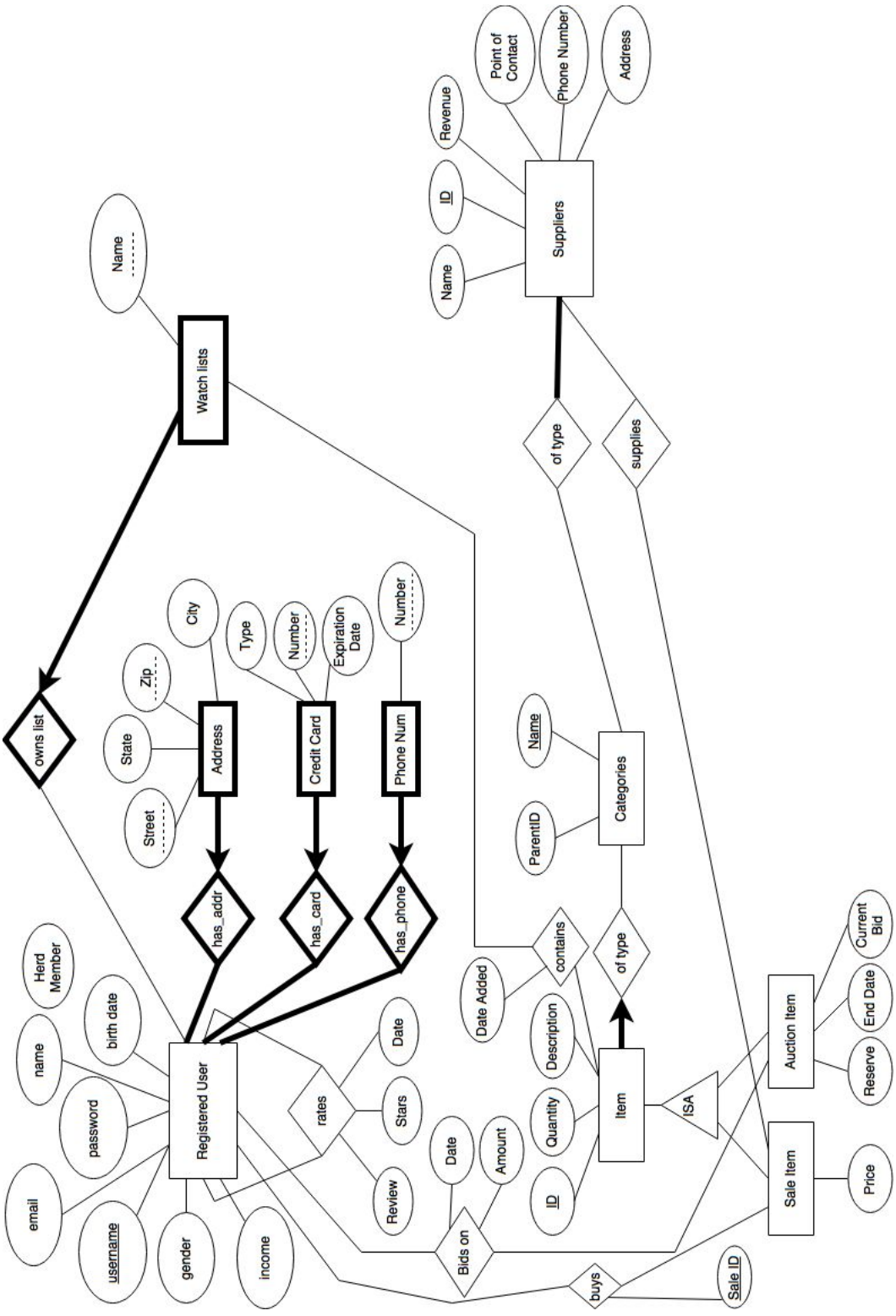

Figure 24: Complete ER Model

4.2 Conclusion

In conclusion Moogle has focused on the categories that are necessary for a company like this to run with the addition of its two features, “Herd Membership” and a watchlist. When drawing from companies that are already in place such as Amazon and eBay, we have organized the categories and thought about how each of them should react with one another in order to compete with the companies already in place. In analysis of the different sections and the overall company as a whole we came up with a total ER diagram that describes how the entities relate to one another and what attributes in each entity needed in order for Moogle’s database to run properly. This diagram will be the basis for the database we want to build for Moogle and moving forward we need to start thinking about how to implement our design. So far this is the conceptual design and principles we want to incorporate within each entity or how each process of Moogle will operate on a database level. This step is important because going forward we have a base design to work off of when actually trying to implement the database itself. This prevents us from forgetting about features or setting up the database in a way that doesn’t allow for a critical feature of the company to operate. In this paper we believe the design is efficient and will help make Moogle a top tier company.