CMPSC 431w Phase I Report

Kurt Mueller

Table of Contents

List of Figures

1. Introduction

This report outlines the foundation of the development of a database-backed web application for an online bookstore. This document will start with explaining the purpose this web application will have and consequently, the functions it is capable of performing. I will also discuss the overall design of the database will be explained. This will get support from illustrations primarily composed of Entity-Relationship (ER) diagrams to show what entities, relationships, and integrity constraints are dealt with and how they all relate to each other to perform the specific functions outlined at the beginning. Additionally, the tools and their role in the project will be addressed in detail. This will outline the tech stack used and show what platforms and tools that I use in order to complete the database-backed web application. This will also be backed up with comparisons to other popular platforms and tools used in web programming and database technology. Lastly, the overall logical design and normalization of the database is explained. This will show the relational schema for the database for each part of the ER diagrams and why the schemas were chosen.

2. Requirement Analysis

This section will address the system requirements and all that it is capable of doing along with the approach to how these features will be implemented. The system is designed to simulate an online bookstore that keeps track of customer information like the books purchased in order to make the overall user experience for all customers more convenient. All of the users will have the ability to interact with different books and other users with managers having additional privileges that will be addressed shortly. This section will split all 14 required functionalities into four different headers and the 5 additional functionalities are included in a fifth header at the end. Functionalities are italicized and bolded.

2.1 User Analysis

This system is composed of two different kinds of users, the *customers* and the *Managers* with customers being the bulk of what the system will contain and cater to. Customers and managers will both be *users*, but users will have an additional attribute called *isManager* which is either True if they are a manager and False if not. The user experience is different for users based on the position; managers will have access to the full functionality of the system while the customer experience is slightly limited in comparison. This difference in capabilities will be handled at the application level by performing a check to see the position of the user before allowing certain functions.

2.2 Registration

Users will have many functions they are able to perform, but first they must ***register*** as a new user to the system. In order to create an account, they must provide a login name (username) and a password that will be hashed to ensure that the accounts are secure. The login name chosen by the user must be unique so that seeing someone's username is enough to know what user the system is interacting with (passwords do not need to unique). Additionally, both of these will be stored as strings on the database likely capped off at 20 characters with the password possibly being more characters depending on the hashing. Lastly, each user must provide their full name, address, and phone number along with their login name and password they created. Details about previous orders for each customer will be stored as well but this is not important when registering. Managers will register in a similar fashion; however only other managers are able to register a ***new manager***. This means that when the system is first created, a manager will be created so that more can be added in the future. A manager can create another manager simply by changing the *isManager* attribute for a user, so registration for managers and customers is identical for simplicity.

2.3 Book-User Interaction Analysis

Being that this is an online bookstore, the system must also store information regarding all the books that are in the system and give *Customers* and *Managers* ways to interact with these books. The data that must be stored in the database for each book include: a unique ISBN, title, author(s),

publisher, language, publication date, number of pages, stock level, price, keywords, subject, and copies sold. After a customer has successfully registered, they are able to ***order*** books. *Customers* are allowed to order one or more books at a time so a *shopping cart* will be needed to store the unique ISBNs for each of the books in the order. The shopping cart will provide the total price of all of the books in the cart from querying based off of the ISBNs of the books. For each book that is successfully ordered, the *stock level* will decrease by one and the *copies sold* will increase by one. When the order is completed, the order details (price, date ordered, ISBNs of books bought, user id) will be stored in an *orders* table. As for browsing for books, customers are able to ***filter*** by searching by *authors*, *publisher*, *title*, *language*, etc.. They can also ***sort*** the books by publish date, average score of comments, or by average score of trusted user feedback (comments, trust, and user feedback are explained shortly). The system will also support ***degrees of separation*** for authors. This will allow customers to not only see books from the same author but from authors that are 1 or 2 degrees separated with that author. For example, it can show *1-degree separated authors* by finding books where the author being looked at co-authored another book with another author. As for *2-degree separated authors*, we can query the database for authors that are in the intersection of sets of two other authors' 1-degree separated authors. These can both be obtained through simple set operations in queries to the database. Lastly, after an order is made, the system will show ***suggestions*** for other books to order. Since we kept track of all of the orders of each customer, the system can then query the orders to show books other users have ordered along with the books the current customer just ordered as suggestions. The query to obtain these will also order by the number of copies sold (decreasing) of the books found.

Managers have some unique interactions with the books in the bookstore. First, they are able to ***add new books*** to the database if more arrive at the warehouse. This means they must also be able to ***adjust the stock level*** for new and existing books since more may arrive to be sold. Additionally, they will need to provide all of the required attributes of a book if it is not in the database at all. These restrictions will be taken care of at the application level. Lastly, managers can access the ***book statistics*** every semester. These statistics include: the m most popular books (top m books with highest copies sold), m most popular authors (top m authors of books with highest copies sold), and m most popular publishers (top m publishers of books with highest copies sold). All of these statistics can easily be obtained by writing queries on the book table, limiting the result by m and sorting by copies sold.

2.4 User-User Interaction Analysis

This system also allows for users to interact with each other through the *trust, rating,* and *comment* features. Starting with the *trust* functions, every customer is able to choose zero or more other users to ***mark as trusted or untrusted***. The login names of all of these trusted users would be stored and each of the trusted users would have their trust score increased by one while each of the untrusted users have their trust score decreased by one. This means that each user has a *trust score* and it is the number of other users that have marked them as trusted minus the number of users that marked them as untrusted. Another crucial part of the system is the ability for customers to write ***comments*** on books. Every comment for a book includes the date it was posted (using datetime library), their rating of the book from 1 to 10 where 10 is the best (Integers only), and

they can also include some text if they would like but it is not required. All comments can be seen by every user, however ***only the creator of the comment can modify their comment***, this will be handled at the application level. Lastly, ***a single user can only make one comment on a book*** to prevent inflating the ratings of the books.

Users are able to interact with comments made by other users by ***giving them a usefulness score*** from 1 – 3, where 3 is very useful. You are not able to rate your own comment, this is easily handled at the application level. These ratings can help customers that want to find the most useful comments. Customers can then ***ask for the top n useful comments*** for a specific book and this will query the comments in the database to sort by average usefulness score (descending) and limit by n. These ratings will also come in handy for the ***User Rewards*** given by the managers. *Managers* can give awards to the top m users with the highest trust score as described earlier, and now for the top m most useful users which is just the average of all of the usefulness scores of all of the user's comments. Users will then have a *useful score* along with the *number of ratings* they have had to easily calculate the user's usefulness score. The manager can then change the *awarded* attribute for the user to *True* if they win one of these awards.

2.5 Additional Functionality Analysis

As for the additional functionality, I decided to give some more features for customers to take advantage of. First, *users* will be able to ***change their password*** if they would like. This option would bring them back to the same sort of procedure as when the registered to pick a new

password. The password must be unique and if it is already used an error will be displayed. Next, *users* are given the ability to add a profile picture to their account. Upon registration, they will not have a ***profile picture*** (or potentially a default picture) and they now can add one to customize their account. This will just be another optional attribute for users in the database and can easily be modified with an update query to the database. Additionally, users can ***add and remove books from their shopping cart*** which will provide the total price of the current cart. Every user has only one shopping cart and when a book is added, the *totalPrice* attribute for the shopping cart will increase by the cost of the added book. Another function that *users* can take advantage of is being able to ***see the list of all users that received awards*** likely shown on the home page. This can be done by querying the User table for users with *awarded* as *True*. Lastly, if a *user* sees a book that they would like but it is no longer in stock, they are able to ***request more copies*** of the book. These requests go straight to the *managers* so that they can see that people want more of the book and should order more. This will likely require another integer attribute in the database for each book to represent the number of *requests* the books have.

3. Conceptual Database Design

Below is the ER-diagram that addresses all of the functionalities described in the previous section.

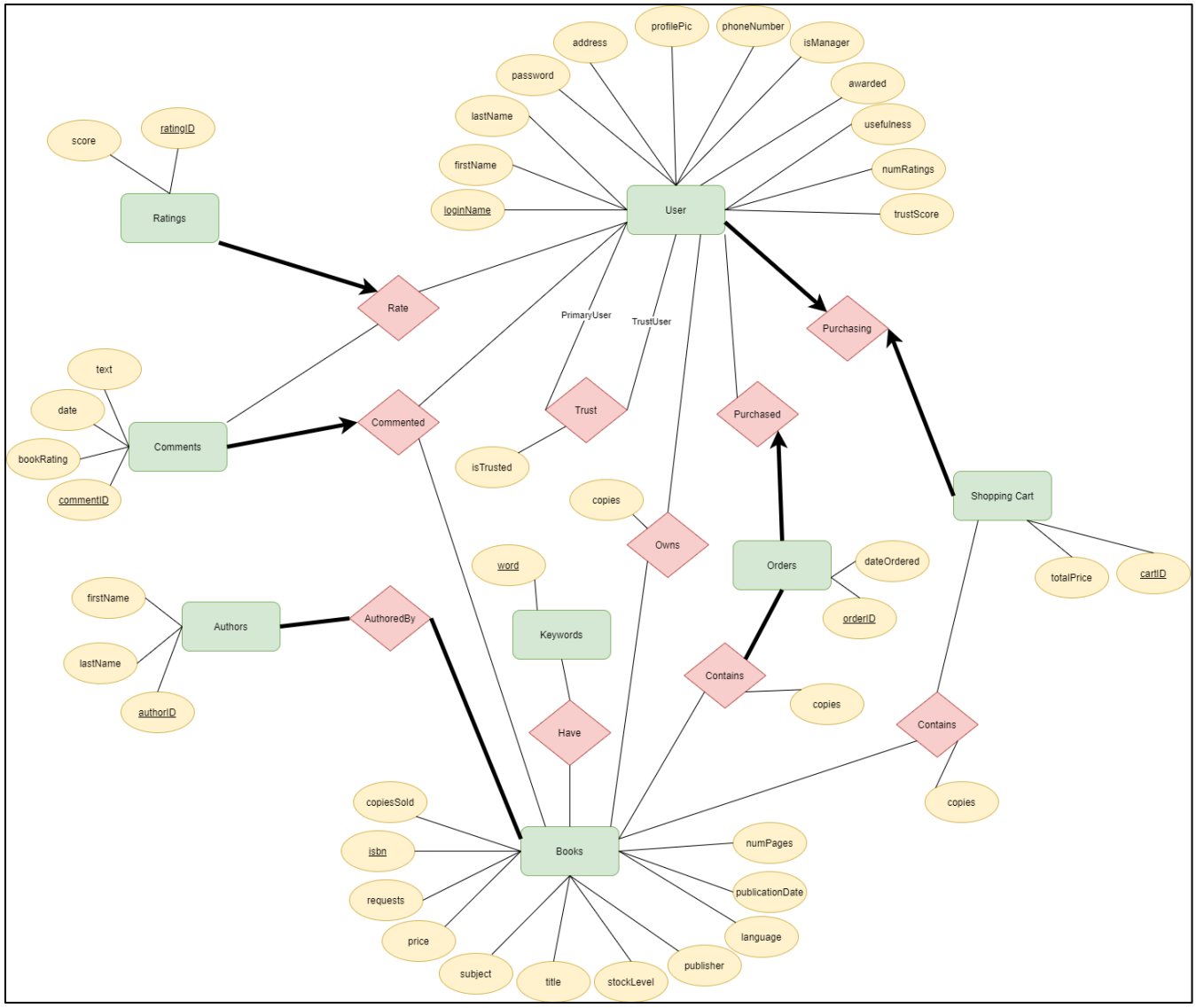

Figure 1 ER-Diagram for the system

3.1 Book Database Design

This part will explain the details of the ER-diagram related to the storing of data for books along with the relations, entities, and integrity constraints that are affiliated with the book data. This section takes a closer look at a specific segment of the ER-diagram having to do with the functions having to do with interacting with the *Books* table in the database.

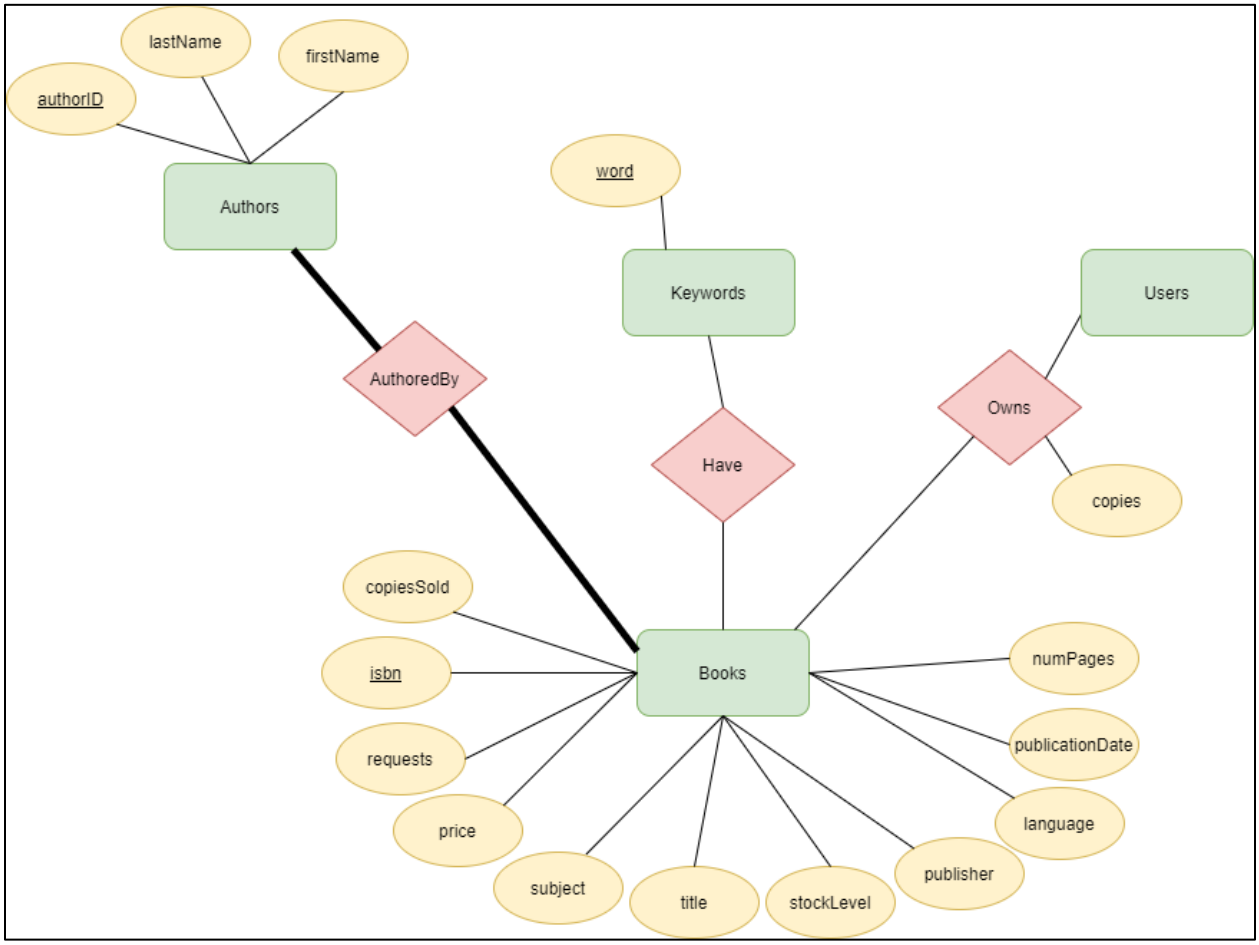

Figure 2 Book Section of the ER-Diagram

The Entities being looked at in this section include: *Books*, *User*, *Keywords*, *Authors.*

The *Books* entity has the attributes: copiesSold, requests, price, subject, title, publisher, language, publicationDate, numPages, stockLevel, isbn. The isbn is unique for each book in this table and will be important for the relations connected to it. For example, the *Authored By* relationship connects the *Books* entity with the *Authors* entity so that it has both the unique identifier for *Authors* (authorID) and the unique attribute for Books (isbn) in its table.

The *Authors* entity contains the firstName, lastName, and authorID as attributes for each author in the table with every author having a unique authorID. We know that each book must have at least one author and authors do not have any cap to the number of books they can write. The ER-diagram reflects this through the relationship between the *Authors* and *Books* entity.

Next, there is a relationship between *Books* and *Keywords* entities since a book can have 0 or many keywords so making keywords an attribute would limit a book to only 0 or 1 keyword. The *Keywords* entity represents a table full of all the keywords used for books in the system. This design will likely be optimized later on since having a table of words isn't incredibly practical. The *Have* relation between these two entities connects the words in the *Keywords* table to the corresponding books that have them as keywords by the isbn numbers.

Lastly, the *Owns* entity is related to *Books* because it is designed to hold all the books that each user owns, meaning the books that the user has ordered. This means when a user finalizes an order, all of the books corresponding to the unique ISBNs from the order are added into this

table attached to the user's unique loginName. This *Owns* entity also saves the number of copies of each book that is owned which may become useful when sorting the book suggestions by decreasing sales amount among the users that bought a certain book. The figure below illustrates how all of these functions are possible with the described entities, relations, and integrity constraints.

The portion of the ER-diagram shown in this section also allows for all the integrity constraints and functionalities to still be possible to enforce. A manager can still ***add new books*** and ***increase the books stock*** when new shipments arrive since this is done at the application level. Users are still able to ***sort books*** by publish date, average numerical score of the comments, and by average trusted user feedback score. This is done since each book has a publish date attribute and the ER-diagram will show the relation between books, comments, and usefulness ratings in the comments section. Since we have an *Authors* relation, we can find authors with ***degrees of separation*** from each other easily through some queries on the table. All of the ***book statistics*** can be easily obtained since the number of copies sold of each book is stored in the table as an attribute. This makes it way easier to find the most popular authors, publishers and books through querying the database. Lastly, the *requests* attribute in the *Books* entity makes if easy for managers to see what books to order after ***users request more copies*** for a specific book and the requests number is increased.

3.2 User Database Design

This part will explain the details of the ER-diagram related to the storing of data for users along with the relations, entities, and integrity constraints that are affiliated with the user data and their given functionalities.

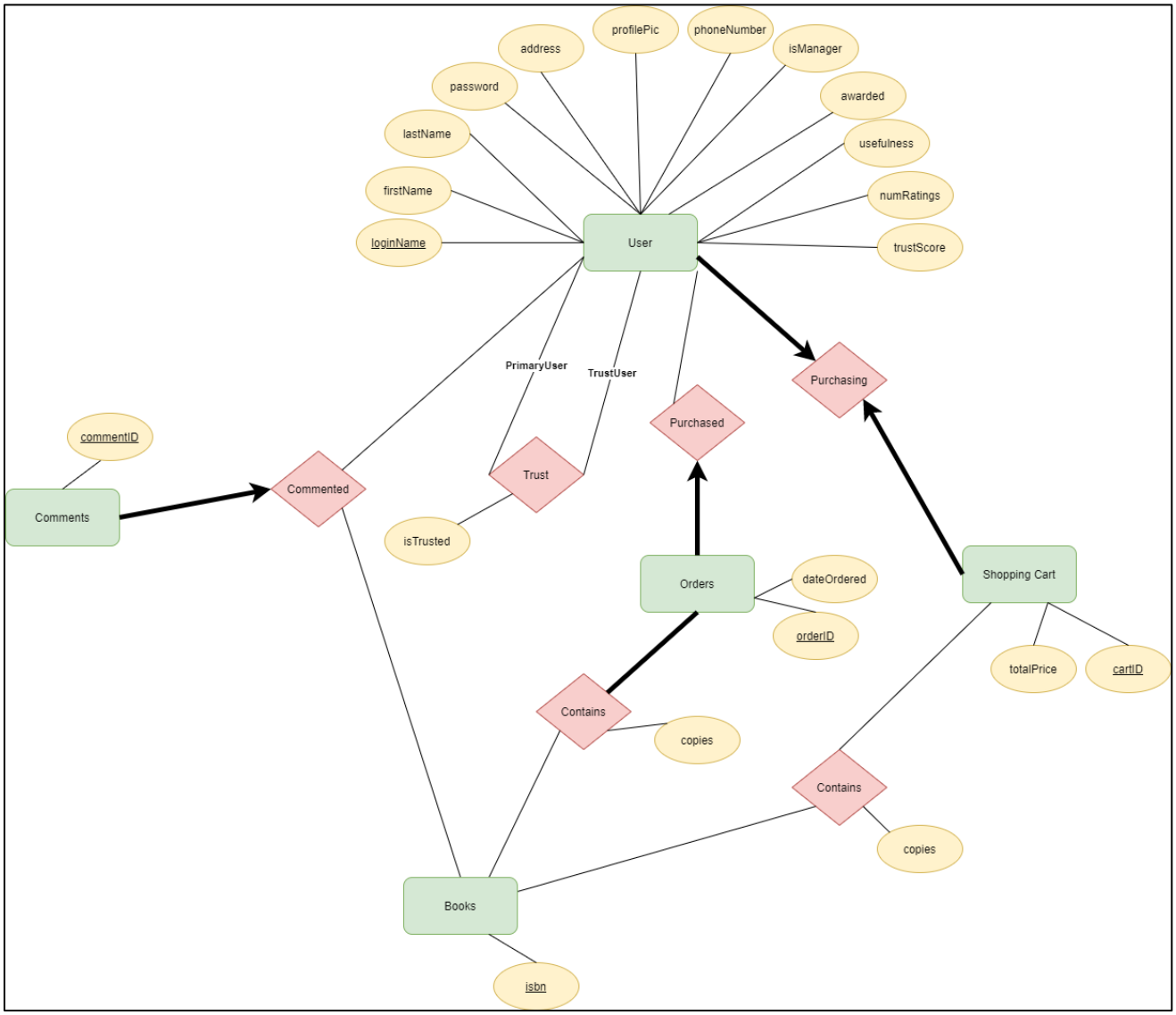

Figure 3 Users Section of the ER-Diagram

The entities included in this portion of the ER-diagram include: *Users, Comments, Orders, Shopping Cart.*

The *Users* entity stores all of the essential information for each user with the attributes: *firstName, lastName, password, address, profilePic, phoneNumber, isManager, awarded, usefulness, numRatings, trustScore, loginName*. The *loginName* is a unique attribute for each user made at registration. The *password* will be hashed before stored. The *profile picture* is an optional attribute. The *isManager* attribute indicates if the user is a manager with a boolean. The *awarded* is also a boolean to say whether they have been awarded by a manager or not. The *usefulness* attribute is the total usefulness rating for every one of the user's comments combined. The *numRatings* is the number of total usefulness ratings their comments have received (for calculating the overall usefulness score). The *trustScore* is just the number of people that trust the user minus the amount that mark them as untrusted.

The *Orders* entity is also crucial for this section and it has *orderID* and *dateOrdered* as attributes. The purpose of this entity is to store all of the orders placed in the system by relating the user by their unique loginName to the book ISBNs that they ordered. The *dateOrdered* attribute is import for performing the ***book statistics*** function for managers. With this date you can find the books sold in the particular quarter by finding orders made between the current date and 3 months before the current date. This is related to the *Users* entity through the *Purchased* relation that has the distinct orderID connected with the unique loginName of the user that made the specific order. This is extremely important for giving users the ability to purchase orders of

books. Notice the integrity constraint that each order can only be made by one user, but each user can make many or no orders.

The *Books* entity resurfaces in this section because it is related to the *Orders* entity through the *Contains* relation. This relation connects the unique orderID with 1 or more distinct book ISBNs while also keeping track of the number of *copies* ordered of each book as shown by the *copies* attribute in the diagram.

The *Shopping Cart* entity is very similar to the *Orders* entity because it does also is related to the *Users* and *Books* entities. This entity has the *totalPrice* and unique *cartID* as attributes with total price being the sum of prices of the books corresponding to the ISBNs in the contains relation. This is so the ***total price of the order can easily be displayed to the user*** before confirming their purchase which then inserts the shopping cart information into the orders table. The purpose of the shopping cart is to store all of the ISBNs of the books that have been added to an "order" but have not yet been completed and purchased. Every user has only one shopping cart and each shopping cart corresponds with only one unique user. Notice that the shopping cart does not have to contain a book unlike an order since an order isn't an order unless something is bought, but a shopping cart exists for each user from immediately after registration. There is also an additional functionality that allows the ***user to add and remove books from the shopping cart*** which can easily be handled at the application level and delete queries on the shopping cart table.

Another ability the users must have is to be able to ***mark other users as trusted or untrusted*** if they would like. This is where the *trust* relation comes in. This relates the user that is marking another user as trusted or untrusted (Primary User) with the user that is being marked (Trust User). This relation has an attribute, *isTrusted*, which stores a Boolean value of false when the trust user is being marked as trusted and false when being marked as untrusted. Notice that the ER-diagram allows for a user to mark zero or more other users as trusted or untrusted as outlined in the system requirements. This relationship also allows for users to now easily ***sort books*** by the average numerical score of the trusted user feedbacks.

Lastly, the *Comments* entity is related to the *Users* because a big part of the system is for users to have the ability to write comments on books. The attributes and other details for this entity are discussed in the next section. Notice that the diagram enforces the integrity constraint that each user is allowed to comment as much as they would like, but every comment corresponds with only one user and book. This makes it so that a user can only ***write one comment for each book***. The *commented* relation makes this all possible since it connects a unique loginName of a user with a distinct ISBN of the book being commented with the unique commentID associated with the comment made. Because of this relation, users can easily ***sort books*** by average numerical score of comments.

3.3 Comment Database Design

This part will explain the details of the ER-diagram related to the storing of data for comments users came write about books along with the relations, entities, and integrity constraints that are affiliated with the user data and their given functionalities.

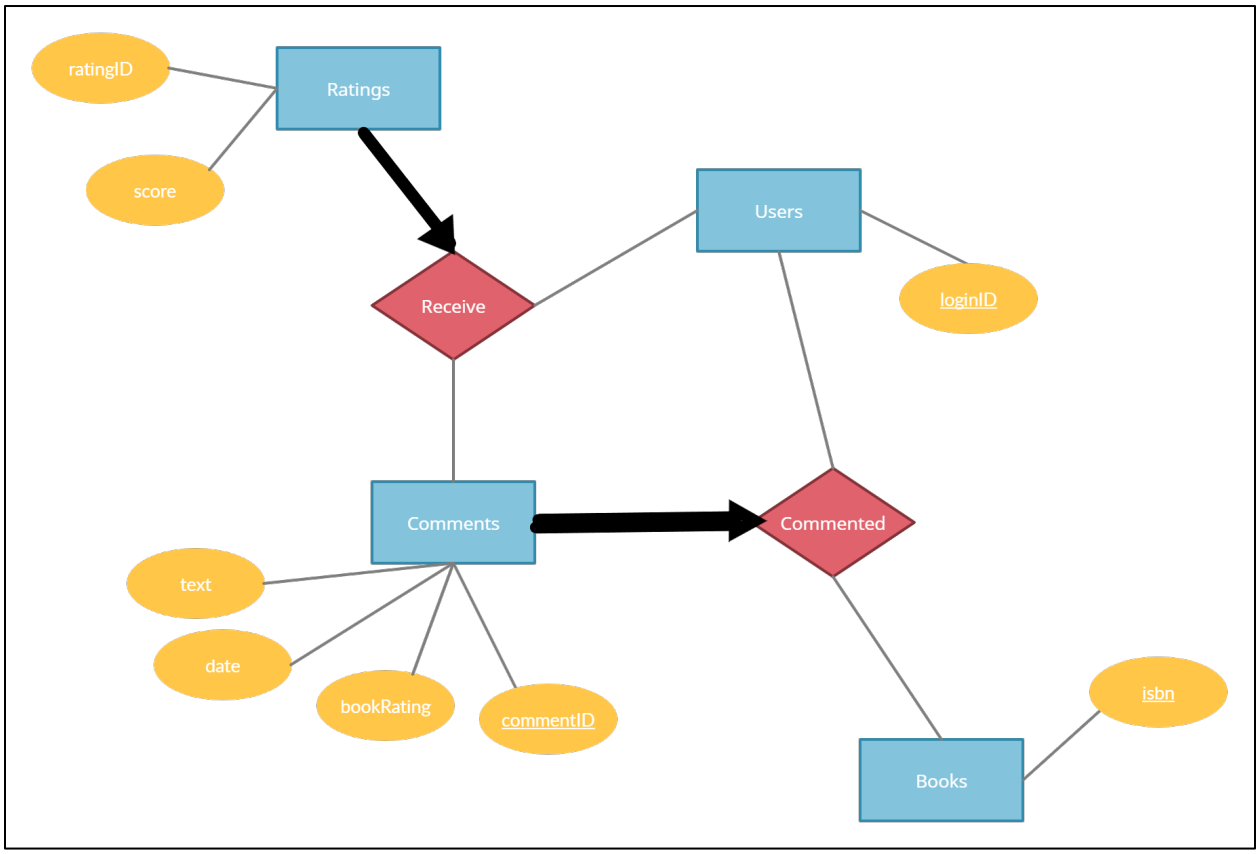

Figure 4 Comments Section of the ER-Diagram

The *Comments* entity stores the important information about each of the comments made my users about a specific book. This entity has the attributes: *text, date, bookRating, commentID.* The text attribute represents the optional text the user can write in the comment. The ability for customers to change their comment will be handled at the application level so this diagram still supports this function. The date is when the comment was written. BookRating is the rating the user gave the book that the comment is about from 1 – 10, 10 being best. The commentID is the unique key for the comment. As mentioned in the user section, this entity is related to both the *users* and *books* entities. The diagram makes sure that for each comment, only one user and book is affiliated with it to keep users from commenting multiple times on the same book.

The *Ratings* entity is the last entity to be covered. The purpose of this entity is to store all of the usefulness ratings that users give on other users' comments. This entity has a *ratingID* and *score* attributes. The ratingID is the unique key for this rating while the score attribute is the rating from 1 – 3, 3 being the most useful, that they think the comment deserves. This entity is related to the *users* and *comments* entities through the *rate* relation which contains the commentID and loginName for the associated ratingID. The diagram makes sure to enforce the integrity constraint that ***every rating is made by exactly one user about exactly one comment***. The purpose for this entity and its relationship with the users and comments entities is to give users a lot of their required functionalities. For example, users can now ***rate comments*** and use these ratings to find useful comments about books to narrow down what book they would like to

order. This also helps ***managers find the top useful users*** by gathering the rating information corresponding to the all the comments made by each unique user by their loginName.

4. Technology Survey

In order to create a database-backed web application for this online bookstore, a tech stack will need to carefully decided to perform all of the necessary tasks as easily as possible. The tech stack will consist of a web programming framework, programming language(s), an IDE to code in, and database management system. This section will describe the tools used for this system and compare to other popular tech stacks for creating database-backed web applications. This section will compare my tech stack consisting of Bootstrap, Flask, Python, and MariaDB to the popular MERN stack and WAMP stack. The MERN stack utilizes MongoDB for the database, Express(.js) for a web framework, React(.js) for the client-side JavaScript framework and Node(.js) for the web server As for the WAMP tech stack, it uses Windows, operating system, Apache web server, MySQL for database, and a mixture of PHP, Perl and python scripts for programming support. My chosen tech stack takes the best, and easiest to learn, parts of each of these popular stacks to best suit my skill set and desired tasks for the system. This section will now split up the layers of the tech stacks for comparison to show how my chosen tools fit best for the project.

4.1 Front-End

For the front-end, MERN uses Node.js and React which is a JavaScript library created by Facebook to improve the speed of frontend development by using a new way of rendering web pages, making it much more dynamic and more responsive to user inputs [2]. This is possible because it utilizes Virtual DOM, which is just a virtual, logical structure of the HTML in memory. What is nice about this choice in the MERN tech stack is that React uses JavaScript to generate HTML rather than what most other tools do which gives HTML more abilities to do what you want instead [1]. This makes may help with the time it will take me to learn front-end development since I will not have to learn as much HTML. As for Node.js, it is used mainly for building fast, large, scalable networks that receive a big number of simultaneous connections [5]. This is a little specific to a certain kind of application and this bookstore will not require this complexity and size. WAMP leaves front-end development open to the developer's choice while using the Apache Web Server to facilitate serving web pages to the users. Apache is more of a back-end part of the process so it will be discussed more later on. This could be nice since it will allow for me to experiment with different front-end languages, however this is a little overwhelming due to my lack of front-end development experience. This makes using the front-end framework, Bootstrap, a more appropriate front-end option for my purposes and skill set. This framework is primarily used for developing websites using CSS and HTML templates and it supports JavaScript plug-in. Bootstrap is extremely useful for creating design templates for the visual components of the website [3]. Additionally, Bootstrap supports JavaScript and jQuery add-ons which may end up

being useful with some UI tasks. Bootstrap also uses a grid system to easily scale websites so that it can look good on all devices and screens of different sizes [4]. This makes it much easier to control where different UI features are placed since it is more intuitive. Lastly, this framework is also very widely used for web development now and is growing fast so there is a lot of great documentation and tutorials to learn quickly.

Pros of Bootstrap:

- Easy to learn because of great documentation and tutorials
- Compatible with many browsers
- Versatile
- Intuitive frontend development

Cons of Bootstrap:

- Must learn more HTML
- It's designed more for mobile-first web applications

The main reason for using Bootstrap is that I do not know much about front-end development to begin with, so I believe it would be beneficial for me to learn a very popular (if not the most popular) front-end framework. This will give a chance to learn more HTML and understand more of how front-end development works. Furthermore, Bootstrap is a framework that supports many add-ons which may be useful later into my frontend development. Lastly, I have found may more

intensive and detailed documentation and tutorials for learning Bootstrap than I did for React. This is why I Bootstrap is a better fit for me than the choices WAMP and MERN provide for front-end development.

4.2 Back-end

As for the back-end, Flask is what my tech stack will be using while MERN uses Express web framework, and WAMP uses Apache as mentioned previously. Express is a fairly flexible Node.js web framework similar to Flask in the sense that it gives the developer a lot of freedom to determine how they would like everything to be organized [7]. Express is a little more geared to handling larger projects than Flask which gives it more complexity to have to figure out which will not even be used for this system. Also, the documentation and tutorials for learning Express are nowhere near as good as Flask' which is important since I would need to learn no matter what framework I use fast to get the project completed. Apache web server on the other hand is very popular among large companies with a lot of traffic on their websites. You are easily able to set up a web server with Flask and Python instead of having to learn all of what Apache is capable of for this system. Flask is designed for smaller projects, there are less features to have to learn so it is known to be a great back-end framework for beginners. Flask also gives you more freedom when implementing your application compared to other frameworks that already load functionalities for you without letting the user decide [5]. Additionally, Flask comes with a development server and debugger to help with all stages of development. There are not a whole lot of dependencies to have to worry about since the two Flask is based on are Werkzeug (a WSGI

utility library) and Jinja2 (template engine) [6]. These dependencies help in reducing the amount of low-level code needed to design web applications. Lastly, Flask does not support databases natively so there are more options to choose for the project's database [8].

Pros of Flask:

- Easy to learn
- Good documentation
- Designed for smaller applications

Cons of Flask:

- Not many setup tasks done automatically
- Developer must manually design more

Flask is definitely the best fit for this project and my skill set. Flask is designed for smaller, less complex applications which is perfect for the size and complexity of this online bookstore. Also, the more open-ended nature of Flask could be helpful when deciding what I want to do with the project since I will have more freedom. Lastly, Flask is recommended for beginners in web development which I have not had much exposure to, so it is an appropriate choice for my skill set. Additionally, Python is the language I chose for my back-end development since I am very comfortable with it and I have not had much experience with other languages commonly paired with Flask like PHP which is used in WAMP stack. Both languages are used often when creating

web applications with Flask and they each have their advantages, but they are extremely similar in the end so Python was the obvious choice for me.

4.3 Data Storage

Finally, we must choose what database management system we would like to use for the database-backed web application. This is crucial for storing all of the data for the books, users, etc. in the online bookstore. MERN utilizes MongoDB while WAMP stack uses MySQL.

MySQL is the an extremely common and well-known database management system maintained by Oracle that covers all aspects of what the project needs for data storage as well as MongoDB. This makes the decision a little harder since these three DBMSs are all easily capable of performing the required tasks for the system. The MERN stack uses MongoDB mainly because it is intended more for modern mobile development so it is able to dynamically transfer your database and all data is saved in JSON format [12]. Obviously, this is not being aimed at being a mobile experience for the purpose of the project so this is not entirely necessary. MariaDB is a fork of the MySQL DBMS and it was given additional commands like JSON, WITH and KILL statements [9]. Overall, MySQL is very similar to MariaDB so it is more of a developer's preference, but MariaDB is very fast especially in comparison to MySQL and has these additional commands in case they are ever needed. Also, MariaDB is very reliable since it is used by many big organizations like Google, Craigslist, Wikipedia, and more. Like most database management systems, MariaDB has many database connectors including Python which we need for the system

Pros of MariaDB:

- Easy to use
- Has many connectors
- Very fast
- Additional Commands

Cons of MariaDB:

- Need to install python connector

Lastly, I feel that it is important to add SQLite to the comparisons since it is commonly paired with Python and Flask which we are using. SQLite is another popular relational database management system for web applications. SQLite only contains one file on the disk so it is more portable than other DBMSs. It is also serverless so no API or library installations are needed to retrieve data from SQLite [10]. Also, Python actually comes with built-in support for SQLite with the sqlite3 module. This makes it very easy to connect to the database with Python [11]. One drawback to SQLite is the security since any user is able to read/write data without being specifically granted access.

This clear winner to use for this system is MariaDB for storing important data to keep the online bookstore running smoothly. MariaDB is best for this mainly because I have had more experience creating my own databases with MariaDB prior to this project so it will be easier to

implement. Additionally, it is able to handle multiple users reading and writing from the database at the same time. In the bookstore, many users will have the ability to write to the database for actions like, updating their profile pictures or contact information. SQLite only allows one write operation to be done at a single time, so this could cause problems with multiple users being active on the website at once. Lastly, it is fairly easy to connect to the database with flask, all you need is a connector and there are plenty of resources to help with that online.

4.4 Final Choice of Tech Stack

In the end, the MERN and WAMP stacks are great and they are very popular for their own reasons. MERN has shaped the way was created mobile-friendly web applications, and WAMP has been even referred to as a "classic, time-tested stack of technologies" [13] that has made it easy for people to decide what tools they would like to use for web development for years. Researching these tech stacks showed me the plethora of options that exist, and how many are some are geared toward certain types of development while others boil down to what the developer is comfortable working with. I chose all of the components of my stack based on how complex my web application will need to be, but also based off of my current skill set and documentation for each tool. Being new to web development almost entirely shifted my focus to finding tools that have been known to have great documentation and are easier to learn. This is extremely important due to the short amount of time provided to do this project. Therefore, my tech stack I plan to work

with is Bootstrap for the front-end framework, Flask for the web framework, Python for the primary programming language, and MariaDB for the database management system.

5. Logical Database Design and Normalization

This section will create the relational schema used for the system's database. The schema will be reduced to Boyce Codd Normal Form in the end after the ER diagram is translated to relations. I will provide the schema through the DDL queries that I would use to create my relations in the database so that the foreign keys, primary keys, and constraints are clear.

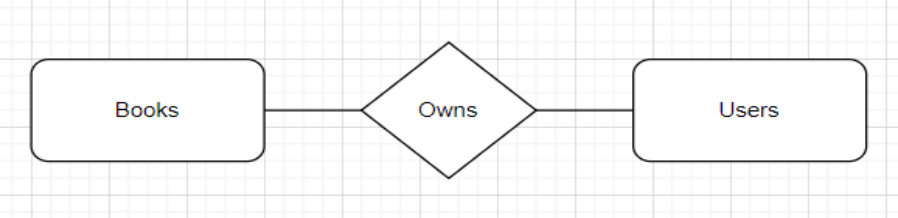

Figure 5 Users, Owns, Books Relation

```
CREATE TABLE Books(
        isbn CHAR(13) NOT NULL,
        copiesSold INT NOT NULL,
        requests INT NOT NULL,
        price DECIMAL(10, 2) NOT NULL,
        subject VARCHAR(40) NOT NULL,
        title TEXT NOT NULL,
        stockLevel INT NOT NULL,
        publisher TEXT NOT NULL,
        language VARCHAR(40) NOT NULL,
        publicationDate DATE NOT NULL,
        numPages INT NOT NULL,
        PRIMARY KEY (isbn))
```

This is clearly in BCNF since every attribute that is not a key is known by the unique isbn attribute. You can see this because the isbn is the Primary key so it must know all other attributes.

```
CREATE TABLE Users(
        loginName VARCHAR(40) NOT NULL,
        firstName VARCHAR(40) NOT NULL,
        lastName VARCHAR(40) NOT NULL,
        password TEXT NOT NULL,
        address TEXT NOT NULL,
        profilePic BLOB,
        phoneNumber VARCHAR(15) NOT NULL,
        isManager BOOL NOT NULL,
        awarded BOOL NOT NULL,
        usefulness INT NOT NULL,
        numRatings INT NOT NULL,
        trustScore INT NOT NULL,
        PRIMARY KEY (loginName))
```

Similarly, to the Books table, this is also in BCNF form due to the distinct primary key *loginName* deciding all other non-key attributes.

```
CREATE TABLE Owns(
        loginName VARCHAR(40) NOT NULL,
        isbn INT(13) NOT NULL,
        PRIMARY KEY (loginName, isbn),
        FOREIGN KEY (loginName)
                REFERENCES Users(loginName),
        FOREIGN KEY (isbn)
                REFERENCES Books(isbn))
```

The Owns table relates the Users and Books tables by allowing the unique loginName of each user be tied to as many book ISBNs as they have ordered. This makes the primary key have to be BOTH the loginName and isbn. This also means that it is in BCNF since the loginName alone is not enough to know the isbn and the isbn alone is not enough to decide the loginName. Thus, the combination of the two as the primary key is the only one that knows all attributes.

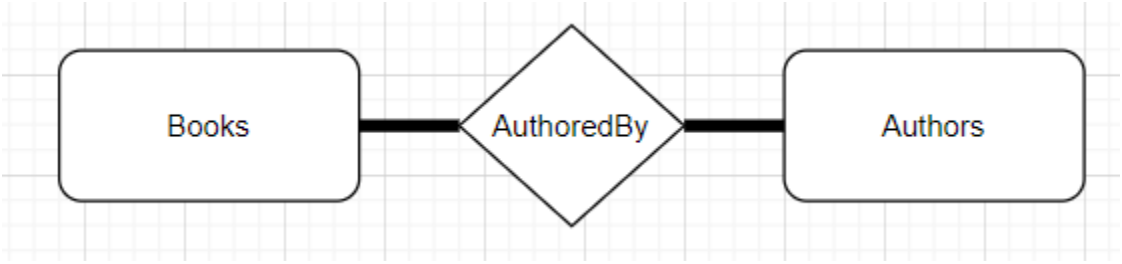

Figure 6 Books, AuthoredBy, Authors Relation

```
CREATE TABLE Authors(
        firstName VARCHAR(40) NOT NULL,
        lastName VARCHAR(40) NOT NULL,
        authorID INT NOT NULL,
        PRIMARY KEY (authorID))
```

The Authors table is in BCNF because each author has a unique authorID so only that primary key knows all other non-key attributes.

```
CREATE TABLE AuthoredBy(
        authorID INT,
        book CHAR(13),
        PRIMARY KEY(authorID, isbn),
        FOREIGN KEY(authorID) REFERENCES Authors(authorID),
        FOREIGN KEY(book) REFERENCES Books(isbn))
```

These tables show the relationship between books and authors. This allows for books to have one or more authors and you can look into the AuthoredBy table to find which book ISBNs are tied to which authors. The primary key has to be both the unique authorID as well as the distinct book ISBN since neither alone is enough to decide all other attributes, thus it is in BCNF as well.

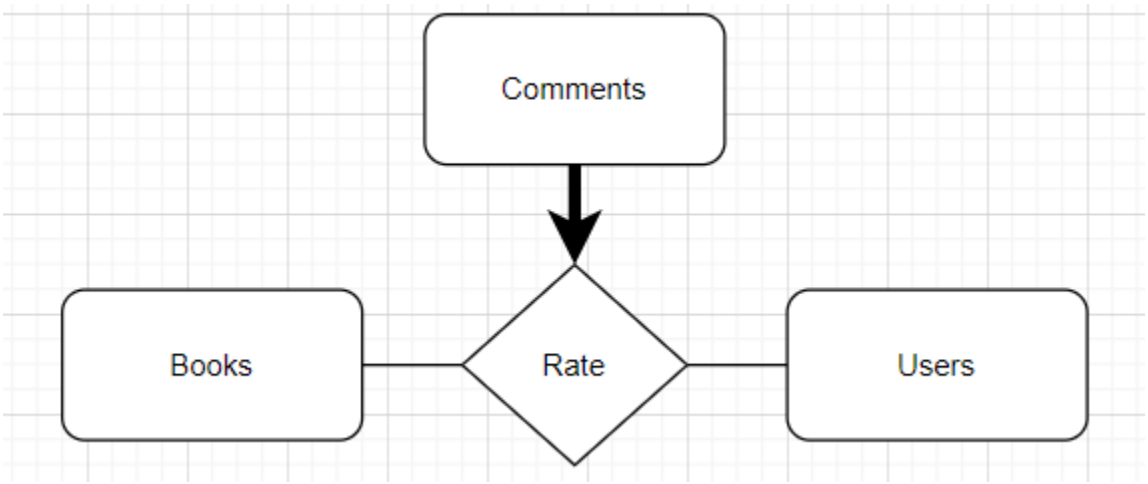

Figure 7 Books, Commented, Users, Comments Relation

```
CREATE TABLE Comments(
        text TEXT,
        date DATETIME NOT NULL,
        bookRating INT(2) NOT NULL,
        CHECK (bookRating > -1 AND bookRating < 11),
        commentID INT NOT NULL,
        PRIMARY KEY (commentID))
```

This table has the a unique commentID primary key for each comment made so only that key attribute decides the other non-key attributes which makes it in BCNF. The date, text, or bookRating are not enough on their own to be keys for sure and they cannot decide the other attributes. Take note that the table is making sure that the bookRating is an integer between 0-10.

```
CREATE TABLE Commented(
        commentID INT,
        user VARCHAR(40),
        book CHAR(13),
        PRIMARY KEY (commentID),
        FOREIGN KEY (commentID) REFERENCES Comments (commentID),
        FOREIGN KEY (user) REFERENCES Users (loginName),
        FOREIGN KEY (book) REFERENCES Books (isbn))
```

The Commented table ties the book, comment, and user together. This is making sure that for each comment, there is exactly one user responsible for writing it and it only refers to one book by the unique ISBN. This is why the commentID is able to be the primary key since a comment has a unique user and book. Because of this, it is in BCNF and only the keys can decide the none keys since neither the loginName or isbn alone are enough to know the rest of the attributes.

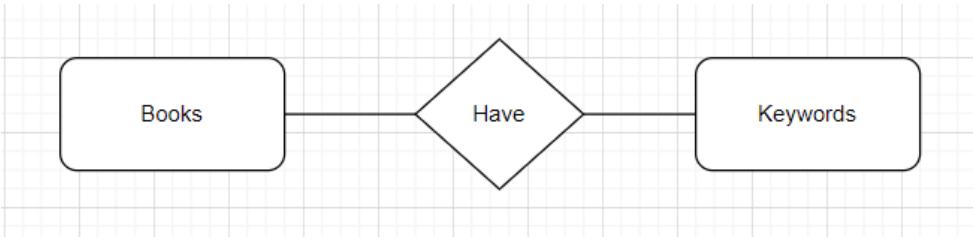

Figure 8 Books, Have, Keywords Relation

```
CREATE TABLE BookKeyword(
        isbn CHAR(13),
        word VARCHAR(20) NOT NULL,
        PRIMARY KEY (isbn, word)),
        FOREIGN KEY (isbn) REFERENCES Books(isbn))
```

This relationship between books, have, and keywords is where some normalizing needs to happen. The Keywords relationship just stores words and all of the words are unique since it is a set. You can combine the Have relation with the Keywords entity to create the BookKeyword table. This table stores all of the keywords for each unique book ISBN. This means ISBNs and keywords can show up more than once so the primary key must be the combination of the two. This also keeps it in BCNF since the primary key is the only way to know the other attribute.

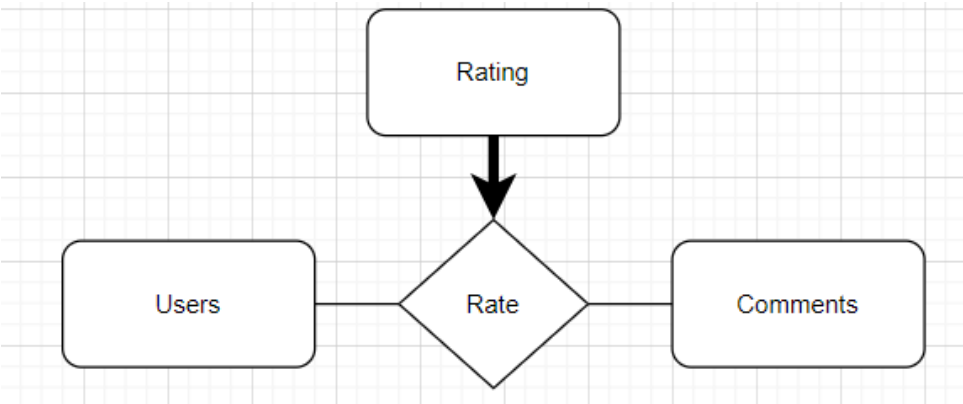

Figure 9 Ratings, Rate, Comments, Users Relation

```
CREATE TABLE Ratings(
        score INT DEFAULT 0,
        ratingID INT NOT NULL,
        PRIMARY KEY (ratingID))
```

The Ratings table is designed to hold all of the usefulness ratings made by users on another user's comment. There will be a unique ratingID made for each comment to keep it in BCNF form and avoid confusion.

```
CREATE TABLE Rate(
        ratingID INT,
        comment INT,
        user VARCHAR(40),
        PRIMARY KEY (ratingID),
        FOREIGN KEY (ratingID) REFERENCES Ratings (ratingID),
        FOREIGN KEY (comment) REFERENCES Comments (commentID),
        FOREIGN KEY (user) REFERENCES Users (loginName))
```

The Rate table is what connects Ratings with comments and users. Here it is crucial that the primary key is the ratingID since every rating must have exactly one user that made the rating and one comment it is about. This also keeps it in BCNF since the non-keys (comment and user) are decided by the primary key attribute (ratingID).

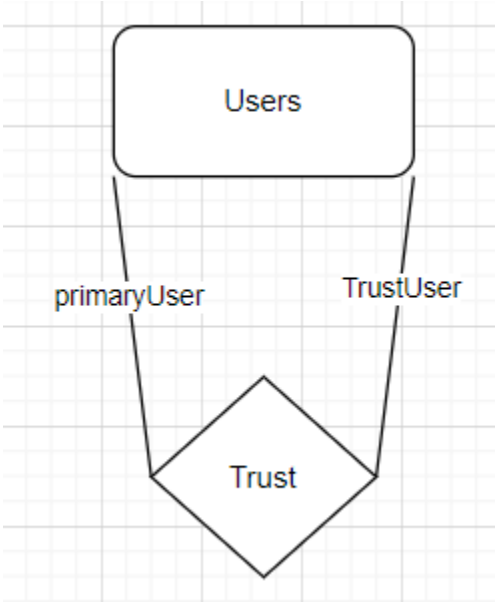

Figure 10 Users, Trust Relation

```
CREATE TABLE Trust(
        primaryUser VARCHAR(40),
        trustUser VARCHAR(40),
        isTrusted BOOL,
        PRIMARY KEY (primaryUser, trustUser),
        FOREIGN KEY (primaryUser) REFERENCES Users (loginName),
        FOREIGN KEY (trustUser) REFERENCES Users (trustUser))
```

The trust table is designed to hold all of the users that the one user marks as trusted or untrusted. The user that is marking other users is the primaryUser while the user being marked as trusted or untrusted is the trustUser. The primary key must be both the primaryUser and trustUser loginNames to keep it in BCNF.

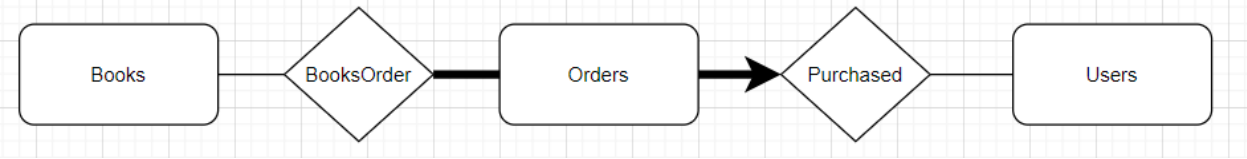

Figure 11 Books, BooksOrder, Orders, Purchased, Users Relation

```
CREATE TABLE Orders(
        orderID INT,
        dateOrdered DATETIME,
        PRIMARY KEY (orderID))
```

The Orders table contains all of the orders that have been made in the system and the date and time it was placed. There is unique orderID created for each order to keep it in BCNF and give it a clear primary key.

```
CREATE TABLE Purchased(
        orderID INT,
        loginName VARCHAR(40),
        PRIMARY KEY (orderID),
        FOREIGN KEY (orderID) REFERENCES Orders (orderID),
        FOREGIN KEY (loginName) REFERENCES Users (loginName))
```

The purchase table is designed to connect the user to the orders they have placed. No order is made twice so this is clearly the primary key but each user can make multiple orders so it is decided by the orderID key making it in BCNF.

```
CREATE TABLE BooksOrder(
        copies INT DEFAULT 1,
        book CHAR(13),
        orderID INT,
        PRIMARY KEY (orderID, book),
        FOREIGN KEY (orderID) REFERENCES Orders (orderID),
        FOREGIN KEY (book) REFERENCES Books (isbn))
```

The BooksOrder table allows the order to more than one different book in a single order specified by the orderID. Each order can have more than one distinct ISBN so the primary key has to be the ISBN of the book and orderID. This makes it so in BCNF since only the orderID and book ISBN pairing is enough to decide all the attributes.

Figure 12 Books, Contains, ShoppingCart, Purchasing, Users Relation

```
CREAT TABLE ShoppingCart(
        totalPrice INT DEFAULT 0,
        cartID INT NOT NULL,
        PRIMARY KEYS (cartID))
```

The Shopping Cart table is nearly the same as the Orders table but it has the total price of all the books contained in the shopping cart so that it can be displayed to the user easily before confirming an order. A unique cartID is made for each shopping cart and each user has exactly one shopping cart. This keeps it in BCNF since the cartID tells you all the other attribute values.

```
CREATE TABLE Purchasing(
        cartID INT,
        owner VARCHAR(40),
        PRIMARY KEY (owner),
        FOREIGN KEY (cartID) REFERENCES ShoppingCart(cartID),
        FOREIGN KEY (owner) REFERENCES Users (loginName))
```

The purchasing table is the shopping cart equivalent to purchased. It also just connects the user that owns the shopping cart to the shopping cart id. The primary key could be ither the loginName of the user or the cartID but I decided to make it the loginName. They are both keys so they both know the other attributes so it is still in BCNF.

```
CREATE TABLE Contains(
        cartID INT,
        book CHAR(13),
        copies INT DEFAULT 1,
        PRIMARY KEY (cartID, book),
        FOREIGN KEY (cartID) REFERENCES ShoppingCart(cartID),
        FOREIGN KEY (book) REFERENCES Books (isbn))
```

Lastly, the Contains table is the shopping cart equivalent to the BooksOrder table that gives the shopping cart the ability to hold more than one distinct book. Just as the BooksOrder table was in BCNF, this one is too since it is structured exactly the same way except a shopping cart is not required to have a book. It can be empty too while an order cannot.

6. Conclusion

The report here will facilitate the process of making the database-backed web application for the online bookstore. From this report, the overall functionality of the system is outlined as well as the data that needs to be stored. Another important aspect is that it outlines how each of the functions are made possible by the design of the database. The last section then brings it all together by normalizing the tables and deciding all of the schema need to store all of the data to allow the store to operate. Lastly, this report goes over all of the tools that will be used for data storage, front-end and back-end development, and overall framework to put it all together in comparison to other popular tools used today.

Project Plan

Goal	Date to Complete By
Complete Task 1 of Phase I	March 24, 2021
Complete Task 2 of Phase I	March 28, 2021
Complete Task 3 of Phase I	March 29, 2021
Finish Phase I	March 31, 2021
Create my database by finalizing my DDL queries and start populating the tables (Task 1)	April 5, 2021
Get the database connected with python in Flask	April 8, 2021
Get a homepage up and begin front-end development	April 10, 2021
Have at least half of the system functionalities working on the website	April 20, 2021
Finish the rest of the functionalities (Task 2)	April 29, 2021
Write up the Reflection (Task3), make the video and turn in the prototype	April 30, 2021

References

[1] Buna, Samer. "Yes, React Is Taking over Front-End Development. The Question Is Why." *FreeCodeCamp.org*, FreeCodeCamp.org, 6 Sept. 2020, www.freecodecamp.org/news/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76/.

[2] Editor. *The Good and the Bad of ReactJS and React Native*, AltexSoft, 27 Feb. 2020, www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/.

[3] Admin, and John Peter. *Bootstrap Vs React : Choose The Best Front End Framework*, Techidology, 12 Dec. 2020, techidology.com/bootstrap-vs-reactjs/#:~:text=Bootstrap%20is%20used%20for%20developing,HTML%20and%20CSS%20design%20templates.&text=React%20tools%20can%20make%20use,enhanced%20with%20the%20Bootstrap%20framework.

[4] Rahman, Syed Fazle. "Why I Love Bootstrap, and Why You Should Too." *SitePoint*, SitePoint, 3 May 2018, www.sitepoint.com/why-i-love-bootstrap-you-should/.

[5] Capan, Tomislav. "Why The Hell Would I Use Node.js? A Case-by-Case Tutorial." Toptal Engineering Blog, Toptal, 13 Aug. 2013, www.toptal.com/nodejs/why-the-hell-would-i-use-node-js.

[6] Amigos-Maker. “What Is Flask Used for?” *DEV Community*, DEV Community, 16 Nov. 2019, dev.to/amigosmaker/what-is-flask-used-for-2do5#:~:text=Flask%20is%20a%20lightweight%20Web,scale%20up%20to%20complex%20applications.&text=As%20a%20developer%20in%20developing,a%20framework%20to%20your%20advantage.

[7] Farzan, M. S. “I Rebuilt the Same Web API Using Express, Flask, and ASP.NET. Here's What IFound.” FreeCodeCamp.org, FreeCodeCamp.org, 11 June 2020, www.freecodecamp.org/news/i-built-a-web-api-with-express-flask-aspnet/.

[8] Grinberg, Miguel. “The Flask Mega-Tutorial Part IV: Database.” *Miguelgrinberg.com*, blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database.

[9] Sarig, Matan. “MariaDB Vs MySQL: Compatibility, Performance, And Syntax.” *Panoply Blog - Data Management, Data Warehousing, and Data Analysis.*, Panoply, 29 Jan. 2021, blog.panoply.io/a-comparative-vmariadb-vs-mysql.

[10] Delgado, Carlos. “Everything You Need to Know about SQLite Mobile Database.” *Our Code World*, Our Code World, 13 Feb. 2019, ourcodeworld.com/articles/read/737/everything-you-need-to-know-about-sqlite-mobile-database.

[11] “Define and Access the Database¶.” *Define and Access the Database - Flask Documentation (1.1.x)*, flask.palletsprojects.com/en/1.1.x/tutorial/database/.

[12] Ivanovs, Alex. “MongoDB vs MariaDB vs MySQL.” Geekflare, 8 Mar. 2021, geekflare.com/mongodb-vs-mariadb-vs-mysql/