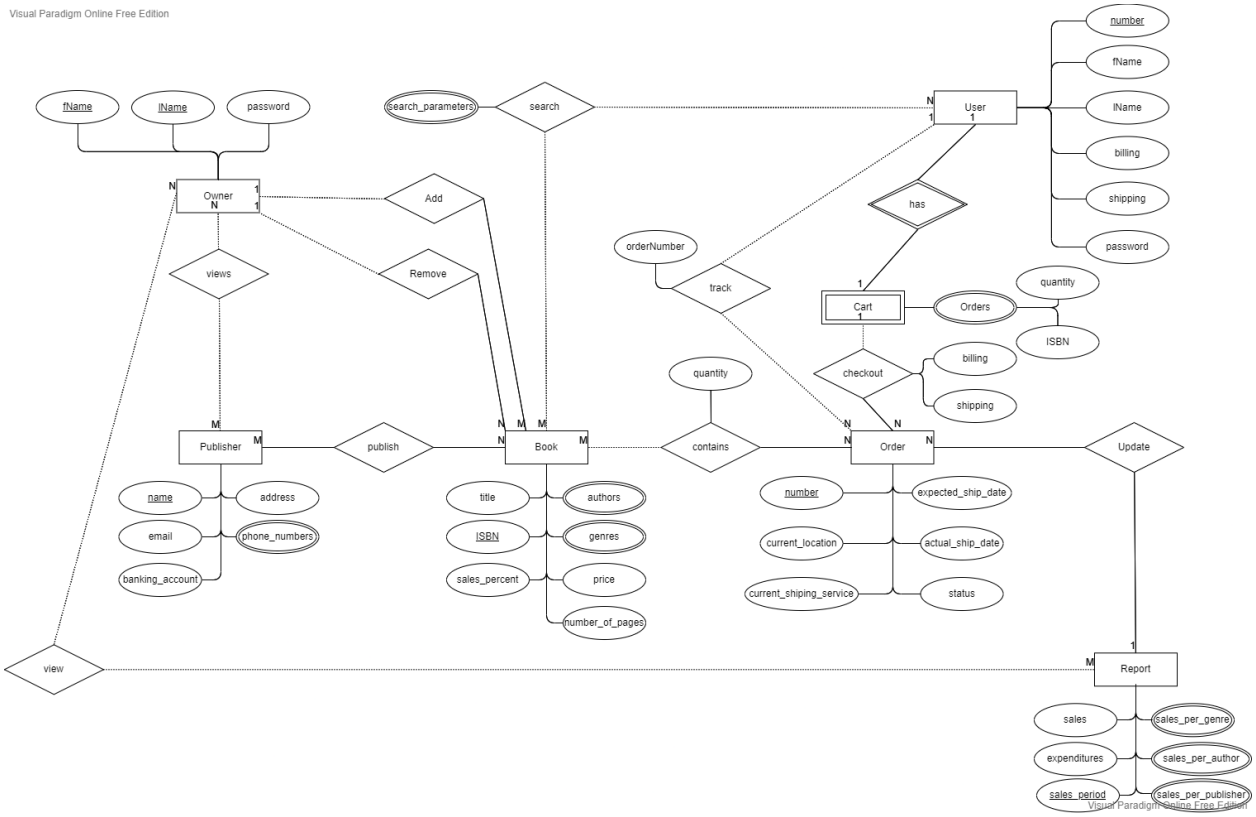
Dustin Doyle Final project

2.1 Conceptual Design:



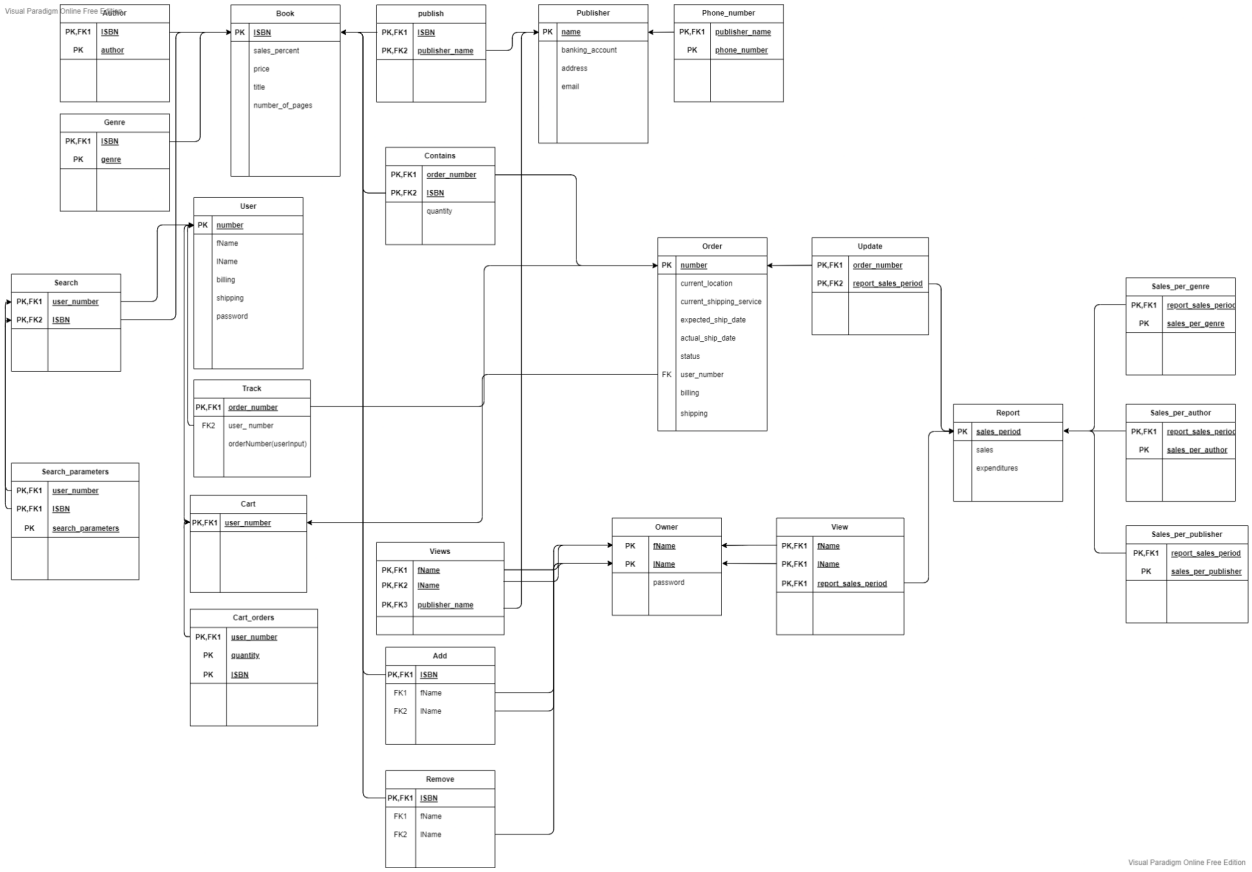
Assumptions in ER model:

1. All users have a single cart.
2. Not all books must be searched, not all users have to search.
3. Not all orders must be tracked by its user.
4. Users and books can have multiple searches.
5. The cart is a weak entity that uses the User number.
6. Not all users checkout their cart.
7. Cart is made up of multivalue orders attribute which is a composite attribute of quantity and ISBN which will be used later to generate an order entity. It should be noted that the user's cart entity will be destroyed on checkout.
8. Each book can have multiple authors, and multiple genres.
9. Each publisher can have multiple phone numbers.
10. Only the user who placed an order can track it using the correct matching order number. This means that other users can not track the order.
11. If the user has made multiple orders then they can track multiple orders.
12. All Publishers must have published a book.
13. There can be more than one Owner.
14. All Owners can view any number of publishers.
15. Publishers are not dependent on owners.
16. That users can place multiple orders.

17. The various reports can be attributes of one large report per sales period.
18. User's number is the user's primary key.
19. That Owner's first and last name are a compound primary key.
20. That Publisher's name is the primary key of the publisher.
21. That ISBN is the book's primary key.
22. That order number is the order's primary key.
23. That a report is made for a unique sales period as in date range that includes day, week, and year. Thus I used date as the primary key of the report.
24. That any owner can access any report from old ones to current ones.
25. That report also does a report for sales by publisher.
26. That publishers existence is dependent on the books stocked, meaning no need for a direct add or remove relationship between owners and publishers like with books.
27. Not all books must be removed. Not all owners need to remove or add books.
28. All books must not be removed thus are dependent on remove, and all books must have been added by an owner at some point thus dependent on add.
29. As a book can only be added or removed once only one owner can remove or add a book.
30. Not all owners must view a report.
31. Assumes the user's cart can not have multiple orders with the same values in them, this means two of the same books of the same number in one's shopping cart.

2.2 Reduction to Relational Schemas:

For some I changed the names to reflect the entity they come from to help clarify things like for example using report_ in front of the various multi value attributes report as this helps to make things more readable and easier to follow.
This is the diagram:



Which can be turned into this long list of relation schemas:

Owner(fName, lName, password)
PK=(fName, lName)

User(number, fName, lName, billing, shipping, password)
PK=(number)

Book(ISBN, sales_percent, price, title, number_of_pages)
PK=(ISBN)

Author(ISBN, author)
PK=(ISBN, author)
FK=(ISBN)

Genre(ISBN, genre)
PK=(ISBN, genre)
FK=(ISBN)

Publisher(name, banking_account, address, email)
PK=(name)

Publish(ISBN, publisher_name)
PK=(ISBN, publisher_name)
FK=(ISBN, publisher_name)

Phone_number(publisher_name, phone_number)
PK=(publisher_name, phone_number)
FK=(publisher_name)

Contains(order_number, ISBN, quantity)
PK=(order_number, ISBN)
FK=(order_number, ISBN)

Order(number, current_location, current_shipping_service, expected_ship_date,
actual_ship_date, status, user_number, billing, shipping)
PK=(number)
FK=(user_number)

Update(order_number, report_sales_period)
PK=(order_number, report_sales_period)
FK=(order_number, report_sales_period)

Report(sales_period, sales, expenditures)
PK=(sales_period)

Sales_per_genre(report_sales_period, sales_per_genre)
PK=(report_sales_period, sales_per_genre)
FK=(report_sales_period)

Sales_per_author(report_sales_period, sales_per_author)
PK=(report_sales_period, sales_per_author)
FK=(report_sales_period)

Sales_per_publisher(report_sales_period, sales_per_publisher)
PK=(report_sales_period, sales_per_publisher)
FK=(report_sales_period)

View(fName, lName, report_sales_period)
PK=(fName, lName, report_sales_period)
FK=(fName, lName, report_sales_period)

Views(fName, lName, publisher_name)
PK=(fName, lName, publisher_name)
FK=(fName, lName, publisher_name)

Add(ISBN, fName, lName)
PK=(ISBN)
FK=(ISBN, fName, lName)

Remove(ISBN, fName, lName)
PK=(ISBN)
FK=(ISBN, fName, lName)

Track(order_number, user_number, orderNumber(userinput))
PK=(order_number)
FK=(order_number, user_number)

Cart(user_number)
PK=(user_number)
FK=(user_number)

Cart_orders(user_number, quantity, ISBN)
PK=(user_number, quantity, ISBN)
FK=(user_number)

Search(user_number, ISBN)
PK=(user_number, ISBN)
FK=(user_number, ISBN)

Search_parameters(user_number, ISBN, search_parameters)
PK=(user_number, ISBN, search_parameters)
FK=(user_number, ISBN)

2.3 Normalization of Relation Schemas:
The set of functional dependencies for my database:

In the Book schema:
ISBN → sales_percent, price, title, number_of_pages

In the Publisher schema:
name → banking_account, address, email

In the Contains schema:
order_number, ISBN → quantity

In the User schema:
number → fName, lName, billing, shipping, password

In the Track schema:
order_number → user_number, orderNumber(userInput)

In the Add schema:
ISBN → fName, lName

In the Remove schema:
ISBN → fName, lName

In the Order schema:
number → current_location, current_shipping_service, expected_ship_date, actual_ship_date, status, user_number, billing, shipping
Also it should be noted that some of these attributes under certain conditions could be additional functional dependencies but with the data I plan to use in these attributes they will be independent of one another. For example status could be dependent on shipping date, but I plan on status being a more general thing as in for example status of order could be canceled, it could be order fulfilled, it can also be used to report errors such as order information lost, order can not be fulfilled, etc. Thus it is not dependent on the other attributes causing any normalization errors. If these attributes were dependent I would need to create a new tracking info entity in my ER model and the subsequent relational schemas.

In the Owner schema:
fName, lName → password

In the Report schema:
sales_period → sales, expenditures

The rest of the schema's do not have any functional dependencies.

Using these we now need to do the normal form tests of our functional dependencies to find out for sure whether or not the schemas have been normalized to a high enough standard. That standard is whether or not they are in BCNF. Thus we will test that all of the schemas with functional dependencies are in BCNF.

The Book schema:
First we must ensure that Book is in BCNF. We have one functional dependency ISBN → sales_percent, price, title, number_of_pages. Given this it's clear that ISBN is our minimum candidate key, as it solely determines all the other attributes based on our single functional dependency. Due to their being only one functional dependency this represents that all of our functional dependencies left sides aka α are a superkey of our relation schema. Thus Book satisfying the conditions for BCNF.

The Publisher schema:
Test that Publisher is in BCNF. Once again we have just one functional dependency that being: name → banking_account, address, email. From this dependency we can see that the name attribute is clearly a superkey, and as it's just one value it's a candidate key. We have just one functional dependency making it clear once again that for all our functional dependencies in Publisher α is a superkey for the relation.

The Contains schema:
Test that Contains is in BCNF. Once again we have just one functional dependency that being: order_number, ISBN → quantity. This one is unique compared to our other relational schemas that also have functional dependencies as it has a composite determinant. However just as with the previous functional dependencies we can see that it is a superkey of the relation. Meaning that once again the relation is in BCNF.

The User schema:
Test that User is in BCNF. Once again we have just one functional dependency that being: number → fName, lName, billing, shipping, password. We see from this functional dependency that number determine all of the attributes meaning that number is a superkey, and also a candidate key due to being only a single attribute. Thus meaning that α is a superkey for all functional dependencies of User. Therefore User is also in BCNF.

The Track schema:
Once again we have just one functional dependency that being: order_number → user_number, orderNumber(userInput). Once again I must state that this orderNumber refers to the user input. In retrospect I should probably rename it to user input instead of orderNumber(userInput), so I will fix this for the final relational schema diagram. Now we can see that using our functional dependency we can get all attributes using order_number of the Track schema this means that our functional dependencies determinant is a superkey. As we have just one functional dependency it means we fulfill the BCNF requirements meaning that the Track schema is in BCNF.

The Add schema:
We have just one functional dependency that being: ISBN → fName, lName. From our functional dependency we can see that ISBN is our candidate key which means it is also a superkey. As we have just one functional dependency it means that for all functional dependencies the determinants are superkeys which fulfills the BCNF requirements.

The Remove schema:
We have just one functional dependency that being: ISBN → fName, lName. As you can see this schema is identical to the add schema however it represents the remove instead of add event. Just as before our functional dependency means that ISBN is our candidate key which means it is also a superkey. As we have just one functional dependency it means that for all functional dependencies the determinants are superkeys which fulfills the BCNF requirements.

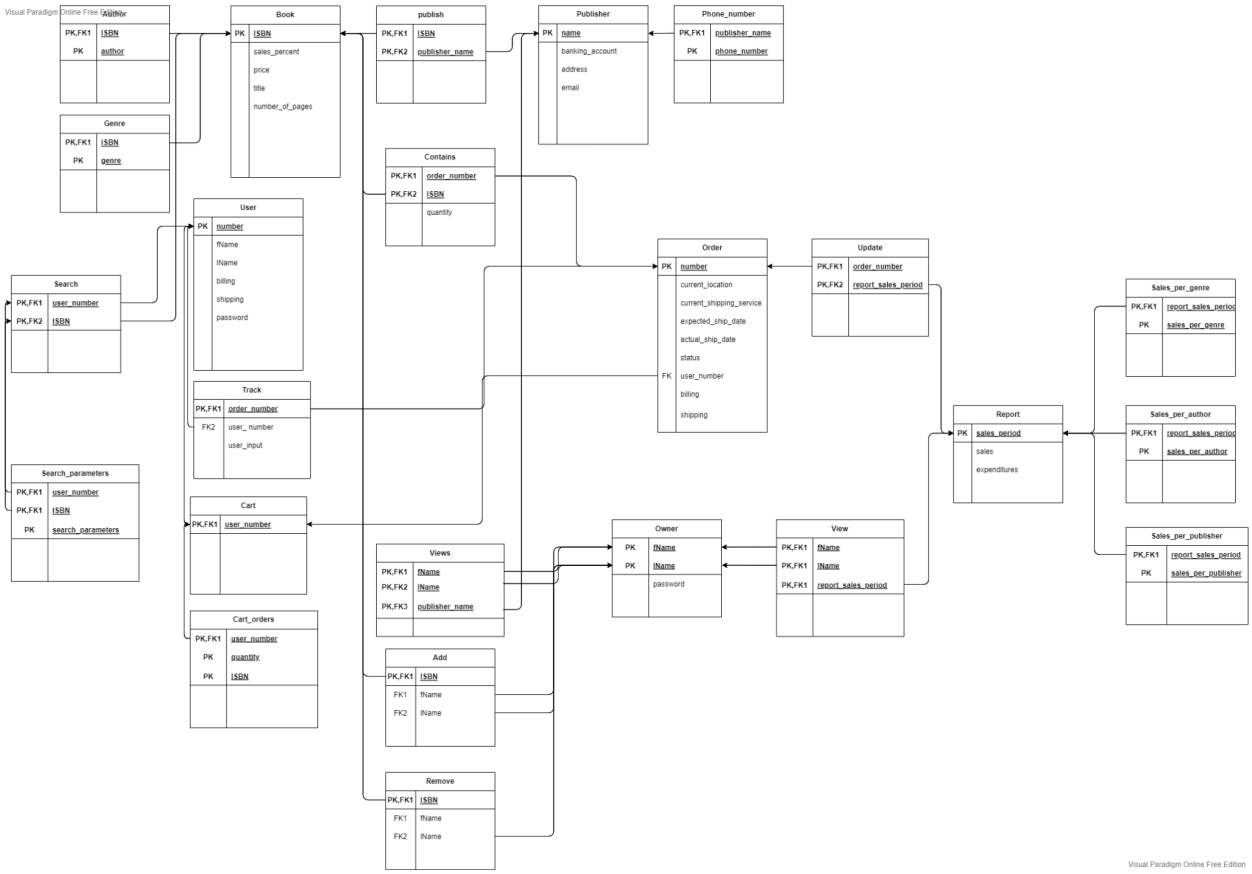
The Order schema:
We have just one functional dependency that being: number → current_location, current_shipping_service, expected_ship_date, actual_ship_date, status, user_number, billing, shipping. Using our functional dependency we can see that number is a superkey, as we have just one functional dependency it means that all of our determinants are superkeys which fulfills the BCNF requirements.

The Owner schema:
We have just one functional dependency that being: fName, lName → password. Once again using our functional dependency we get all attributes of the schema showing that it is in fact a superkey. This one is also a composite superkey. However this does not affect the BNCF rule thus we once again see that for all our functional dependencies aka just our one and only functional dependency the determinants are superkeys which fulfills the BCNF requirements.

The Report schema:
We have just one functional dependency that being: sales_period → sales, expenditures. Just as with all of the other schemas using our functional dependency we can get all attributes using our determinants which means that the determinant is a superkey. Since we have just one functional dependency it means that our schema fulfills the BCNF requirements.

Schemas without any functional dependencies are automatically in BNCF as they have no functional dependencies and they can not be unionized to any of the other schemas without changing the functionality of the database. Thus all of our relational schemas are already in BCNF without any changes. This means I can carry on using my current relational schemas without issue.

2.4 Database Schema Diagram:
Because I already made the database diagram as I converted my ER model into relational schemas and because nothing needed to be changed other than renaming Track's orderNumber(userinput) to user_input. Thus this is my final schema diagram of my bookstore database:



2.5 Implementation:
Changes and notes:
When implementing this database model I did not use the owner entity and instead used a user with an attribute type which determines if the user is customer or owner. Maybe in the future it could be used to implement additional types such as staff members.

Additionally I had to change the table name Order to BookOrder because Order is already defined by sql.

Changed Update to Update_Report as Update was also already defined by sql.

Realized during implementation that for more Sales_per parts that are inside of the reports that the Sales_per attributes need to split into two values the sales and the per part. For example I need in Sales_per_genre to split the Sales_per_genre into a sales as in money and a genre. For some reasons I had only thought of just storing the money values not needing a way to identify the entity that they pair with. Additionally had to remove the report_sales from primary

keys as that caused a bunch of separate reports not one report that is added to like I wanted thus new schemas for them look like this:

Sales_per_genre(report_sales_period, sales_per_genre, genre)
PK=(report_sales_period, genre)
FK=(report_sales_period, genre)

Sales_per_author(report_sales_period, sales_per_author, author)
PK=(report_sales_period, author)
FK=(report_sales_period)

Sales_per_publisher(report_sales_period, sales_per_publisher, publisher)
PK=(report_sales_period, publisher)
FK=(report_sales_period)

Had to change View and Views to View_Report and View_Publisher respectively because they were already defined.

Also had to change the names of schemas Add and Remove to be Add_Book, and Remove_Book since once again add and remove were already defined.

All of my relational schemas tables are created in the bookstore.sql file

Currently the program will make an order that does not contain anything if you try to checkout your cart without putting anything into it. I should probably fix this but I am running out of time so I will need to leave it for now.

Description of application architecture:
First the file called bookstore.sql is used to generate the database tables into a sql database. Next my code requires you to use npm install to download all the various modules needed to run my server and its routes. The main module's are express which is used to handle the server communications and sending information and webpages to the users browser. Then sqlite3 which is used to implement the SQL database in my web application. The general layout of my program is based on a web app that was provided to me in class as a part of comp2406. This web application however does not run using regular buttons and post/get requests but instead uses various routes determined by the imputed URL. Many of the web applications functions are based on the ability to parse the URL into varios queries and chunks of information. This can be seen from the selection of basic tests that my program lists on startup:

```
console.log('http://localhost:3000/addToCart?ISBN=42069&quantity=25')
        console.log('http://localhost:3000/index.html')
        console.log('http://localhost:3000/users')
```

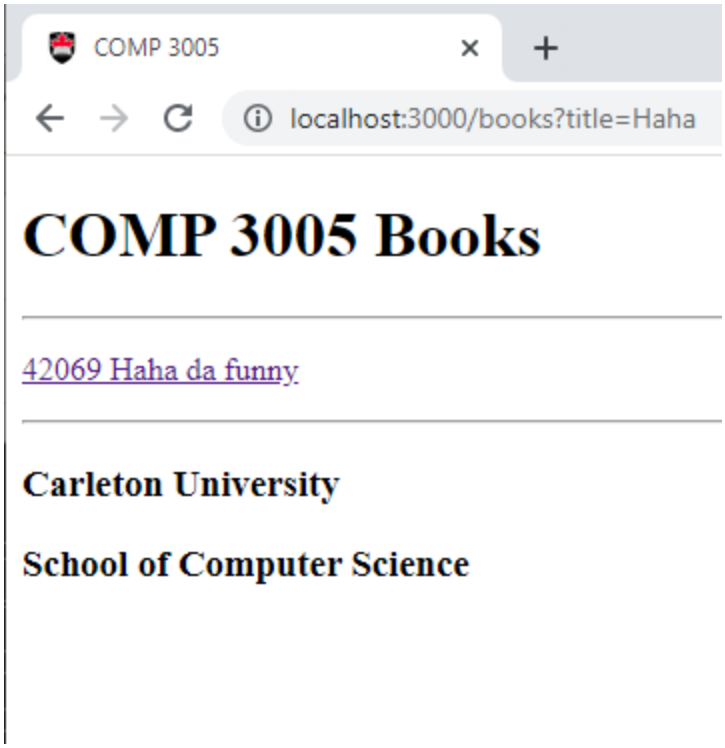
```
console.log('http://localhost:3000/checkoutCart?billing=ScamBank&shipping=
ScamMeStreet')
        console.log('http://localhost:3000/books?title=Haha')
        console.log('http://localhost:3000/removeBook?title=Haha')

console.log('http://localhost:3000/addBook?title=newBook&ISBN=100001&sales
_percent=24&price=15.99&number_of_pages=39')
        console.log('http://localhost:3000/book/42069')
        console.log('http://localhost:3000/viewReports')
    }
```

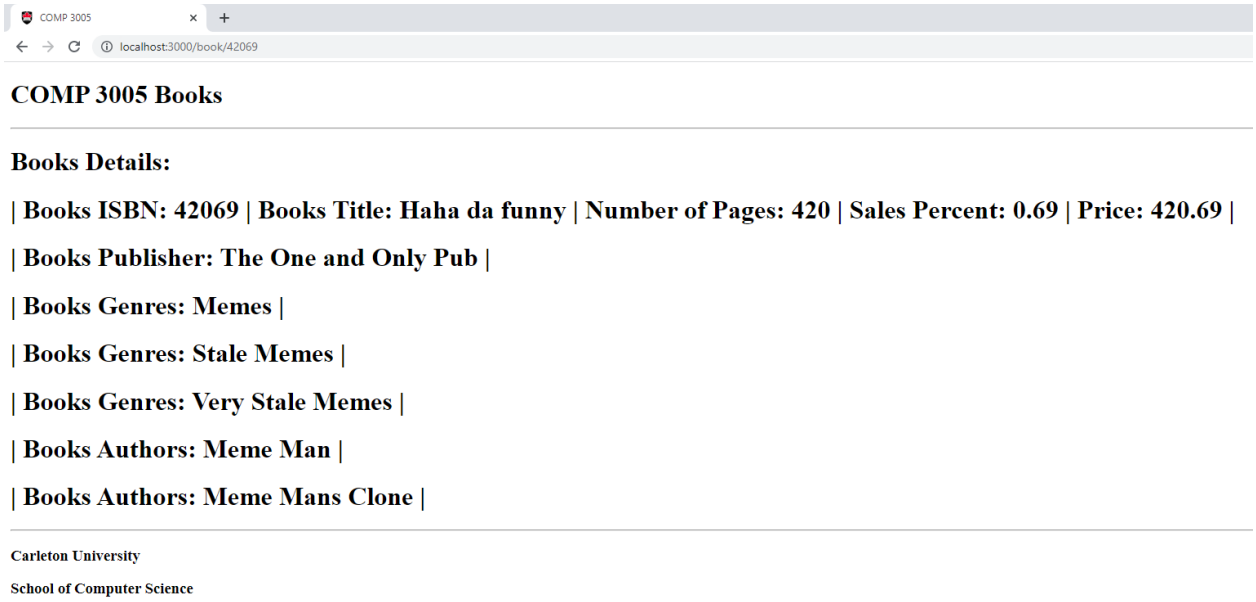
We can see that the functions of each of these URLs are hinted to in their makeup for example /addToCart adds to cart using the ISBN of the book and the quantity of said book, and /checkoutCart provides us with a quarry that we input our billing and shipping info into.

My web application supports authentication, which enables the adding and removing of books, the viewing of reports and users. The none authentication requiring activities which are currently supported is adding books to cart, viewing the homepage, checking out users cart, viewing a list of all books that match or partial match with the inputted parameters, then viewing a book on detail is also supported which will show additional information about the book other then just book number and book name like in the long list. You can either click on the number and name which is a link or type in http://localhost:3000/book/<booknumber> with booknumber being the ISBN value.

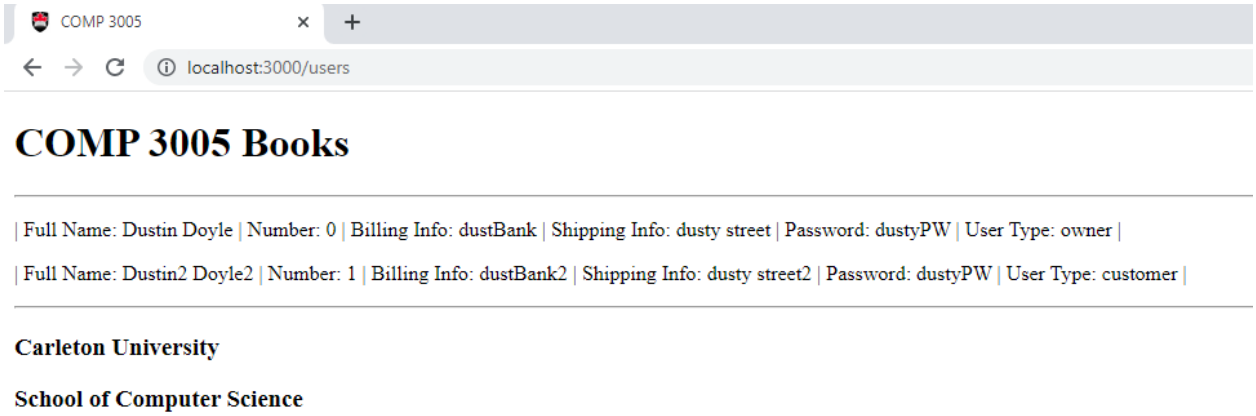
To better show what my code does I will include these images:



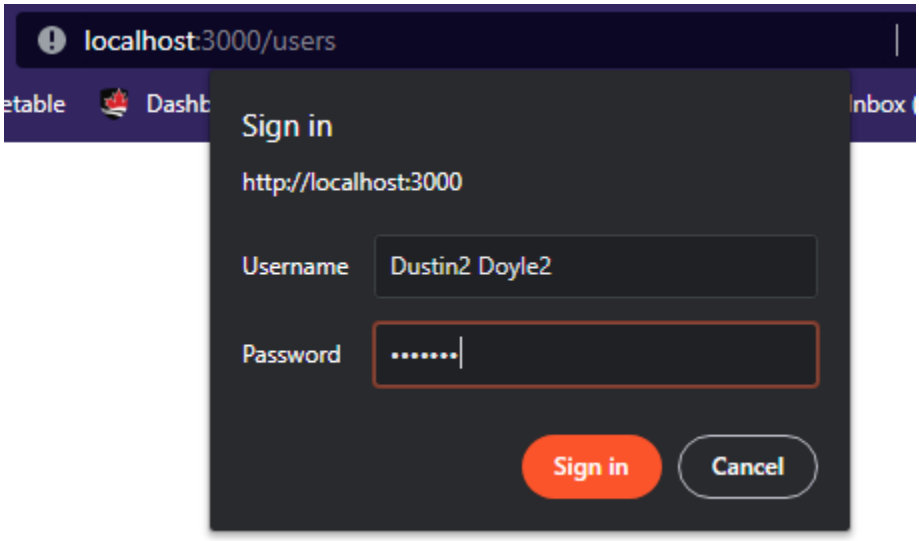
The search list looks like this and you can be a user type customer or owner while using this route.



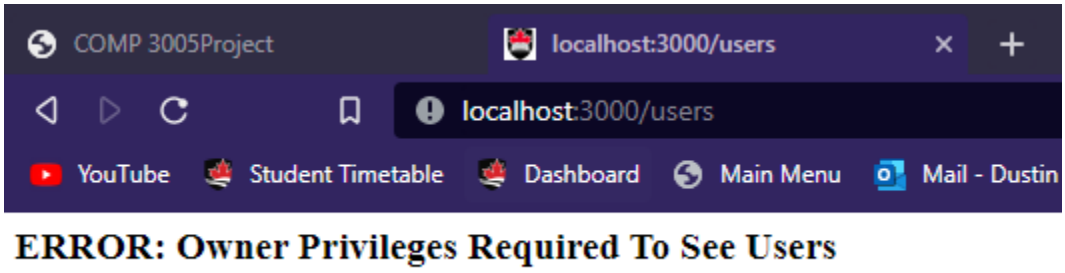
This is what it looks like when we click on the link or use the url to view the detailed view which includes the list of Genres and Authors, along with the publisher and the other information about the book.



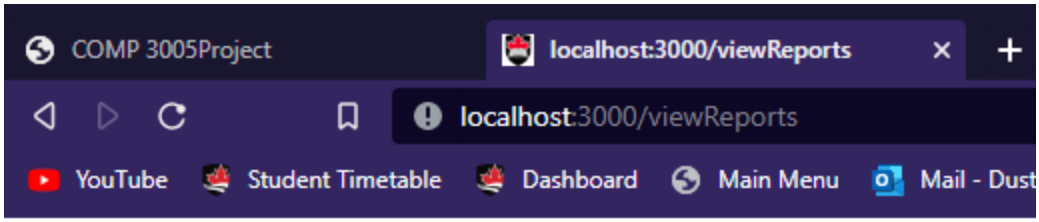
This is what it looks like when you are the owner and can view the database's users.



This is what user authentication looks like and you you are authenticated as long as you do not close all of your browser tabs meaning even if you close the tab it will remember you and keep you authenticated.

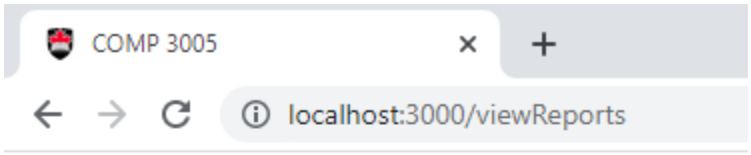


When you are not logged in as a user that is a Owner and try to view users.



ERROR: Owner Privileges Required To See Reports

Same as before but for reports.



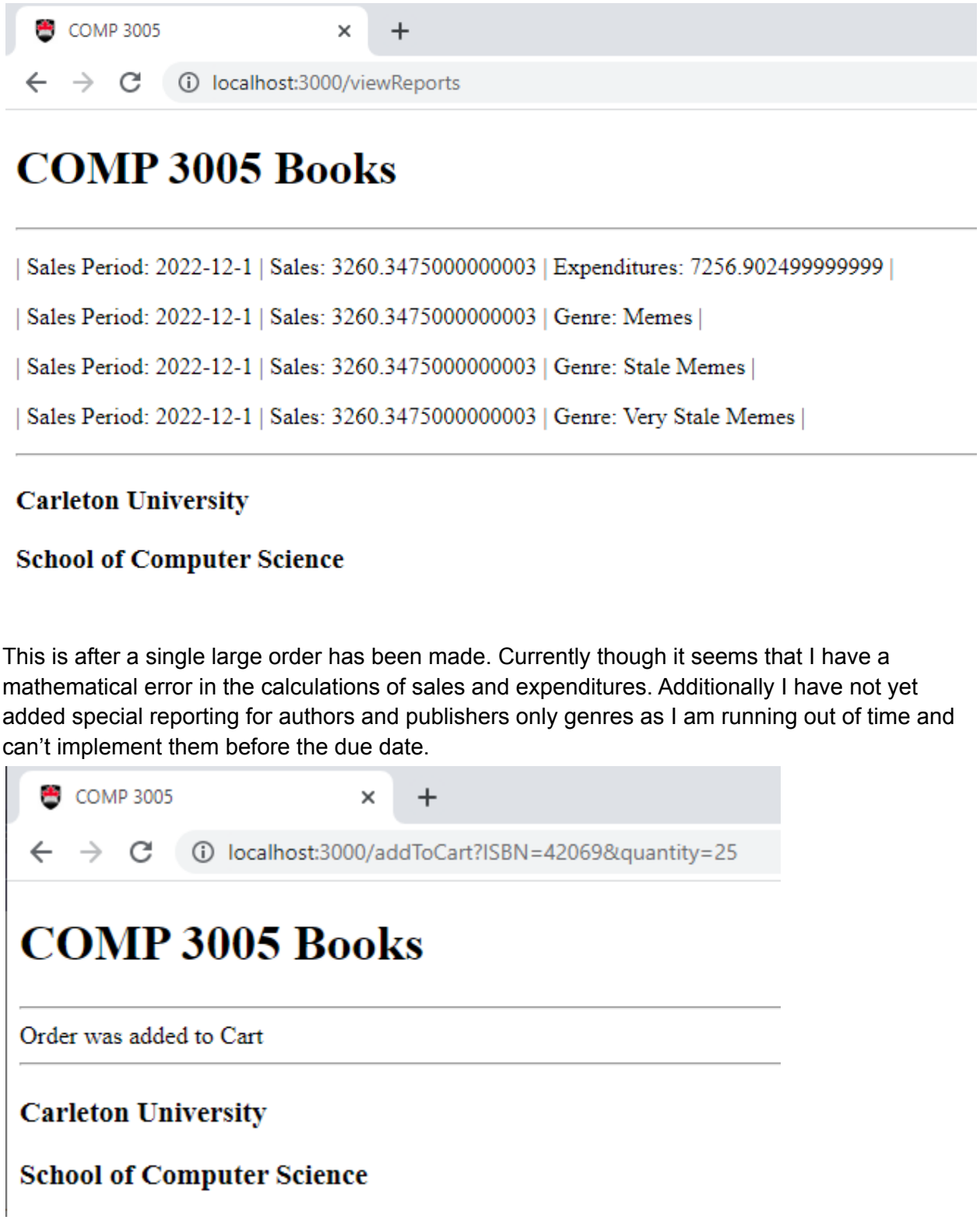
COMP 3005 Books

| Sales Period: 2022-12-1 | Sales: 0 | Expenditures: 0 |

Carleton University

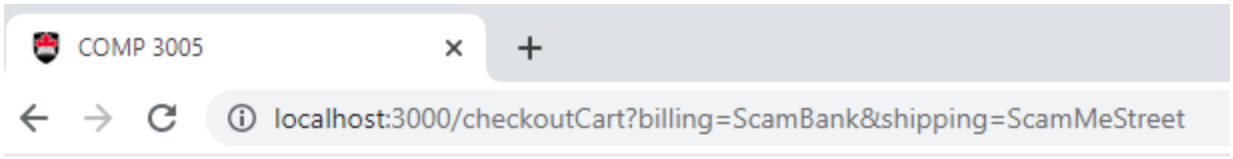
School of Computer Science

This shows reports when there are no orders yet.



This is after a single large order has been made. Currently though it seems that I have a mathematical error in the calculations of sales and expenditures. Additionally I have not yet added special reporting for authors and publishers only genres as I am running out of time and can't implement them before the due date.

This is what adding to the cart looks like so far I haven't implemented a way to tell you when it fails to add to cart.



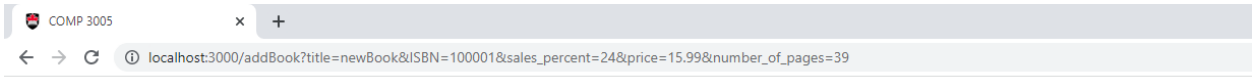
COMP 3005 Books

Cart has been checked out

Carleton University

School of Computer Science

This is what checking out currently looks like and of course you can see the query which is how we perform actions on this web application.



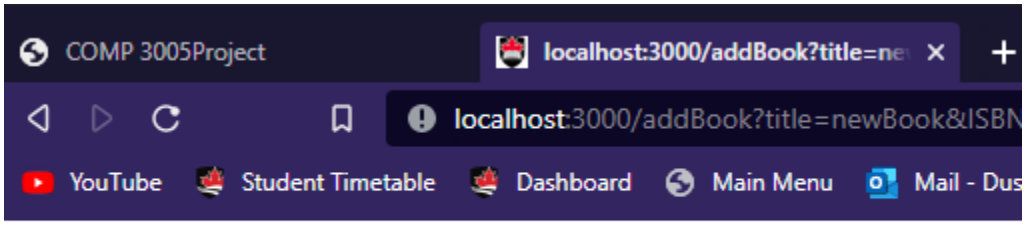
COMP 3005 Books

Added the book created by this url query: "/addBook?title=newBook&ISBN=100001&sales_percent=24&price=15.99&number_of_pages=39"

Carleton University

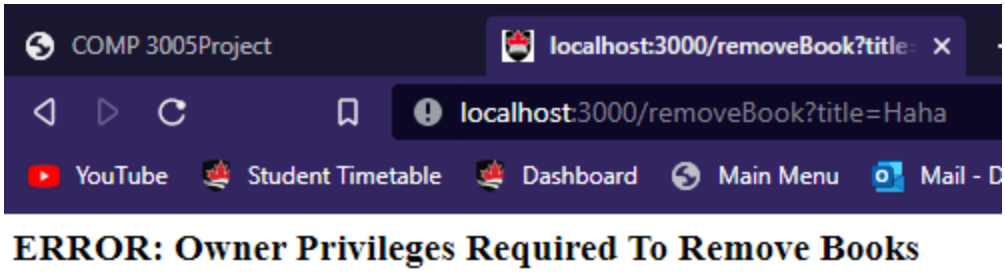
School of Computer Science

This is what it looks like when we add a book to the database as an owner type user.

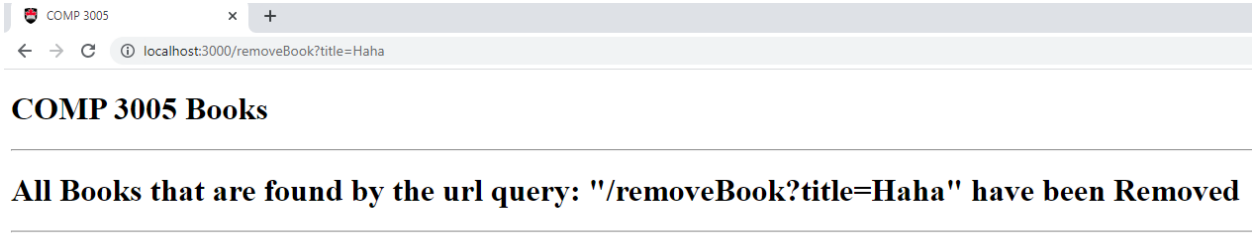


ERROR: Owner Privileges Required To Add Books

This is what it looks like if you are not an owner user.



This is what a non-owner's attempt at removing a book looks like.



This is the owner removing all books that are found with the removebook query. This uses the same finding method as the search list so it will in fact remove all books whose title contains haha regardless of capitalization or not.

Now in order to support all of this it required the use of many, many, many database queries mainly in the checkout route as we need to quarry and update a lot of data across many different relational tables in our database. But as can be seen by the various images even though I could not fully complete everything I wanted to I was able to get the majority of things working nicely. Though one thing I found was that there are a good number of unnecessary tables which I probably could have done without as I have not properly utilized a lot of them for example there is no need to store the add and remove info unless we want to log that information. Which to be fair could likely be very handy if you accidentally delete something or someone keeps adding books you don't want to the database, but in the current state a lot of my relational schemas have not been used outside of the major ones.

2.6 Bonus Features:
When it comes to bonus features the main one is likely the use of the web application as I think there would have been plenty of easier ways to implement the database then trying to use this url parsing express web application. Additionally the support of multiple removes from a single query is neat. Then there is the ability to view users which is another little extra feature. I also like that since the program is an actual web application there is some greater level of interaction between the users and the program.

2.7 GitHub Repository
Link: https://github.com/DustinDoyleCarleton/Comp3005Final

2.8 Appendix I:

I might not quite be awake at 9am so maybe a bit later at something like 10:30am I should be awake by then. Thus I will probably be available from about 10:30am to 5pm on December 12th unless something changes.