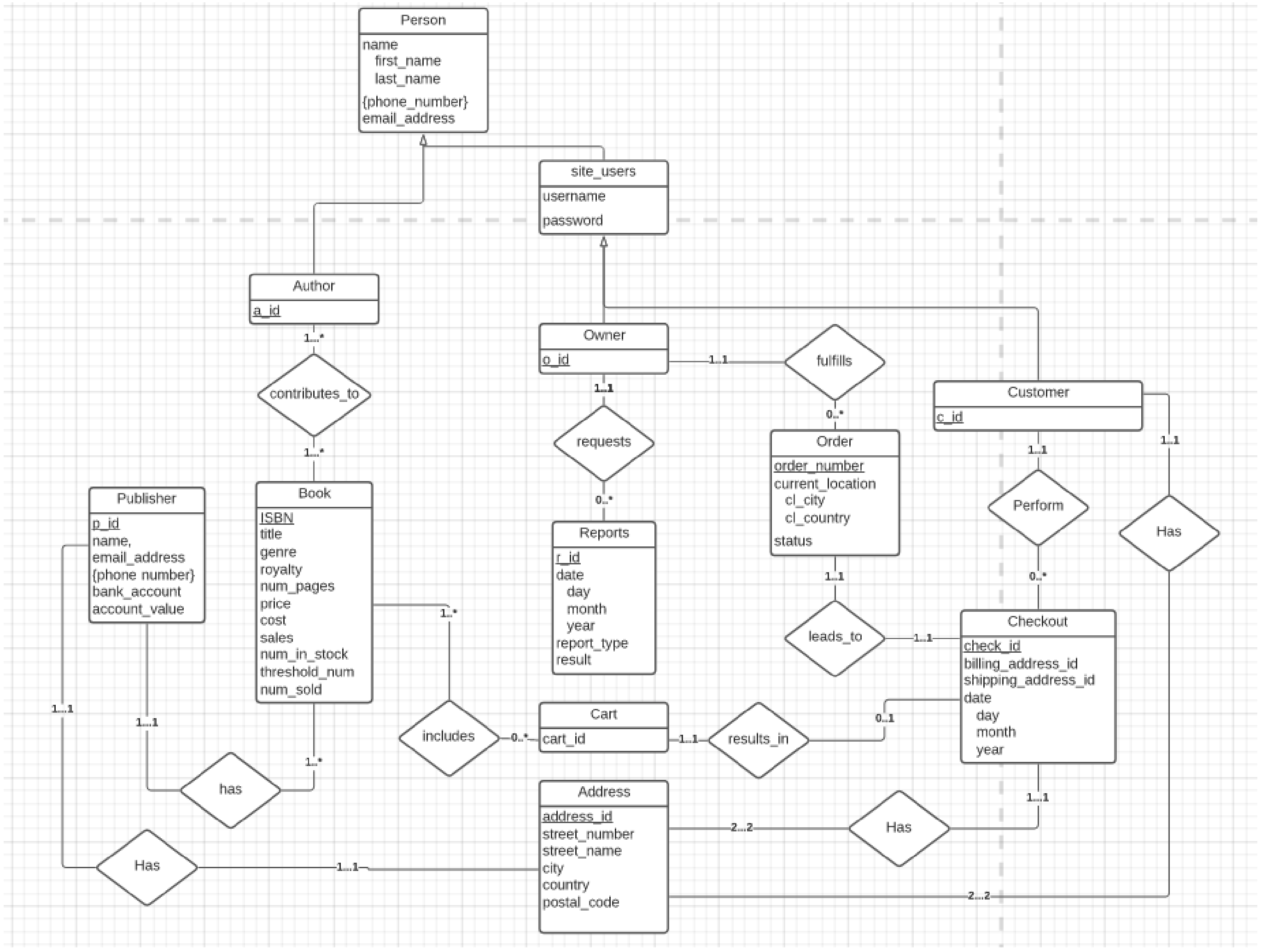
Conceptual Design

ERD



Assumptions

- Carts are only every generated after user adds a book
- Therefore, Carts must have atleast one book
- A single Book has a single publisher
- All publishers and all persons could have many phone numbers
- Not all Carts will be converted to Checkouts
- All Checkouts are converted to Orders (no failed checkouts)
- A Book can have many Authors
- Authors, Owners, and Customers all share Person attributes
- Owners must request Reports
- An Owner is responsible for fulfilling all orders
- A customer is only added to the DB if they choose to register
- A customer will only be prompted to register at Checkout
- An owner must log in before accessing the owner menu
- A customer can opt not to check out, even after registering
- A customer must be registered in order to Checkout

- An email must belong to one and only one customer. Two people cannot share an email.
- A customer has two addresses attributed to them. A billing and shipping address.
- A checkout also has two addresses, a billing and shipping address.
- A publisher only has one address, their billing address.

Reduction to Relational Schema (No Normalization)

'*' = FK

book(ISBN, p_id*, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

cart(c_id)

cart_books(c_id, ISBN)

author_phone_number(a_id, phone_number)

owner_phone_number(o_id, phone_number)

customer_phone_number(c_id, phone_number)

author(a_id, first_name, last_name, email_address)

book_authors(ISBN, a_id)

publisher(p_id, address_id*, name, email_address, bank_account , account_value)

publisher_phone_number(p_id, phone_number)

owner(o_id, first_name, last_name, email_address, username, password)

reports(r_id, o_id*, day, month, year, report_type, result)

order(order_number , o_id*, check_id*, cl_city, cl_country, status)

customer(c_id, shipping_address_id*, billing_address_id*, first_name, last_name, email_address, username, password)

checkout(check_id, billing_address*, shipping_address*, c_id*, cart_id*, day, month, year)

address(address_id, street_number, street_name, city, country, postal_code)

Normalization of Relational Schemas

Functional Dependencies

F_{book} = {

ISBN → (p_id, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

}

F_{author} = {

a_id → (first_name, last_name, email_address),

email_address → (first_name, last_name)

}

$F_{publisher}$ = {

p_id → (address_id, name, email_address, bank_account , account_value),

email_address → name,

bank_account → account_value

}

F_{owner} = {

o_id → (first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → o_id

}

$F_{reports}$ = {

r_id → (o_id, day, month, year, report_type, result)

}

F_{order} = {

order_number → (o_id, check_id, cl_city, cl_country, status)

}

$F_{customer}$ = {

c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → c_id

}

$F_{checkout}$ = {

check_id → (shipping_address_id, billing_address_id, c_id, cart_id, day, month, year)

}

$F_{address}$ = {

address_id → (street_number, street_name, city, country, postal_code),

postal_code → (city, country)

}

Normalization into 3NF

1. If there exists FD's in F_c such that α $_1$ → β $_1$ and α $_1$ → β $_2$, remove them all and replace them with their union α $_1$ → β $_1$ β $_2$
2. Test each FD in F_c for extraneous attributes, remove those that are found from F_c
3. For each FD in F_c, derive R_i = α β
4. Check that atleast one of the derived relations R_i contains a candidate key for R. If none do, derive one that does
5. Check if any relation R_j is a subset of any other relation R_k, remove R_j

---

Book

book = {ISBN, p_id, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold}

F = {

ISBN → (p_id, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

}

F_c = F

F_c = {

ISBN → (p_id, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

}

1. Since there exists only 1 FD, no union can be formed
2. Since there exists only 1 FD, none of the attributes can be extraneous, therefore nothing to remove from F_c

3. Therefore, R_1 = {ISBN, p_id, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold}
4. ISBN from R_1 is a candidate key for the relation book
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation book therefore satisfies 3NF, and is in good normal form. No decomposition required.

book(<u>ISBN</u>, p_id*, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

Cart

cart = {cart_id}

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {cart_id} as cart_id → cart_id is the only inferrable (trivial) FD
4. cart_id from R_1 is a candidate key for the relation cart
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation cart therefore satisfies 3NF, and is in good normal form. No decomposition required.

cart(<u>cart_id</u>)

Cart_Books

cart_books = (cart_id, ISBN)

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {cart_id, ISBN} as (cart_id, ISBN) → (cart_id, ISBN) is the only inferrable (trivial) FD
4. (cart_id, ISBN) from R_1 is a candidate key for the relation cart_books
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation cart_books therefore satisfies 3NF, and is in good normal form. No decomposition required.

cart_books(cart_id, ISBN)

Author_Phone_Number

author_phone_number = (a_id, phone_number)

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {a_id, phone_number} as (a_id, phone_number) → (a_id, phone_number) is the only inferrable (trivial) FD
4. (a_id, phone_number) from R_1 is a candidate key for the relation author_phone_number
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation author_phone_number therefore satisfies 3NF, and is in good normal form. No decomposition required.

author_phone_number(a_id, phone_number)

Owner_Phone_Number

owner_phone_number = (o_id, phone_number)

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {o_id, phone_number} as (o_id, phone_number) → (o_id, phone_number) is the only inferrable (trivial) FD
4. (o_id, phone_number) from R_1 is a candidate key for the relation owner_phone_number
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation owner_phone_number therefore satisfies 3NF, and is in good normal form. No decomposition required.

owner_phone_number(o_id, phone_number)

Customer_Phone_Number

customer_phone_number = (c_id, phone_number)

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {c_id, phone_number} as (c_id, phone_number) → (c_id, phone_number) is the only inferrable (trivial) FD
4. (c_id, phone_number) from R_1 is a candidate key for the relation customer_phone_number
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation customer_phone_number therefore satisfies 3NF, and is in good normal form. No decomposition required.

customer_phone_number(<u>c_id</u>, <u>phone_number</u>)

Author

author = {a_id, first_name, last_name, email_address}

F = {

a_id → (first_name, last_name, email_address),

email_address → (first_name, last_name))

}

F_c = F

F_c = {

a_id → (first_name, last_name, email_address),

email_address → (first_name, last_name)

}

1. No two FD's have the same α , therefore no union can be formed

2. Testing for Extraneous Attributes

   - Case for a_id → (first_name, last_name, email_address)

     - Is first_name extraneous in a_id → (first_name, last_name, email_address) ?

    - Replace a_id → (first_name, last_name, email_address) with a_id → (last_name, email_address)
    - a_id^+ = { }
        1. a_id → a_id : a_id^+ = { a_id }
        2. a_id → (last_name, email_address) : a_id^+ = { a_id, last_name, email_address}
        3. email_address → (first_name, last_name) : a_id^+ = {last_name, email_address, first_name}

        - Since first_name can be found in a_id^+ , it is extraneous. We will replace the FD a_id → (first_name, last_name, email_address) with the FD a_id → (last_name, email_address) in F_c
            - Therefore, the new adjusted Canonical Cover

        F_c = {

        a_id --> (last_name, email_address),

        email_address --> (first_name, last_name),

        }
- Is last_name extraneous in a_id → (last_name, email_address) ?
    - Replace a_id → (last_name, email_address) with a_id → (email_address)
    - a_id^+ = { }
        1. a_id → a_id : a_id^+ = { a_id }
        2. a_id → (email_address) : a_id^+ = { a_id, email_address }
        3. email_address → (first_name, last_name) : a_id^+ = {email_address, first_name, last_name}

        - Since last_name can be found in a_id^+ , it is extraneous. We will replace the FD a_id → (last_name, email_address) with the FD a_id → (email_address) in F_c
            - Therefore, the new adjusted Canonical Cover

        F_c = {

        a_id --> (email_address),

        email_address --> (first_name, last_name),

        }
- Is email_address extraneous in a_id → (email_address) ?

- Since there is only 1 attribute on either the LHS and RHS, neither can be extraneous.

- Case for email_address → (first_name, last_name)
  - Is first_name extraneous in email_address → (first_name, last_name) ?
    - Replace email_address → (first_name, last_name) with email_address → (last_name)
    - email_address$^+$ = { }
      1. email_address → email_address : email_address$^+$ = { email_address }
      2. email_address → (last_name) : email_address$^+$ = { email_address, last_name }
      - Nothing else can be inferred, since first_name is not in p_id$^+$, first_name is not an extraneous attribute, and must stay in the FD
  - Is last_name extraneous in email_address → (first_name, last_name) ?
    - Replace email_address → (first_name, last_name) with email_address → (first_name)
    - email_address$^+$ = { }
      1. email_address → email_address : email_address$^+$ = { email_address }
      2. email_address → (first_name) : email_address$^+$ = { email_address, first_name }
      - Nothing else can be inferred, since last_name is not in p_id$^+$, last_name is not an extraneous attribute, and must stay in the FD

3. Therefore, R_1 = {a_id, email_address} and R_2 = {email_address, first_name, last_day}

4. a_id from R_1 is a candidate key for the relation author

5. Neither of the above relations are subsets of one another, so they will both persist.

author(a_id, email_address*)

author_email(email_address, first_name, last_name)

Book_Authors

book_authors = (ISBN, a_id)

F = { }

F_C = F

F_C = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_C
3. Therefore, R_1 = {ISBN, a_id} as (ISBN, a_id) → (ISBN, a_id) is the only inferrable (trivial) FD
4. (ISBN, a_id) from R_1 is a candidate key for the relation book_authors
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation book_authors therefore satisfies 3NF, and is in good normal form. No decomposition required.

book_authors(<u>ISBN</u>, <u>a_id</u>)

Publisher

publisher = (p_id, address_id*, name, email_address, bank_account, account_value)

F = {

p_id → (address_id, name, email_address, bank_account , account_value),

email_address → (name),

bank_account → account_value

}

F_C = F

F_C = {

p_id → (address_id, name, email_address, bank_account , account_value),

email_address → (name),

bank_account → account_value

}

1. No two FD's have the same α , therefore no union can be formed
2. Testing for Extraneous Attributes
   - Case for p_id → (address_id, name, email_address, bank_account , account_value)
     - Is address_id extraneous in p_id → (address_id, name, email_address, bank_account , account_value) ?
       - Replace p_id → (address_id, name, email_address, bank_account , account_value) with p_id → (name, email_address, bank_account , account_value)
       - p_id^+ = { }

1. p_id → p_id : p_id$^+$ = { p_id }
2. p_id → (name, email_address, bank_account , account_value) : p_id$^+$ = { p_id, name, email_address, bank_account , account_value }

- Nothing else can be inferred, since address_id is not in p_id$^+$, address_id is not an extraneous attribute, and must stay in the FD

- Is name extraneous in p_id → (address_id, name, email_address, bank_account , account_value) ?
  - Replace p_id → (address_id, name, email_address, bank_account , account_value) with p_id → (address_id, email_address, bank_account , account_value)
  - p_id$^+$ = { }
    1. p_id → p_id : p_id$^+$ = { p_id }
    2. p_id → (address_id, email_address, bank_account , account_value) : p_id$^+$ = { p_id, address_id, email_address, bank_account , account_value }
    3. email_address → (name) : p_id$^+$ = { p_id, address_id, email_address, bank_account, account_value, name }

    - Since name can be found in p_id$^+$, it is extraneous. We will replace the FD p_id → (address_id, name, email_address, bank_account , account_value) with the FD p_id → (address_id, email_address, bank_account , account_value) in F_C
    - Therefore, the new adjusted Canonical Cover

  F_C = {

  ```
  p_id --> (address_id, email_address, bank_account ,
  account_value),

  email_address --> (name),

  bank_account --> account_value
  ```

  }

- Is email_address extraneous in p_id → (address_id, email_address, bank_account, account_value) ?
  - Replace p_id → (address_id, email_address, bank_account, account_value) with p_id → (address_id, bank_account, account_value)
  - p_id$^+$ = { }

1. p_id → p_id : p_id$^+$ = { p_id }
2. p_id → (address_id, bank_account, account_value) : p_id$^+$ = { p_id, address_id, bank_account, account_value }

- Nothing else can be inferred, since email_address is not in p_id$^+$, email_address is not an extraneous attribute, and must stay in the FD

- Is bank_account extraneous in p_id → (address_id, email_address, bank_account, account_value) ?
  - Replace p_id → (address_id, email_address, bank_account, account_value) with p_id → (address_id, email_address, account_value)
  - p_id$^+$ = { }
    1. p_id → p_id : p_id$^+$ = { p_id }
    2. p_id → (address_id, email_address, account_value) : p_id$^+$ = { p_id, address_id, email_address, account_value }
    3. email_address → name : p_id$^+$ = { p_id, address_id, email_address, account_value, name }

    - Nothing else can be inferred, since bank_account is not in p_id$^+$, bank_account is not an extraneous attribute, and must stay in the FD

- Is account_value extraneous in p_id → (address_id, email_address, bank_account, account_value) ?
  - Replace p_id → (address_id, email_address, bank_account, account_value) with p_id → (address_id, email_address, bank_account)
  - p_id$^+$ = { }
    1. p_id → p_id : p_id$^+$ = { p_id }
    2. p_id → (address_id, email_address, bank_account) : p_id$^+$ = { p_id, address_id, email_address, bank_account }
    3. email_address → name : p_id$^+$ = { p_id, address_id, email_address, bank_account, name }
    4. bank_account → account_value : p_id$^+$ = { p_id, address_id, email_address, bank_account, name, account_value }

    - Since account_value can be found in p_id$^+$, it is extraneous. We will replace the FD p_id → (address_id, email_address, bank_account, account_value) with the FD p_id → (address_id, email_address, bank_account) in F_c
    - Therefore, the new adjusted Canonical Cover

  F_c = {

```
p_id --> (address_id, email_address, bank_account),

email_address --> name,

bank_account --> account_value
```

}

- Case for email_address → name
  - Since there is only 1 attribute on either the LHS and RHS, no attributes in this FD can be extraneous.
- Case for bank_account → account_value
  - Since there is only 1 attribute on either the LHS and RHS, no attributes in this FD can be extraneous.

3. Therefore, R_1 = {p_id, address_id, email_address, bank_account}, R_2 = {email_address, name}, and R_3 = {bank_account, account_value }
4. p_id from R_1 is a candidate key for the relation book_authors
5. None of R_1, R_2, or R_3 are subsets of one another, so we must decompose the relation publsiher into three new relations. These will be the schemas for the new relations...

publisher (p_id, address_id*, email_address*, bank_account*)

publisher_email (email_address, name)

publisher_bank (bank_account, account_value)

Publisher_Phone_Number

publisher_phone_number = {p_id, phone_number}

F = { }

F_c = F

F_c = { }

1. Since there exists 0 FD's, no union can be formed
2. Since there exists 0 FD's, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {p_id, phone_number} as (p_id, phone_number) → (p_id, phone_number) is the only inferrable (trivial) FD
4. (p_id, phone_number) from R_1 is a candidate key for the relation publisher_phone_number

5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation publisher_phone_number therefore satisfies 3NF, and is in good normal form. No decomposition required.

publisher_phone_number(p_id, phone_number)

Owner

owner = {o_id, first_name, last_name, email_address, username, password}

F = {

o_id → (first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → o_id

}

F_c = F

F_c = {

o_id → (first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → o_id

}

1. No two FD's have the same α , therefore no union can be formed
2. Testing for Extraneous Attributes
   - Case for o_id → (first_name, last_name, email_address, username, password)
     - Is first_name extraneous in o_id → (first_name, last_name, email_address, username, password) ?
       - Replace o_id → (first_name, last_name, email_address, username, password) with o_id → (last_name, email_address, username, password)
       - o_id^+ = { }
         1. o_id → o_id : o_id^+ = { o_id }
         2. o_id → (last_name, email_address, username, password) : o_id^+ = { o_id, last_name, email_address, username, password }
         3. email_address → (first_name, last_name) : o_id^+ = { o_id, last_name, email_address, username, password, first_name }

    - Since first_name can be found in o_id^+ , it is extraneous. We will replace the FD o_id → (first_name, last_name, email_address, username, password) with the FD o_id → (last_name, email_address, username, password) in F_c

    - Therefore, the new adjusted Canonical Cover

  F_c = {

  ```
  o_id --> (last_name, email_address, username, password),

  email_address --> (first_name, last_name),

  (username, password) --> o_id
  ```

  }

- Is last_name extraneous in o_id → (last_name, email_address, username, password) ?

  - Replace o_id → (last_name, email_address, username, password) with o_id → (email_address, username, password)

  - o_id^+ = { }

    1. o_id → o_id : o_id^+ = { o_id }
    2. o_id → (email_address, username, password) : o_id^+ = { o_id, email_address, username, password }
    3. email_address → (first_name, last_name) : o_id^+ = { o_id, email_address, username, password, first_name, last_name}

    - Since last_name can be found in o_id^+ , it is extraneous. We will replace the FD o_id → (last_name, email_address, username, password) with the FD o_id → (email_address, username, password) in F_c

    - Therefore, the new adjusted Canonical Cover

  F_c = {

  ```
  o_id --> (email_address, username, password),

  email_address --> (first_name, last_name),

  (username, password) --> o_id
  ```

  }

- Is email_address extraneous in o_id → (email_address, username, password) ?

        - Replace o_id → (email_address, username, password) with o_id → (username, password)
        - o_id^+ = { }
            1. o_id → o_id : o_id^+ = { o_id }
            2. o_id → (username, password) : o_id^+ = { o_id, username, password }
            - Nothing else can be inferred, since email_address is not in o_id^+, email_address is not an extraneous attribute, and must stay in the FD
    - Is username extraneous in o_id → (email_address, username, password) ?
        - Replace o_id → (email_address, username, password) with o_id → (email_address, password)
        - o_id^+ = { }
            1. o_id → o_id : o_id^+ = { o_id }
            2. o_id → (email_address, password) : o_id^+ = { o_id, email_address, password }
            3. email_address → (first_name, last_name) : p_id^+ = {o_id, email_address, password, first_name, last_name }
            - Nothing else can be inferred, since username is not in o_id^+, username is not an extraneous attribute, and must stay in the FD
    - Is password extraneous in o_id → (email_address, username, password) ?
        - Replace o_id → (email_address, username, password) with o_id → (email_address, username)
        - o_id^+ = { }
            1. o_id → o_id : o_id^+ = { o_id }
            2. o_id → (email_address, username) : o_id^+ = { o_id, email_address, username }
            3. email_address → (first_name, last_name) : p_id^+ = {o_id, email_address, username, first_name, last_name }
            - Nothing else can be inferred, since password is not in o_id^+, password is not an extraneous attribute, and must stay in the FD
- Case for email_address → (first_name, last_name)
    - Is first_name extraneous in email_address → (first_name, last_name) ?
        - Replace email_address → (first_name, last_name) with email_address → last_name
        - $email\_address^+$ = { }

1. email_address → email_address : email_address$^+$ = { email_address }
2. email_address → last_name : email_address$^+$ = { email_address, last_name, }

- Nothing else can be inferred, since first_name is not in email_address$^+$, first_name is not an extraneous attribute, and must stay in the FD

- Is last_name extraneous in email_address → (first_name, last_name) ?
    - Replace email_address → (first_name, last_name) with email_address → first_name
    - email_address$^+$ = { }

1. email_address → email_address : email_address$^+$ = { email_address }
2. email_address → first_name : email_address$^+$ = {email_address, first_name}

- Nothing else can be inferred, since last_name is not in email_address$^+$, last_name is not an extraneous attribute, and must stay in the FD

- Case for (username, password) → o_id
    - Is username extraneous in (username, password) → o_id ?
        - Can we imply password → o_id
        - password$^+$ = { }

1. password → password : password$^+$ = { password }

- Nothing else can be inferred, since password alone cannot imply o_id, username must stay in the FD.

- Is password extraneous in (username, password) → o_id ?
    - Can we imply username → o_id
    - username$^+$ = { }

1. username → username : username$^+$ = { username }

- Nothing else can be inferred, since username alone cannot imply o_id, password must stay in the FD.

3. Therefore, R_1 = {o_id, email_address, username, password}, R_2 = {email_address, first_name, last_name}, and R_3 = {o_id, username, password}.
4. o_id from R_1 is a candidate key for the relation owner
5. R_3 is a subset of R_1, so it will be deleted, R_1 and R_2 will persist and be the product of this decomposition...

owner(o_id, email_address*, username, password)

owner_email(email_address, first_name, last_name)

Reports

reports = (r_id, o_id, day, month, year, report_type, result)

F = {

  r_id → (o_id, day, month, year, report_type, result)

}

F_c = F

F_c = {

  r_id → (o_id, day, month, year, report_type, result)

}

1. Since there exists only 1 FD, no union can be formed
2. Since there exists only 1 FD, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {r_id, o_id, day, month, year, report_type, result}
4. r_id from R_1 is a candidate key for the relation reports
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation reports therefore satisfies 3NF, and is in good normal form. No decomposition required.

reports(r_id, o_id*, day, month, year, report_type, result)

Order

order = (order_number, o_id, check_id, cl_city, cl_country, status)

F = {

  order_number → (o_id, check_id, cl_city, cl_country, status)

}

F_c = F

F_c = {

  order_number → (o_id, check_id, cl_city, cl_country, status)

}

1. Since there exists only 1 FD, no union can be formed

2. Since there exists only 1 FD, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {order_number, o_id, check_id, cl_city, cl_country, status}
4. order_number from R_1 is a candidate key for the relation order
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation order therefore satisfies 3NF, and is in good normal form. No decomposition required.

order(<u>order_number</u> , o_id*, check_id*, cl_city, cl_country, status)

Customer

customer = (c_id, shipping_address_id*, billing_address_id*, first_name, last_name, email_address, username, password)

F = {

c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → c_id

}

F_c = F

F_c = {

c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password),

email_address → (first_name, last_name),

(username, password) → c_id

}

1. No two FD's have the same α , therefore no union can be formed
2. Testing for Extraneous Attributes
   - Case for c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password)
     - Is shipping_address_id extraneous in c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) ?
       - Replace c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) with c_id → (billing_address_id, first_name,

last_name, email_address, username, password)

- c_id$^+$ = { }

  1. c_id → c_id : c_id$^+$ = { c_id }
  2. c_id → (billing_address_id, first_name, last_name, email_address, username, password) : c_id$^+$ = { c_id, billing_address_id, first_name, last_name, email_address, username, password }

  - Nothing else can be inferred, since shipping_address_id is not in c_id$^+$, shipping_address_id is not an extraneous attribute, and must stay in the FD

- Is billing_address_id extraneous in c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) ?

  - Replace c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) with c_id → (shipping_address_id, first_name, last_name, email_address, username, password)

  - c_id$^+$ = { }

    1. c_id → c_id : c_id$^+$ = { c_id }
    2. c_id → (shipping_address_id, first_name, last_name, email_address, username, password) : c_id$^+$ = { c_id, shipping_address_id, first_name, last_name, email_address, username, password }

    - Nothing else can be inferred, since billing_address_id is not in c_id$^+$, billing_address_id is not an extraneous attribute, and must stay in the FD

- Is first_name extraneous in c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) ?

  - Replace c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) with c_id → (shipping_address_id, billing_address, last_name, email_address, username, password)

  - c_id$^+$ = { }

    1. c_id → c_id : c_id$^+$ = { c_id }
    2. c_id → (shipping_address_id, billing_address, last_name, email_address, username, password) : c_id$^+$ = { c_id, shipping_address_id, billing_address, last_name, email_address, username, password }
    3. email_address → (first_name, last_name : c_id$^+$ = { c_id, shipping_address_id, billing_address, last_name, email_address, username, password, first_name }

    - Since first_name can be found in c_id$^+$, it is extraneous. We will replace the FD c_id → (shipping_address_id, billing_address_id, first_name, last_name, email_address, username, password) with the FD c_id → (

shipping_address_id, billing_address, last_name, email_address, username, password) in F_c

- Therefore, the new adjusted Canonical Cover

F_c = {

```
c_id --> (shipping_address_id, billing_address_id,
last_name, email_address, username, password),

email_address --> (first_name, last_name),

(username, password) --> c_id
```

}

- Is last_name extraneous in c_id → (shipping_address_id, billing_address_id, last_name, email_address, username, password) ?
  - Replace c_id → (shipping_address_id, billing_address_id, last_name, email_address, username, password) with c_id → (shipping_address_id, billing_address_id, email_address, username, password)
  - c_id^+ = { }
    1. c_id → c_id : c_id^+ = { c_id }
    2. c_id → (shipping_address_id, billing_address, email_address, username, password) : c_id^+ = { c_id, shipping_address_id, billing_address, email_address, username, password }
    3. email_address → (first_name, last_name) : c_id^+ = { c_id, shipping_address_id, billing_address, email_address, username, password, first_name, last_name }

    - Since last_name can be found in c_id^+ , it is extraneous. We will replace the FD c_id → (shipping_address_id, billing_address_id, last_name, email_address, username, password) with the FD c_id → (shipping_address_id, billing_address_id, email_address, username, password) in F_c
    - Therefore, the new adjusted Canonical Cover

F_c = {

```
c_id --> (shipping_address_id, billing_address_id,
email_address, username, password),

email_address --> (first_name, last_name),
(username, password) --> c_id
```

}

- Is email_address extraneous in c_id → (shipping_address_id, billing_address_id, email_address, username, password) ?
  - Replace c_id → (shipping_address_id, billing_address_id, email_address, username, password) with c_id → (shipping_address_id, billing_address_id, username, password)
  - c_id$^+$ = { }
    1. c_id → c_id : c_id$^+$ = { c_id }
    2. c_id → (shipping_address_id, billing_address, username, password) : c_id$^+$ = { c_id, shipping_address_id, billing_address, username, password }
    - Nothing else can be inferred, since email_address is not in c_id$^+$, email_address is not an extraneous attribute, and must stay in the FD
- Is username extraneous in c_id → (shipping_address_id, billing_address_id, email_address, username, password) ?
  - Replace c_id → (shipping_address_id, billing_address_id, email_address, username, password) with c_id → (shipping_address_id, billing_address_id, email_address, password)
  - c_id$^+$ = { }
    1. c_id → c_id : c_id$^+$ = { c_id }
    2. c_id → (shipping_address_id, billing_address, email_address, password) : c_id$^+$ = { c_id, shipping_address_id, billing_address, email_address, password }
    3. email_address --> (first_name, last_name) : c_id$^+$ = { c_id, shipping_address_id, billing_address, email_address, password, first_name, last_name }
    - Nothing else can be inferred, since username is not in c_id$^+$, username is not an extraneous attribute, and must stay in the FD
- Is password extraneous in c_id → (shipping_address_id, billing_address_id, email_address, username, password) ?
  - Replace c_id → (shipping_address_id, billing_address_id, email_address, username, password) with c_id → (shipping_address_id, billing_address_id, email_address, username)
  - c_id$^+$ = { }

1. c_id → c_id : c_id^+ = { c_id }
2. c_id → (shipping_address_id, billing_address, email_address, username) : c_id^+ = { c_id, shipping_address_id, billing_address, email_address, username }
3. email_address --> (first_name, last_name) : c_id^+ = { c_id, shipping_address_id, billing_address, email_address, username, first_name, last_name }
- Nothing else can be inferred, since password is not in c_id^+, password is not an extraneous attribute, and must stay in the FD

- Case for email_address → (first_name, last_name)
  - Is first_name extraneous in email_address → (first_name, last_name) ?
    - Replace email_address → (first_name, last_name) with email_address → last_name
    - $email\_address^+$ = { }
      1. email_address → email_address : $email\_address^+$ = { email_address }
      2. email_address → last_name : $email\_address^+$ = { email_address, last_name }
      - Nothing else can be inferred, since first_name is not in $email\_address^+$, first_name is not an extraneous attribute, and must stay in the FD
  - Is last_name extraneous in email_address → (first_name, last_name) ?
    - Replace email_address → (first_name, last_name) with email_address → first_name
    - $email\_address^+$ = { }
      1. email_address → email_address : $email\_address^+$ = { email_address }
      2. email_address → first_name : $email\_address^+$ = { email_address, first_name}
      - Nothing else can be inferred, since last_name is not in $email\_address^+$, last_name is not an extraneous attribute, and must stay in the FD
- Case for (username, password) → c_id
  - Is username extraneous in (username, password) → c_id ?
    - Can we imply password → c_id ?
    - $password^+$ = { }
      1. password → password : $password^+$ = { password }
      - Nothing else can be inferred, since password alone cannot imply c_id, username is not an extraneous attribute, and must stay in the FD

    - Is password extraneous in (username, password) → c_id ?
        - Can we imply username → c_id ?
        - username$^+$ = { }
            1. username → username : username$^+$ = { username }
            - Nothing else can be inferred, since username alone cannot imply c_id, password is not an extraneous attribute, and must stay in the FD

3. Therefore, R_1 = {c_id, shipping_address_id, billing_address_id, email_address, username, password}, R_2 = {email_address, first_name, last_name}, and R_3 = {username, password, c_id}.
4. c_id from R_1 is a candidate key for the relation customer
5. R_3 is a subset of R_1, so it will be deleted, but R_1 and R_2 will persist and be the product of this decomposition...

customer(c_id, shipping_address_id*, billing_address_id*, email_address*, username, password)

customer_email (email_address , first_name, last_name)

Checkout

checkout = (check_id, billing_address, shipping_address, c_id, cart_id, day, month, year)

F = {

check_id → (shipping_address_id, billing_address_id, c_id, cart_id, day, month, year)

}

F_c = F

F_c = {

check_id → (shipping_address_id, billing_address_id, c_id, cart_id, day, month, year)

}

1. Since there exists only 1 FD, no union can be formed
2. Since there exists only 1 FD, none of the attributes can be extraneous, therefore nothing to remove from F_c
3. Therefore, R_1 = {check_id, shipping_address_id, billing_address_id, c_id, cart_id, day, month, year}
4. check_id from R_1 is a candidate key for the relation checkout
5. R_1 is the only relation, and therefore is not a subset of any other relation

The relation checkout therefore satisfies 3NF, and is in good normal form. No decomposition required.

checkout(check_id, billing_address_id*, shipping_address_id*, c_id*, cart_id*, day, month, year)

Address

address = (address_id, street_number, street_name, city, country, postal_code)

F = {

  address_id → (street_number, street_name, city, country, postal_code),

  postal_code → (city, country)

}

F_C = F

F_C = {

  address_id → (street_number, street_name, city, country, postal_code),

  postal_code → (city, country)

}

1. No two FD's have the same α , therefore no union can be formed
2. Testing for Extraneous Attributes
    - Case for address_id → (street_number, street_name, city, country, postal_code)
        - Is street_number extraneous in address_id → (street_number, street_name, city, country, postal_code) ?
            - Replace address_id → (street_number, street_name, city, country, postal_code) with address_id → (street_name, city, country, postal_code)
            - address_id$^+$ = { }
                1. address_id → address_id : address_id$^+$ = { address_id }
                2. address_id → (street_name, city, country, postal_code) : address_id$^+$ = { address_id, street_name, city, country, postal_code }
                - Nothing else can be inferred, since street_number is not in address_id$^+$, street_number is not an extraneous attribute, and must stay in the FD
        - Is street_name extraneous in address_id → (street_number, street_name, city, country, postal_code) ?
            - Replace address_id → (street_number, street_name, city, country, postal_code) with address_id → (street_number, city, country, postal_code)
            - address_id$^+$ = { }
                1. address_id → address_id : address_id$^+$ = { address_id }

2. address_id → (street_number, city, country, postal_code) : address_id$^+$ = { address_id, street_number, city, country, postal_code }
   - Nothing else can be inferred, since street_name is not in address_id$^+$, street_name is not an extraneous attribute, and must stay in the FD

- Is city extraneous in address_id → (street_number, street_name, city, country, postal_code) ?
  - Replace address_id → (street_number, street_name, city, country, postal_code) with address_id → (street_number, street_name, country, postal_code)
  - address_id$^+$ = { }
    1. address_id → address_id : address_id$^+$ = { address_id }
    2. address_id → (street_number, street_name, country, postal_code) : address_id$^+$ = { address_id, street_number, street_name, country, postal_code }
    3. postal_code → (city, country) : address_id$^+$ = { address_id, street_number, street_name, country, postal_code, city }
    - Since city can be found in address_id$^+$, it is extraneous. We will replace the FD address_id → (street_number, street_name, city, country, postal_code) with the FD address_id → (street_number, street_name, country, postal_code) in F_c
    - Therefore, the new adjusted Canonical Cover

  F_c = {

  ```
  address_id --> (street_number, street_name, country,
  postal_code),

  postal_code --> (city, country)
  ```

  }

- Is country extraneous in address_id → (street_number, street_name, city, country, postal_code) ?
  - Replace address_id → (street_number, street_name, country, postal_code) with address_id → (street_number, street_name, postal_code)
  - address_id$^+$ = { }
    1. address_id → address_id : address_id$^+$ = { address_id }
    2. address_id → (street_number, street_name, postal_code) : address_id$^+$ = { address_id, street_number, street_name, postal_code }

3. postal_code → (city, country) : address_id$^+$ = { address_id, street_number, street_name, postal_code, city, country }

    - Since country can be found in address_id$^+$, it is extraneous. We will replace the FD address_id → (street_number, street_name, country, postal_code) with the FD address_id → (street_number, street_name, postal_code) in F_c
    - Therefore, the new adjusted Canonical Cover

F_c = {

```
address_id --> (street_number, street_name,
postal_code),

postal_code --> (city, country)
```

}

- Is postal_code extraneous in address_id → (street_number, street_name, postal_code) ?
    - Replace address_id → (street_number, street_name, postal_code) with address_id → (street_number, street_name)
    - address_id$^+$ = { }
        1. address_id → address_id : address_id$^+$ = { address_id }
        2. address_id → (street_number, street_name) : address_id$^+$ = { address_id, street_number, street_name}
        - Nothing else can be inferred, since postal_code is not in address_id$^+$, postal_code is not an extraneous attribute, and must stay in the FD

- Case for postal_code → (city, country)
    - Is city extraneous in postal_code → (city, country) ?
        - Replace postal_code → (city, country) with postal_code → (country)
        - postal_code$^+$ = { }
            1. postal_code → postal_code : postal_code$^+$ = { postal_code }
            2. postal_code → (country) : postal_code$^+$ = { postal_code, country }
            - Nothing else can be inferred, since city is not in postal_code$^+$, city is not an extraneous attribute, and must stay in the FD
    - Is country extraneous in postal_code → (city, country) ?
        - Replace postal_code → (city, country) with postal_code → (city)

- postal_code$^+$ = { }
    1. postal_code → postal_code : postal_code$^+$ = { postal_code }
    2. postal_code → (city) : postal_code$^+$ = { postal_code, city }
    - Nothing else can be inferred, since country is not in postal_code$^+$, country is not an extraneous attribute, and must stay in the FD

3. Therefore, R_1 = {address_id, street_number, street_name, postal_code} and R_2 = {postal_code, city, country}.
4. address_id from R_1 is a candidate key for the relation address
5. Neither of theese relations are subset of one another, so R_1 and R_2 will persist and be the product of this decomposition...

address(address_id, street_number, street_name, postal_code*)

region(postal_code, city, country)

The Resulting Relational Schemas After 3NF

book(ISBN, p_id*, title, genre, royalty, num_pages, price, cost, num_in_stock, threshold_num, num_sold)

cart(cart_id)

cart_books(cart_id, ISBN)

author_phone_number(a_id, phone_number)

owner_phone_number(o_id, phone_number)

customer_phone_number(c_id, phone_number)

author(a_id, email_address)

author_email(email_address, first_name, last_name)

book_authors(ISBN, a_id)

publisher (p_id, address_id*, email_address*, bank_account*)

publisher_email (email_address, name)

publisher_bank (bank_account, account_value)

publisher_phone_number(p_id, phone_number)

owner(o_id, email_address*, username, password)

owner_email(email_address, first_name, last_name)

reports(r_id, o_id*, day, month, year, report_type, result)

order(order_number , o_id*, check_id*, cl_city, cl_country, status)

customer(c_id, shipping_address_id*, billing_address_id*, email_address*, username, password)

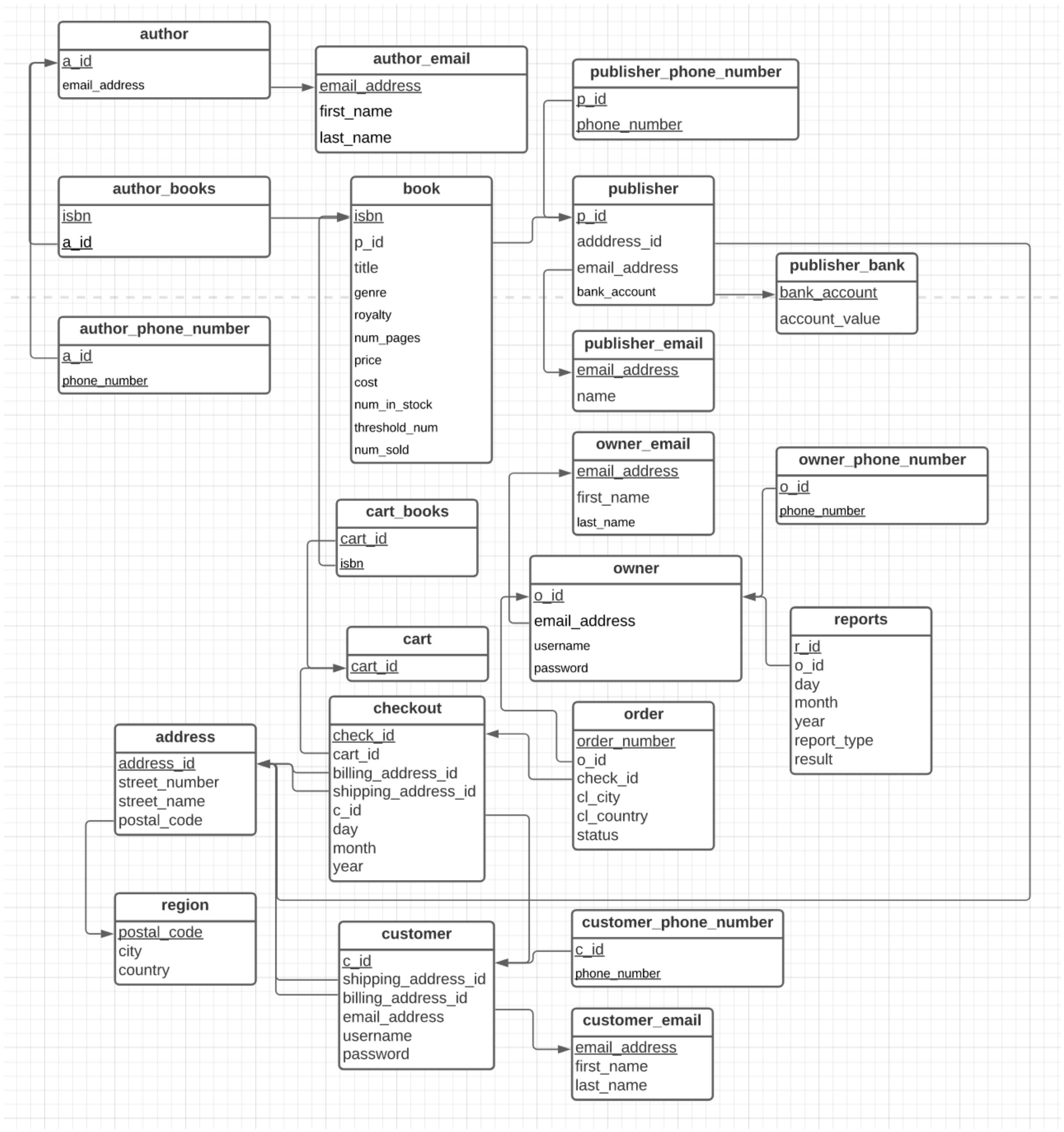
customer_email (email_address , first_name, last_name)

checkout(check_id, billing_address_id*, shipping_address_id*, c_id*, cart_id*, day, month, year)

address(address_id, street_number, street_name, postal_code*)

region(postal_code, city, country)

Database Schema Diagram



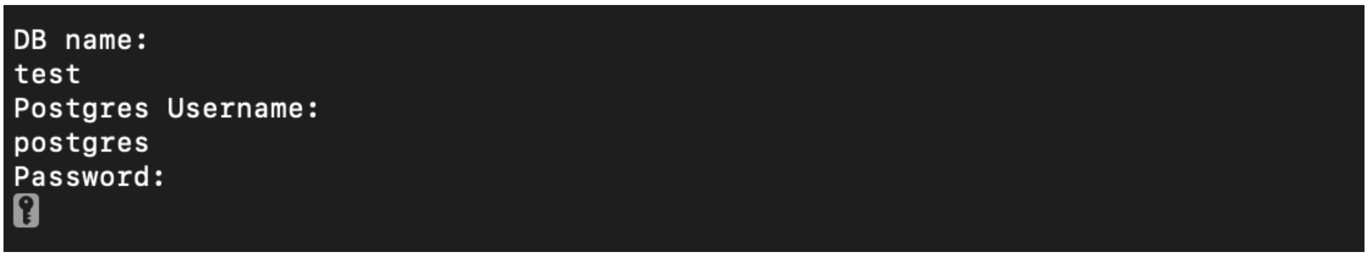
Implementation

Notes

- This appication was written in ruby
- Though I will not show it in this report, 99.9% of the time, if you provide invalid input, the program has been designed to catch it, and re prompt the user to provide valid data. input cleansing is an exhaustive task, and we did as much as we could. If you try to break the program you may find spots where our input validity detection is weaker. We will not bother displaying this error detection in the below sequence.

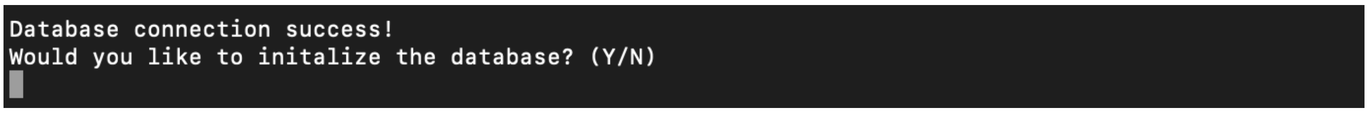
Application Design

Opening



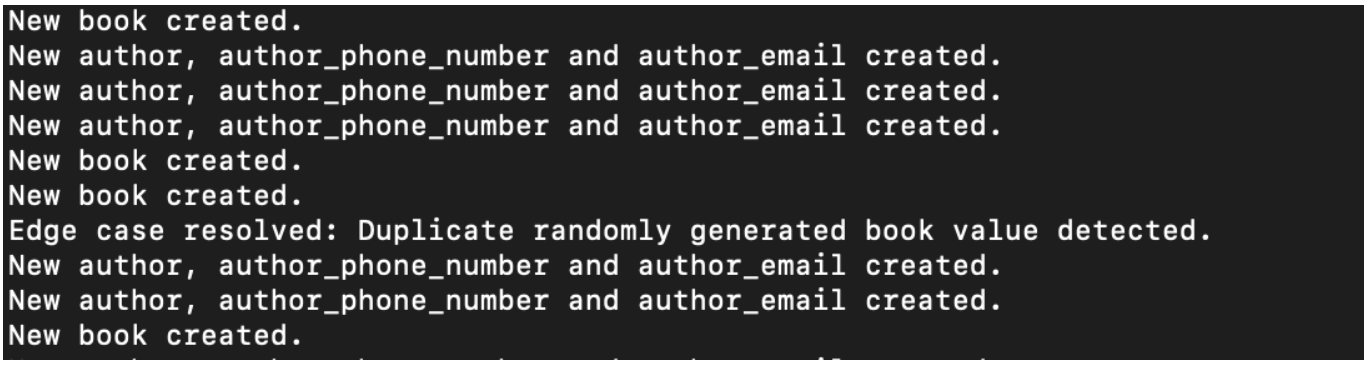
```
DB name:
test
Postgres Username:
postgres
Password:
```

Once you launch the program, you are prompted to provide the DB name which you have generated on your local version of pgAdmin / postgres. You must then provide your postgres username, and your postgres password. This is to setup a connection to the DB, this has nothing to do with being an owner or a client. Note: the password is hidden as you type it in, in the event you are simple like me and used the same passwrd for everything but didn't want your partner to access your banking info when paired programming on the project.

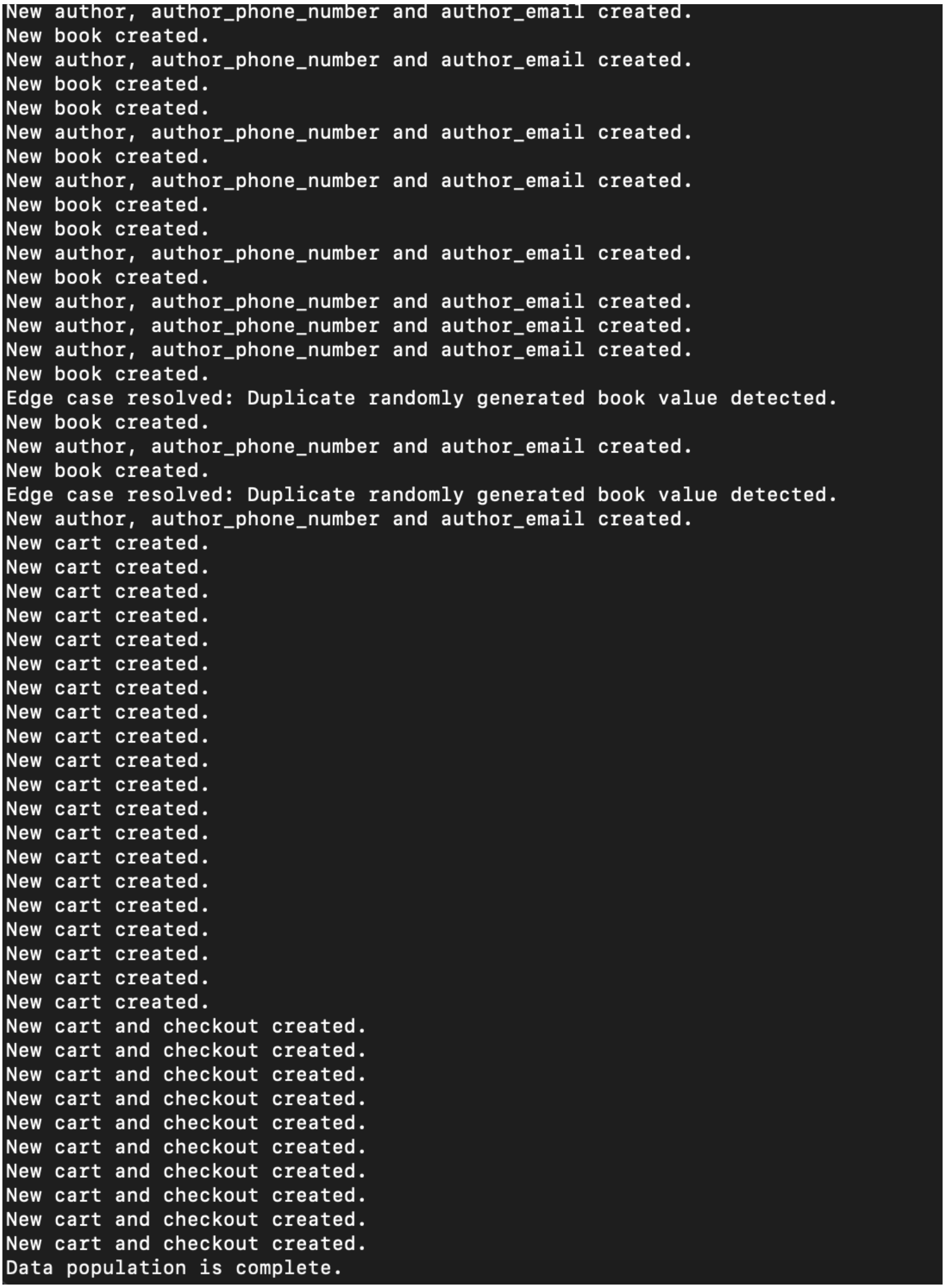


```
Database connection success!
Would you like to initalize the database? (Y/N)
```

You will then be asked wether or not you want to initialize the DB. If you say no, you will go forward with w.e version of the database is currently connected to the DB connection you made in the previous screen. This allows for you to close the program, return, and still have all of your data stored. This option is assuming you have initialized the DB atleast once before. If you ask to initialize, a program will run which deletes all the old data, drops all the old tables, reconstructs all the tables, and populated all the tables with randomly generated data.

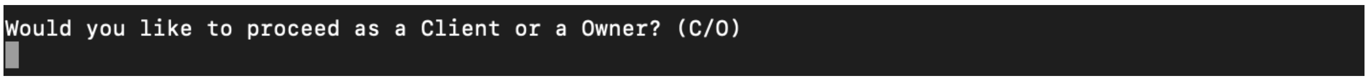


```
New book created.
New author, author_phone_number and author_email created.
New author, author_phone_number and author_email created.
New author, author_phone_number and author_email created.
New book created.
New book created.
Edge case resolved: Duplicate randomly generated book value detected.
New author, author_phone_number and author_email created.
New author, author_phone_number and author_email created.
New book created.
```



```
New author, author_phone_number and author_email created.
New book created.
New author, author_phone_number and author_email created.
New book created.
New book created.
New author, author_phone_number and author_email created.
New book created.
New author, author_phone_number and author_email created.
New book created.
New book created.
New author, author_phone_number and author_email created.
New book created.
New author, author_phone_number and author_email created.
New author, author_phone_number and author_email created.
New author, author_phone_number and author_email created.
New book created.
Edge case resolved: Duplicate randomly generated book value detected.
New book created.
New author, author_phone_number and author_email created.
New book created.
Edge case resolved: Duplicate randomly generated book value detected.
New author, author_phone_number and author_email created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
New cart and checkout created.
Data population is complete.
```

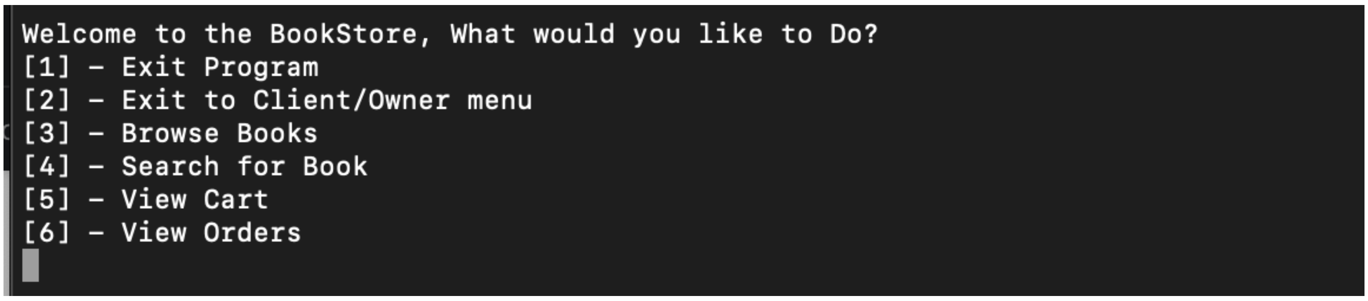
This is a small sample section of the feedback that is priinted to console when the DB initializes. All these records are connected to one another, and follow all the needed restriction of the DB. The ruby Faker Gem was used to help populate fields when possible. You will notice that anytime the Faker gem were to cause an error, say maybe adding in a non unqiue value when it needed to have been, we capture these cases, resolve them, and print "edge case resolved" to the console on occurance. This is useful for debugging and letting the user know that everything started up as expected.



```
Would you like to proceed as a Client or a Owner? (C/O)
```

Next you are prompted to enter as either a client or an owner. We will first see the Client side of things, and later the owner side.

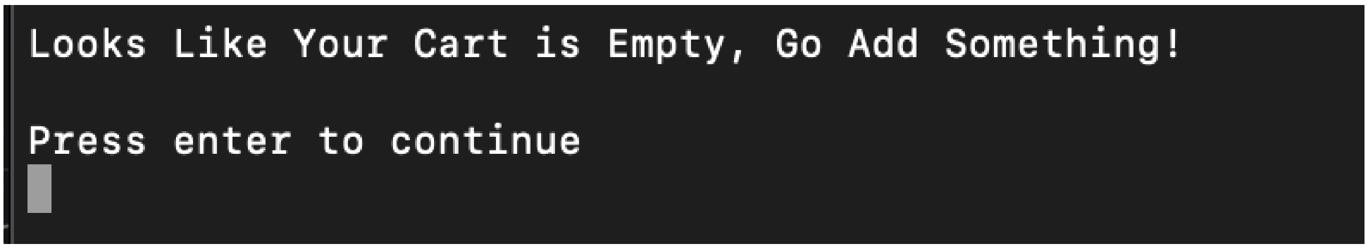
The Client Side



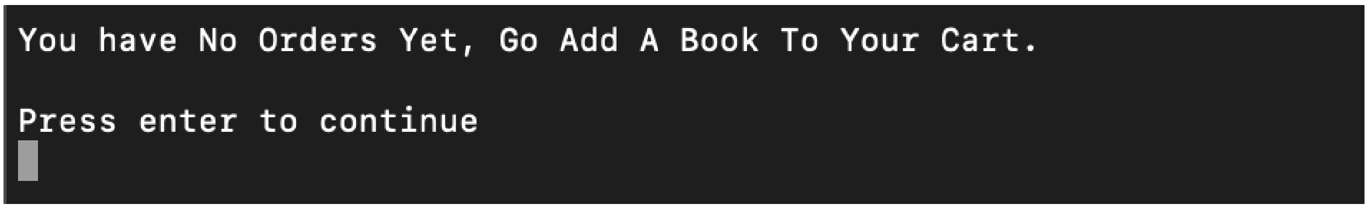
```
Welcome to the BookStore, What would you like to Do?
[1] - Exit Program
[2] - Exit to Client/Owner menu
[3] - Browse Books
[4] - Search for Book
[5] - View Cart
[6] - View Orders
```

Upon pressing 'C', you will be brought to the stores main page. Here you have a nmber of options of things to do. 1 will quit the program all together. 2 will bring you back to the client / owner menu you saw in the previous section, and the other 4 options we will explore in more detail. Currently, since we only just entered the applicaiton, we do not have a cart, and we do not have any checkouts (because we have not yet signed in)

If we click on 5 or 6 at this time, you will see the following messages;

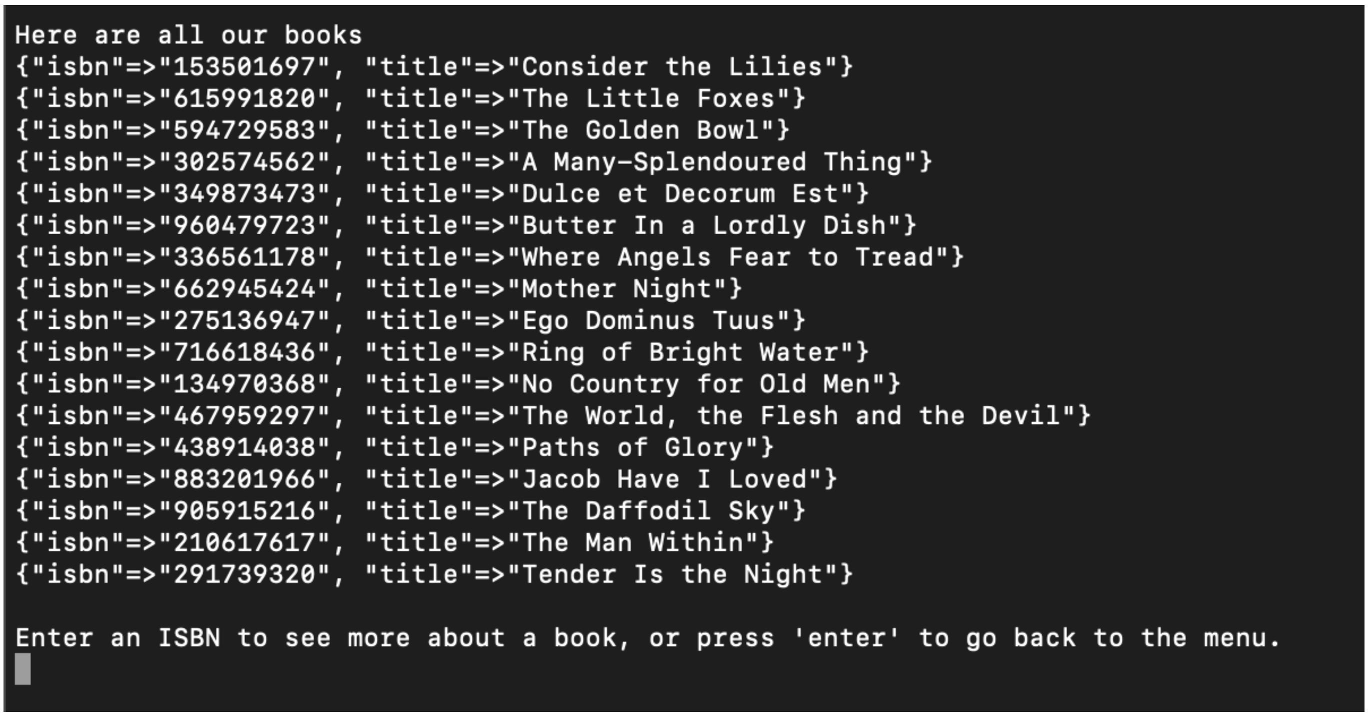


```
Looks Like Your Cart is Empty, Go Add Something!

Press enter to continue
```



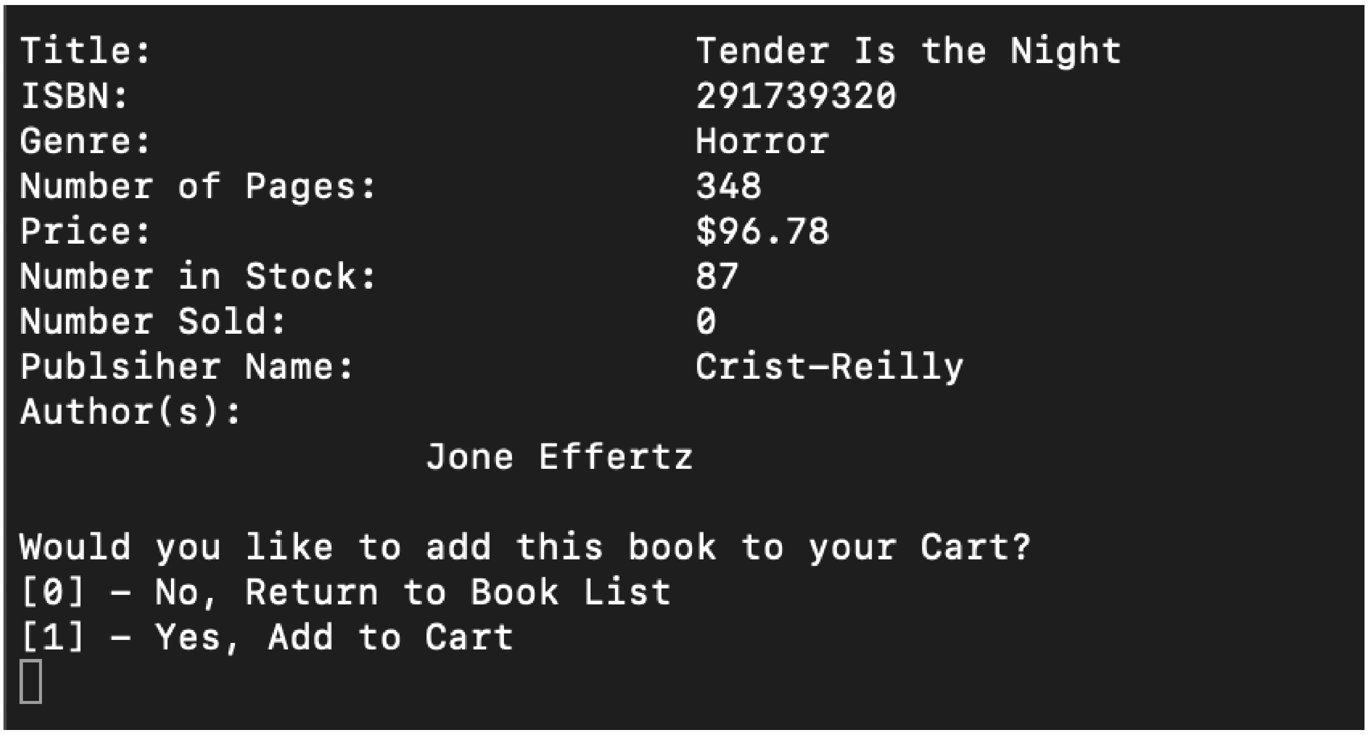
```
You have No Orders Yet, Go Add A Book To Your Cart.

Press enter to continue
```

In either case we will be brought back to the main client side menu from earlier.



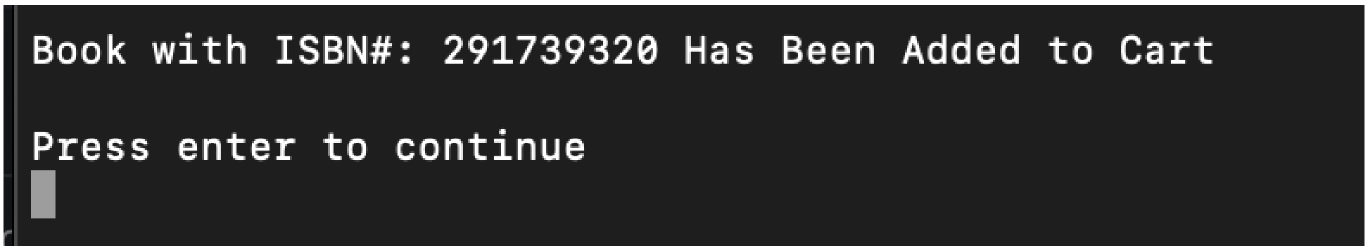
```
Here are all our books
{"isbn"=>"153501697", "title"=>"Consider the Lilies"}
{"isbn"=>"615991820", "title"=>"The Little Foxes"}
{"isbn"=>"594729583", "title"=>"The Golden Bowl"}
{"isbn"=>"302574562", "title"=>"A Many-Splendoured Thing"}
{"isbn"=>"349873473", "title"=>"Dulce et Decorum Est"}
{"isbn"=>"960479723", "title"=>"Butter In a Lordly Dish"}
{"isbn"=>"336561178", "title"=>"Where Angels Fear to Tread"}
{"isbn"=>"662945424", "title"=>"Mother Night"}
{"isbn"=>"275136947", "title"=>"Ego Dominus Tuus"}
{"isbn"=>"716618436", "title"=>"Ring of Bright Water"}
{"isbn"=>"134970368", "title"=>"No Country for Old Men"}
{"isbn"=>"467959297", "title"=>"The World, the Flesh and the Devil"}
{"isbn"=>"438914038", "title"=>"Paths of Glory"}
{"isbn"=>"883201966", "title"=>"Jacob Have I Loved"}
{"isbn"=>"905915216", "title"=>"The Daffodil Sky"}
{"isbn"=>"210617617", "title"=>"The Man Within"}
{"isbn"=>"291739320", "title"=>"Tender Is the Night"}

Enter an ISBN to see more about a book, or press 'enter' to go back to the menu.
```

If we select 3, we will enter the browse books screen. This queries the DB for all book on record, and shows their isbn and title. You will then be prompted to either copy paste an isbn below and click enter to view more details about the book, or just click enter to go back to the main menu.



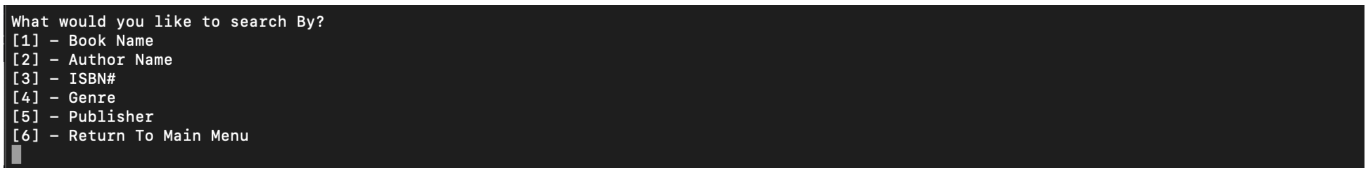
```
Title:                              Tender Is the Night
ISBN:                               291739320
Genre:                              Horror
Number of Pages:                    348
Price:                              $96.78
Number in Stock:                    87
Number Sold:                        0
Publsiher Name:                     Crist-Reilly
Author(s):
                    Jone Effertz

Would you like to add this book to your Cart?
[0] - No, Return to Book List
[1] - Yes, Add to Cart
```

If we pasted in the isbn for *Tender Is The Night*, you will see more infor about that title. We can enter 0 to back to viewing the books if we are not intereste,d or 1 to add it to the cart.



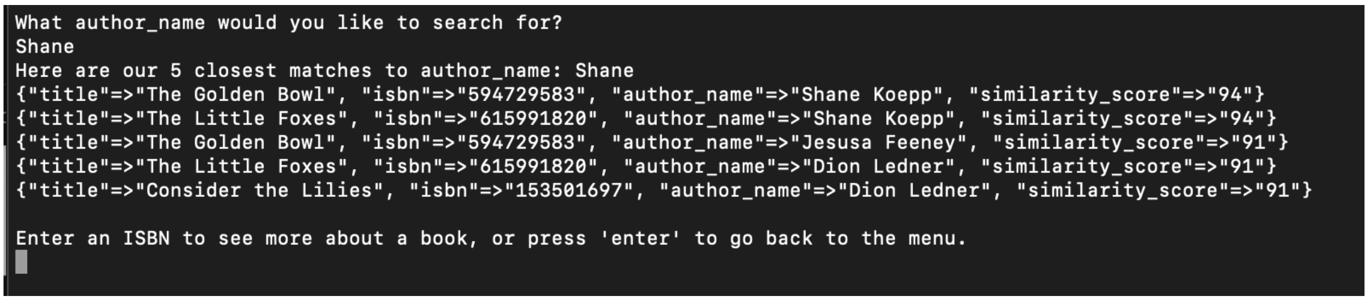
```
Book with ISBN#: 291739320 Has Been Added to Cart

Press enter to continue
```

After pressing one, we are brought to a screen that confirms the item was added to the cart. Behind the scenes, this also instantiated a new cart entity, and stored its cart_id as state in the applicaiton, so any other books we add or remove going forward get added to the same cart. Pressing enter will bring you back to the book browse. I'll go ahead and add in two more books and return to the main menue.



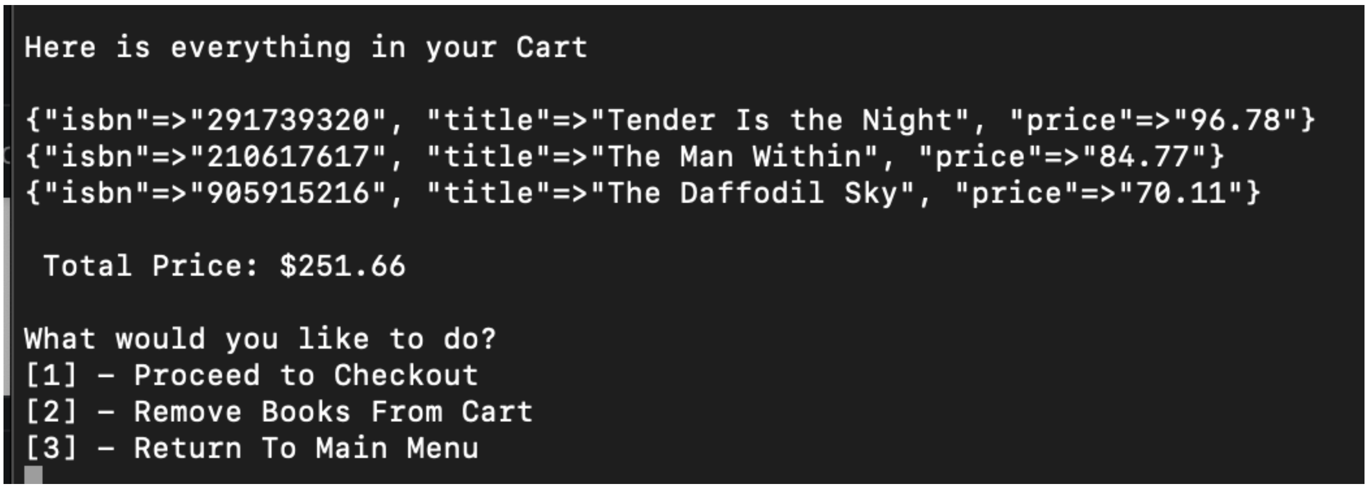
```
What would you like to search By?
[1] - Book Name
[2] - Author Name
[3] - ISBN#
[4] - Genre
[5] - Publisher
[6] - Return To Main Menu
```

If we click on 4 from the main menue, we get brought into the main menu screen. Here we can choose what to search by and we also hve the option to leave this menu. Most of the logic behind thse options are similar, so lets take a peak at what happens when we search by author...



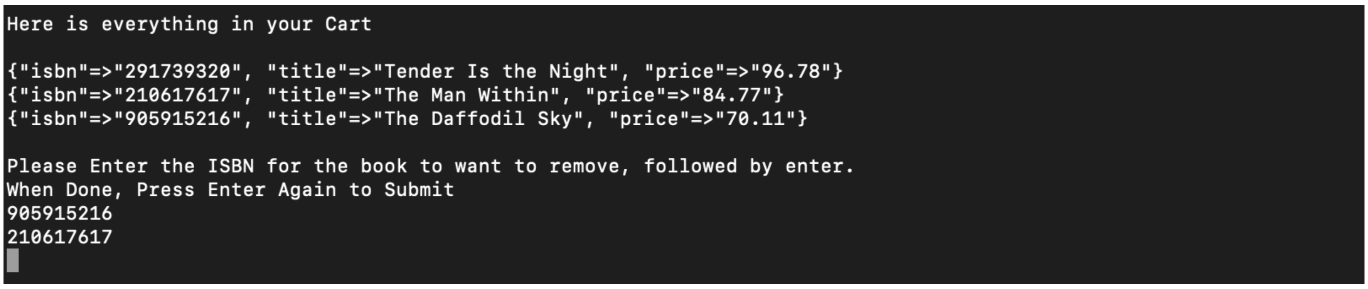
```
What author_name would you like to search for?
Shane
Here are our 5 closest matches to author_name: Shane
{"title"=>"The Golden Bowl", "isbn"=>"594729583", "author_name"=>"Shane Koepp", "similarity_score"=>"94"}
{"title"=>"The Little Foxes", "isbn"=>"615991820", "author_name"=>"Shane Koepp", "similarity_score"=>"94"}
{"title"=>"The Golden Bowl", "isbn"=>"594729583", "author_name"=>"Jesusa Feeney", "similarity_score"=>"91"}
{"title"=>"The Little Foxes", "isbn"=>"615991820", "author_name"=>"Dion Ledner", "similarity_score"=>"91"}
{"title"=>"Consider the Lilies", "isbn"=>"153501697", "author_name"=>"Dion Ledner", "similarity_score"=>"91"}

Enter an ISBN to see more about a book, or press 'enter' to go back to the menu.
```

Here we ssearched upp the author name *Shane* and weere returns the top 5 similar options. These similarity searches are done by Levenshtein value. This means, the fewer character insertions, deletions, and replacements needed, the more similar the strings are considered to be. For instance, it only took 6 inssertions to go from *Shane* -> *Shane Koepp*, so we subtracted 6 from 100 to get a similarity score of 94. In the same vein, it would take 9 modificaitons to *Dion Ledner* -> *Shane Koepp*, which is fewer then other authors in the DB, so her books are shown next. Just like on the books browse menu earlier, we can either add these to the cart just like before, or go back to the menu. Let's go back.



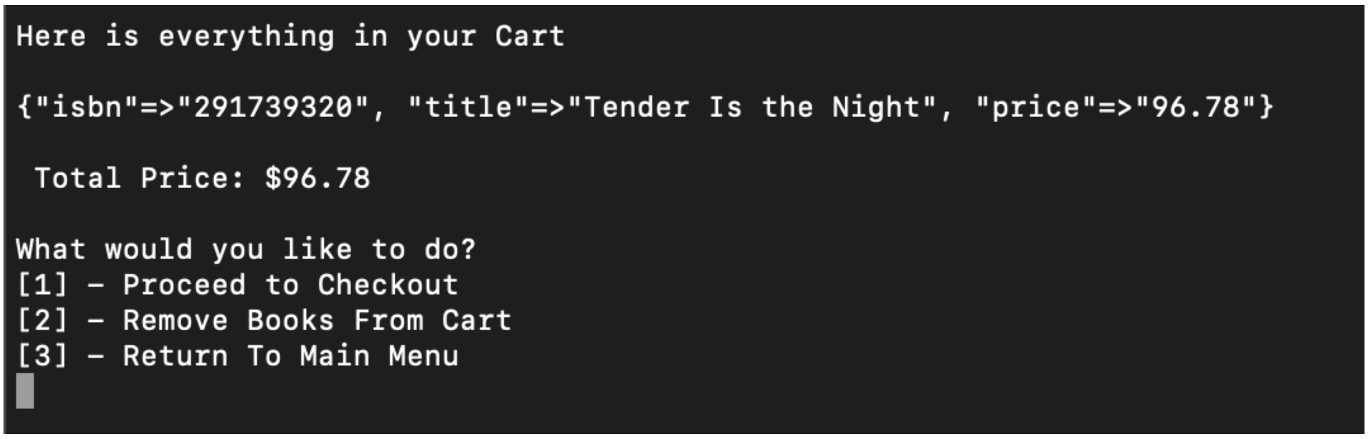
```
Here is everything in your Cart

{"isbn"=>"291739320", "title"=>"Tender Is the Night", "price"=>"96.78"}
{"isbn"=>"210617617", "title"=>"The Man Within", "price"=>"84.77"}
{"isbn"=>"905915216", "title"=>"The Daffodil Sky", "price"=>"70.11"}

 Total Price: $251.66

What would you like to do?
[1] - Proceed to Checkout
[2] - Remove Books From Cart
[3] - Return To Main Menu
```

At this point, if we tried to view orders, it would still boot us out, as we have not made any, but we know do have a valid cart that is populated with isbn's. Here we can see everythign in the cart, their isbns, titles, and prices. We can remove books by pressing 2



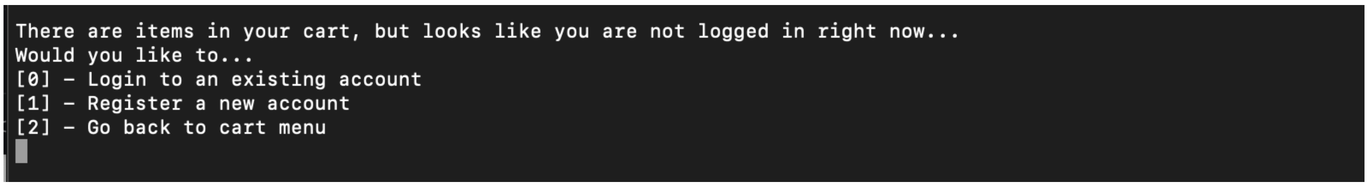
```
Here is everything in your Cart

{"isbn"=>"291739320", "title"=>"Tender Is the Night", "price"=>"96.78"}
{"isbn"=>"210617617", "title"=>"The Man Within", "price"=>"84.77"}
{"isbn"=>"905915216", "title"=>"The Daffodil Sky", "price"=>"70.11"}

Please Enter the ISBN for the book to want to remove, followed by enter.
When Done, Press Enter Again to Submit
905915216
210617617
```

Here, we aree prompted to put in the isbns we wish to remove. At this point I implemented the ability to provide many isbns at once rather then needed to do it in many motions. So long as they are new line separated, they will all be removed from your cart.



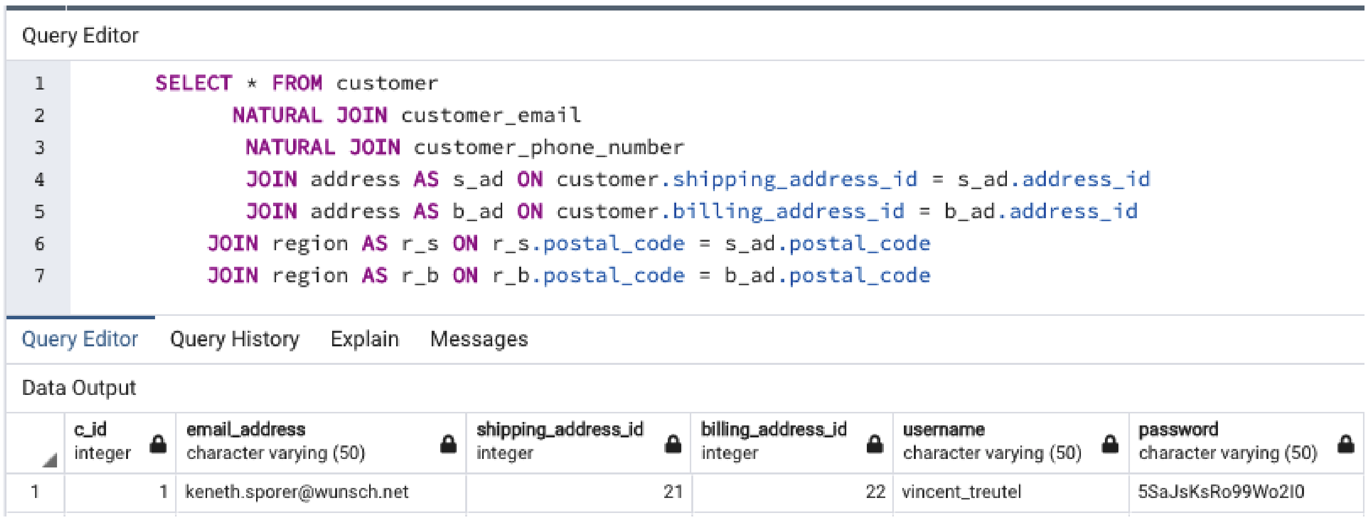
```
Here is everything in your Cart

{"isbn"=>"291739320", "title"=>"Tender Is the Night", "price"=>"96.78"}

 Total Price: $96.78

What would you like to do?
[1] - Proceed to Checkout
[2] - Remove Books From Cart
[3] - Return To Main Menu
```

As you can see, both of the titles were removed from the cart, and the total price has adjusted accordingly. I will go add in another book so that the final screens are more interesting, then we will progress into *Proceed to Checkout*.



```
There are items in your cart, but looks like you are not logged in right now...
Would you like to...
[0] - Login to an existing account
[1] - Register a new account
[2] - Go back to cart menu
```

This option will check and make sure we didnt just delete everything from our cart and try to checkout. If the cart were empty, the only option it would have given us would have been to go back, but since we have things in our cart, we can can either login, register, or leave. Lets start by logging in.



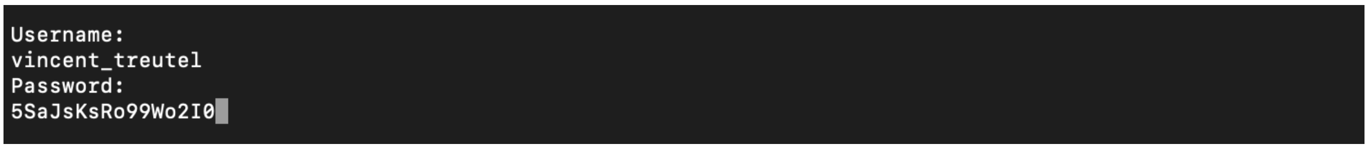
Query Editor

```
SELECT * FROM customer
      NATURAL JOIN customer_email
       NATURAL JOIN customer_phone_number
       JOIN address AS s_ad ON customer.shipping_address_id = s_ad.address_id
       JOIN address AS b_ad ON customer.billing_address_id = b_ad.address_id
     JOIN region AS r_s ON r_s.postal_code = s_ad.postal_code
     JOIN region AS r_b ON r_b.postal_code = b_ad.postal_code
```

Data Output

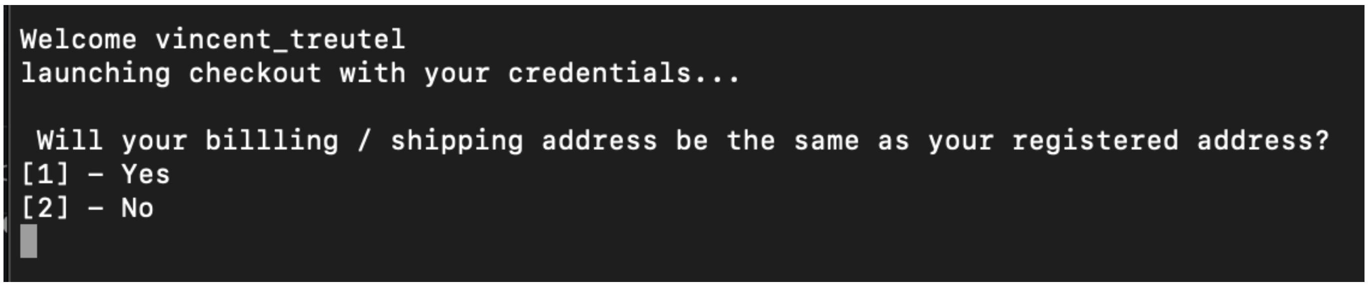
	c_id integer	email_address character varying (50)	shipping_address_id integer	billing_address_id integer	username character varying (50)	password character varying (50)
1	1	keneth.sporer@wunsch.net	21	22	vincent_treutel	5SaJsKsRo99Wo2I0

We can go into pgAdmin, fetch the customers that were auto populated when we initialized the DB, and grab the username and password from one of the records. We will just use the firsst one.



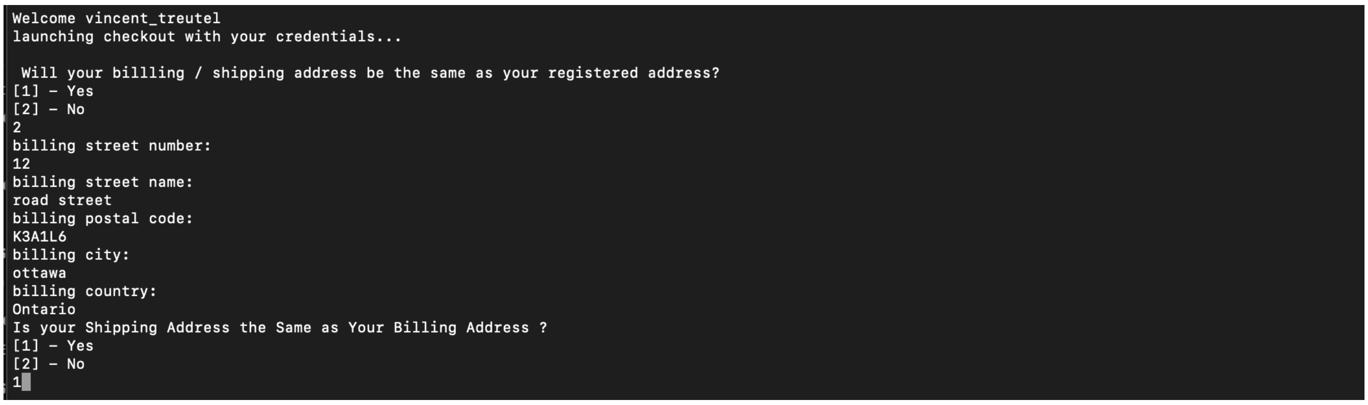
```
Username:
vincent_treutel
Password:
5SaJsKsRo99Wo2I0
```

provide these details after being prompted from the loging page



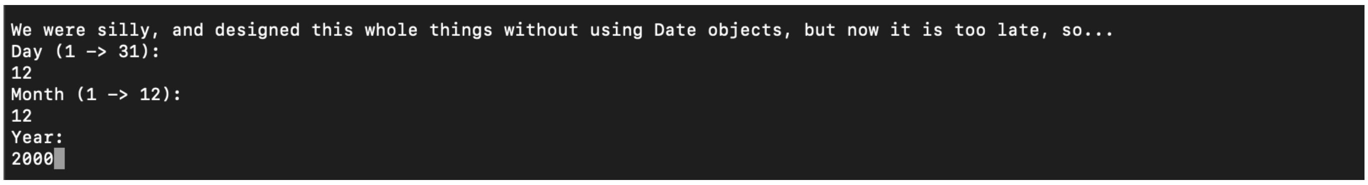
```
Welcome vincent_treutel
launching checkout with your credentials...

 Will your billling / shipping address be the same as your registered address?
[1] - Yes
[2] - No
```

You will then be welcomed, and a user state, containing your id, will be stored in the application, going forward, you will not need to provide as much information about yourself for subsequent checkouts. This time around, you will be asked if your billing / shipping for this particular checkout are the same or different from those you have on record. Letss pretend they are not.



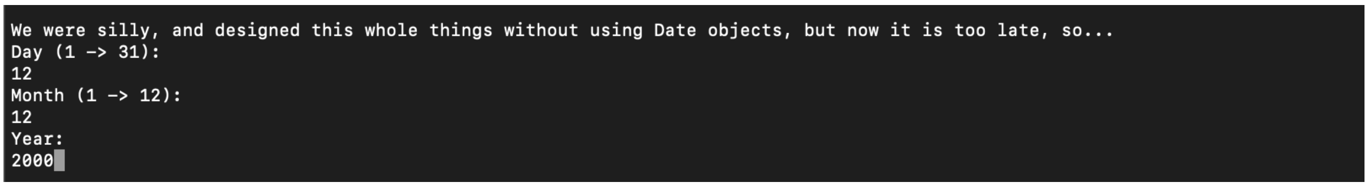
```
Welcome vincent_treutel
launching checkout with your credentials...

 Will your billling / shipping address be the same as your registered address?
[1] - Yes
[2] - No
2
billing street number:
12
billing street name:
road street
billing postal code:
K3A1L6
billing city:
ottawa
billing country:
Ontario
Is your Shipping Address the Same as Your Billing Address ?
[1] - Yes
[2] - No
1
```

After providing all of your billing information, you will be asked if your shipping and billing are the ssame for this order. If you say no, you will be prompted all of the above information again. Both of these addresses you provide will generate unique insances in the DB, and your billing_id and shipping_id will be different. In the event that you say yes, they are the same, we just have both you billing_id and shipping_id point to the same address_id, the one you just created for billing info.



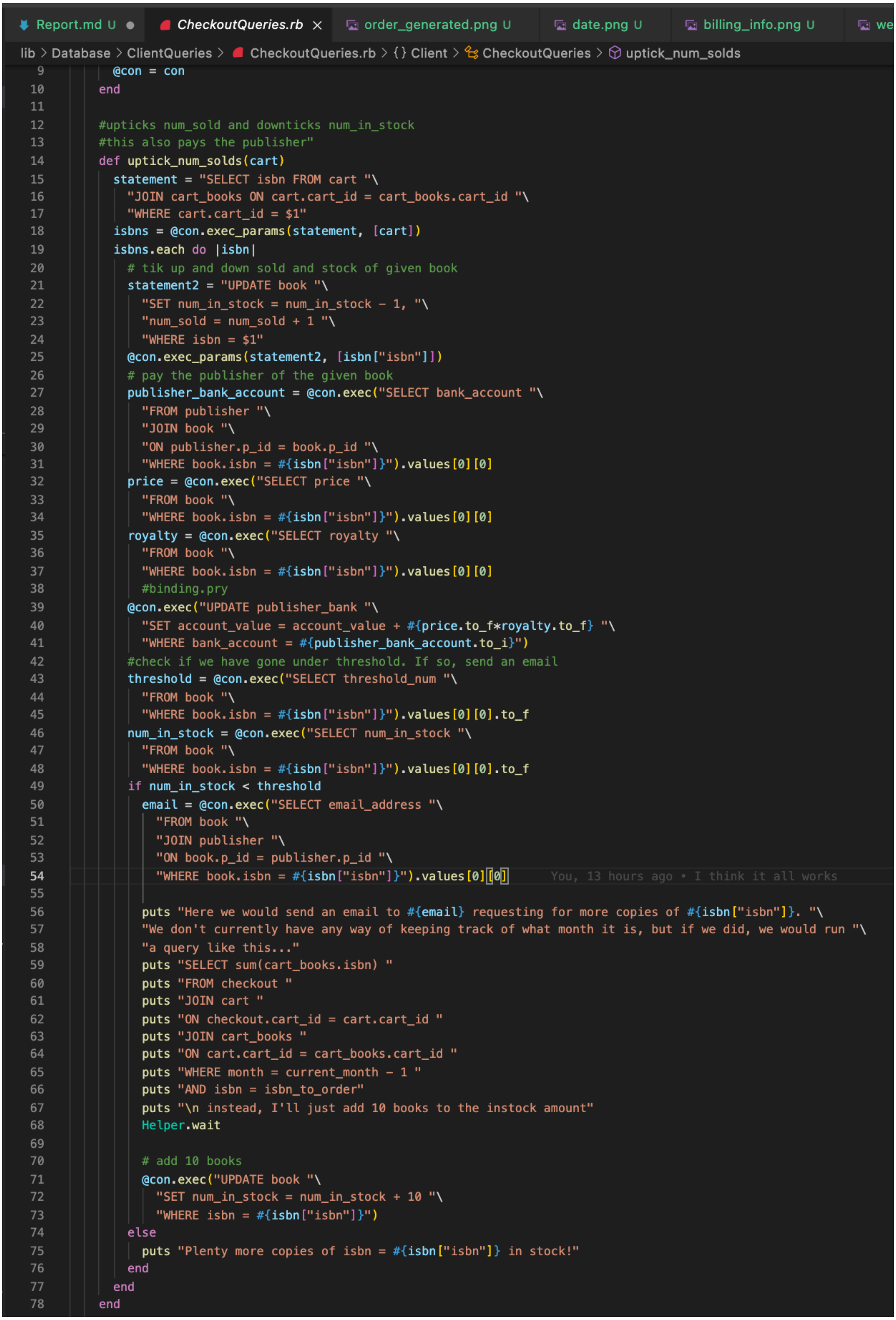
```
We were silly, and designed this whole things without using Date objects, but now it is too late, so...
Day (1 -> 31):
12
Month (1 -> 12):
12
Year:
2000
```

This part I regret designing this way, but alas. You will be prompted to put in the day / month / year. We should have used a Date or DateTime object and had this auto populate, this would have saved a lot of headache down the road. But we do not have enough time to redo all of this, so we will embrace this jank, and chalf it up to a learning opportunity.



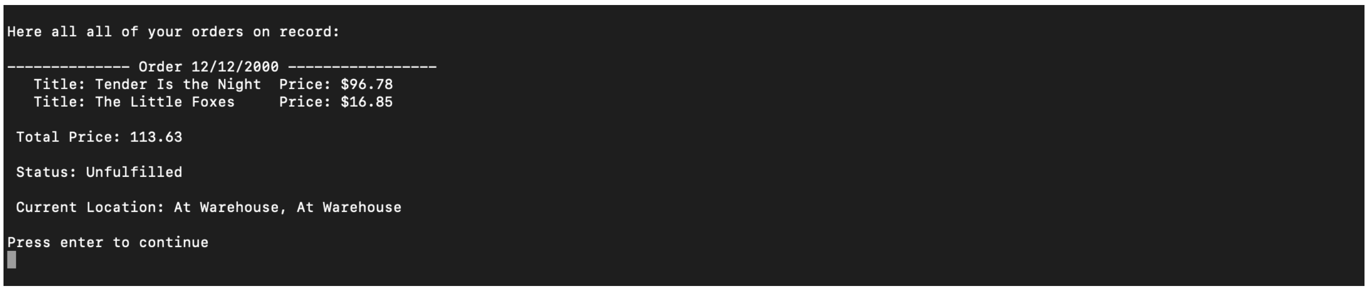
```
We were silly, and designed this whole things without using Date objects, but now it is too late, so...
Day (1 -> 31):
12
Month (1 -> 12):
12
Year:
2000
```

If the order goes through succesfuly you will be notified, and we provide a little bit of feedback that there are more copies of those books in stock. If the stock number dips before the threshold number, this is the screen where you will see an *email* get sent to the appropriate publisher. If this were a real application, we obviously would not show that here, but this seemed like the most appropriate space to give this feedback all thigns considered. This is also the spot where the following actions take place behind the scenes.


Report.md U
CheckoutQueries.rb
order_generated.png U
date.png U
billing_info.png U
we
lib > Database > ClientQueries > CheckoutQueries.rb > {} Client > CheckoutQueries > uptick_num_solds

```ruby
    @con = con
  end

  #upticks num_sold and downticks num_in_stock
  #this also pays the publisher"
  def uptick_num_solds(cart)
    statement = "SELECT isbn FROM cart "\
      "JOIN cart_books ON cart.cart_id = cart_books.cart_id "\
      "WHERE cart.cart_id = $1"
    isbns = @con.exec_params(statement, [cart])
    isbns.each do |isbn|
      # tik up and down sold and stock of given book
      statement2 = "UPDATE book "\
        "SET num_in_stock = num_in_stock - 1, "\
        "num_sold = num_sold + 1 "\
        "WHERE isbn = $1"
      @con.exec_params(statement2, [isbn["isbn"]])
      # pay the publisher of the given book
      publisher_bank_account = @con.exec("SELECT bank_account "\
        "FROM publisher "\
        "JOIN book "\
        "ON publisher.p_id = book.p_id "\
        "WHERE book.isbn = #{isbn["isbn"]}").values[0][0]
      price = @con.exec("SELECT price "\
        "FROM book "\
        "WHERE book.isbn = #{isbn["isbn"]}").values[0][0]
      royalty = @con.exec("SELECT royalty "\
        "FROM book "\
        "WHERE book.isbn = #{isbn["isbn"]}").values[0][0]
        #binding.pry
      @con.exec("UPDATE publisher_bank "\
        "SET account_value = account_value + #{price.to_f*royalty.to_f} "\
        "WHERE bank_account = #{publisher_bank_account.to_i}")
      #check if we have gone under threshold. If so, send an email
      threshold = @con.exec("SELECT threshold_num "\
        "FROM book "\
        "WHERE book.isbn = #{isbn["isbn"]}").values[0][0].to_f
      num_in_stock = @con.exec("SELECT num_in_stock "\
        "FROM book "\
        "WHERE book.isbn = #{isbn["isbn"]}").values[0][0].to_f
      if num_in_stock < threshold
        email = @con.exec("SELECT email_address "\
          "FROM book "\
          "JOIN publisher "\
          "ON book.p_id = publisher.p_id "\
          "WHERE book.isbn = #{isbn["isbn"]}").values[0][0]      You, 13 hours ago • I think it all works

        puts "Here we would send an email to #{email} requesting for more copies of #{isbn["isbn"]}. "\
        "We don't currently have any way of keeping track of what month it is, but if we did, we would run "\
        "a query like this..."
        puts "SELECT sum(cart_books.isbn) "
        puts "FROM checkout "
        puts "JOIN cart "
        puts "ON checkout.cart_id = cart.cart_id "
        puts "JOIN cart_books "
        puts "ON cart.cart_id = cart_books.cart_id "
        puts "WHERE month = current_month - 1 "
        puts "AND isbn = isbn_to_order"
        puts "\n instead, I'll just add 10 books to the instock amount"
        Helper.wait

        # add 10 books
        @con.exec("UPDATE book "\
          "SET num_in_stock = num_in_stock + 10 "\
          "WHERE isbn = #{isbn["isbn"]}")
      else
        puts "Plenty more copies of isbn = #{isbn["isbn"]} in stock!"
      end
    end
  end
```

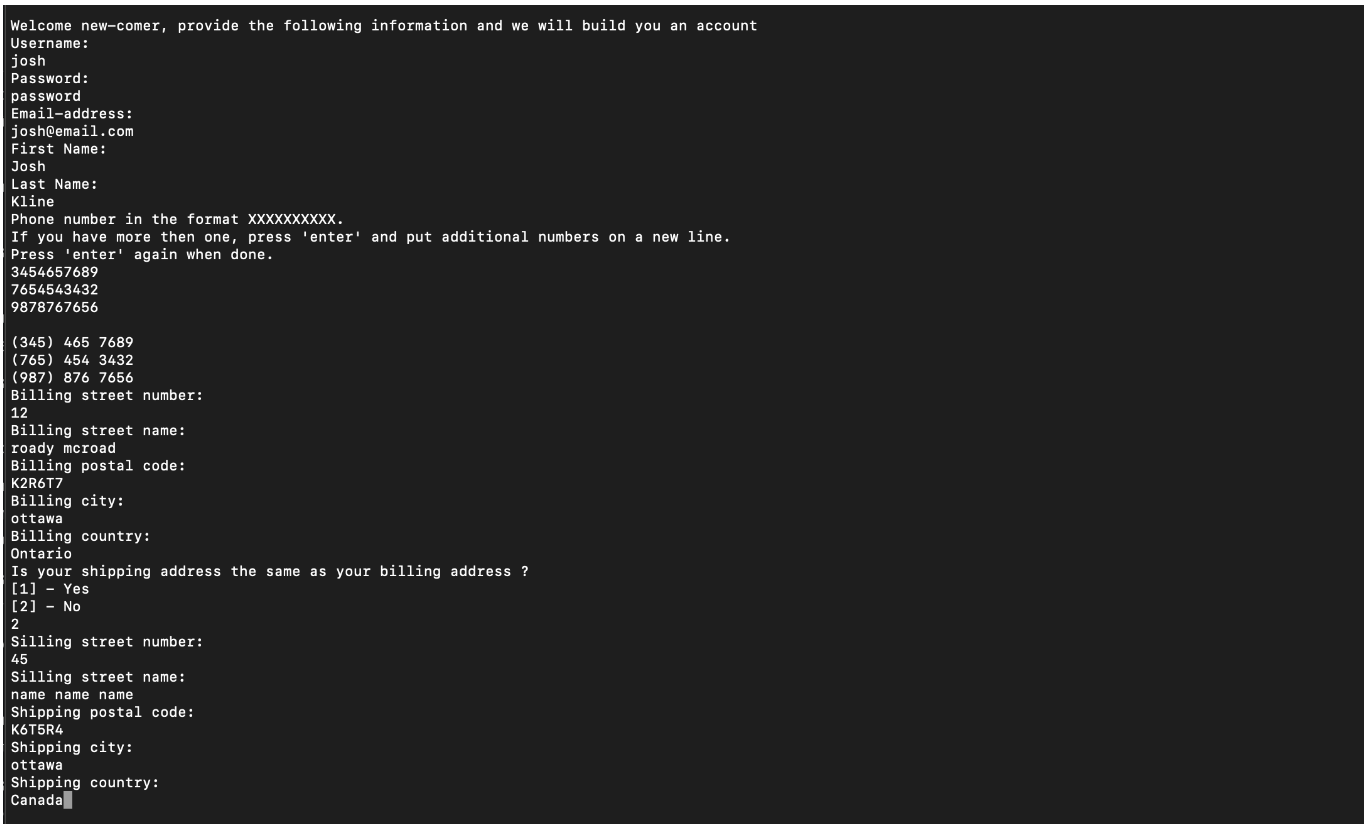
This function fires in the backgorund, update the num_in_stock, num_sold, threshold email trigger, and publisher pay out triggers all at once. It is dense, but it works. this is also called iteratively on every isbn in the cart, so every book has its values adjusted accordingly.



```
Here all all of your orders on record:

------------- Order 12/12/2000 -----------------
   Title: Tender Is the Night  Price: $96.78
   Title: The Little Foxes     Price: $16.85

 Total Price: 113.63

 Status: Unfulfilled

 Current Location: At Warehouse, At Warehouse

Press enter to continue
```

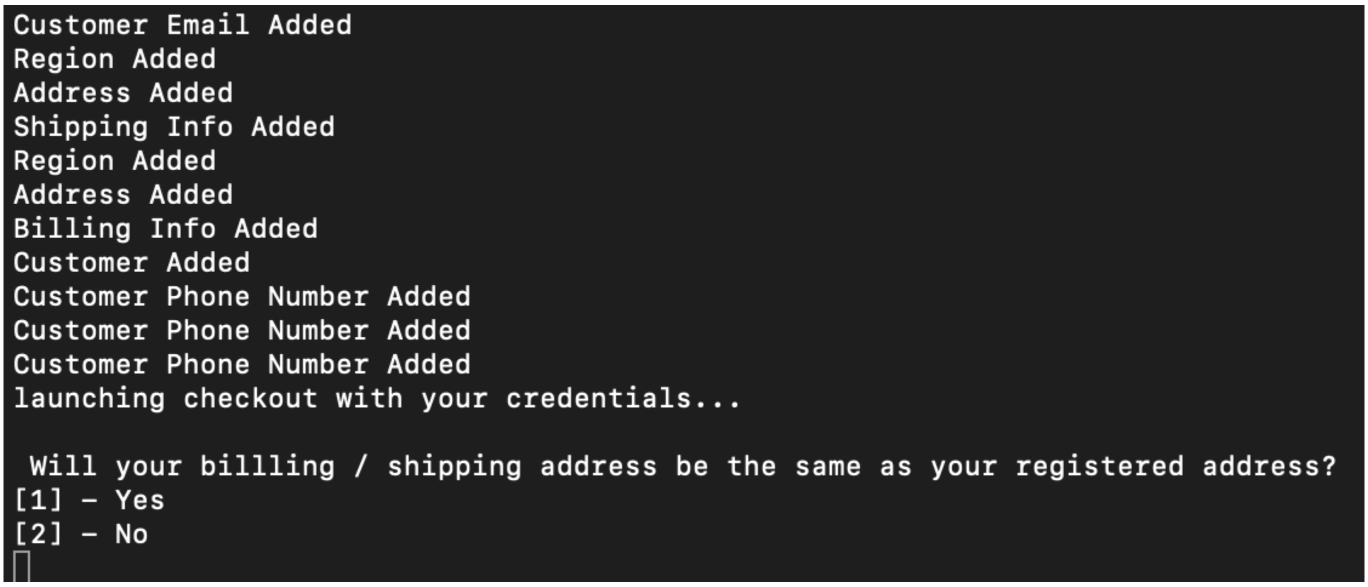
Now if we go back to the main menu, and click on orders, we will be shown all of our orders, their dates, their contents, the total price, the status and the current location. On the owner side of the application,, they can switch the oorder from unfulfilled to fulfilled, and provide geographic locations while it is in transit. Project description suggested viewing orders 1 at a time by order number, but this seemed very odd and unrealistic. When you buy things on amazon, and you view your past orders, they are there, easily accessible, and you can see all the relevant infomation. Having to go find an order numberr and look it up manually is tedious and does not make sense. As such I implemented it this way instead. This still requires performing the same type of query on an order number, but rather then having the user pass in a specific number, I am iterating through all of the order_numbers associated with the given user_id who is currently stored in state, and generating a order report for every one on record. I would argue this is a bonus feature.

The last thing to show on the client side is the proocess of registring a new user. Currently, we are signed in, but no longer have a valid cart. If we try to view cart at this point, it would boot us out, but if we try to view orders, like we just did, it will let us through because we have a user_id in state. In order to remove the user_id from state we need to return to the client/owner option mmenu, and re-enter as a client. We then need to add something to the cart, and go to checkout, and select register new user. To save time, I will do that without photos, and we will continue from there.



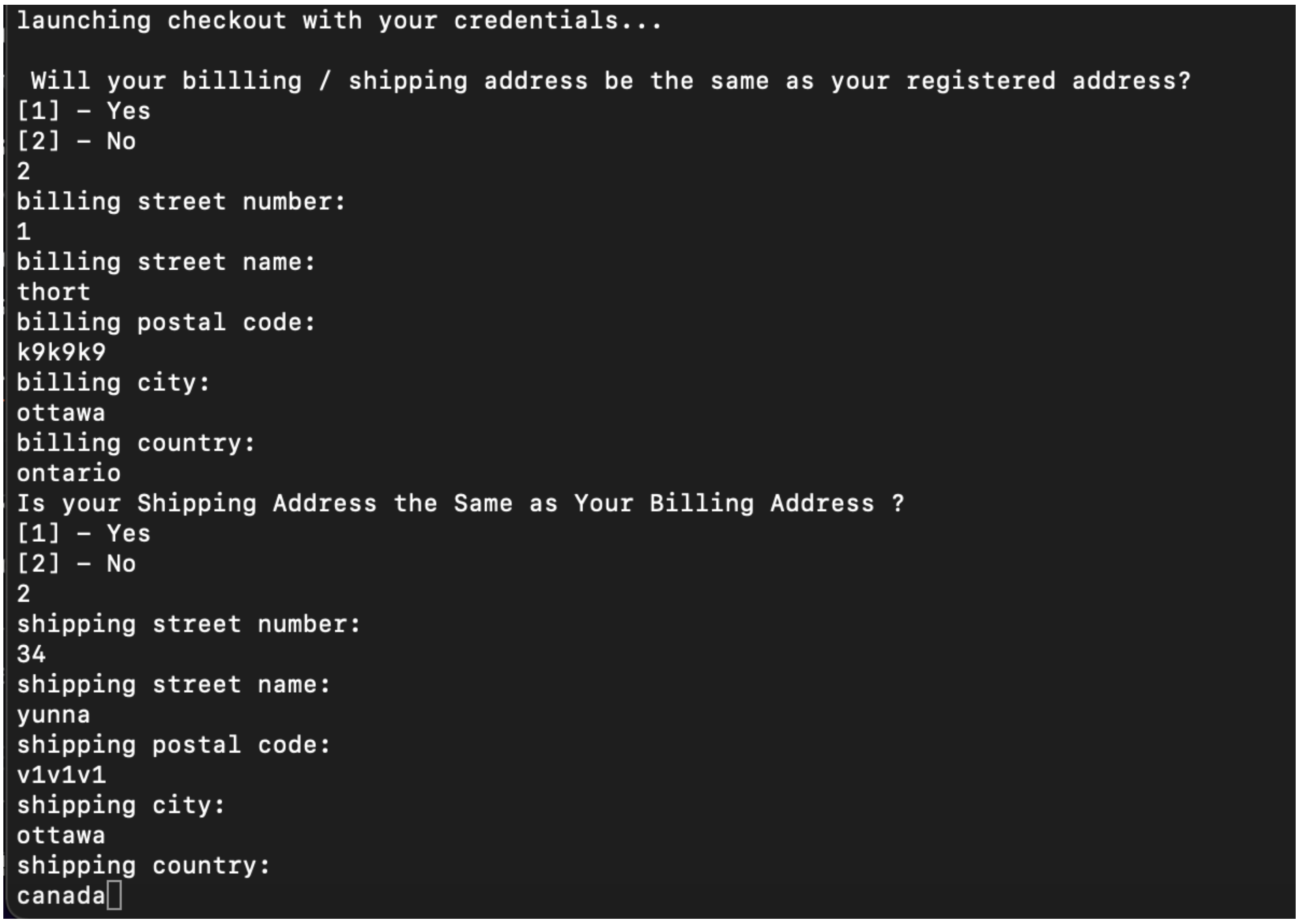
```
Welcome new-comer, provide the following information and we will build you an account
Username:
josh
Password:
password
Email-address:
josh@email.com
First Name:
Josh
Last Name:
Kline
Phone number in the format XXXXXXXXXX.
If you have more then one, press 'enter' and put additional numbers on a new line.
Press 'enter' again when done.
3454657689
7654543432
9878767656

(345) 465 7689
(765) 454 3432
(987) 876 7656
Billing street number:
12
Billing street name:
roady mcroad
Billing postal code:
K2R6T7
Billing city:
ottawa
Billing country:
Ontario
Is your shipping address the same as your billing address ?
[1] - Yes
[2] - No
2
Silling street number:
45
Silling street name:
name name name
Shipping postal code:
K6T5R4
Shipping city:
ottawa
Shipping country:
Canada
```

Here we provide all our info, and this time we decided the billing and shipping are different for the registered account. We also have the option of provided as many phone numbers as neede,d which will be formatted and then stored in the DB.



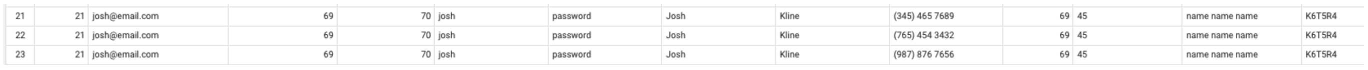
```
Customer Email Added
Region Added
Address Added
Shipping Info Added
Region Added
Address Added
Billing Info Added
Customer Added
Customer Phone Number Added
Customer Phone Number Added
Customer Phone Number Added
launching checkout with your credentials...

 Will your billling / shipping address be the same as your registered address?
[1] - Yes
[2] - No
```

Upon submitting we see that all the entities populated succesfully, and are again prompted if the checkout specific billing and shipping match the registered billing and shipping we just provided. Lets say no this time. We will also assum the checkout shpping and billing are not equal. this one is a real doozy!



```
launching checkout with your credentials...

 Will your billling / shipping address be the same as your registered address?
[1] - Yes
[2] - No
2
billing street number:
1
billing street name:
thort
billing postal code:
k9k9k9
billing city:
ottawa
billing country:
ontario
Is your Shipping Address the Same as Your Billing Address ?
[1] - Yes
[2] - No
2
shipping street number:
34
shipping street name:
yunna
shipping postal code:
v1v1v1
shipping city:
ottawa
shipping country:
canada
```

Provide all the info and we are good to go. We will then be asked to provide the date again, and the rest of the sequence is exactly like it was for the login sequence.



21	21	josh@email.com	69	70	josh	password	Josh	Kline	(345) 465 7689	69	45	name name name	K6T5R4
22	21	josh@email.com	69	70	josh	password	Josh	Kline	(765) 454 3432	69	45	name name name	K6T5R4
23	21	josh@email.com	69	70	josh	password	Josh	Kline	(987) 876 7656	69	45	name name name	K6T5R4

We can also see if we query in pgAdmin that ouur new user has been generated, and all 3 phone numbers have been stored.

The Owner Side

Bonus Features

- When searching for a book by title, author, genre, or publisher, we perform an approximate search. Do do this, we return the 5 results who's attribute's string requires the fewest insertion, deletions, or replacements of any given character to comprise the desired search string. This does a decent job at behaving like an approximation engine, though the short coming is in searching for small strings. For example, if you search for "t" in book titles, even if there is one book called "ttt" and another called "z", "z" will return with higher priority since it only requires 1 replacement, whereas "ttt" requires 2 deletions. If the search string is around the average length of the availible string to compare against, and the search strings actually contains some valid sequeence of characters, it performs decently well. For the isbn aproximation, wee instead compare the absolute difference between the search isbn and those on record.
- We show similar books when searching by any given parameter. You will always return the top 5 matches, assuming there are 5, for any given search.
- We implemented a sense of "state" throughout the client side of the aplication. Once you go to checkout ffor the first time, you will be promted to either sign in or create a new account. After doing so, your credentials will be stored such that you can navigate around the store in any direction you want, and when it comes time to checkout again, you will not need to provide your information for a second time. We keep track of the logged in users credentials, the connection point to the database stored on the hostss computer, and the cart currently in action through out the alication.

Github Repository

link

Appendix I

Availibility on Dec 18th For presentation

- 11:00 am - 11:20 am
- 12:00 pm - 12:20 pm
- 12:40 pm - 1:00 pm

Shortcomings

I am including these to show that we aknowlegde the issues, and with more timee, could have fixed them as we are aware of the solution implementation. This project has taken a very long time to put together and out of the interest of preparing for final exams, we simply do not have enough time to correct these little things. They are, however, worthy of note, and interesting to discuss.

- We modelled isbn to be the PK for a book. As a result, book_cart requires both a PK from book, and PK from cart. The consequence of this is that we can not add 2 copies of the same book into any given cart. One way of solving this could have been to let individual books have individual id's, and for isbn, we could have added in many copies of the same book, with unique id's but repeating isbns. A second way we could have solved this wuld be to simply keep track of a multiplier in the cart. If ssometone attempts to add a second copy of the isbn, instead of rejecting that, we could catch the error thee DB throws, and instead tick up a "num_copies_in_cart propert by 1. We could then charge the customer num_copies_in_cart X book_prce, and pay the publisher in a similar manner. When ticking up or down the num_copies_sold and num_copies_in_stock, we again could have just added or removed based on the multiplier.
- We made the somewhat silly decision of capturing day / month / year as distinct attributes, and allowed the user to provide this information. What would have made mroe sense, and is how things are done in the real world, would have been to use a Date or DateTime object, which generated a universally formatted timestamp without the need of user input. As a consequence of having NOT done this, and not having a sense of time, we don't currently have a way of checking "last months sales". In part, because someone can cheeckout and claim that it is march 3rd, 1805, even if it is really December 16th 2021. We do send a faux email to the correct publisher asking for more books, but in order to get the correct amount, we would need to implement a query like this ... SELECT sum(cart_books.isbn) FROM checkout JOIN cart ON checkout.cart_id = cart.cart_id JOIN cart_books ON cart.cart_id = cart_books.cart_id WHERE month = current_month - 1 AND isbn = isbn_to_order
- You may notice there is one sequence when running the program that is a bit ugly. When a checkout is complete and an order is generated, it asks you to aknowledge to go forward. Then it tells you you cannot checkout with an empty cart, and assks you to aknowledge. Then it tells you you can't view an empty cart, and asks you aknowledge. Then finally it returns control to you and lets you interact with the application. This is an artifact of how I implemented state. Any time you go into a deeper level of the application, and either change the connection to the DB, the user, or the cart, I update the state at that layer of depth, and pass the modified state up a level to the menu you were in prviously. In order to go from a "deep" level where the state was changed, to a "shallow level" where the state needs to be updated, so that the new state can permiate the rest of the application, I need to send the user back 1 level at a time. Upon order complete, one of the state changes is setting your current cart to nil, so as you back through all the menus to ass the state along, it keeps rejecting you because you do not have a valid cart. Ideally, I could bump you back to the start menu in one go, but I was having trouble implementing this, and did not have enough time to sort it out.
-