Coorporate State University Baden-Württemberg Mannheim

Project report

Project Databases - Booktropolis

Computer Science and Business

Specification Data Science

Editors:	Lasse Friedrich, Benedikt Prisett
Matriculation number:	9924680, 5709658
Class:	Databases 1 & 2
Lecturer:	Petko Rutesic
Course:	WWI21DSA
Director of studies:	Dennis Pfisterer
Processing period:	08.05.2023 – 14.07.2023

Contents

1 Introduction

In the age of digital transformation, libraries–the sanctuaries of knowledge and learning– are not left behind. To cater to the evolving demands of effective information management and retrieval, modern libraries are now embracing database management systems. One such progressive step in this direction is the introduction of our comprehensive solution, *Booktropolis*.

Why a database management system for libraries? As the volume of books and scholarly content proliferates, the task of effectively managing these resources becomes challenging. This is where a well-designed database system comes in. It helps in the systematic organization, management, and retrieval of information with just a few clicks.

Booktropolis is a user-friendly system that goes beyond the standard database management system. This system offers a seamless user interface that not only tracks each book, its various copies, the different genres, authors, and publishers, but also efficiently manages the lending process. It tracks the checked-out books, their due dates, and the borrowers, all in real time.

While *Booktropolis* is a potent tool for technical administrators, it does not overlook the needs of library staff. The system's upcoming features promise to empower library staff, making their tasks more streamlined and less nasty to deal with. In addition to the robust management features, *Booktropolis* also allows users to leave reviews and rate their reads, fostering a more interactive and engaging library environment.

Experience the future of library management with *Booktropolis*, where we're bridging the gap between libraries and their trustees, making library management more efficient, one book at a time!

2 Specification

This preliminary requirements specification was formulated as an initial blueprint prior to the commencement of the project; it served as a guideline and foundational structure for the development of the library database. However, as the project unfolded, some of the specifics of this initial proposal were subject to changes and adaptations, reflecting the evolving needs and realities of the project, hence the final implementation may not meet all the specifics of the initial plan.

- An author has a first and last name, an artist name, nationality, birthdate, and sex. Each book has an author, and one author can write several books.
- The library database keeps track of the people who lend books and when they have to be returned. There are different restrictions on how many and what types of books can be checked out by either normal users or professional researchers.
- A user has a first and last name, an ID, an address, and an email address.
- The library staff can easily view which books are currently checked out, as well as which users have how many books and the corresponding deadlines. Each staff manages one or multiple locations.
- Books are stored in locations that have an address, building, floor, and shelf number.
- The library stores information about which users and staff are currently at which location.

3 Database Design

3.1 ER-Diagramm

The database is composed of ten tables, each serving a unique purpose.

The **Author** table stores information about authors, including their ID, first and last names, gender, birthdate, nationality, and artist name.

The **Address** table holds address details, such as the ID, street, city, postal code, and state.

The **Publisher** table contains information about publishers, including their ID, name, email, website, and a foreign key to their address stored in the Address table.

The **Customer** table keeps customer details, including their ID, first and last names, email, phone number, birthdate, and a foreign key to their address, also stored in the Address table.

The **Building** table holds building details, including the ID, floor number, wheelchair accessibility, and a foreign key to its address, which is stored in the Address table.

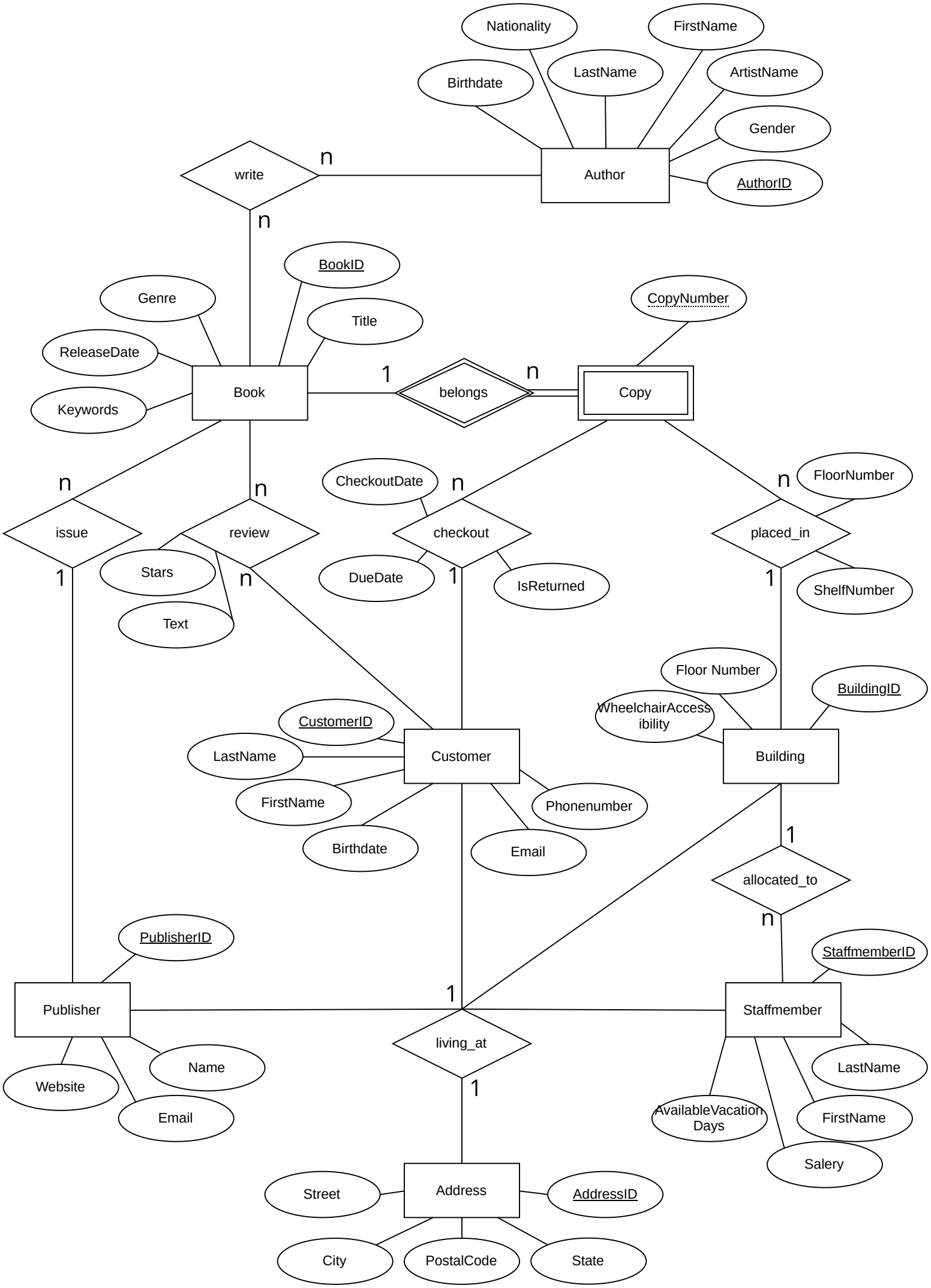
The **Staffmember** table stores staff member details, including their ID, first and last names, salary, available vacation days, and foreign keys to their building and address, stored in the Building and Address tables respectively.

The **Book** table contains information about books, including their ID, title, genre, release date, keywords, and a foreign key to their publisher, stored in the Publisher table.

The **Copy** table stores details about book copies, including their ID, checkout date, due date, return status, and foreign keys to their book, customer, and building, stored in the Book, Customer, and Building tables respectively.

The **Write** table is a junction table that links the Author and Book tables, storing foreign keys to both.

Lastly, the **Review** table stores reviews made by customers for books, including star ratings, review text, and foreign keys to the book and customer, stored in the Book and Customer tables respectively.


Nationality
FirstName
Birthdate
LastName
ArtistName
Gender
write
n
Author
AuthorID
n
BookID
Genre
Title
CopyNumber
ReleaseDate
Book
1
belongs
n
Copy
Keywords
n
n
CheckoutDate
n
n
FloorNumber
issue
review
checkout
placed_in
Stars
DueDate
1
IsReturned
1
ShelfNumber
n
Text
Floor Number
BuildingID
WheelchairAccessibility
CustomerID
LastName
Customer
Building
FirstName
Phonenumber
Birthdate
Email
1
allocated_to
n
PublisherID
StaffmemberID
1
Publisher
Staffmember
living_at
Name
LastName
Website
Email
AvailableVacationDays
FirstName
1
Salery
Street
Address
AddressID
City
PostalCode
State

3.2 3NF Analysis

In the context of normal forms, the library database schema is designed to adhere to the First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).

With respect to the **First Normal Form (1NF)**, every table has a primary key such as `BookID`, `CopyID`, `AuthorID`, `UserID`, `CheckoutID`, `StaffID`, `LocationID`, `BuildingID`, `PublisherID`, and `ReviewID`. All values in each column are atomic, indicating there are no repeating groups or nested structures. Thus, the schema satisfies the 1NF requirements.

Turning to the **Second Normal Form (2NF)**, the schema, already being in 1NF, has primary keys that are either single-column or composite. Composite keys are used in the `BookAuthor` and `BookLocation` tables to represent many-to-many relationships. Importantly, no partial dependencies exist in the schema, meaning no non-key attributes depend on a part of a composite key. Therefore, the schema meets the 2NF conditions.

As for the **Third Normal Form (3NF)**, the schema, already complying with 2NF, maintains its attributes depending only on the primary key and not on any non-key attributes, thereby avoiding transitive dependencies. There may be a debate on whether the email of the publisher or the customer and their phone number may cause a transitive dependency as they could potentially uniquely identify the entity. However, this is not the case here, as multiple publishers can belong to one corporate group and therefore share the same email address. Similarly, parents might create accounts for their children using their contact details, leading to multiple accounts with the same email and phone number. Even the postal code cannot be a reliable unique identifier due to the possibility of different cities sharing the same postal code. Hence, the schema also fulfills the requirements of the 3NF.

In summary, the database schema for the library system adheres to all three fundamental normal forms.

4 SQL Occurences

4.1 Database Views

Database views, virtual tables based on the result-set of an SQL statement, are crucial to the *Booktropolis* system. They facilitate efficient querying of data by pre-computing complex joins and aggregations, allowing users to fetch data quickly without worrying about underlying table structures.

CustomerCheckoutView

The *CustomerCheckoutView* is a non-materialized view that provides an overview of the books currently checked out, along with their due dates and the customers who have borrowed them.

```
1  CREATE VIEW CustomerCheckoutView AS
2  SELECT
3      Copy.CopyID AS CopyID,
4      Customer.FirstName AS CustomerFirstName,
5      Customer.LastName AS CustomerLastName,
6      Book.Title AS BookTitle,
7      string_agg(Author.FirstName || ' ' || Author.LastName, ', ') AS
           ↪ Authors,
8      Copy.DueDate AS DueDate
9  FROM
10     Customer
11 JOIN
12     Copy ON Customer.CustomerID = Copy.CustomerID
13 JOIN
14     Book ON Copy.BookID = Book.BookID
15 JOIN
16     Write ON Book.BookID = Write.BookID
17 JOIN
18     Author ON Write.AuthorID = Author.AuthorID
```

```
19 GROUP BY
20     Copy.CopyID,
21     Customer.FirstName,
22     Customer.LastName,
23     Book.Title,
24     Copy.DueDate
25 ORDER BY DueDate;
```

BookAuthorPublisherMaterializedView

The *BookAuthorPublisherMaterializedView* is a materialized view, which means that the query result is stored physically, and a refresh mechanism is required to keep it in sync with the underlying tables. This view facilitates faster data retrieval for often queried yet rarely updated data by saving the computation time. In this case, it provides information about books, authors, and publishers in a pre-computed, easy-to-query manner.

```
1  CREATE MATERIALIZED VIEW BookAuthorPublisherMaterializedView AS
2  SELECT
3      Book.Title AS BookTitle,
4      Author.FirstName AS AuthorFirstName,
5      Author.LastName AS AuthorLastName,
6      Publisher.Name AS PublisherName
7  FROM
8      Book
9  JOIN
10     Write ON Book.BookID = Write.BookID
11 JOIN
12     Author ON Write.AuthorID = Author.AuthorID
13 JOIN
14     Publisher ON Book.PublisherID = Publisher.PublisherID;
```

In conclusion, the choice between a materialized and non-materialized view depends on the specific use-case. Non-materialized views are best suited for frequently updated data, while materialized views shine in scenarios where data is infrequently updated, and fast retrieval is crucial. Based on this we implemented the described views.

4.2 Stored Procedures

In the context of Booktropolis, the procedure `UpdateReturnStatus()` plays a crucial role in managing book rental operations. It's a PostgreSQL stored procedure that acts as a trigger function to update the `CheckoutDate` and `DueDate` of a book copy in the `Copy` table whenever its `IsReturned` status changes.

```
1  CREATE OR REPLACE FUNCTION UpdateReturnStatus() RETURNS TRIGGER
2  LANGUAGE plpgsql
3  AS $$
4  BEGIN
5    IF (NEW.IsReturned = True) THEN
6      NEW.CheckoutDate := NULL;
7      NEW.DueDate := NULL;
8    ELSIF (NEW.IsReturned = False) THEN
9      NEW.CheckoutDate := CURRENT_DATE;
10     NEW.DueDate := CURRENT_DATE + INTERVAL '30 days';
11   END IF;
12   RETURN NEW;
13 END;
14 $$;
15
16 CREATE TRIGGER tr_UpdateReturnStatus
17 BEFORE UPDATE ON Copy
18 FOR EACH ROW
19 EXECUTE PROCEDURE UpdateReturnStatus();
```

This is the logic behind the `UpdateReturnStatus()` procedure:

- If a book copy is marked as returned (`IsReturned = True`), it will nullify its `CheckoutDate` and `DueDate`. This ensures that no residual data from the previous checkout remains in these fields once the copy is returned.
- Conversely, if a book copy is marked as not returned (`IsReturned = False`), it will set the `CheckoutDate` to the current date and the `DueDate` to 30 days later. This helps to define a clear borrowing period whenever a book is checked out.

The trigger `tr_UpdateReturnStatus` is fired before any update operation on the `Copy` table. It automatically executes the `UpdateReturnStatus()` procedure, ensuring that the aforementioned logic is always applied, and thus maintaining data consistency.

4.3 SQL Queries

Based on the described requirements we have implemented several SQL Queries, that extract data from the database. The Queries use complex structures such as joins, renaming, nested queries, set operations and aggregate functions.

This SQL query retrieves the title of the book 'Science 101', along with the first and last names of all customers who have checked out this book. It does this by joining the Customer, Copy, and Book tables on their respective IDs and filtering for the specific book title, which can be changed out by any other one.

```
1 -- Find all customers who have checked out a specific book:
2 SELECT Book.Title, Customer.FirstName, Customer.LastName
3 FROM Customer
4 JOIN Copy ON Customer.CustomerID = Copy.CustomerID
5 JOIN Book ON Copy.BookID = Book.BookID
6 WHERE Book.Title = 'Science␣101';
```

This SQL query counts the total number of returned copies of the book titled 'Journey to the East'. It joins the Copy and Book tables on their respective IDs, filters for the specific book title and returned copies, and then groups the results by book title to count the copies.

```
1 -- Get the total number of available copies for a particular book
2 SELECT Book.Title, COUNT(*) AS "TotalCopies"
3 FROM Copy
4 JOIN Book ON Book.BookID = Copy.BookID
5 WHERE Book.Title = 'Journey␣to␣the␣East' AND Copy.IsReturned = true
6 GROUP BY Book.Title;
```

This SQL query retrieves a list of all books, their average review ratings, and the total number of reviews they've received. The results are sorted in descending order by average

rating. The LEFT JOIN ensures that books without any reviews are still included in the results.

```
-- Find the top-rated books along with their average ratings:
SELECT Book.Title, AVG(Review.Stars) AS AverageRating, COUNT(Review.
    ↪ BookID) AS ReviewCount
FROM Book
LEFT JOIN Review ON Book.BookID = Review.BookID
GROUP BY Book.BookID
ORDER BY AverageRating DESC;
```

This SQL query retrieves the titles of all distinct books written by American authors that have more than three reviews. It does so by joining Book, Write, Author, and a subquery that selects books with more than three reviews.

```
-- Find the titles of books written by American authors that have
    ↪ received more than three reviews.
SELECT DISTINCT Book.Title
FROM Book
JOIN Write ON Book.BookID = Write.BookID
JOIN Author ON Write.AuthorID = Author.AuthorID
JOIN (
  SELECT BookID, COUNT(*) AS ReviewCount
  FROM Review
  GROUP BY BookID
  HAVING COUNT(*) > 3
) AS ReviewCountTable ON Book.BookID = ReviewCountTable.BookID
WHERE Author.Nationality = 'American';
```

This SQL query retrieves a list of authors and publishers who have released books after June 18, 2014. For each person or company, it returns their name and whether they are an author or publisher. The results are ordered in ascending order by type ('Author' or 'Publisher'). This is achieved through a common table expression, a subquery, and a UNION operation to combine the two types of results.

```
-- Find all the authors and publisher that have realeased a book
    ↪ after 18.06.2023. List the Name or Lastname and indecate of
    ↪ what type they are.
```

```
WITH BooksWithReleaseDate AS (
  SELECT BookID
  FROM Book
  WHERE ReleaseDate > '2014-06-18'
)
SELECT LastName AS Name, 'Author' AS Type
FROM Author
WHERE AuthorID IN (
  SELECT AuthorID
  FROM Write
  WHERE BookID IN (SELECT BookID FROM BooksWithReleaseDate)
)
UNION
SELECT Name, 'Publisher' AS Type
FROM Publisher
WHERE PublisherID IN (
  SELECT PublisherID
  FROM Book
  WHERE BookID IN (SELECT BookID FROM BooksWithReleaseDate)
)
ORDER BY Type ASC;
```

This query first selects customers who have written a review and then intersects this with the list of customers who have not returned a book they've reviewed. The result is the first and last names of customers who both have written a review and have not returned a book they have reviewed.

```
-- Find all customers who have reviewed the book they have currently
   ↪ checked out.
SELECT FirstName, LastName
FROM Customer
WHERE CustomerID IN (
  SELECT CustomerID
  FROM Review
)
INTERSECT
```

```sql
SELECT FirstName, LastName
FROM Customer
WHERE CustomerID IN (
  SELECT CustomerID
  FROM Copy
  WHERE IsReturned = false
  AND BookID IN (
    SELECT BookID
    FROM Review
    WHERE CustomerID = Copy.CustomerID
  )
);
```

This SQL query returns a list of publishers, genres of books they've published, the count of books for each genre, and the average rating of the books by genre. It does this by joining and aggregating data from Publisher, Book, and Review tables.

The query joins the Publisher and Book tables, then does a LEFT JOIN with a subquery to get the count of books for each publisher by genre. It also does another LEFT JOIN with Review to get the average rating of the books.

The use of COALESCE(BookCount.BookCount, 0) ensures that if there are no books for a specific publisher and genre (which would return a NULL value for the BookCount), it gets replaced with 0.

```sql
-- For each publisher find the number of books they have published in
   ↪ each genre (if there are any) and calculate the average rating
   ↪ for each publisher genre combination
SELECT
    Publisher.Name AS PublisherName,
    Book.Genre AS Genre,
    COALESCE(BookCount.BookCount, 0) AS BookCount,
    AVG(Review.Stars) AS AverageRating
FROM
    Publisher
    JOIN Book ON Publisher.PublisherID = Book.PublisherID
    LEFT JOIN (
```

```
        SELECT
            PublisherID,
            Genre,
            COUNT(*) AS BookCount
        FROM
            Book
        GROUP BY
            PublisherID,
            Genre
    ) AS BookCount ON Publisher.PublisherID = BookCount.PublisherID AND
        ↪ Book.Genre = BookCount.Genre
    LEFT JOIN Review ON Book.BookID = Review.BookID
GROUP BY
    Publisher.PublisherID,
    Publisher.Name,
    Book.Genre,
    BookCount.BookCount
ORDER BY
    PublisherName;
```

4.4 Secondary Indexes

A secondary index is a data structure that improves the speed of many search operations in a database table. Unlike the primary index, which is usually based on the primary key, a secondary index is built on non-primary key columns. By doing so, it allows fast access to data in various different ways, not just by the primary key.

```
-- Index on Book.Title for fast retrieval of books by title
CREATE INDEX idx_book_title ON Book (Title);

-- Index on Copy.BookID for efficient joining with Book table
CREATE INDEX idx_copy_bookid ON Copy (BookID);

-- Index on Copy.CustomerID for efficient joining with Customer table
CREATE INDEX idx_copy_customerid ON Copy (CustomerID);
```

```
9
10 -- Index on Review.BookID for efficient joining with Book table
11 CREATE INDEX idx_review_bookid ON Review (BookID);
```

The indices above were created for the following reasons:

- `idx_book_title`: This index on the `Title` column of the `Book` table speeds up searches for books by their title.
- `idx_copy_bookid` and `idx_review_bookid`: These indices enhance the efficiency of join operations between the `Book` table and the `Copy` or `Review` tables. Join operations are common in our database queries, and an index on the join column significantly reduces the computational cost.
- `idx_copy_customerid`: This index on the `CustomerID` column in the `Copy` table accelerates join operations with the `Customer` table.

In summary, secondary indices, like the ones we have created, are an especially important to optimize data retrieval operations from the advances queries and enhance overall database performance.

5 Application

The application is built using the Streamlit library, which provides a simple way to create interactive web applications using Python. It is intended to provide an intuitive and user-friendly interface for performing CRUD (Create, Read, Update, Delete) operations on the database.

The interface of the application allows users to select from different operations such as creating, reading, updating, and deleting data in the database. Each operation is presented in a form-like interface where users can input the necessary data.

For data connectivity and manipulation, psycopg2 and pandas libraries are used. psycopg2 is a PostgreSQL database adapter for Python, while pandas is used for data manipulation and analysis.

The Booktropolis application connects to the PostgreSQL database using the DATABASE_URL, which is read from the .env file using the dotenv library.

The CRUD operations are executed via corresponding functions: create_operation, read_operation, update_operation, and delete_operation.

The *create_operation* function facilitates the insertion of new records into the chosen table. This function constructs an SQL INSERT INTO query with the given table name, field names, and corresponding values obtained from user input. The input values are provided in a Streamlit form, which once submitted, triggers the execution of the SQL query via a cursor object.

```
1 def create_operation(table_name, table_fields):
2     cur = conn.cursor()
3 
4     fields_str = ', '.join(table_fields.keys())
5     values_str = ', '.join([f"'{value}'" for value in table_fields.
          ↪ values()])
6 
7     cur.execute(f"""
8         INSERT INTO {table_name} ({fields_str})
9         VALUES ({values_str});
```

```
    """)

    conn.commit()
```

The *read_operation* function retrieves records from the database. This function constructs an SQL SELECT query to fetch all records from the chosen table. The data fetched is displayed on the Streamlit interface as a pandas DataFrame.

```
def read_operation(table_name):
    query = f"SELECT * FROM {table_name};"
    df = pd.read_sql_query(query, conn)

    st.write(df)
```

The *update_operation* function is responsible for modifying existing records in the table. The function creates an SQL UPDATE query using the user-inputted field values. The query updates the records where the condition based on a primary key is met.

```
def update_operation(table_name, table_fields, condition_value):
    cur = conn.cursor()

    set_values = ', '.join([f"{key} = '{value}'" for key, value in
        ↪ table_fields.items()])

    cur.execute(f"""
        UPDATE {table_name}
        SET {set_values}
        WHERE {list(table_fields.keys())[0]} = '{condition_value}';
    """)

    conn.commit()
```

The *delete_operation* function enables the removal of records from a table. This function constructs an SQL DELETE FROM query using the table name and a condition based on user input. The query deletes records from the table where the condition is met.

```
def delete_operation(table_name, condition_field, condition_value):
    cur = conn.cursor()
```

```
3
4       cur.execute(f"""
5           DELETE FROM {table_name}
6           WHERE {condition_field} = '{condition_value}';
7       """)
8
9       conn.commit()
```

By this means, all CRUD operations in Booktropolis are simplified and user-friendly, providing a robust system for managing book-related data.

6 Appendix

6.1 Database Definition in DDL

```
-- Create Author table
CREATE TABLE Author (
  AuthorID SERIAL PRIMARY KEY,
  FirstName VARCHAR(255),
  LastName VARCHAR(255),
  Gender VARCHAR(10),
  Birthdate DATE,
  Nationality VARCHAR(255),
  Artistname VARCHAR(255)
);

-- Create Address table
CREATE TABLE Address (
  AddressID SERIAL PRIMARY KEY,
  Street VARCHAR(255),
  City VARCHAR(255),
  PostalCode VARCHAR(20),
  State VARCHAR(255)
);

-- Create Publisher table
CREATE TABLE Publisher (
  PublisherID SERIAL PRIMARY KEY,
  Name VARCHAR(255),
  Email VARCHAR(255),
  Website VARCHAR(255),
  AddressID INT UNIQUE,
  FOREIGN KEY (AddressID) REFERENCES Address(AddressID) ON DELETE SET
    ↪ NULL
```

```
29 );
30
31 -- Create Customer table
32 CREATE TABLE Customer (
33   CustomerID SERIAL PRIMARY KEY,
34   FirstName VARCHAR(255),
35   LastName VARCHAR(255),
36   Email VARCHAR(255),
37   Phonenumer VARCHAR(20),
38   Birthdate DATE,
39   AddressID INT UNIQUE,
40   FOREIGN KEY (AddressID) REFERENCES Address(AddressID) ON DELETE SET
       ↪ NULL
41 );
42
43 -- Create Building table
44 CREATE TABLE Building (
45   BuildingID SERIAL PRIMARY KEY,
46   FloorNumber INT,
47   WheelchairAccessibility BOOLEAN,
48   AddressID INT UNIQUE,
49   FOREIGN KEY (AddressID) REFERENCES Address(AddressID) ON DELETE
       ↪ CASCADE
50 );
51
52 -- Create Staffmember table
53 CREATE TABLE Staffmember (
54   StaffmemberID SERIAL PRIMARY KEY,
55   FirstName VARCHAR(255),
56   LastName VARCHAR(255),
57   Salary DECIMAL(10, 2),
58   AvailableVacationDays INT,
59   BuildingID INT,
60   AddressID INT UNIQUE,
61   FOREIGN KEY (BuildingID) REFERENCES Building(BuildingID) ON DELETE
```

```
    ↪ SET NULL,
  FOREIGN KEY (AddressID) REFERENCES Address(AddressID) ON DELETE
    ↪ CASCADE
);

-- Create Book table
CREATE TABLE Book (
  BookID SERIAL PRIMARY KEY,
  Title VARCHAR(255),
  Genre VARCHAR(255),
  ReleaseDate DATE,
  Keyword VARCHAR(255),
  PublisherID INT,
  FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID) ON
    ↪ DELETE SET NULL
);

-- Create Copy table
CREATE TABLE Copy (
  CopyID SERIAL PRIMARY KEY,
  BookID INT,
  CustomerID INT,
  CheckoutDate DATE,
  DueDate DATE,
  IsReturned BOOLEAN,
  BuildingID INT,
  FloorNumber INT,
  ShelfNumber INT,
  FOREIGN KEY (BookID) REFERENCES Book(BookID) ON DELETE CASCADE,
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
    ↪ SET NULL,
  FOREIGN KEY (BuildingID) REFERENCES Building(BuildingID) ON DELETE
    ↪ SET NULL
);

```

```
92  -- Create Write table
93  CREATE TABLE Write (
94    AuthorID INT,
95    BookID INT,
96    PRIMARY KEY (AuthorID, BookID),
97    FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID) ON DELETE CASCADE
        ↪ ,
98    FOREIGN KEY (BookID) REFERENCES Book(BookID) ON DELETE CASCADE
99  );
100
101 -- Create Review table
102 CREATE TABLE Review (
103   BookID INT,
104   CustomerID INT,
105   Stars INT,
106   Text VARCHAR(255),
107   PRIMARY KEY (BookID, CustomerID),
108   FOREIGN KEY (BookID) REFERENCES Book(BookID) ON DELETE CASCADE,
109   FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE
        ↪ SET NULL
110 );
```

6.2 Database Dump in DDL

```
1  -- Insert into Address table
2  INSERT INTO Address (AddressID, Street, City, PostalCode, State)
     ↪ VALUES
3  (1, '123␣Elm␣Street', 'New␣York', '10001', 'New␣York'),
4  (2, '456␣Pine␣Avenue', 'Los␣Angeles', '90001', 'California'),
5  (3, '789␣Oak␣Boulevard', 'Chicago', '60007', 'Illinois'),
6  (4, '987␣Maple␣Lane', 'Seattle', '98101', 'Washington'),
7  (5, '321␣Cedar␣Road', 'Houston', '77002', 'Texas'),
8  (6, '654␣Birch␣Street', 'San␣Francisco', '94101', 'California'),
9  (7, '890␣Willow␣Avenue', 'Miami', '33101', 'Florida'),
```

```
(8, '234 Oakwood Drive', 'Denver', '80201', 'Colorado'),
(9, '567 Walnut Lane', 'Boston', '02101', 'Massachusetts'),
(10, '876 Pinecrest Road', 'Atlanta', '30301', 'Georgia'),
(11, '109 Maple Street', 'Dallas', '75201', 'Texas'),
(12, '432 Cedar Avenue', 'Philadelphia', '19101', 'Pennsylvania'),
(13, '753 Elmwood Avenue', 'Phoenix', '85001', 'Arizona'),
(14, '246 Pine Street', 'San Diego', '92101', 'California'),
(15, '987 Oakwood Lane', 'Portland', '97201', 'Oregon'),
(16, '320 Maple Road', 'New Orleans', '70112', 'Louisiana'),
(17, '901 Cedar Avenue', 'Nashville', '37201', 'Tennessee'),
(18, '567 Pine Street', 'Dallas', '75201', 'Texas'),
(19, '890 Elmwood Avenue', 'Atlanta', '30301', 'Georgia'),
(20, '123 Maple Road', 'Seattle', '98101', 'Washington');

-- Insert into Building table
INSERT INTO Building (BuildingID, FloorNumber, WheelchairAccessibility
    ↪ , AddressID) VALUES
(1, 5, TRUE, 18),
(2, 2, FALSE, 19),
(3, 1, TRUE, 20);

-- Insert into Author table
INSERT INTO Author (AuthorID, FirstName, LastName, Gender, Birthdate,
    ↪ Nationality, Artistname) VALUES
(1, 'John', 'Doe', 'Male', '1980-01-01', 'American', 'JDoe'),
(2, 'Jane', 'Doe', 'Female', '1985-01-01', 'British', 'Jadoe'),
(3, 'James', 'Smith', 'Male', '1975-01-01', 'Canadian', 'JSmith'),
(4, 'Emily', 'Williams', 'Female', '1990-03-15', 'Australian', '
    ↪ EWilliams'),
(5, 'Michael', 'Johnson', 'Male', '1988-06-20', 'American', 'MJohnson'
    ↪ );

-- Insert into Publisher table
INSERT INTO Publisher (PublisherID, Name, Email, Website, AddressID)
    ↪ VALUES
```

```
40 (1, 'PubHouse', 'contact@pubhouse.com', 'www.pubhouse.com', 1),
41 (2, 'BookWorm Inc.', 'info@bookworm.com', 'www.bookworm.com', 2),
42 (3, 'Literature Ltd.', 'support@literature.com', 'www.literature.com',
     ↪ 3),
43 (4, 'NovelVerse Publishing', 'info@novelverse.com', 'www.novelverse.
     ↪ com', 4);
44
45 -- Insert into Customer table
46 INSERT INTO Customer (CustomerID, FirstName, LastName, Email,
     ↪ Phonenumer, Birthdate, AddressID) VALUES
47 (1, 'Jane', 'Smith', 'jane.smith@example.com', '+123456789', '
     ↪ 1990-01-01', 5),
48 (2, 'Sam', 'Johnson', 'sam.johnson@example.com', '+234567890', '
     ↪ 1988-01-01', 6),
49 (3, 'Sara', 'Williams', 'sara.williams@example.com', '+345678901', '
     ↪ 1992-01-01', 7),
50 (4, 'Mark', 'Brown', 'mark.brown@example.com', '+456789012', '
     ↪ 1995-03-15', 8),
51 (5, 'Emma', 'Davis', 'emma.davis@example.com', '+567890123', '
     ↪ 1991-06-20', 9),
52 (6, 'Alex', 'Wilson', 'alex.wilson@example.com', '+678901234', '
     ↪ 1993-09-25', 10),
53 (7, 'Oliver', 'Taylor', 'oliver.taylor@example.com', '+789012345', '
     ↪ 1994-12-10', 11),
54 (8, 'Sophia', 'Miller', 'sophia.miller@example.com', '+890123456', '
     ↪ 1989-04-05', 12);
55
56 -- Insert into Staffmember table
57 INSERT INTO Staffmember (StaffmemberID, FirstName, LastName, Salary,
     ↪ AvailableVacationDays, BuildingID, AddressID) VALUES
58 (1, 'Mark', 'Johnson', 50000.00, 20, 1, 13),
59 (2, 'Anna', 'Miller', 55000.00, 18, 1, 14),
60 (3, 'Jacob', 'Brown', 60000.00, 15, 2, 15),
61 (4, 'Emily', 'Davis', 52000.00, 17, 3, 16);
62
```

```
63 -- Insert into Book table
64 INSERT INTO Book (BookID, Title, Genre, ReleaseDate, Keyword,
       ↪ PublisherID) VALUES
65 (1, 'Journey to the East', 'Adventure', '2000-01-01', 'adventure', 1),
66 (2, 'Science 101', 'Education', '2005-01-01', 'education, 101', 2),
67 (3, 'Cooking Made Easy', 'Cookbook', '2010-01-01', 'cookbook', 3),
68 (4, 'The Enigma of Elysium', 'Mystery', '2016-08-12', 'Elysium', 3),
69 (5, 'Quantum Chronicles: Beyond Time', 'Science Fiction', '2022-02-28'
       ↪ , 'quantum', 4),
70 (6, 'The Culinary Alchemist', 'Cookbook', '2013-11-05', 'alchemist',
       ↪ 1),
71 (7, 'A Brush with Destiny', 'Biography', '2017-04-20', 'destiny', 2),
72 (8, 'Shadows of Serendipity', 'Fantasy', '2019-10-15', 'serendipity',
       ↪ 3),
73 (9, 'The Art of Deception', 'Thriller', '2014-06-18', 'art', 4),
74 (10, 'Whispers of the Moon', 'Poetry', '2011-09-08', 'moon', 1);
75
76 -- Insert into Copy table
77 INSERT INTO Copy (CopyID, BookID, CustomerID, CheckoutDate, DueDate,
       ↪ IsReturned, BuildingID, FloorNumber, ShelfNumber) VALUES
78 (1, 1, 1, '2023-05-12', '2023-06-11', FALSE, 1, 1, 1),
79 (2, 1, NULL, NULL, NULL, TRUE, 2, 2, 2),
80 (3, 1, NULL, NULL, NULL, TRUE, 3, 3, 3),
81 (4, 2, 3, '2023-01-04', '2023-02-03', FALSE, 1, 1, 2),
82 (5, 3, 4, '2023-01-05', '2023-02-04', FALSE, 2, 2, 1),
83 (6, 4, NULL, NULL, NULL, TRUE, 3, 3, 5),
84 (7, 4, NULL, NULL, NULL, TRUE, 1, 1, 2),
85 (8, 4, 6, '2023-03-08', '2023-04-07', FALSE, 2, 2, 5),
86 (9, 5, NULL, NULL, NULL, TRUE, 3, 3, 2),
87 (10, 6, NULL, NULL, NULL, TRUE, 1, 1, 3),
88 (11, 6, 2, '2023-03-11', '2023-04-09', FALSE, 2, 2, 6),
89 (12, 6, 4, '2023-05-12', '2023-06-10', FALSE, 3, 1, 2),
90 (13, 7, 5, '2023-05-13', '2023-06-11', FALSE, 1, 1, 4),
91 (14, 7, 8, '2023-03-14', '2023-04-23', FALSE, 2, 2, 3),
92 (15, 8, NULL, NULL, NULL, TRUE, 3, 2, 3),
```

```
(16, 8, NULL, NULL, NULL, TRUE, 1, 1, 5),
(17, 9, 4, '2023-01-17', '2023-01-16', FALSE, 2, 1, 1),
(18, 10, NULL, NULL, NULL, TRUE, 3, 3, 6),
(19, 10, 6, '2023-05-19', '2023-06-18', FALSE, 1, 1, 6),
(20, 10, 7, '2023-05-20', '2023-06-19', FALSE, 2, 1, 2);

-- Insert into Write table
INSERT INTO Write (AuthorID, BookID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(1, 6),
(2, 7),
(3, 8),
(4, 9),
(5, 10),
(1, 3),
(2, 5),
(5,9);

-- Insert into Review table
INSERT INTO Review (BookID, CustomerID, Stars, Text) VALUES
(1, 1, 5, 'An␣adventurous␣journey␣that␣will␣keep␣you␣hooked!'),
(2, 2, 4, 'A␣fascinating␣exploration␣of␣scientific␣concepts.'),
(3, 3, 3, 'Simple␣and␣delicious␣recipes␣for␣everyday␣cooking.'),
(4, 4, 5, 'An␣enigmatic␣and␣thrilling␣mystery␣you␣won''t␣be␣able␣to␣
    ↪ put␣down.'),
(5, 5, 4, 'A␣mind-bending␣science␣fiction␣adventure␣through␣time␣and␣
    ↪ space.'),
(6, 6, 3, 'Unleash␣your␣inner␣culinary␣alchemist␣with␣this␣magical␣
    ↪ cookbook.'),
(7, 7, 5, 'A␣captivating␣biography␣that␣paints␣a␣vivid␣portrait␣of␣
    ↪ destiny.'),
```

```
(8, 8, 4, 'Enter a world of shadows and serendipity in this enchanting fantasy.'),
(9, 1, 3, 'A gripping thriller that will keep you guessing until the very end.'),
(10, 2, 5, 'Whispered words of beauty and emotion in this captivating poetry collection.'),
(1, 3, 4, 'A heartwarming tale of friendship and resilience.'),
(2, 4, 3, 'An introspective journey into the depths of the human psyche.'),
(3, 6, 5, 'A poignant memoir that will touch your soul.'),
(4, 5, 4, 'An epic historical saga filled with passion and drama.'),
(5, 7, 2, 'A disappointing read that fell short of expectations.'),
(6, 8, 5, 'An empowering self-help book that will change your life.'),
(7, 1, 4, 'A thought-provoking exploration of existential themes.'),
(8, 2, 3, 'A light and entertaining romance for a cozy evening.'),
(9, 3, 5, 'A masterfully crafted thriller with unexpected twists.'),
(10, 4, 2, 'A poorly written novel that lacked depth and originality.'),
(1, 2, 4, 'A thrilling page-turner that will keep you on the edge of your seat.'),
(2, 3, 3, 'An informative guide to mastering the art of negotiation.'),
(3, 4, 5, 'A heartwarming story about the power of love and forgiveness.'),
(4, 8, 2, 'A disappointing sequel that failed to live up to the original.'),
(5, 6, 4, 'An inspiring memoir of overcoming adversity and achieving success.'),
(6, 7, 5, 'A beautifully written literary masterpiece that explores the human condition.'),
(7, 8, 3, 'A lighthearted and humorous novel perfect for a weekend read.'),
(8, 6, 4, 'A thought-provoking philosophical inquiry into the nature of existence.'),
(9, 8, 2, 'A confusing and poorly structured book with undeveloped
```

```
    ↪ characters.'),
146 (10, 1, 5, 'A␣captivating␣historical␣fiction␣that␣transports␣you␣to␣
    ↪ another␣era.'),
147 (1, 7, 4, 'A␣thrilling␣and␣suspenseful␣story␣with␣unexpected␣twists.')
    ↪ ,
148 (3, 2, 5, 'A␣must-read␣cookbook␣with␣mouthwatering␣recipes␣for␣every␣
    ↪ occasion.'),
149 (5, 3, 4, 'A␣mind-bending␣sci-fi␣adventure␣that␣will␣leave␣you␣
    ↪ pondering␣the␣nature␣of␣reality.');
```