**Conceptual Design:**

The following assumptions are made regarding cardinalities and participation in the ER diagram:

The ISBN of a book is unique, and thus no two books may have the same ISBN. A book must have exactly one genre, exactly one publisher, and at least one author. It may be the case that a book may have the same title, genre, publisher, etc, so long as it has different ISBNs.

A genre’s id is unique and its name is also unique, no two genres may have the same name.

An author’s id is unique. However, it may be the case that two authors may share the same first name and/or last name. An author can write many books.

A publisher’s id is unique. In addition, a publisher’s email address, phone number, and bank account number are unique, no two publishers may have the same email address, phone number, and/or bank account number. It may be the case that multiple publishers with the same name coexist.

A user’s id is unique. However, it may be the case that two users may share the same first name, last name, billing address, and/or shipping address. Importantly, a user’s email is unique – no two users may have the same email address. A user can make many orders. Users won’t have phones as an email address is sufficient to contact them. Additionally, users won’t have passwords as that is beyond the scope of the project.

An order’s id is unique. It may be the case that two orders may share the same user id, total cost, billing address, shipping address, order status, and/or date. However, an order’s tracking number must be unique, and thus no two orders may share the same tracking number. An order must be made by exactly one user. Additionally, each order must be correlated with at least one suborder.

A suborder details exactly one book ordered (by ISBN) and the quantity of that book ordered. A suborder must belong to exactly one order. It may be the case that a book is present in many different suborders.

Addresses such as billing and shipping addresses are stored as a single string.

Further assumptions are made regarding the database:

When removing books from the store, it won’t be the case that the books are deleted from the database, but instead said books are no longer visible to the user at the storefront. In doing so, financial records (e.g. book orders) are kept intact.

A genre cannot be deleted so long as at least one book is part of that genre, however it may be the case that a genre exists with no books falling under that description.

An author cannot be deleted so long as at least one book is written by that author, however it may be the case that an author is not the writer of any of the books in our database.

A publisher cannot be deleted so long as at least one book is published by that publisher, however it may be the case that a publisher has not published any books within our database.

A user can only be deleted from the database should no orders be associated with that user.

Should an order be deleted, its corresponding suborders will be deleted as well – however it generally will not be the case that orders will be deleted, as this would interfere with the bookstore’s financial bookkeeping.

Generally books won’t be deleted from the database as this would result in nulling or deleting corresponding suborders. The functionality of removing books from the bookstore has been discussed above.

In the scope of the project, these deletions won’t be operations available to the user on either interface (removing books operates by hiding said books from the storefront as explained earlier). These assumptions have been made to give context to cascade effects (or lack thereof for some relations) in the database implementation.

**Reduction to Relation Schema**:

books(ISBN, title, genre\_id, publisher\_id, num\_pages, available, stock, price, publisher\_percentage)

genres(genre\_id, genre\_name)

authors(author\_id, author\_name)

publishers(publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account)

users(user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address, user\_shipping\_address)

orders(order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status, date)

suborders(order\_id, suborder\_id, ISBN, quantity)

book\_author(ISBN, author\_id)

**Normalization of Relation Scheme**:

The following are the set of nontrivial functional dependencies for the books relation:

F = {

ISBN -> title

ISBN -> genre\_id

ISBN -> publisher\_id

ISBN -> num\_pages

ISBN -> available

ISBN -> stock

ISBN -> price

ISBN -> publisher\_percentage }

ISBN is the sole key in each nontrivial functional dependency. To examine whether ISBN is a superkey for the relation, we determine (ISBN)+, the closure of ISBN under F:

result = ISBN

ISBN -> genre\_id: result = ISBN, genre\_id

ISBN -> publisher\_id: result = ISBN, genre\_id, publisher\_id

ISBN -> num\_pages: result = ISBN, genre\_id, publisher\_id, num\_pages

ISBN -> available: result = ISBN, genre\_id, publisher\_id, num\_pages, available

ISBN -> stock: result = ISBN, genre\_id, publisher\_id, num\_pages, available, stock

ISBN -> price: result = ISBN, genre\_id, publisher\_id, num\_pages, available, stock, price

ISBN -> publisher\_percentage: result = ISBN, genre\_id, publisher\_id, num\_pages, available, stock, price, publisher\_percentage

(ISBN)+ = (ISBN, genre\_id, publisher\_id, num\_pages, available, stock, price, publisher\_percentage)

Thus we find that ISBN is a superkey for the books relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the books relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the genres relation:

F = {

genre\_id -> genre\_name

genre\_name -> genre\_id }

genre\_id and genre\_name are the keys in the nontrivial functional dependency. To examine whether genre\_id is a superkey for the relation, we determine (genre\_id)+, the closure of genre\_id under F:

result = genre\_id

genre\_id -> genre\_name: result = genre\_id, genre\_name

(genre\_id)+ = (genre\_id, genre\_name)

Next, examining whether genre\_name is a superkey for the relation, we determine (genre\_name)+, the closure of genre\_name under F:

result = genre\_name

genre\_name -> genre\_id: result = genre\_id, genre\_name

(genre\_name)+ = (genre\_id, genre\_name)

Thus we find that genre\_name is a superkey for the genre relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the genres relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the authors relation:

F = {

author\_id -> author\_name }

author\_id is the sole key in each nontrivial functional dependency. To examine whether author\_id is a superkey for the relation, we determine (author\_id)+, the closure of author\_id under F:

result = author\_id

author \_id -> author\_first\_name: result = author\_id, author\_name

(author\_id)+ = (author\_id, author\_name)

Thus we find that author\_id is a superkey for the authors relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the authors relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the publishers relation:

F = {

publisher\_id -> publisher\_name

publisher\_id -> publisher\_email

publisher\_id -> publisher\_phone

publisher\_id -> publisher\_address

publisher\_id -> publisher\_bank\_account

publisher\_email -> publisher\_id

publisher\_phone -> publisher\_id

publisher\_bank\_account -> publisher\_id }

publisher\_id, publisher\_email, publisher\_phone, publisher\_bank\_account are the keys the nontrivial functional dependencies. We first examine whether publisher\_id is a superkey for the relation, determining (publisher\_id) +, the closure of publisher\_id under F:

result = publisher\_id

publisher\_id -> publisher\_name: result = publisher\_id, publisher\_name

publisher\_id -> publisher\_email: result = publisher\_id, publisher\_name, publisher\_email

publisher\_id -> publisher\_phone: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone

publisher\_id -> publisher\_address: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address

publisher\_id -> publisher\_bank\_account: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account

(publisher\_id)+ = (publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account)

Thus we find that publisher\_id is a superkey for the publishers relation.

Next we examine whether publisher\_email is a superkey for the publishers relation, determining (publisher\_email) +, the closure of publisher\_email under F:

result = publisher\_email

publisher\_email -> publisher\_id: result = publisher\_id, publisher\_email

publisher\_id -> publisher\_name: result = publisher\_id, publisher\_name, publisher\_email

publisher\_id -> publisher\_phone: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone

publisher\_id -> publisher\_address: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address

publisher\_id -> publisher\_bank\_account: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account

(publisher\_email)+ = (publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account)

Thus we find that publisher\_email is a superkey for the publishers relation.

Next we examine whether publisher\_phone is a superkey for the publishers relation, determining (publisher\_phone) +, the closure of publisher\_phone under F:

result = publisher\_phone

publisher\_phone -> publisher\_id: result = publisher\_id, publisher\_phone

publisher\_id -> publisher\_name: result = publisher\_id, publisher\_name, publisher\_phone

publisher\_id -> publisher\_email: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone

publisher\_id -> publisher\_address: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address

publisher\_id -> publisher\_bank\_account: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account

(publisher\_phone)+ = (publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account)

Thus we find that publisher\_phone is a superkey for the publishers relation.

Finally we examine whether publisher\_bank\_account is a superkey for the publishers relation, determining (publisher\_bank\_account) +, the closure of publisher\_bank\_account under F:

result = publisher\_bank\_account

publisher\_bank\_account -> publisher\_id: result = publisher\_id, publisher\_bank\_account

publisher\_id -> publisher\_name: result = publisher\_id, publisher\_name, publisher\_bank\_account

publisher\_id -> publisher\_email: result = publisher\_id, publisher\_name, publisher\_email, publisher\_bank\_account

publisher\_id -> publisher\_phone: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_bank\_account

publisher\_id -> publisher\_address: result = publisher\_id, publisher\_name, publisher\_email, publisher\_phone, publisher\_address, publisher\_bank\_account

(publisher\_bank\_account)+ = (publisher\_id, publisher\_name, publisher\_email, publisher\_phone,, publisher\_address, publisher\_bank\_account)

Thus we find that publisher\_bank\_account is a superkey for the publishers relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the publishers relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the users relation:

F = {

user\_id -> user\_first\_name

user\_id -> user\_last\_name

user\_id -> user\_email

user\_id -> user\_billing\_address

user\_id -> user\_shipping\_address

user\_email -> user\_id

user\_id -> publisher\_percentage }

user\_id and user\_email are the keys in the nontrivial functional dependencies. To examine whether user\_id is a superkey for the relation, we determine (user\_id)+, the closure of user\_id under F:

result = user\_id

user\_id -> user\_first\_name: result = user\_id, user\_first\_name

user\_id -> user\_last\_name: result = user\_id, user\_first\_name, user\_last\_name

user\_id -> user\_email: result = user\_id, user\_first\_name, user\_last\_name, user\_email

user\_id -> user\_billing\_address: result = user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address

user\_id -> user\_shipping\_address: result = user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address, user\_shipping\_address

(user\_id)+ = (user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address, user\_shipping\_address)

Thus we find that user\_id is a superkey for the users relation.

Next we examine whether user\_email is a superkey for the users relation, determining (user\_email) +, the closure of user\_email under F:

result = user\_email

user\_email -> user\_id: result = user\_id, user\_email

user\_id -> user\_first\_name: result = user\_id, user\_email, user\_first\_name

user\_id -> user\_last\_name: result = user\_id, user\_email, user\_first\_name, user\_last\_name

user\_id -> user\_billing\_address: result = user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address

user\_id -> user\_shipping\_address: result = user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address, user\_shipping\_address

(user\_email)+ = (user\_id, user\_first\_name, user\_last\_name, user\_email, user\_billing\_address, user\_shipping\_address)

Thus we find that user\_email is a superkey for the users relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the users relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the orders relation:

F = {

order\_id -> user\_id

order\_id -> tracking\_number

order\_id -> total

order\_id -> billing\_address

order\_id -> shipping\_address

order\_id -> order\_status

order\_id -> date

tracking\_number -> order\_id }

order\_id and tracking\_number are the keys in the nontrivial functional dependencies. To examine whether order\_id is a superkey for the relation, we determine (order\_id)+, the closure of order\_id under F:

result = order\_id

order\_id -> user\_id: result = order\_id, user\_id

order\_id -> tracking\_number: result = order\_id, user\_id, tracking\_number

order\_id -> total: result = order\_id, user\_id, tracking\_number, total

order\_id -> billing\_address: result = order\_id, user\_id, tracking\_number, total, billing\_address

order\_id -> shipping\_address: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address

order\_id -> order\_status: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status

order\_id -> date: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status, date

(order\_id)+ = (order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status, date)

Thus we find that order\_id is a superkey for the orders relation.

Next we examine whether tracking\_number is a superkey for the orders relation, determining (tracking\_number) +, the closure of tracking\_number under F:

result = tracking\_number

tracking\_number -> order\_id: result = order\_id, tracking\_number

order\_id -> user\_id: result = order\_id, user\_id, tracking\_number

order\_id -> total: result = order\_id, user\_id, tracking\_number, total

order\_id -> billing\_address: result = order\_id, user\_id, tracking\_number, total, billing\_address

order\_id -> shipping\_address: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address

order\_id -> order\_status: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status

order\_id -> date: result = order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status, date

(tracking\_number)+ = (order\_id, user\_id, tracking\_number, total, billing\_address, shipping\_address, order\_status, date)

Thus we find that tracking\_number is a superkey for the orders relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the orders relation is thus in BCNF.

The following are the set of nontrivial functional dependencies for the suborders relation:

F = {

order\_id, suborder\_id -> ISBN

order\_id, suborder\_id -> quantity }

order\_id, suborder\_id is the sole key in each nontrivial functional dependency. To examine whether order\_id, suborder\_id is a superkey for the relation, we determine (order\_id, suborder\_id)+, the closure of order\_id, suborder\_id under F:

result = order\_id, suborder\_id

order\_id, suborder\_id -> ISBN: result = order\_id, suborder\_id, ISBN

order\_id, suborder\_id -> quantity: result = order\_id, suborder\_id, ISBN, quantity

(order\_id, suborder\_id)+ = (order\_id, suborder\_id, ISBN, quantity)

Thus we find that order\_id, suborder\_id is a superkey for the suborders relation. As for all nontrivial functional dependencies, α -> β, α is a superkey for the relation, the suborders relation is thus in BCNF.

The book\_author relation has no nontrivial functional dependencies. Thus as every functional dependency in this relation is trivial, it is by definition in BCNF.

We have therefore found that all our reduced relations are in BCNF and thus are in good normal form. In addition, as each relation was already originally in BCNF, we can assert that each relation is lossless and dependency preserving – these relations were never decomposed.