

LIBER

Online Book Store

Design Report

by

Bardia Parmoun
101143006

Christopher Shen
101149908

to

Professor Ahmed El-Roby

COMP 3005

December 17th, 2021

1. Problem Statement

Design and implement an application for an online bookstore (Look Inna Book). This application lets clients browse a collection of books that are available in the bookstore. A client can search the bookstore by book name, author name, ISBN, genre, etc.. When a book is selected, information on the author(s), genre, publisher, number of pages, price, etc. can be viewed. A client can select as many books as she likes to be added to the checkout basket. A client needs to be registered in the bookstore to be able to checkout. When checking out, the client inserts billing and shipping information (can be different than those used in registration), and completes the order. The bookstore has the feature of tracking an order via an order number. A client can use this order number to track where the order is currently. Although shipping is carried out by a third-party shipping service, the online bookstore should have the tracking information available for when the client inquires about an order using the order number. Assume all books are shipped from only one warehouse (no multiple order numbers for multiple books shipped from multiple warehouses). The bookstore owners can add new books to their collections, or remove books from their store. They also need to store information on the publishers of books such as name, address, email address, phone number(s), banking account, etc.. The banking account for publishers is used to transfer a percentage of the sales of books published by these publishers. This percentage is variable and changes from one book to another. The owners should have access to reports that show sales vs. expenditures, sales per genres, sales per author, etc.. The application should also be able to automatically place orders for new books if the remaining quantity is less than a given threshold (e.g., 10 books). This is done by sending an email to the publisher of the limited books to order a number of books equal to how many books were sold in the previous month (you do not have to implement the email sending component).

2. Project Report

2.1 Conceptual Design

2.1.1 List of assumptions for the design

Entities

Address entity set: This is the entity set for holding information about a specific address like where an order will be shipped to, the address of a warehouse, the address of a specific publisher etc, the address will be distinguished by a unique `address_id`, which will server as the primary key for the entity set, other attributes are just required to actually use an address, these would be `building_num`, `street`, `city`, `state`, `country` and `postal_code`, the reason for address being its own entity set is to easily assign a single address to multiple other entities/relations (like if many orders are to the same address).

BankAccount entity set: Since books are being sold from publishers to the store and then sold to customers, the BankAccount acts as a black box where money is withdrawn from to buy

books and deposited too when the store needs to pay for a publisher's cut assuming that all we need to access the funds is the account number and its amount to check whether or not a withdrawal (for a payment) is possible. The only primary key needed is that of the account number, as each account number should be unique.

Book entity set: This entity set serves to represent all the books that may be available for sale from the book store, it stores all required information such as; ISBN, title, author_name, genre, etc, as well information on how many copies are available, cost to the store, cost to buyers, publishers cut and restock threshold, all of which we have decided to store along with each individual book since every book may have a different value for these attributes. For the primary key, only the ISBN attribute is needed, since ISBN serves as a book id (International Standard Book Number), and is unique to every book, other attributes are free to be non-unique, such as sharing publishers, authors, etc.

Orders entity set: The entity set with the most relationships, the Order entity set has a unique id, order_number which acts as the primary key, there is also attributes keeping track of the current status of the order (pending, shipped, delivered, etc) as well as attributes for the date the order was placed, the address the order will be going to, there is also an attribute final_total which is the final dollar amount of the order. We are making the assumption that all the tracking data (for status) is handled by a third-party shipping company that will provide tracking updates, which will be tied to an order_number so that a client can check on their individual orders.

Publisher entity set: This entity set keeps track of tuples containing information needed on all publishers whose books are available for sale in the book store, such as their unique publisher_id, name, email, etc. The primary key chosen for this entity set was just the publisher_id as it acts like any other id, it is unique to each publisher, while other attributes may not be.

Sales entity set: The purpose of the Sales entity set is to allow a way for the store (and its staff) a way to check how many of each book has been sold during a given time, in our case we have assumed that this will be stored by month, thus the inclusion of the month and year attributes, the way sales are tracked is through keeping a quantity sold tally for each ISBN (book) that has been sold by the store, The reason for this is to allow for an easy query to get any information needed on sales figures, by keeping track of the books and the quantity sold, we can easily use queries to get figures on books by specific authors, genres, publishers, etc. The primary keys chosen were the ISBN, month and year since like before, ISBN is unique to every book, month and year since we are assuming that data is collected by month (year + month also lets us to specific queries on a variable range of months/years).

Staff entity set: The Staff entity set acts as the “owner” account for the store, and has a unique staff_id along with an associated email (the store's email). This entity represents the staff

members of the book store that have permission to order more books, as well as the ability to modify any of the tuples present in the book entity set. The only primary key that will be needed is that of the staff's id, since there may be multiple staff members with the authority to make requests and modify books for the store. We are making the assumption that staff members have the authority to request more books and adjust book listings and the store shares one email (thus email is not a primary key).

Client entity set: This entity set contains all the necessary attributes for a registered client, they each have a unique client_id acting as the primary key along with extra attributes such as their name, associated email, and address (default address). We are making the assumption that only a client account is only linked to one email and one address.

Warehouse entity set: This is a fairly basic entity set that provides information on warehouses that can handle the stores orders, as such each warehouse is uniquely identified by a warehouse_id which serves as its primary key, address is included but is not part of the primary key as a large warehouse have separate smaller sections with different purposes (thus unique id, shared address).

Relationships

check_out: This relationship is for relating Orders to the books being bought. The relationship is total participation 1..* to 0..*. On the Order side it is total participation because an order must contain books, you cannot order nothing (not a valid purchase), it is 1..* since many different Orders can be placed on the same book. The book side is 0..* since not every book is part of an Order, but at the same time many books can be in the same Order.

emails: This relationship represents the staff members from the store requesting more of a certain book from their associated publishers, the relationship is a 0..* to 0..*, the Staff side is 0..* since not all staff members send emails, but there may be a staff member responsible for emailing multiple publishers. The publisher side is also 0..*, this is because a publisher may have many of its books available for purchase at the bookstore, or they might have no books available for purchase yet (even though they have no books in the store, the staff members may want to get books from them in the future).

handle_account: This represents the payments that are being made for an order to some banking account, the relationship is a total participation 1..* to 0..*. The Order side of the relationship is a total participation with 1..* since every Order made must have a client BankAccount that is paying the Order and one or more publishers whose books were sold and they now need the cut (%) paid to them. The BankAccount side is 0..* since in any transaction, not every bank account participates in every transaction, but there also might be more than one publisher that needs to have their cut (%) of sales sent to them when a client buys books.

manage: This is a simple relationship which shows which staff members are responsible/have done changes to any of the books that are being sold in the book store (ie, adding more books, changing pricing etc), this is a 0..* to total participation 1..*, it is 0..* on the staff side since not every staff member may be assigned to the management of the books, while the books side is total since every book being sold must have been added by some staff member (can't just magically appear to be sold) and is 1..* since one book may be managed by more than one staff member at a time. (ie, we are assuming that more than one staff member can change what price the book is being sold for and change what the restock threshold should be).

order_address: This relationship represents the Address than an Order will be going to (ie, destination for the order), this relationship is total participation 1..* to one. The Order side is total participation since every Order must be made to be sent to some Address (you cannot send an Order to nowhere) and it is 1..* since many Orders could be placed and sent to the same place (ie, a Client places many Orders to his house). The Address side is one since you cannot split a single Order into being sent to two different Addresses.

publisher_account: This relates a publisher to their designated bank account, to which funds for book sales (ie, the % cut per book sold) will be sent to. This relationship is total participation 1..1 to 0..1, the publisher side is total participation because a publisher must have a bank account to which money can be sent to and 1..1 since we are assuming that the Publisher only want money sent to one central publisher account. The BankAccount side is 0..1 since not every BankAccount is one that belongs to a Publisher, some belong to Clients, it is capped at 1 as previously explained, a Publisher is only going to have a single main account, thus 0..1.

publisher_address: This represents the Publisher having an Address. The relationship is a one to one relationship. The publisher is only one since a publisher can only have one primary address it is at (like a main company address), and the address side is also one since a single address cannot have two publishers at it (this is an assumption that is being made).

update_sales: Whenever an order occurs, it will update some sales figures stored as the Sales entity. This relationship is total participation 1..* to total participation 1..*. The Order side is total since every Order will change the sales figure for at least one book (you cannot have an empty order..) and is 1..* since multiple Orders of the same book will update the same Sales entity. Meanwhile on the Sales side total since no Sales entity can occur without an Order taking place and is 1..* because, if a book does not sell at all within the given time period (1 month), that books sale tuple for that month will not exist, thus if that books sales tuple exists, it must have come from some Order, multiple Sales entities can be updated by a single Order as well if that Order contained multiple different books, thus 1..*.

send_order: This is a fairly simple relationship that shows which Order is being delegated to which Warehouse, this is a total participation 1..* to 1..1 relationship. The Order side is total since every Order must be passed on to a Warehouse in order to be fulfilled, it is 1..* since many Orders may share a shipping Warehouse. The Warehouse side is 1..1 since we are making the assumption that each Warehouse is being used to ship out Orders in some capacity (ie, no Warehouse is doing nothing), its limited to 1 since no Order can be split across more than 1 Warehouse (as per the project requirements).

tracks: The Tracks relationship is what relates a Client to the Order(s) they may have made, this relationship is a one to total participation 1..*. The Client since any order made by a Client must be associated (trackable) but the client itself. On the Order side it is total since every Order must have come from a Client and it is 1..* since a Client can make more than one Order. (similar to previous assumptions, only registered Clients have an actual Client entity tuple and can Order).

client_account: This serves a similar purpose to the publisher_account, it is to relate a Client to some BankAccount, this relationship is 1..* to 0..*. The Client side is 1..* since we are assuming that upon registering as a Client they provide a sort of default payment method (default BankAccount they will pay from), it is limited by * as they can then add more BankAccounts if they like, similar to how you can add multiple payment methods to an Amazon account. The BankAccount side is 0..* since some of the BankAccounts as previously mentioned are associated with Publishers and not Clients, it is limited by * since a Client can have multiple BankAccounts.

client_address: This represents the Client accounts having a default Address assigned to their account (ie, they gave a default address when they registered). The relationship is total participation 1..* to 0..1, the Client side is total participation since all client accounts need to have some address that they have orders shipped to and is 1..* since multiple clients might share the same address, the Address side is 0..1 since not every address is necessarily that of a client, (could be a warehouse address, etc) and is limited to 1 since the client will only have one default address.

warehouse_address: This represents the Warehouse having an Address. The relationship is a one to one relationship. The warehouse is only one since a warehouse can only have one location/address it is at, and the address side is also one since a single address cannot have two warehouses at it (this is an assumption that is being made).

2.1.2 ER Diagram of the Database

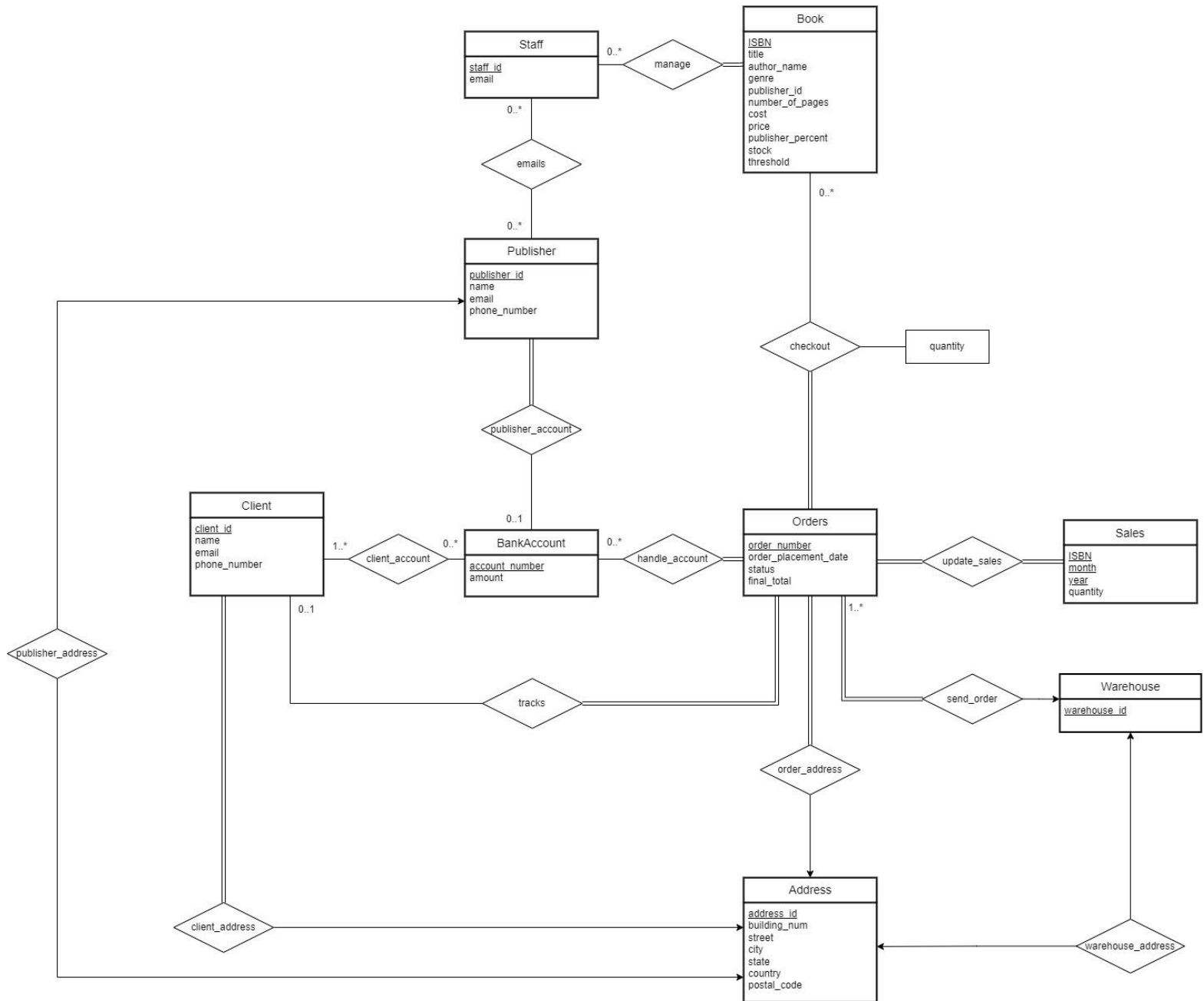


Figure 1: ER Diagram of the online bookstore database.

2.2 Reduction to Relational Schemas

Many to one and one to one relations are reduced down to single attributes. Those relations include client_address, warehouse_address, order_address, publisher_address, and send_order.

2.2.1 Entities

- Address(address_id, building_num, street, city, state, country, postal_code)
- BankAccount(account_number, amount)
- Book(ISBN, title, author_name, genre, publisher_id, number_of_pages, cost, price, publisher_percent, stock, threshold)
- Order(order_number, order_placement_date, status, final_total, address_id, warehouse_id)
 - order_address relation could be reduced to a single attribute.
 - warehouse_address relation could be reduced to a single attribute.
- Publisher(publisher_id, name, email, phone_number, address_id)
 - publisher_address relation could be reduced to a single attribute.
- Sales(ISBN, month, year, quantity)
- Staff(staff_id, email)
- Client(client_id, name, email, phone_number, address_id)
 - client_address relation could be reduced to a single attribute.
- Warehouse(warehouse_id, address_id)
 - warehouse_address relation could be reduced to a single attribute.

2.2.2 Relationships

- check_out(ISBN, order_number, quantity)
 - many to many relation with total participation of Order
- emails(staff_id, publisher_id)
 - many to many relation
- handle_account(order_number, account_number)
 - many to many relation with total participation of Order
- manage(ISBN, staff_id)
 - many to many relation with total participation of Book
- publisher_account(publisher_id, account_number)
 - zero or one to one relation with total participation of Publisher
- update_sales(order_number, ISBN, month, year)
 - many to many relation with total participation of both sides.
- send_order(order_number, warehouse_id)
 - many to one relation from Order with total participation of Order.
- tracks(order_number, client_id)
 - many to one relation from Order with total participation of Order.
- client_account(client_id, account_number)
 - many to many relation.

2.3 Normalization of Relation Schemas

2.3.1 Functional dependencies

Entities

Address:

- $\text{address_id} \rightarrow \text{building_num}, \text{street}, \text{city}, \text{state}, \text{country}, \text{postal_code}$
- $\text{postal_code} \rightarrow \text{state}, \text{country}$

BankAccount:

- $\text{account_number} \rightarrow \text{amount}$

Book:

- $\text{ISBN} \rightarrow \text{title}, \text{author_name}, \text{genre}, \text{publisher_id}, \text{number_of_pages}, \text{cost}, \text{price}, \text{publisher_percent}, \text{stock}, \text{threshold}$

Order:

- $\text{order_number} \rightarrow \text{order_placement_date}, \text{status}, \text{final_total}, \text{address_id}, \text{warehouse_id}$
-

Publisher:

- $\text{publisher_id} \rightarrow \text{name}, \text{email}, \text{phone_number}, \text{address_id}$

Sales:

- $\text{ISBN}, \text{month}, \text{year} \rightarrow \text{quantity}$

Staff:

- $\text{staff_id} \rightarrow \text{email}$

Client:

- $\text{client_id} \rightarrow \text{name}, \text{email}, \text{phone_number}, \text{address_id}$

Warehouse:

- $\text{warehouse_id} \rightarrow \text{address_id}$

Relations

checkout:

- many to many relation, no functional dependency can be made

emails:

- many to many relation, no functional dependency can be made.

handle_account:

- many to many relation, no functional dependency can be made.

manage:

- many to many relation, no functional dependency can be made.

publisher_account:

- Since one side is 0..1 there is a possibility for null values so no functional dependency can be made.

update_sales:

- many to many relation, no functional dependency can be made.

tracks:

- Since one side is 0..1 there is a possibility for null values so no functional dependency can be made.

client_account:

- many to many relation, no functional dependency can be made.

To check if our relation schemas are in a good normal form, we check functional dependencies listed above with their corresponding relations.

2.3.2 Tests for good normal form

When looking at our entity sets, it is clear that all entity sets other than Address are already in good normal form (BCNF) since their only functional dependencies are just their primary keys. The only entity set that has more than one functional dependency in our case is the Address entity set, which will need to be tested.

1. Address

Letting attributes in entity set Address= R

R = {address_id, building_num, street, city, state, country, postal_code}

Testing the functional dependencies of Address:

1. address_id → building_num, street, city, state, country, postal_code

Since address_id is the primary key of this relation we can conclude that this dependency does not violate BCNF.

2. $\text{postal_code} \rightarrow \text{state, country}$

postal_code is not a primary key of this relation:

$$(\text{postal_code})^+ = \text{state, country} \subset R$$

This dependency violates BCNF, so we need to decompose R using the algorithm defined in lectures.

$$R_1 = \{\text{postal_code, state, country}\}$$

$$F_1 = \{\text{postal_code} \rightarrow \text{state, country}\}$$

postal_code is the primary key of this new relation so we can conclude that this dependency is not violating BCNF.

$$R_2 = \{\text{address_id, building_num, street, city, postal_code}\}$$

$$F_2 = \{\text{address_id} \rightarrow \text{building_num, street, city, state, country, postal_code}\}$$

address_id is the primary key of this new relation so we can conclude that this dependency is not violating BCNF.

It is clear that all dependencies of F_1 and F_2 do not violate BCNF and thus R_1 and R_2 are in BCNF.

Therefore the BCNF decomposition of address is:

$$R_1 = \{\text{postal_code, state, country}\}$$

$$R_2 = \{\text{address_id, building_num, street, city, postal_code}\}$$

We can call R_1 entity set “region” and keep the name “address” for the R_2 entity set.

This decomposition is also lossless.

2.4 Database Schema Diagram

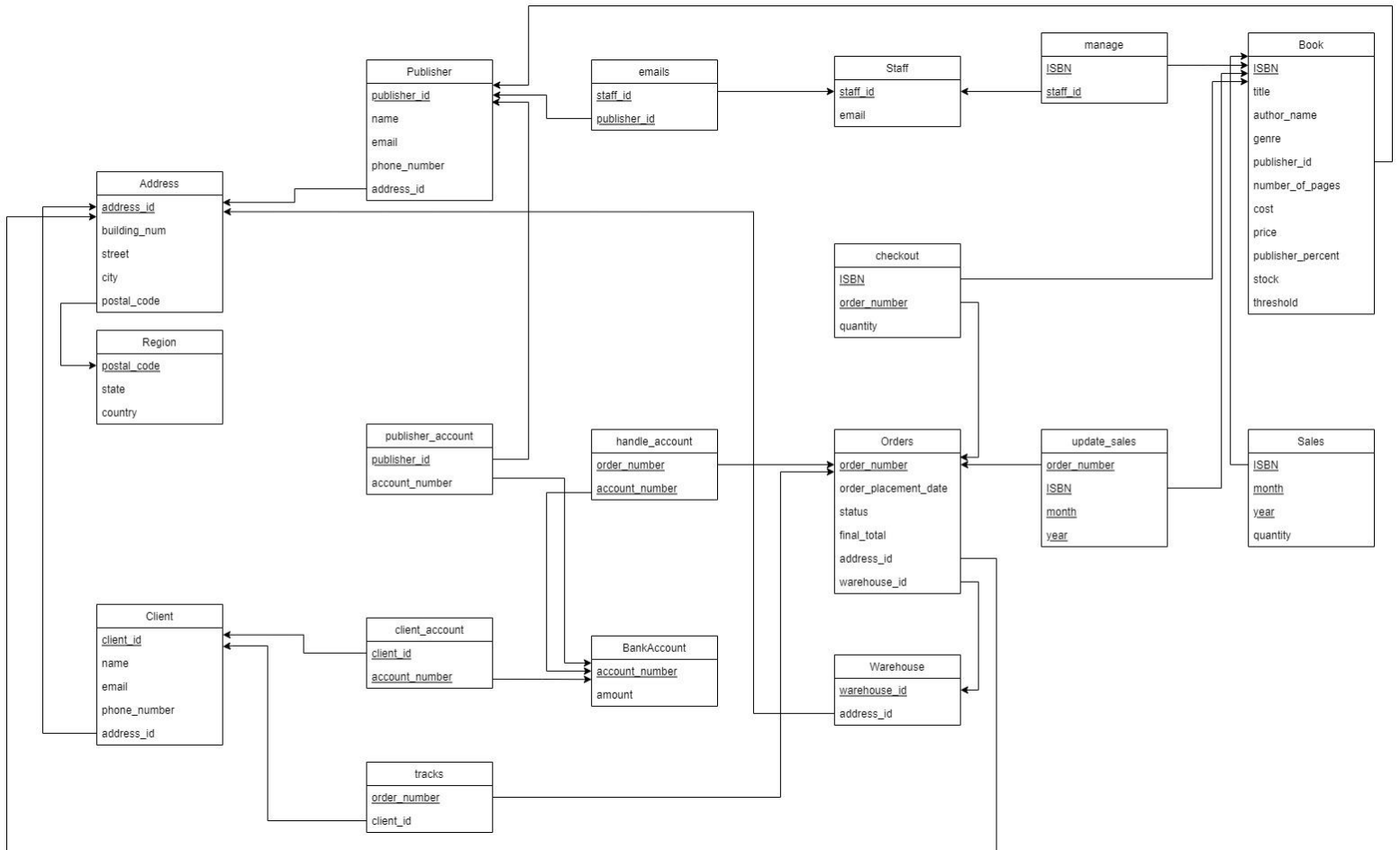


Figure 2: Schema Diagram of the online bookstore database.

2.5 Implementation

Here is the link to the video demo of the project:

[INSERT LINK]

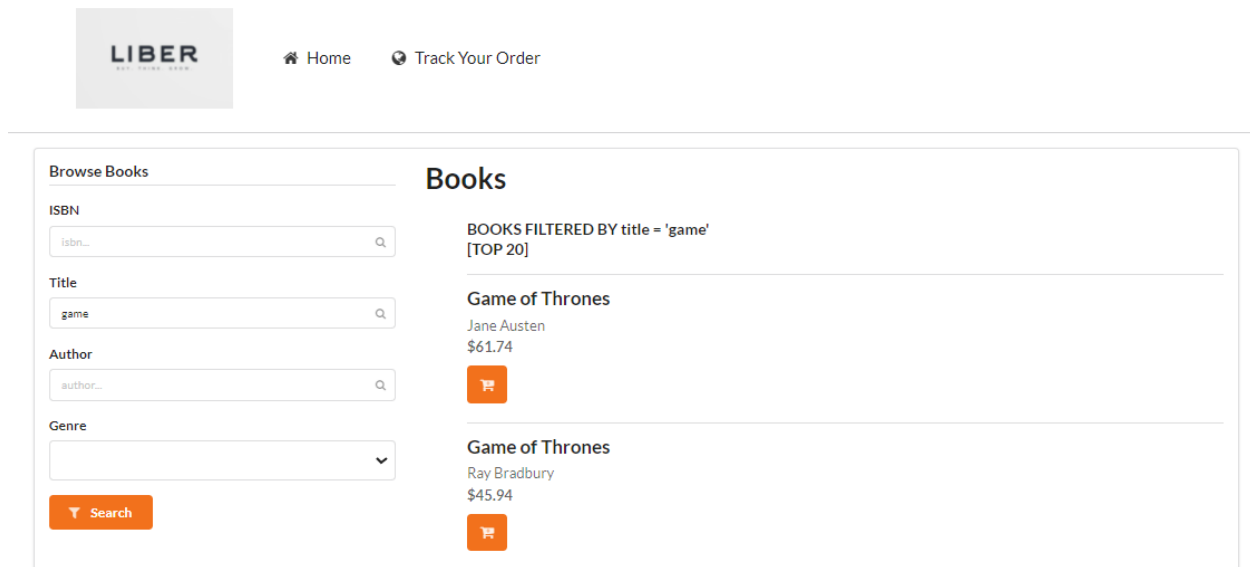
2.6 Bonus Features

Here is a list of the bonus features that were included in the project:

1. Custom Searches:

The SQL queries were designed in such a way by only typing a some parts of the title or the author name of the books, all the books that match the name will be returned.

Here is an example of the custom searches:

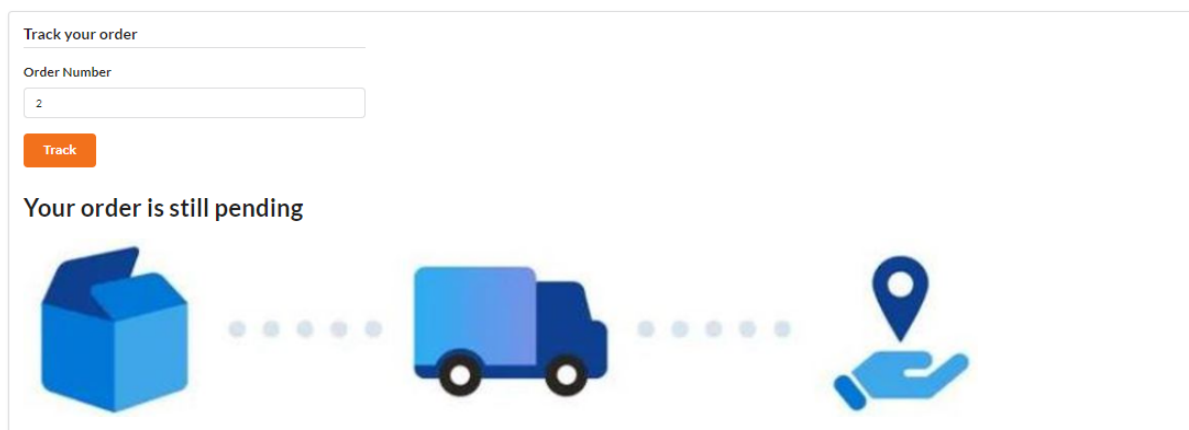


The screenshot shows the LIBER website's search interface. At the top, there is a navigation bar with the LIBER logo and links for Home and Track Your Order. Below the navigation bar, there is a search section titled "Browse Books" with input fields for ISBN, Title, Author, and Genre. The Title field contains the text "game". A search button is located below the input fields. To the right of the search section, there is a "Books" section titled "BOOKS FILTERED BY title = 'game' [TOP 20]". This section displays two book results: "Game of Thrones" by Jane Austen priced at \$61.74, and "Game of Thrones" by Ray Bradbury priced at \$45.94. Each book result includes a small orange button with a shopping cart icon.

Figure 3: An example of using custom searches

2. Order Shipping Status

The website has the option of providing the customers with a tracking number that they can use to track their orders. Every 5 minutes the status of the orders will change and custom diagram will be displayed showing the status of said orders.



The screenshot shows the LIBER website's order tracking interface. At the top, there is a section titled "Track your order" with an input field for the Order Number. The Order Number field contains the text "2". A "Track" button is located below the input field. Below the "Track" button, there is a message that says "Your order is still pending". Below the message, there is a shipping diagram showing a blue box, a blue truck, and a blue location pin, connected by dotted lines.

Figure 4: Tracking an order when it is just placed.



Figure 5: Tracking an order after it has been shipped.



Figure 6: Tracking an order after it has been delivered.

3. Changing the details of already existing books

The bookstore also has the option of changing the details of the books that already exist in the system. By searching the ISBN of the book, they can change details such as the cost of the book, its price, publisher's cut, threshold, and its stock.

Book To Manage

ISBN

438-14-11430-36-3

Search

Book Information Panel

Book Found Through ISBN

ISBN

438-14-11430-36-3

Title

Gargantua and Pantagruel

Author

Cormac McCarthy

Genre

Biography

Publisher

Joe and Smiths

Number of Pages

422

Cost

6.19

Price

65.97

Publisher % Cut

34.00

Stock

42

Threshold

9

Save Changes

Figure 7: Changing the details of the books that already exist in the store

4. Generating graphs for the reports

2.7 Github Repository

Here is the link to the github repository where the codebase for this project can be found:

<https://github.com/bardia-p/LIBER>

Appendix I

List of availabilities:

Dec 20, 2021:

10:30 AM

2:00 PM

6:00 PM

(Any time 9:00 AM to 10:00 PM)