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 users browse a collection of books that are available in the bookstore. A user 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 user can select as many books as she likes to be added to the checkout basket. A user needs to be registered in the bookstore to be able to checkout. When checking out, the user 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 user 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 user 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:

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being sold from publishers to the store and then sold to customers, the BankAccount acts as a black box where money is withdrawn from to buy books and deposited too when the store needs to pay for a publisher's cut assuming that all we need to access the funds is the account number and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, authoer_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.

Cart entity set: The Cart entity set represents a users shopping cart, where they may place books that they intend to purchase (add and remove to cart), in our case, the cart has three attributes, cart_id which is unique to each cart and servers as the primary key, user_id which would indicate which user the cart belongs to and cart_total which is the current total price of all books in the cart (at the time). We have decided not to include the user_id as part of the primary key for two reasons, one, a user has the ability to have multiple Carts, ie, if they decide to split their shopping into multiple sessions but want to plan it all out now second being that we are only allowed to let users with accounts checkout/purchase the books as per the requirements, this means that if the user_id is null we check whether or not the owner of the cart is in fact a register user without causing issues that would arise with a primary key being null (thus, we leave is as a non-primary key).

Order 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, where the order destination will be and where the orders payment will be taken from (the account maybe be different from the one registered to a users account as they are allowed to change it on placing an order, same goes for destination, the user can specify an address different from their own), 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 user 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).

User entity set: This entity set contains all the necessary attributes for a registered user, they each have a unique user_id acting as the primary key along with extra attributes such as their name, associated email, address (default address) and account_number (default payment option). We are making the assumption that only one email can be linked to a user account (sign up email) and only one default payment option, which is why email and account_number are not primary keys.

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 Carts of books being bought. The relationship is total participation 1..1 to 0..1. On the Order side it is total participation because an order must be on a cart of books, you cannot order nothing (not a valid purchase). The Cart side is 0..1 since not every cart is part of an Order, since only registered users can actually Order/buy books from the store, meaning that if someone adds a bunch of books to their cart, but don't end up buying them, that cart was never Ordered/bought, we are assuming that every Order can only be on 1 cart, ie, we assume that we a user cannot checkout/Order 2 Carts at once, in that case we would just combine the 2 Carts into one large cart, thus 0..1.

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 user 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 user buys books.

in_cart: The in_cart relationship is used to associate which books in the store are currently in someone's cart (ie, which books are being considered for purchase), the relationship is a 0..* to 0..*, (many to many), this is because no books can be in any cart, or even all books could be in one cart, ie, there is no limitation to the cart and book combinations, thus it is many to many. There is also an extra attribute associated with this relationship, which is essentially just a running total of the value of all books currently in the cart.

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).

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 Users, it is capped at 1 as previously explained, a Publisher is only going to have a single main account, thus 0..1.

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

user_acocunt: This serves a similar purpose to the publisher_account, it is to relate a User to some BankAccount, this relationship is 1..* to 0..*. The User side is 1..* since we are assuming that upon registering as a User 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 Users, it is limited by * since a User can have multiple BankAccounts.

2.1.2 ER Diagram of the Database

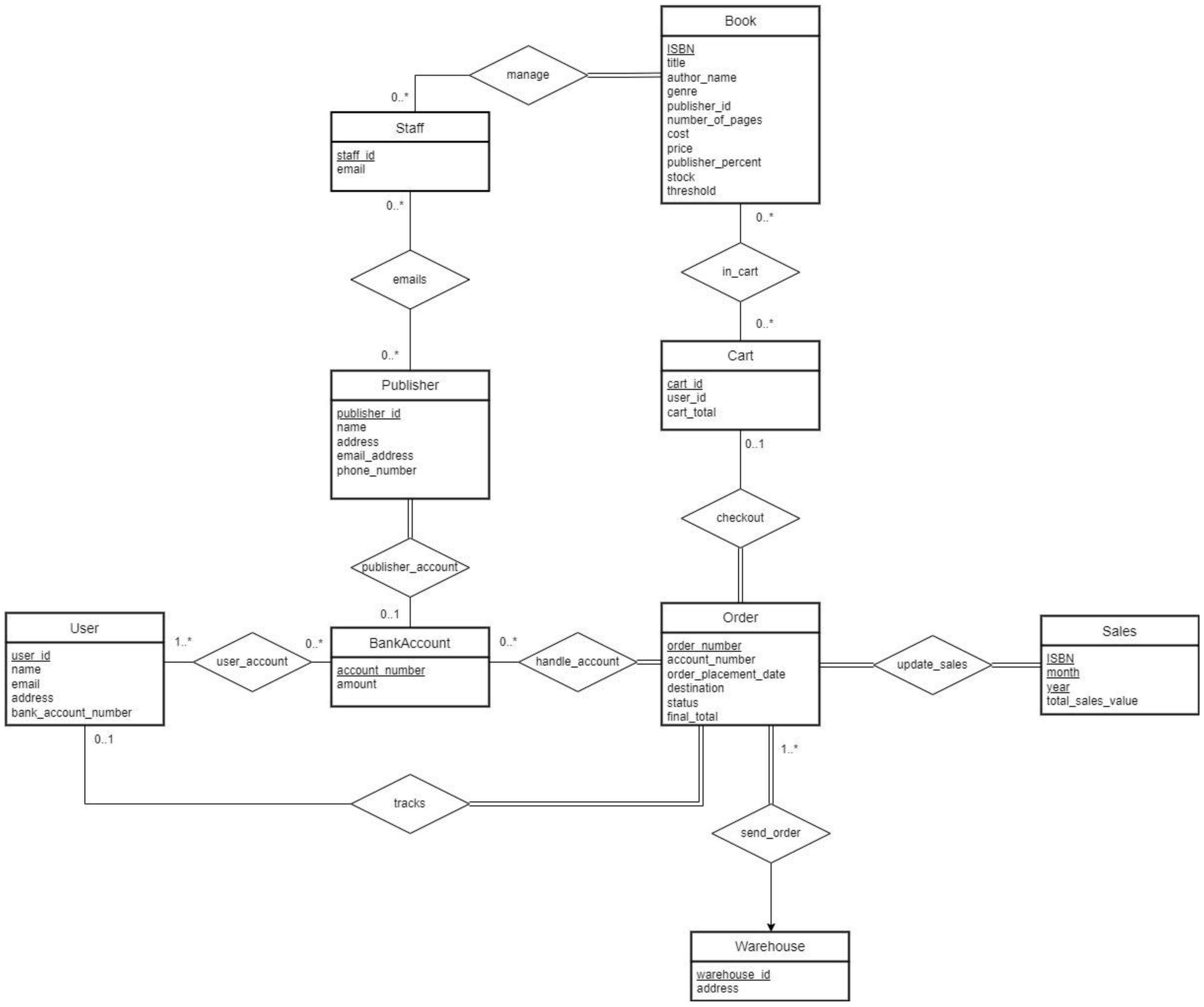

Figure 1: ER Diagram of the online bookstore database.

2.2 Reduction to Relational Schemas

2.2.1 Entities

- BankAccount(account_number, amount)
- Book(ISBN, title, author_name, genre, publisher_id, number_of_pages, cost, price, publisher_percent, stock, threshold)
- Cart(cart_id, user_id, cart_total)
- Order(order_number, account_number, order_placement_date, destination, status, final_total)
- Publisher(publisher_id, name, address, email, phone_number)
- Sales(ISBN, month, year, total_sales_value)
- Staff(staff_id, email)
- User(user_id, name, email, address, account_number)
- Warehouse(warehouse_id, address)

2.2.2 Relationships

check_out(order_number, cart_id)
- zero or one to one 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

in_cart(ISBN, cart_id)
- many to many relation

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, user_id)
- many to one relation from Order with total participation of Order.

user_account(user_id, account_number)

- many to many relation.

2.3 Normalization of Relation Schemas

2.3.1 Functional dependencies

Entities

BankAccount:
account_number → amount

Book:
ISBN → title, author_name, genre, publisher_id, number_of_pages, cost, price, publisher_percent, stock, threshold

Cart:
cart_id → user_id

Order:
order_number → account_number, order_placement_date, destination, status, final_total

Publisher:
publisher_id → name, address, email, phone_number

Sales:
ISBN, month, year → total_sales_value

Staff:
staff_id → email

User:
user_id → name, email, address, account_number

Warehouse:
warehouse_id → address

Relations

checkout:
order_number → cart_id

emails:
many to many relation, no functional dependency can be made.

handle_account:
many to many relation, no functional dependency can be made.

in_cart:
many to many relation, no functional dependency can be made.

manage:
many to many relation, no functional dependency can be made.

publisher_account:
publisher_id → account_number

update_sales:
many to many relation, no functional dependency can be made.

send_order:
order_number → warehouse_id

tracks:
order_number → user_id

user_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

Testing the functional dependencies of the entities:
It can be concluded that all the functional dependencies that were created for the relations are trivial. This is because those dependencies were all created using the primary keys of the tables and as result they can all be considered as trivial.

Testing the functional dependencies of the relations:
1. checkout:
order_number → cart_id
Since order_number is the primary key of this relation we can conclude that this dependency is trivial and checkout is in BCNF.

2. publisher_account:
publisher_id → account_number
Since publisher_id is the primary key of this relation we can conclude that this dependency is trivial and publisher_account is in BCNF.

3. send_order:
order_number → warehouse_id
Since order_number is the primary key of this relation we can conclude that this dependency is trivial and send_order is in BCNF.

4. tracks:
order_number → user_id
Since order_number is the primary key of this relation we can conclude that this dependency is trivial and tracksis in BCNF.

Since it was concluded that all the relations and entities in the ER diagram are in BCNF there are no reductions required.

2.4 Database Schema Diagram

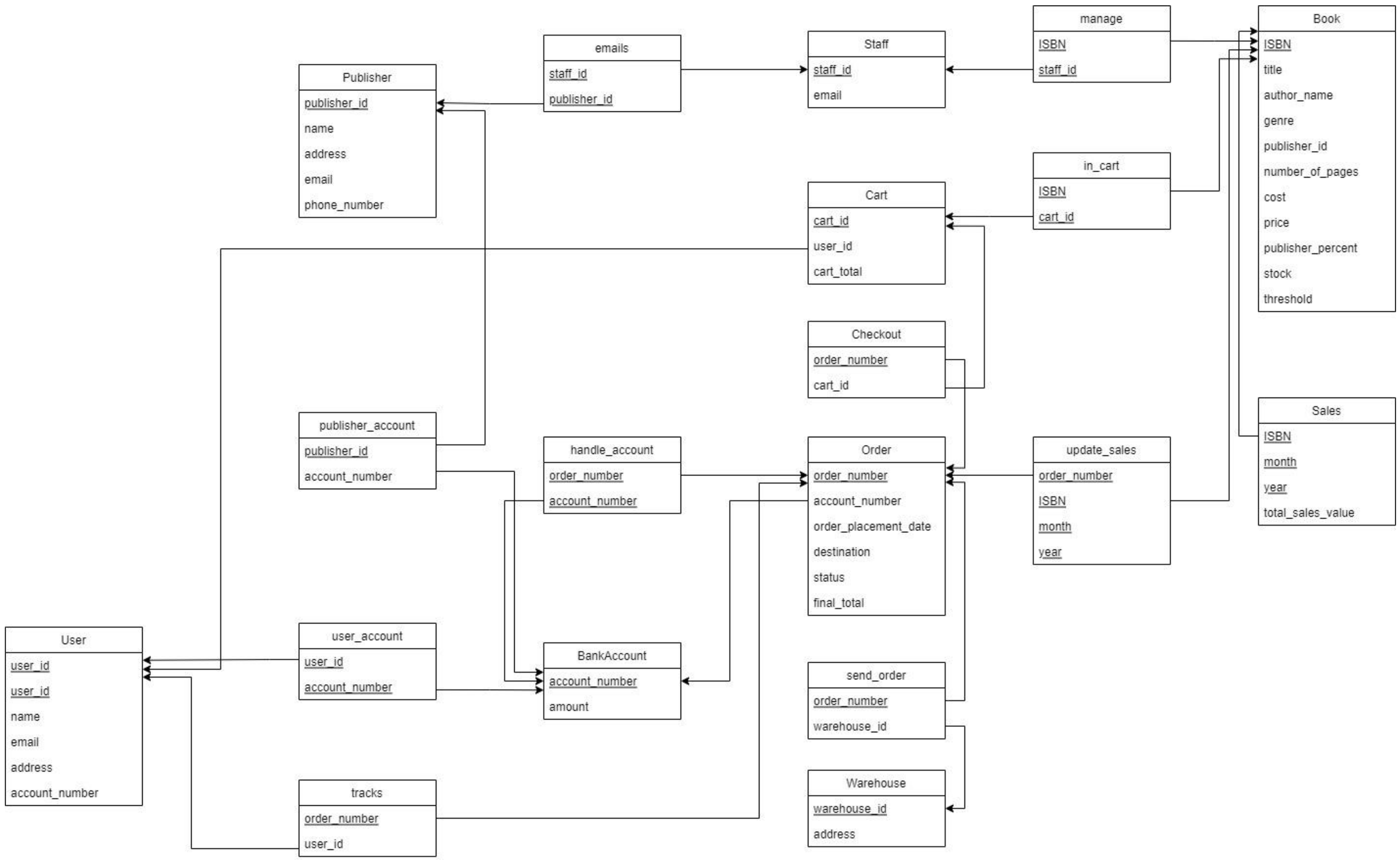

Figure 2: Schema Diagram of the online bookstore database.

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/Online-Bookstore

Appendix I

List of availabilities: