Introduction

Background

The rapid growth of internet users worldwide and the increasing demand for accessible, convenient shopping experiences have revolutionized the way people buy and sell products. Among these products, books have consistently been a popular choice for consumers who prefer to shop online. The demand for online bookstores has grown significantly in recent years, paving the way for an efficient and user-friendly system to manage the various aspects of an online bookstore's operations.

The OnlineBookstore is a comprehensive database application designed to address the needs of an online bookstore by managing essential operations such as customer management, inventory management, order processing, and data analysis. The system aims to streamline the bookstore's processes and provide an easy-to-use platform for both customers and administrators. By developing a robust and efficient application, the OnlineBookstore aims to improve the overall shopping experience for customers while also optimizing the management and reporting processes for online bookstore operators.

Functions and Services

The OnlineBookstore system offers a wide range of functions and services that cater to both customers and administrators. These functions and services are designed to provide a seamless shopping experience for customers and efficient management for administrators.

The system allows administrators to manage customer information, including their contact details and shipping addresses. Customers can create and update their profiles, making it easy for them to shop and track their orders. The application also offers features like password recovery and email notifications, ensuring a secure and user-friendly experience for customers.

Administrators can easily manage the bookstore's inventory by adding, updating, or removing books from the database. The system tracks essential book information, such as title, author, ISBN, publication date, price, and publisher. This feature enables administrators to maintain accurate records and ensure that customers can access up-to-date information on available books.

The OnlineBookstore system streamlines the order processing workflow by allowing customers to browse, search, and filter books, add them to their shopping cart, and complete the checkout process. Administrators can then manage and fulfill these orders, update the order status, and notify customers about their order's progress. This feature ensures a smooth and efficient ordering process for both customers and administrators.

Data Analysis and Reporting: The system provides a suite of analytical tools and reports that allow administrators to gain insights into the bookstore's operations. By examining sales data, popular books, author-publisher collaborations, and customer demographics, administrators can make informed decisions on inventory, pricing, and marketing strategies. This feature helps the online bookstore stay competitive in the market and respond to customer preferences more effectively.

The OnlineBookstore system is a comprehensive and robust database application that caters to the needs of an online bookstore by providing a seamless shopping experience for customers and efficient management for administrators. By offering a wide range of functions and services, the system aims to improve the overall shopping experience for customers and optimize the management and reporting processes for online bookstore operators.

Part A)

A1) Entities and Attributes:

1. Customer:
    - customer_id (Primary Key): Unique identifier for each customer
    - first_name: Customer's first name
    - last_name: Customer's last name
    - email: Customer's email address
    - phone_number: Customer's phone number
    - address: Customer's mailing address
2. Book:
    - book_id (Primary Key): Unique identifier for each book
    - title: Book's title
    - isbn: Book's International Standard Book Number
    - publication_date: Date when the book was published
    - price: Price of the book
    - publisher_id (Foreign Key): Reference to the publisher of the book
3. Author:
    - author_id (Primary Key): Unique identifier for each author
    - first_name: Author's first name
    - last_name: Author's last name
    - bio: Short biography of the author
    - birth_date: Author's date of birth
4. Publisher:
    - publisher_id (Primary Key): Unique identifier for each publisher
    - name: Publisher's name
    - address: Publisher's mailing address
    - email: Publisher's email address
    - phone_number: Publisher's phone number
5. Order:

- order_id (Primary Key): Unique identifier for each order
- customer_id (Foreign Key): Reference to the customer who placed the order
- order_date: Date when the order was placed
- total_amount: Total amount of the order
- shipping_address: Shipping address for the order

6. BookAuthor (intermediary table for "written_by" relationship):
    - book_id (Foreign Key): Reference to the associated book
    - author_id (Foreign Key): Reference to the associated author
7. OrderBook (intermediary table for "contains" relationship):
    - order_id (Foreign Key): Reference to the associated order
    - book_id (Foreign Key): Reference to the associated book
    - quantity: Number of copies of the book in the order
8. AuthorPublisher (intermediary table for "collaborates_with" relationship):
    - author_id (Foreign Key): Reference to the associated author
    - publisher_id (Foreign Key): Reference to the associated publisher

A2) Relationships:

1. Customer - Order (one-to-many): "places" A customer "places" multiple orders, but each order belongs to only one customer. This relationship connects the customers with their respective orders.
2. Book - Author (many-to-many): "written_by" (BookAuthor intermediary table) A book is "written_by" multiple authors, and an author can have multiple books. This relationship captures the association between books and their authors using the "BookAuthor" intermediary table.
3. Book - Publisher (many-to-one): "published_by" A book is "published_by" only one publisher, but a publisher can publish multiple books. This relationship links books to their respective publishers.
4. Order - Book (many-to-many): "contains" (OrderBook intermediary table) An order "contains" multiple books, and a book can be a part of multiple orders. This relationship represents the books included in each order using the "OrderBook" intermediary table.
5. Author - Publisher (many-to-many): "collaborates_with" (AuthorPublisher intermediary table) An author "collaborates_with" multiple publishers, and a publisher can work with multiple authors. This relationship represents the collaboration between authors and publishers using the "AuthorPublisher" intermediary table.

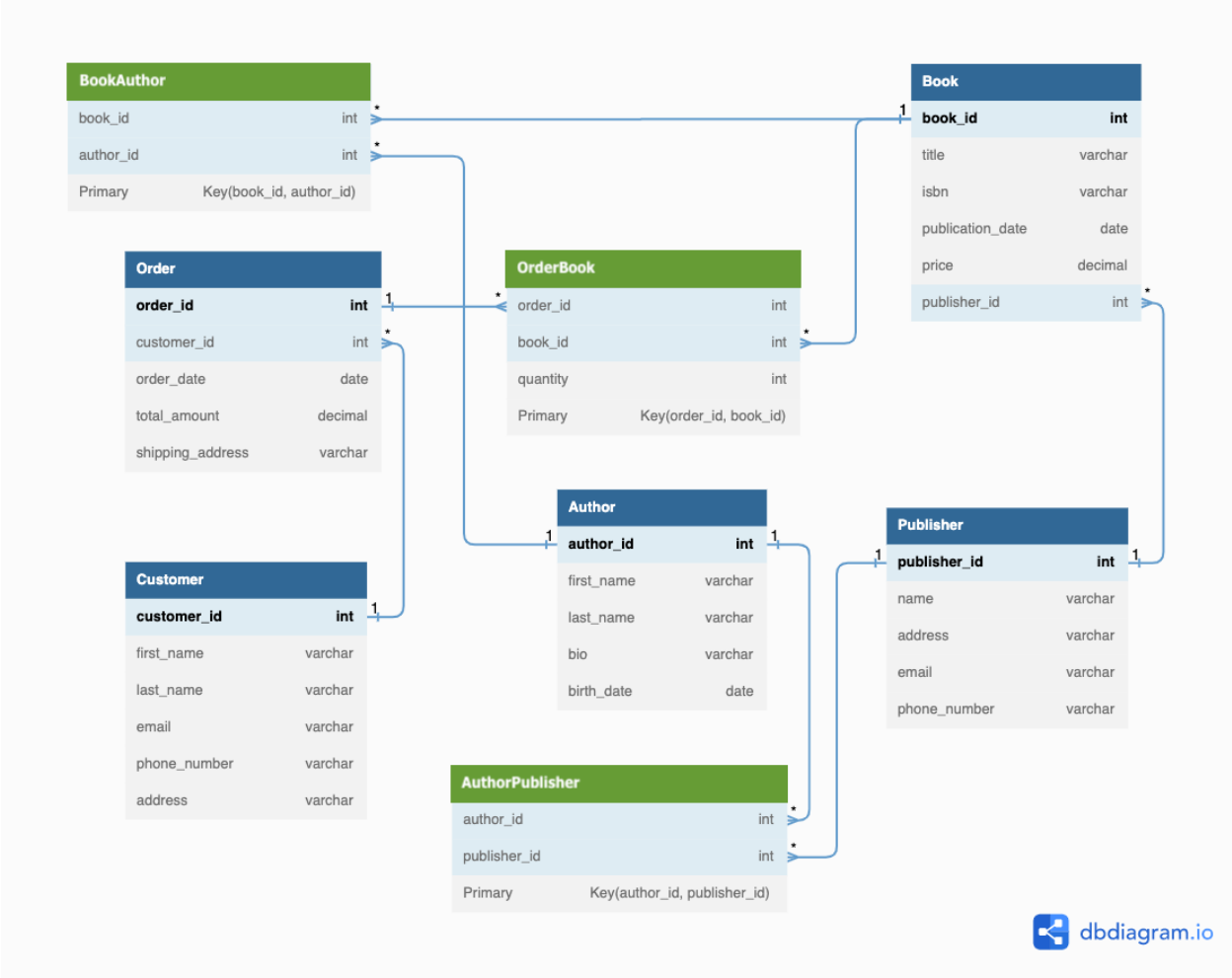
Part B)

B1) Set of relations for the database design:

1. Customer (customer_id, first_name, last_name, email, phone_number, address)
    - Primary Key: customer_id
2. Book (book_id, title, isbn, publication_date, price, publisher_id)
    - Primary Key: book_id

- Foreign Key: publisher_id (References Publisher)

3. Author (author_id, first_name, last_name, bio, birth_date)
    - Primary Key: author_id
4. Publisher (publisher_id, name, address, email, phone_number)
    - Primary Key: publisher_id
5. Order (order_id, customer_id, order_date, total_amount, shipping_address)
    - Primary Key: order_id
    - Foreign Key: customer_id (References Customer)
6. BookAuthor (book_id, author_id)
    - Primary Key: (book_id, author_id)
    - Foreign Key: book_id (References Book)
    - Foreign Key: author_id (References Author)
7. OrderBook (order_id, book_id, quantity)
    - Primary Key: (order_id, book_id)
    - Foreign Key: order_id (References Order)
    - Foreign Key: book_id (References Book)
8. AuthorPublisher (author_id, publisher_id)
    - Primary Key: (author_id, publisher_id)
    - Foreign Key: author_id (References Author)
    - Foreign Key: publisher_id (References Publisher)



B2) Nontrivial functional dependencies for each relation in the schema:

1. Customer (customer_id, first_name, last_name, email, phone_number, address)
    - customer_id -> first_name, last_name, email, phone_number, address
2. Book (book_id, title, isbn, publication_date, price, publisher_id)
    - book_id -> title, isbn, publication_date, price, publisher_id
    - isbn -> title, publication_date, price, publisher_id
3. Author (author_id, first_name, last_name, bio, birth_date)
    - author_id -> first_name, last_name, bio, birth_date
4. Publisher (publisher_id, name, address, email, phone_number)
    - publisher_id -> name, address, email, phone_number
5. Order (order_id, customer_id, order_date, total_amount, shipping_address)
    - order_id -> customer_id, order_date, total_amount, shipping_address
6. BookAuthor (book_id, author_id)
    - No nontrivial functional dependencies, as this table represents a many-to-many relationship.

7. OrderBook (order_id, book_id, quantity)
    - (order_id, book_id) -> quantity
8. AuthorPublisher (author_id, publisher_id)
    - No nontrivial functional dependencies, as this table represents a many-to-many relationship.

B3) Checking BCNF

1. Customer:
    - The only nontrivial functional dependency is customer_id -> first_name, last_name, email, phone_number, address. customer_id is the primary key, so the relation is in BCNF.
2. Book:
    - The only nontrivial functional dependency is book_id -> title, isbn, publication_date, price, publisher_id. book_id is the primary key, so the relation is in BCNF.
3. Author:
    - The only nontrivial functional dependency is author_id -> first_name, last_name, bio, birth_date. author_id is the primary key, so the relation is in BCNF.
4. Publisher:
    - The only nontrivial functional dependency is publisher_id -> name, address, email, phone_number. publisher_id is the primary key, so the relation is in BCNF.
5. Order:
    - The only nontrivial functional dependency is order_id -> customer_id, order_date, total_amount, shipping_address. order_id is the primary key, so the relation is in BCNF.
6. BookAuthor:
    - There are no nontrivial functional dependencies. The primary key is a composite key (book_id, author_id), and there are no other attributes. The relation is in BCNF.
7. OrderBook:
    - The only nontrivial functional dependency is (order_id, book_id) -> quantity. The primary key is a composite key (order_id, book_id), and there are no other attributes. The relation is in BCNF.
8. AuthorPublisher:
    - There are no nontrivial functional dependencies. The primary key is a composite key (author_id, publisher_id), and there are no other attributes. The relation is in BCNF.

All relations in the schema are in Boyce-Codd Normal Form (BCNF) with respect to the functional dependencies. No new relations are produced during the normalization process, and all relations pass the BCNF test. The relations have no redundancies because the primary key determines all other attributes in each relation, and there are no nontrivial functional dependencies on non-key attributes.

B4) Discussing 3NF and 4NF

A relation is in 3NF if it meets the following conditions:

1. It is in Second Normal Form (2NF).
2. There are no transitive dependencies between non-key attributes.

Since all my relations are in BCNF, they are already in 3NF by definition. BCNF is a stronger form of 3NF, meaning that if a relation is in BCNF, it is also in 3NF. Therefore, there is no need to check for 3NF, as the relations are already in a more normalized form.

Now, let's discuss the Fourth Normal Form (4NF). A relation is in 4NF if it meets the following conditions:

1. It is in Boyce-Codd Normal Form (BCNF).
2. It has no multi-valued dependencies.

The relations are already in BCNF, so I only need to check for multi-valued dependencies. Multi-valued dependencies occur when two or more independent multi-valued facts about an entity are combined into a single relation.

In my schema, I have already taken care of potential multi-valued dependencies by creating intermediary tables for many-to-many relationships: BookAuthor, OrderBook, and AuthorPublisher. These intermediary tables separate independent multi-valued facts about the entities, thus eliminating any multi-valued dependencies.

Conclusion: The relations in the schema are already in BCNF, which is a stronger form of 3NF, so there is no need to check for 3NF. Regarding 4NF, I have created intermediary tables to handle many-to-many relationships, which eliminate any potential multi-valued dependencies. Therefore, neither Third Normal Form nor Fourth Normal Form would further improve the structures of the relations, as they are already optimized.

Part C)

In the database, I generated the data using Python's **Faker** library. The **Faker** library helps me in generating random and realistic data for testing purposes. I created a Python script that uses the **Faker** library to generate random data for each entity in the database.

To ensure the uniqueness of key attributes, I follow these methods:

1. Primary Keys: For primary key attributes such as **customer_id**, **book_id**, **author_id**, **publisher_id**, and **order_id**, I use sequential integers starting from 1. By doing so, I ensure that primary keys are unique across their respective tables.
2. Unique constraints: For some attributes, such as **email** in the **Customer** and **Publisher** tables and **isbn** in the **Book** table, I apply unique constraints. To generate unique values for these attributes, I used a set to store the already generated emails and ISBNs. This ensures that no two records will have the same value for these fields. Here's the code snippet:

```
myEmailSet = set()
for i in range(1, num_records + 1):
   email = fake.email()
   while email in myEmailSet:
      email = fake.email()
```

myEmailSet.add(email)

3. Composite keys: In intermediary tables, such as **BookAuthor**, **OrderBook**, and **AuthorPublisher**, I use composite primary keys. These keys consist of a combination of foreign keys from the tables they connect, which ensures the uniqueness of records in those intermediary tables.

For interesting joins among multiple relations, I have created the following relationships:

1. Customer - Order (one-to-many): A customer places multiple orders, but each order belongs to only one customer. This relationship connects customers with their respective orders.
2. Book - Author (many-to-many): A book is written by multiple authors, and an author can have multiple books. This relationship captures the association between books and their authors using the "BookAuthor" intermediary table.
3. Book - Publisher (many-to-one): A book is published by only one publisher, but a publisher can publish multiple books. This relationship links books to their respective publishers.
4. Order - Book (many-to-many): An order contains multiple books, and a book can be a part of multiple orders. This relationship represents the books included in each order using the "OrderBook" intermediary table.
5. Author - Publisher (many-to-many): An author collaborates with multiple publishers, and a publisher can work with multiple authors. This relationship represents the collaboration between authors and publishers using the "AuthorPublisher" intermediary table.

These relationships enable interesting joins among multiple relations, allowing for complex queries and data analysis.

Part D)

As the developer of this system, I built the user interface using Django, a Python web framework, alongside HTML, CSS, and JavaScript for the frontend. The application connects to a MySQL database to store and retrieve data. I ensured that the interface is user-friendly, allowing users to interact with the database without needing to know SQL.

Here is a list of functions that I offer to users through the system:

List Customers: This function displays a list of all customers in the database. Users can search for customers by name using the search bar. The function is implemented in SQL using a SELECT statement to retrieve customer data from the Customer table.

Add Customer: Users can add a new customer to the database by filling out a form with the customer's details, such as first name, last name, email, phone number, and address. This function is implemented in SQL using an INSERT statement to add a new row to the Customer table.

List Books: This function shows a list of all books in the database, along with their prices. Users can sort the list by price, either from high to low or low to high, and filter by price range. This function is implemented in SQL using SELECT statements with ORDER BY and WHERE clauses to retrieve and filter book data from the Book table.

Add Book: Users can add a new book to the database by filling out a form with the book's details, such as title, ISBN, publication date, price, and publisher. This function is implemented in SQL using an INSERT statement to add a new row to the Book table.

List Orders: This function displays a list of all orders in the database, along with their order dates, total amounts, and shipping addresses. Users can filter the list by order date and total amount. This function is implemented in SQL using SELECT statements with WHERE clauses to retrieve and filter order data from the Order table.

Add Order: Users can add a new order to the database by selecting a customer, specifying the order date, total amount, and shipping address, and choosing books and their quantities. This function is implemented in SQL using INSERT statements to add new rows to the Order and OrderBook tables.

To build the user interface, I used Django's template system to create reusable HTML templates for each function. I employed CSS to style the interface and JavaScript to handle user interactions, such as searching and filtering. I also used Django views and models to interact with the MySQL database, executing SQL queries and modifications as needed.

My Experience in this project:

During the development of this database application, I encountered several difficulties, which I managed to overcome through research, perseverance, and adjustments to my approach. Some of these difficulties included:

1. Database normalization: Ensuring that the database schema adheres to the principles of normalization was challenging. I spent considerable time analyzing the entities, attributes, and relationships to ensure that the schema is in BCNF, 3NF, and 4NF. Through this process, I learned the importance of normalization in reducing redundancy and improving database performance.
2. Many-to-many relationships: Handling many-to-many relationships and designing intermediary tables was another challenge. By using composite primary keys consisting of foreign keys from the related tables, I was able to create intermediary tables that effectively managed these relationships, allowing for interesting joins among multiple relations.
3. Generating realistic test data: To properly test the application, I needed a significant amount of realistic data. By using Python's Faker library, I was able to generate random yet realistic data for each entity in the database, which greatly helped with testing the application's functionality and performance.

4. Designing a user-friendly interface: Creating a user interface that allows users to interact with the database without needing to know SQL was a challenge. By using Django, HTML, CSS, and JavaScript, I was able to develop an intuitive and easy-to-use interface that met this goal.
5. Ensuring data consistency and integrity: To maintain data integrity, I had to ensure that the primary keys, unique constraints, and foreign key relationships were properly enforced. By carefully designing the schema, setting appropriate constraints, and using error handling in the application, I was able to preserve data consistency and integrity.

Throughout this project, I learned a great deal about database design, normalization, handling relationships, generating test data, and developing user-friendly interfaces. This experience has not only improved my skills as a developer but also deepened my understanding of the principles and best practices in database application development. By overcoming these challenges, I gained valuable insights and practical knowledge that will serve me well in future projects.