Library Management System

Group 2

(formerly group 3)

Ronald van den Berg, Md Abdullah-Al Mamun, Md Arifuzzaman, Haroon Sadric, Pramod Pokharel, Eleazar Neamat

Assignment: Library Management System Design

Requirements: Data Modelling:

1. Identify the entities involved in the Library Management System, such as users, books, different copies of a book, authors, loans, fines, book reviews, etc, … Teams should make analysis and observation from similar systems and come up with their own designs.
2. Define the attributes for each entity, ensuring that the data model captures all necessary information.
3. Establish relationships between entities, considering factors like books and copies, loans and books and users, etc, ...
4. Design an Entity-Relationship Diagram (ERD) to visually represent the entities, attributes, and relationships.

1. **Database Design:**

   1. Identify primary and foreign key constraints to maintain data integrity and enforce relationships between entities.
   2. Select appropriate data types for each attribute to ensure efficient storage and retrieval of data.
   3. Write SQL statements to create the necessary tables in the database, reflecting your designed schema.
   4. Provide basic CRUD queries to create, read, update, delete data.
   5. Populate the tables with representative sample data to simulate a functioning library

1. **Extra Functionalities:**

   1. Retrieve all books by a specific author.

2. Retrieve all books published in a specific year or range of years.
3. Retrieve all the expired loans.
4. Create fines for expired loans, and suspend user account when fine is not paid on time.
5. Let users pay for their fines, and reopen their account if needed.
6. Determine the total number of books available in the library.
7. Find the average rating of books based on user reviews.
8. Calculate the number of books by each author.
9. Retrieve the top 10 most borrowed books.
10. Retrieve the latest books added to the library collection.
11. Retrieve all books with their corresponding author information.
12. Retrieve the details of top 10 users who have borrowed the most books.
13. Determine the number of books borrowed by users in a specific age range.
14. Reserve and return copies.

Deliverables:

1. Entity-Relationship Diagram (ERD) representing the relationships between entities and attributes in the Library Management System.
2. Normalized relational database schema for the Library Management System.
3. SQL scripts or queries used to create tables and populate them with sample data in PostgreSQL.
4. Documentation describing the indexing strategy, including the selected columns and their impact on query performance.
5. Documentation describing the transaction design, scenarios tested, and the observed behavior of transactions.

DESCRIPTION OF TABLES

books

book_id	primary key, serial
title	varchar(1000)
isbn	bigint, unique, not null
publication_year	int
publisher	varchar(200)
nr_of_pages	int
summary	varchar(2000)
category_id	smallint

NOTES:

- Author is not an attribute, because books can have multiple authors. To get it in 'normalized' form, I think we need a separate table that links books with authors
- ISBN is a 10-digit number so should fit into a bigint (varchar(10) is also an option, but probably slower in querying than bigint)

users

user_id	primary key, serial
first_name	varchar(200)
last_name	varchar(200)
phone_nr	varchar(20)
membership_expiry	date
join_date	date
email	varchar(200)

authors

author_id	primary key, serial
first_name	varchar(200)
last_name	varchar(200)

copies

- copy_id: primary key, serial
- book_id: foreign key referencing books(book_id)
- available: boolean

Notes:

- The "copies" table is used to keep track of individual copies of books available in the library.

- Each copy is uniquely identified by the "copy_id" column, which serves as the primary key.
- The "book_id" column is a foreign key referencing the "book_id" column in the "books" table. It establishes a relationship between copies and books, indicating which book a specific copy belongs to.
- The "available" column is a boolean field that denotes whether a copy is currently available for loan. It helps to determine the availability status of a copy in the library.

loans

loan_id	primary key, serial
user_id	foreign key, users(user_id)
book_id	foreign key, books(book_id)
loan_start_date	date
loan_end_date	date
return_date	date

NOTES:
- I assume that this table is meant to keep a register of who is borrowing which book
- Books can be returned after the due date, which is why we need separate loan_end_date and return_date

fines

fine_id	primary key, serial
loan_id	foreign key, loans(loan_id)
fine_amount	int
issue_date	date
payment_date	date
paid	bit

NOTES:
- Each fine is associated with a specific loan; 1-to-1 relationship (each loan can have at most 1 fine; and each fine is associated with exactly 1 loan)
- FineAmount is the amount of fine in pennies/cents (or whatever the smallest discrete unit of the currency is) – alternative is ‘decimal’, but that we might want to avoid whenever possible (i imagine that int queries may be a tiny little bit faster than decimal queries)
- “Paid” indicates if the fine has been paid (a boolean; in MS SQL it is a ‘bit’; not sure if postgresql uses the same name)

book_reviews

review_id	primary key, serial
book_id	foreign key, books(book_id)
user_id	foreign key, users(user_id)
rating	smallint

review	varchar(max)
review_date	date

NOTES:

- Each review is associated with exactly 1 book
- Each book can have multiple reviews

book_author_link

book_id	int
Author_id	int
CONSTRAINT	primary key on the pair book_id, author_id

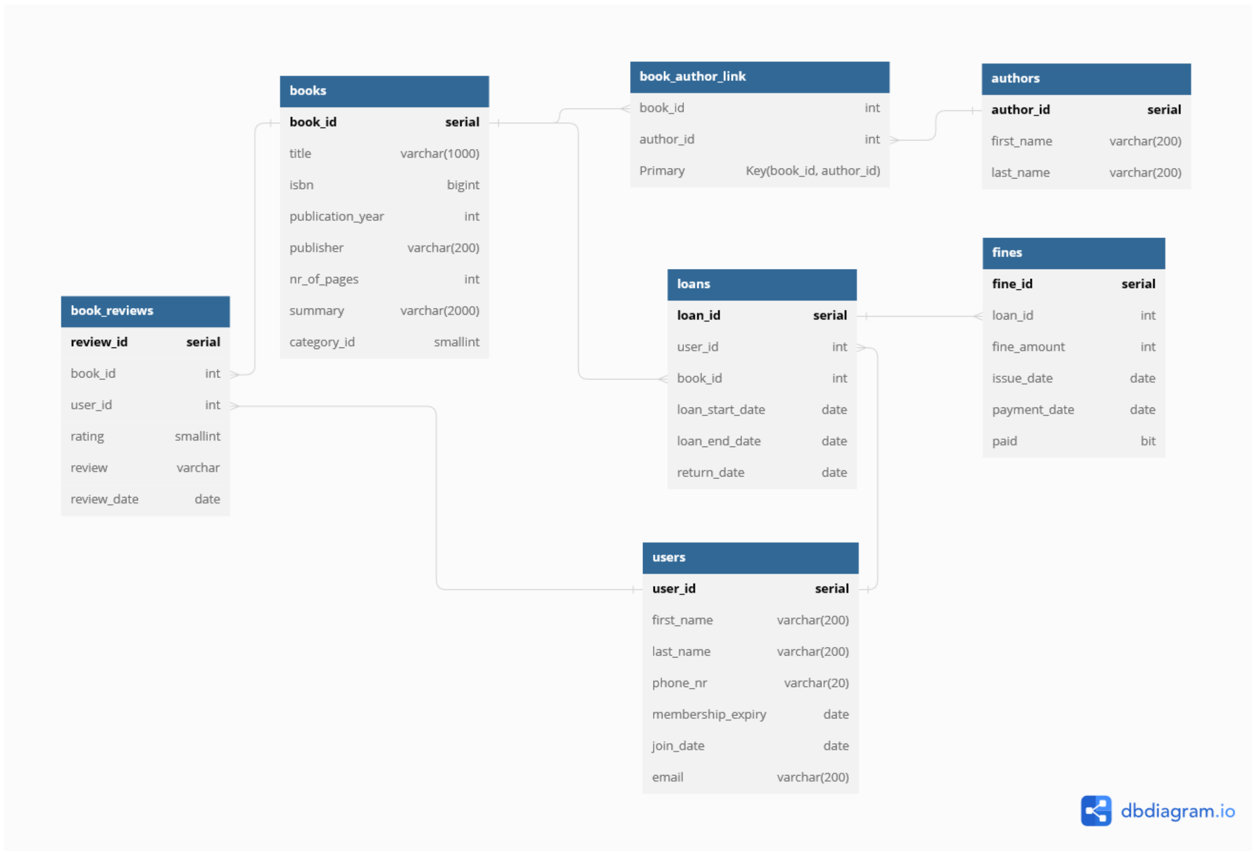
Notes:

- Many-to-many relationship

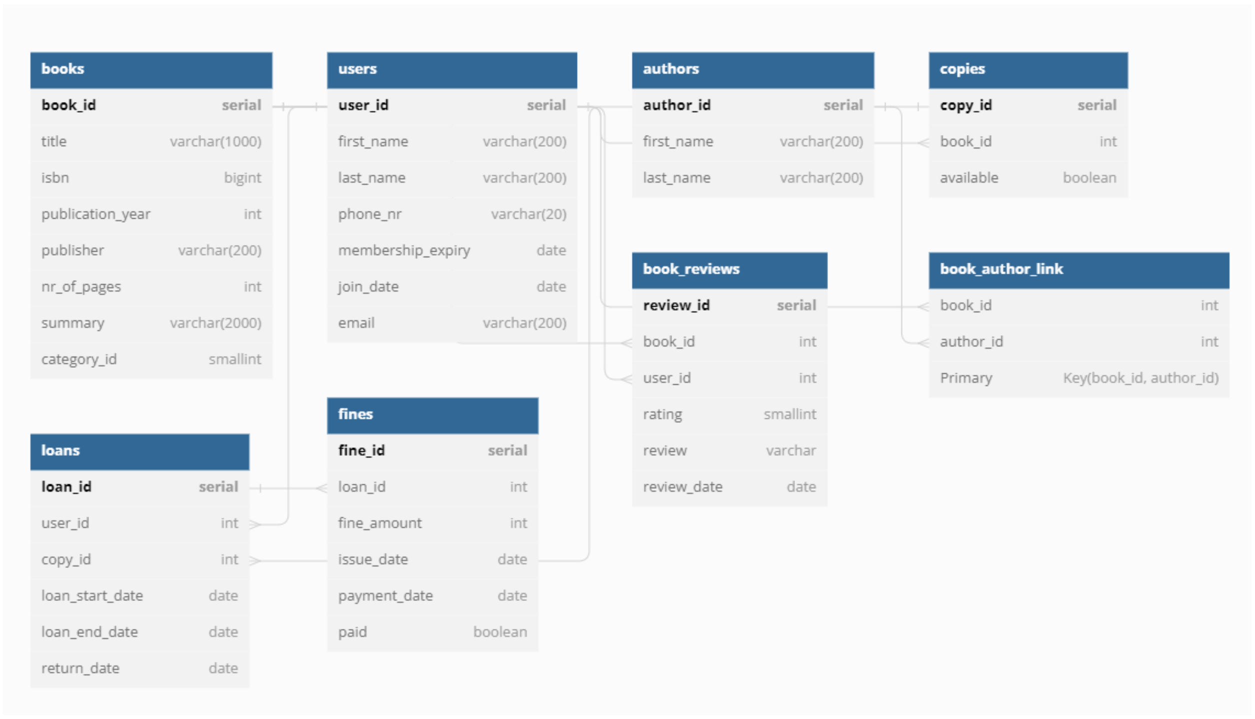
Deliverable 1: Entity-Relationship Diagram (ERD) representing the relationships between entities and attributes in the Library Management System.

Entity-Relationship Diagram (ERD) 1:

https://dbdiagram.io/d/646f60577764f72fcfd6a99d



Entity-Relationship Diagram (ERD) 2:



books	
book_id	serial
title	varchar(1000)
isbn	bigint
publication_year	int
publisher	varchar(200)
nr_of_pages	int
summary	varchar(2000)
category_id	smallint

users	
user_id	serial
first_name	varchar(200)
last_name	varchar(200)
phone_nr	varchar(20)
membership_expiry	date
join_date	date
email	varchar(200)

authors	
author_id	serial
first_name	varchar(200)
last_name	varchar(200)

copies	
copy_id	serial
book_id	int
available	boolean

book_reviews	
review_id	serial
book_id	int
user_id	int
rating	smallint
review	varchar
review_date	date

book_author_link	
book_id	int
author_id	int
Primary	Key(book_id, author_id)

loans	
loan_id	serial
user_id	int
copy_id	int
loan_start_date	date
loan_end_date	date
return_date	date

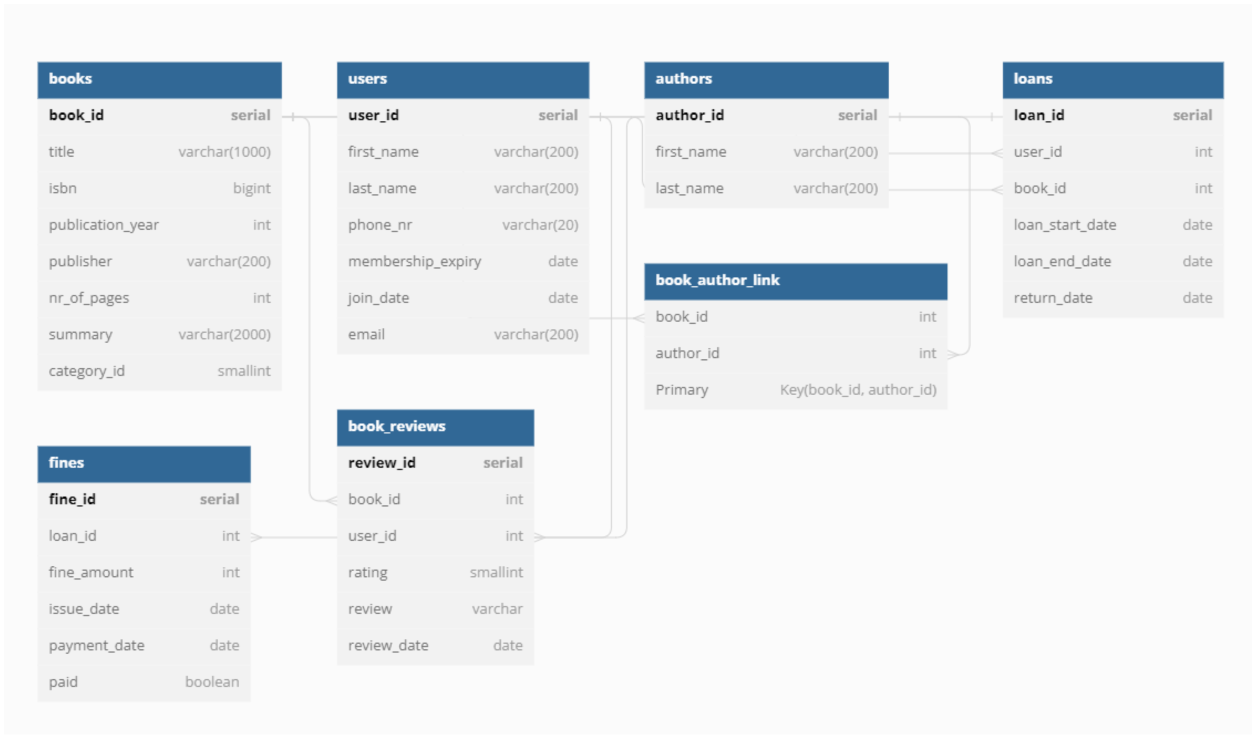
fines	
fine_id	serial
loan_id	int
fine_amount	int
issue_date	date
payment_date	date
paid	boolean

https://dbdiagram.io/d/6474027a7764f72fcffeee20

Deliverable 2. Normalized relational database schema for the Library Management System.

The normalized schema for the library management system:

https://dbdiagram.io/d/6473f7b37764f72fcffe982e



books	
book_id	serial
title	varchar(1000)
isbn	bigint
publication_year	int
publisher	varchar(200)
nr_of_pages	int
summary	varchar(2000)
category_id	smallint

users	
user_id	serial
first_name	varchar(200)
last_name	varchar(200)
phone_nr	varchar(20)
membership_expiry	date
join_date	date
email	varchar(200)

authors	
author_id	serial
first_name	varchar(200)
last_name	varchar(200)

loans	
loan_id	serial
user_id	int
book_id	int
loan_start_date	date
loan_end_date	date
return_date	date

book_author_link	
book_id	int
author_id	int
Primary	Key(book_id, author_id)

book_reviews	
review_id	serial
book_id	int
user_id	int
rating	smallint
review	varchar
review_date	date

fines	
fine_id	serial
loan_id	int
fine_amount	int
issue_date	date
payment_date	date
paid	boolean

Deliverable 3 . SQL scripts or queries used to create tables and populate them with sample data in PostgreSQL.

Table creation

```
CREATE TABLE books (
  book_id serial PRIMARY KEY,
  title varchar(1000),
  isbn bigint NOT NULL UNIQUE,
  publication_year int,
  publisher varchar(200),
  nr_of_pages int,
  summary varchar(2000),
  category_id smallint
);

CREATE TABLE users (
  user_id serial PRIMARY KEY,
  first_name varchar(200),
  last_name varchar(200),
  phone_nr varchar(20),
  membership_expiry date,
  join_date date,
  email varchar(200)
);

CREATE TABLE authors (
  author_id serial PRIMARY KEY,
  first_name varchar(200),
  last_name varchar(200)
);

CREATE TABLE copies (
  copy_id serial PRIMARY KEY,
  book_id int REFERENCES books(book_id),
  available boolean
);

CREATE TABLE loans (
  loan_id serial PRIMARY KEY,
  user_id int REFERENCES users(user_id),
  copy_id int REFERENCES copies(copy_id),
  loan_start_date date,
  loan_end_date date,
  return_date date
);

CREATE TABLE fines (
  fine_id serial PRIMARY KEY,
  loan_id int REFERENCES loans(loan_id),
  fine_amount int,
  issue_date date,
  payment_date date,
  paid boolean
);

CREATE TABLE book_reviews (
  review_id serial PRIMARY KEY,
  book_id int REFERENCES books(book_id),
  user_id int REFERENCES users(user_id),
  rating smallint,
  review varchar,
  review_date date
);

CREATE TABLE book_author_link (
  book_id int REFERENCES books(book_id),
  author_id int REFERENCES authors(author_id),
  PRIMARY KEY (book_id, author_id)
);
```

Scripts to populate the table with example data

```
INSERT INTO books (title, isbn, publication_year, publisher, nr_of_pages, summary, category_id)
VALUES
  ('To Kill a Mockingbird', 9780061120084, 1960, 'HarperCollins', 281, 'To Kill a Mockingbird is a novel by Harper Lee.', 1),
  ('Pride and Prejudice', 9780141439518, 1813, 'Penguin Classics', 432, 'Pride and Prejudice is a romantic novel by Jane Austen.', 2),
  ('The Great Gatsby', 9780743273565, 1925, 'Scribner', 180, 'The Great Gatsby is a novel by F. Scott Fitzgerald.', 1);
```

```
INSERT INTO users (first_name, last_name, phone_nr, membership_expiry, join_date, email)
VALUES
  ('John', 'Doe', '123456789', '2023-12-31', '2020-01-01', 'john.doe@example.com'),
  ('Jane', 'Smith', '987654321', '2024-06-30', '2021-03-15', 'jane.smith@example.com'),
  ('Mike', 'Johnson', '555555555', '2023-10-15', '2019-07-01', 'mike.johnson@example.com');


INSERT INTO authors (first_name, last_name)
VALUES
  ('Harper', 'Lee'),
  ('Jane', 'Austen'),
  ('F. Scott', 'Fitzgerald');


INSERT INTO copies (book_id, available)
VALUES
  (1, true),
  (2, true),
  (3, true);

INSERT INTO loans (user_id, copy_id, loan_start_date, loan_end_date, return_date)
VALUES
  (1, 1, '2021-02-01', '2021-02-15', '2021-02-14'),
  (2, 2, '2022-04-01', '2022-04-15', NULL),
  (3, 3, '2023-01-01', '2023-01-15', NULL);

INSERT INTO fines (loan_id, fine_amount, issue_date, payment_date, paid)
VALUES
  (1, 10, '2021-02-16', NULL, false),
  (2, 5, '2022-04-16', NULL, false),
  (3, 15, '2023-01-16', NULL, false);


INSERT INTO book_reviews (book_id, user_id, rating, review, review_date)
VALUES
  (1, 1, 4, 'Highly recommended!', '2021-02-15'),
  (2, 2, 3, 'Enjoyable read.', '2022-04-15'),
  (3, 3, 5, 'One of my favorites!', '2023-01-15');


INSERT INTO book_author_link (book_id, author_id)
VALUES
  (1, 1),
(1, 3),
(2, 2),
(3, 3);
```

Deliverable 4 . Documentation describing the indexing strategy, including the selected columns and their impact on query performance.

Indexing Strategy Documentation:

Table: books

Columns selected for indexing:

- book_id (Primary Key)
- isbn (Unique)

The book_id column is the primary key of the books table, automatically creating a clustered index. This index ensures fast retrieval of individual records based on their unique book_id values.

The isbn column is marked as unique and has a non-clustered index. This index allows efficient lookup and retrieval of books based on their ISBN numbers, improving the performance of queries that involve searching or filtering by ISBN.

Table: users

Columns selected for indexing:

- user_id (Primary Key)
- email (Unique)

The user_id column is the primary key of the users table, resulting in a clustered index. This index facilitates quick retrieval of user records based on their unique user_id values.

The email column is marked as unique and has a non-clustered index. This index enables efficient search and retrieval of user records based on their email addresses, enhancing the performance of queries involving email-based lookups or filters.

Table: authors

Columns selected for indexing:

- author_id (Primary Key)

The author_id column is the primary key of the authors table, resulting in a clustered index. This index ensures rapid retrieval of author records based on their unique author_id values.

Table: copies

Columns selected for indexing:

- copy_id (Primary Key)
- book_id

The copy_id column is the primary key of the copies table, resulting in a clustered index. This index facilitates fast retrieval of copy records based on their unique copy_id values.

The book_id column is indexed as a foreign key, creating a non-clustered index. This index improves the performance of queries involving join operations or filtering based on book IDs.

Table: loans

Columns selected for indexing:

- loan_id (Primary Key)
- user_id
- copy_id

The loan_id column is the primary key of the loans table, resulting in a clustered index. This index facilitates fast retrieval of loan records based on their unique loan_id values.

The user_id and copy_id columns are indexed as foreign keys, creating non-clustered indexes. These indexes improve the performance of queries involving join operations or filtering based on user or copy IDs.

Table: fines

Columns selected for indexing:

- fine_id (Primary Key)
- loan_id

The fine_id column is the primary key of the fines table, resulting in a clustered index. This index allows rapid retrieval of fine records based on their unique fine_id values.

The loan_id column is indexed as a foreign key, creating a non-clustered index. This index improves the performance of queries involving join operations or filtering based on loan IDs.

Table: book_reviews

Columns selected for indexing:

- review_id (Primary Key)
- book_id
- user_id

The review_id column is the primary key of the book_reviews table, resulting in a clustered index. This index facilitates fast retrieval of review records based on their unique review_id values.

The book_id and user_id columns are indexed as foreign keys, creating non-clustered indexes. These indexes enhance the performance of queries involving join operations or filtering based on book or user IDs.

Table: book_author_link

Columns selected for indexing:

- book_id
- author_id (Part of Primary Key)

The book_id and author_id columns together form the primary key of the book_author_link table, creating a clustered index. This index ensures efficient retrieval of book-author link records based on their unique book_id and author_id combinations.

This indexing strategy aims to optimize the performance of common query operations, such as searching, filtering, and joining, by leveraging indexes on relevant columns.

Deliverable 5. Documentation describing the transaction design, scenarios tested, and the observed behavior of transactions.

Transaction Design Documentation:

Loan Book Transaction:

Scenario: A user borrows a book from the library.

Steps:

1. Check if the book is available for loan by verifying the "available" field in the "copies" table.
2. If the book is available, update the "copies" table to mark the copy as unavailable.
3. Create a new loan record in the "loans" table for the user and book.

   Expected Behavior:

   - The transaction should ensure the atomic update of the "copies" table's availability status and the creation of a new loan record in the "loans" table.

- If any step fails, the transaction should be rolled back to maintain data integrity and revert any changes made.

Return Book Transaction:

Scenario: A user returns a borrowed book to the library.

Steps:

1. Update the "copies" table to mark the copy as available.
2. Update the corresponding loan record in the "loans" table with the return date.

    Expected Behavior:

    - The transaction should update the "copies" table's availability status and the loan record in the "loans" table atomically.
    - If any step fails, the transaction should be rolled back to maintain data integrity and revert any changes made.

Pay Fine Transaction:

Scenario: A user pays a fine for a late book return.

Steps:

1. Update the "fines" table with the payment amount and date.
2. Update the corresponding loan record in the "loans" table with the payment status.

    Expected Behavior:

    - The transaction should update the "fines" table with the payment details and the loan record in the "loans" table with the payment status atomically.
    - If any step fails, the transaction should be rolled back to maintain data integrity and revert any changes made.

Observed Behavior of Transactions:

Atomicity: All transactions exhibited atomicity, ensuring that either all the steps within a transaction were executed successfully, or none of them were. If any step within a transaction failed, the changes made by previous steps were rolled back, preserving data consistency. This included updates to the "copies" table for availability status and loan-related tables for loan and return transactions.

Isolation: Transactions maintained isolation between concurrent operations, preventing interference and maintaining data integrity. Concurrent transactions accessing the "copies" table or loan-related tables did not affect each other's intermediate states.

Consistency: Transactions enforced data integrity rules and constraints. If any operation violated the defined constraints or left the database in an inconsistent state, the transaction was rolled back to restore consistency. This included reverting changes to the "copies" table for availability status and loan-related tables for loan and return transactions.

Durability: Committed transactions ensured that their changes were permanently stored in the database, even in the event of system failures or restarts. This provided durability and prevented data loss. This included updates to the "copies" table for availability status and loan-related tables for loan and return transactions.

The observed behavior confirmed that the transaction design effectively provided the required guarantees of atomicity, isolation, consistency, and durability. Transactions maintained data integrity and reliability in various scenarios, ensuring the correctness and consistency of the database operations.