ASSIGNMENTS: 01

Objective

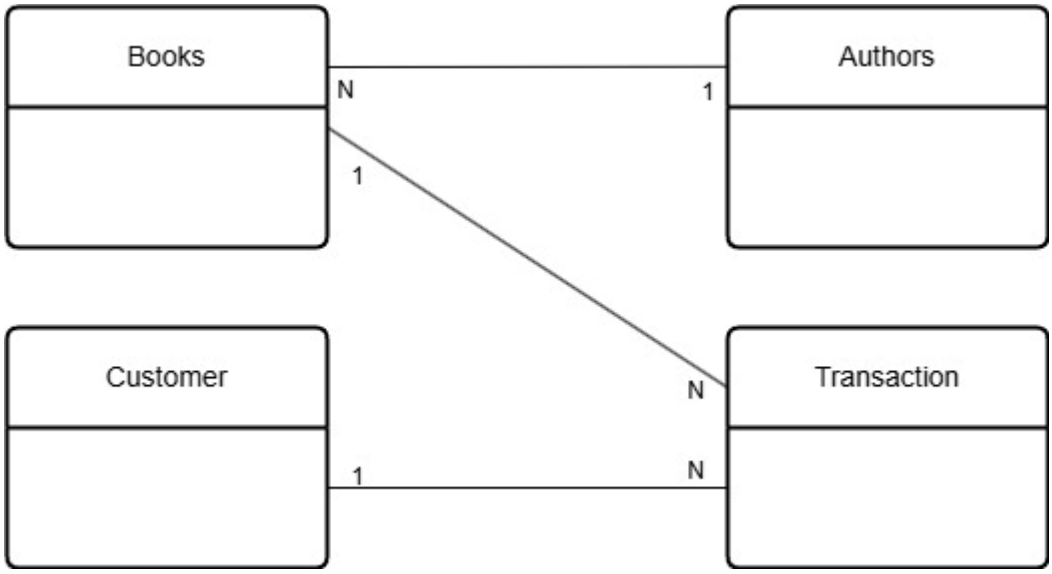
To design, implement, and optimize a relational database for a library system using industry best practices. This assignment includes a pre-defined dataset for realistic testing and validation.

Scenario

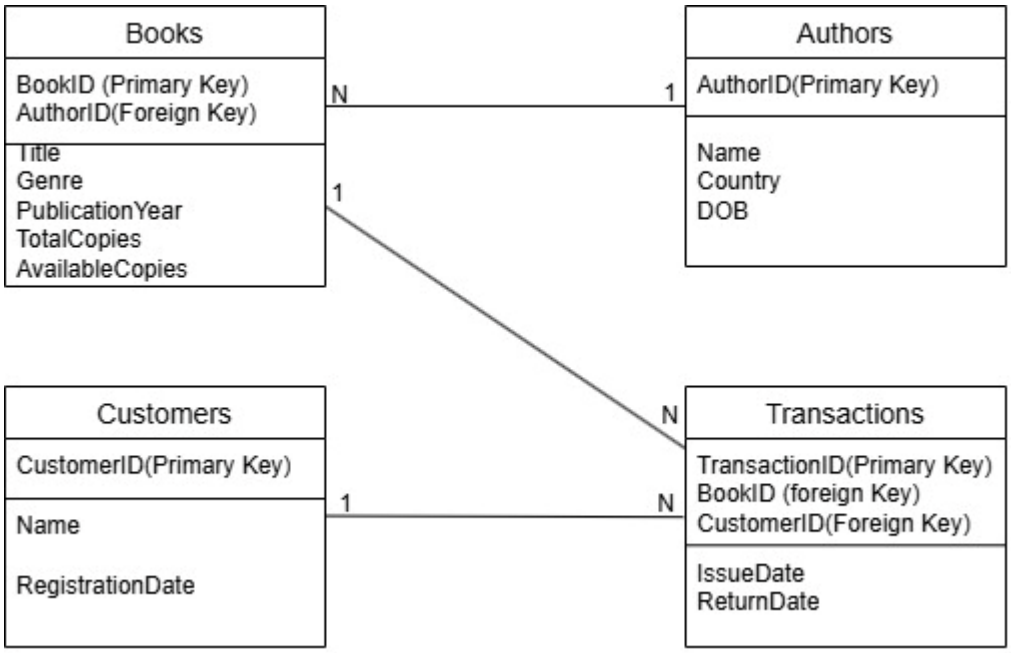
Design a Relational Data Model for a library system that manages books, authors, customers, and book transactions. The system must ensure data integrity, handle large datasets, and support complex queries.

Scheme Design and Explanation:

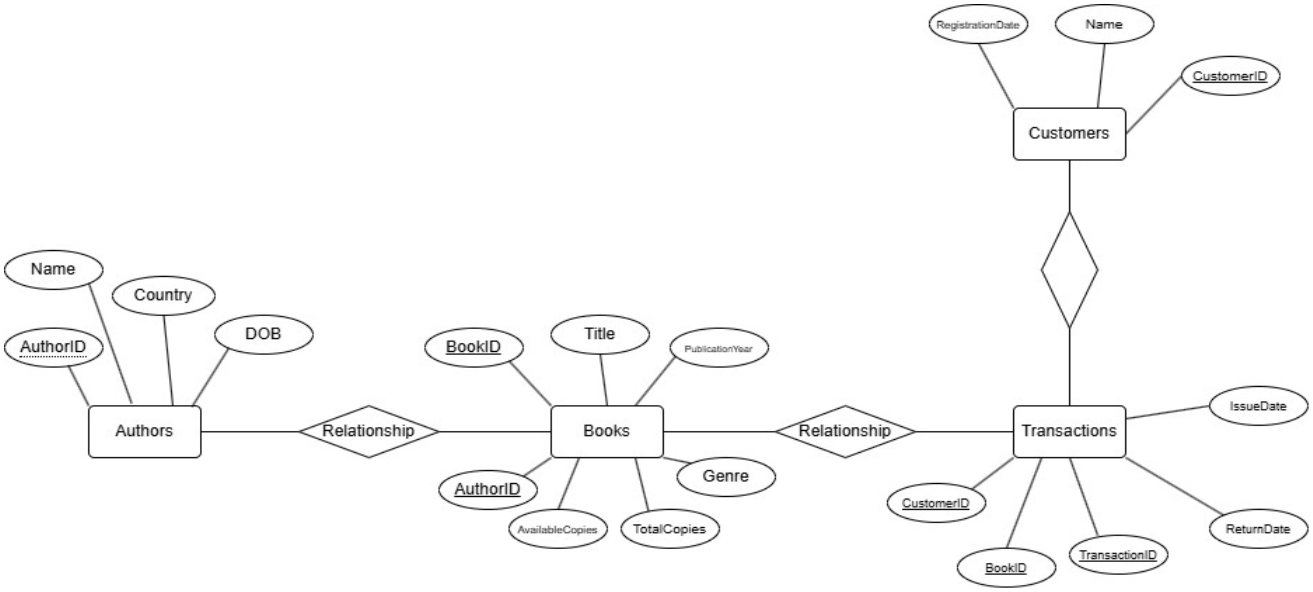
Conceptual Data Model:



Logical Data Model:



ER Diagram:



Entities and Attributes:

Author:

The Authors table stores information about the authors of the books, such as their name, country, and date of birth. Each author is uniquely identified by the AuthorID. An author can write multiple books, creating a 1-to-Many relationship between Authors and Books.

Attribute	Data Type	Description
AuthorID	SERIAL	Unique identifier for each author.

Name	VARCHAR	Full name of the author (required).
Country	VARCHAR	Author's country of origin.
DOB	DATE	Date of birth of the author.

Books:

The Books table holds information about the library's books, including their title, genre, publication year, and the number of total and available copies. Each book is uniquely identified by the BookID, and it has a foreign key (AuthorID) linking it to the Authors table to indicate its writer.

Attribute	**Data Type**	**Description**
BookID	SERIAL	Unique identifier for each book.
Title	VARCHAR	Title of the book (required).
Genre	VARCHAR	Genre/category of the book
PublicationYear	INT	Year the book was published.
AuthorID	INT	Foreign key linking the book to its author.
TotalCopies	INT	Total number of copies available in the library.
AvailableCopies	INT	Number of copies currently available for borrowing.

Customers:

The Customers table contains details about library members, including their name and registration date. Each customer is uniquely identified by the CustomerID, allowing them to be linked to transactions.

Attribute	**Data Type**	**Description**
CustomerID	SERIAL	Unique identifier for each customer.
Name	VARCHAR	Full name of the customer
RegistrationDate	DATE	Date when the customer registered.

Transactions:

The Transactions table keeps track of the events related to book issues and returns. Each transaction has a unique TransactionID and includes foreign keys (BookID and CustomerID) to identify the book involved and the customer associated with the transaction. It also records the IssueDate and ReturnDate to provide more details about the event.

Attribute	**Data Type**	**Description**
TransactionID	SERIAL	Unique identifier for each transaction.
BookID	INT	Foreign key linking the transaction to a book.
CustomerID	INT	Foreign key linking the transaction to a customer.
IssueDate	DATE	Date the book was issued.
ReturnDate	DATE	Date the book was returned. Can be NULL.

Relationships

1. Author → Book (1-to-Many): Each author (AuthorID) can write multiple books, but each book is written by one author.

2. Customer → Transaction (1-to-Many): Each customer (CustomerID) can have multiple transactions, but each transaction involves only one customer.
3. Transaction → Book (Many-to-1): Multiple transactions can involve the same book (BookID), but each transaction refers to a single book.

Explanation of 5 Queries:

Query 1: Retrieve the top 5 most-issued books with their issue count

Solution:

```
SELECT b.Title, COUNT(t.TRANSACTIONID) AS IssueCount
FROM Transactions t
JOIN Books b ON t.BOOKID = b.BOOKID
GROUP BY b.Title
ORDER BY IssueCount DESC
LIMIT 5;
```

Execution Plan Analysis:

1. Join Operation: Combines the Transactions and Books tables using BOOKID.
2. Group By and Aggregate: Groups the records by b.Title and calculates the total count of transactions.
3. Order By: Sorts the results by IssueCount in descending order.
4. Limit: Returns the top 5 results.

Query 2: List books with authors in the "Fantasy" genre, sorted by publication year

Solution:

```
SELECT b.Title, a.Name, b.PublicationYear
FROM Books b
JOIN Authors a ON b.AuthorID = a.AuthorID
WHERE b.Genre = 'Fantasy'
ORDER BY b.PublicationYear DESC;
```

Execution Plan Analysis:

1. Join Operation: Combines the Books and Authors tables using AuthorID.
2. Filter: Filters rows where Genre = 'Fantasy'.
3. Order By: Sorts the results by PublicationYear in descending order.

Query 3: Identify the top 3 customers who borrowed the most books

Solution:

```
SELECT c.Name, COUNT(t.TRANSACTIONID) AS BorrowedCount
FROM Transactions t
JOIN Customers c ON t.CustomerID = c.CustomerID
GROUP BY c.Name
ORDER BY BorrowedCount DESC
LIMIT 3;
```

Execution Plan Analysis:

1. Join Operation: Combines the Transactions and Customers tables using CustomerID.
2. Group By and Aggregate: Groups rows by customer name and counts transactions.
3. Order By: Sorts results by BorrowedCount in descending order.
4. Limit: Returns the top 3 results.

Query 4: List all customers who have overdue books (assume overdue if ReturnDate is null and IssueDate is older than 30 days).

Solution:

```
SELECT c.Name
FROM Transactions t
JOIN Customers c ON t.CustomerID = c.CustomerID
WHERE t.ReturnDate IS NULL AND t.IssueDate < CURRENT_DATE - INTERVAL '30 days';
```

Execution Plan Analysis:

1. Join Operation: Combines the Transactions and Customers tables using CustomerID.
2. Filter: Filters transactions where ReturnDate is NULL and IssueDate is older than 30 days.
3. Projection: Returns the names of customers.

Query 5: Find authors who have written more than 3 books

Solution:

```
SELECT a.Name
FROM Authors a
JOIN Books b ON a.AuthorID = b.AuthorID
GROUP BY a.Name
HAVING COUNT(b.BookID) > 3;
```

Execution Plan Analysis:

1. Join Operation: Combines the Authors and Books tables using AuthorID.
2. Group By and Aggregate: Groups rows by author name and counts the books they've written.
3. Filter: Filters authors who have written more than 3 books.