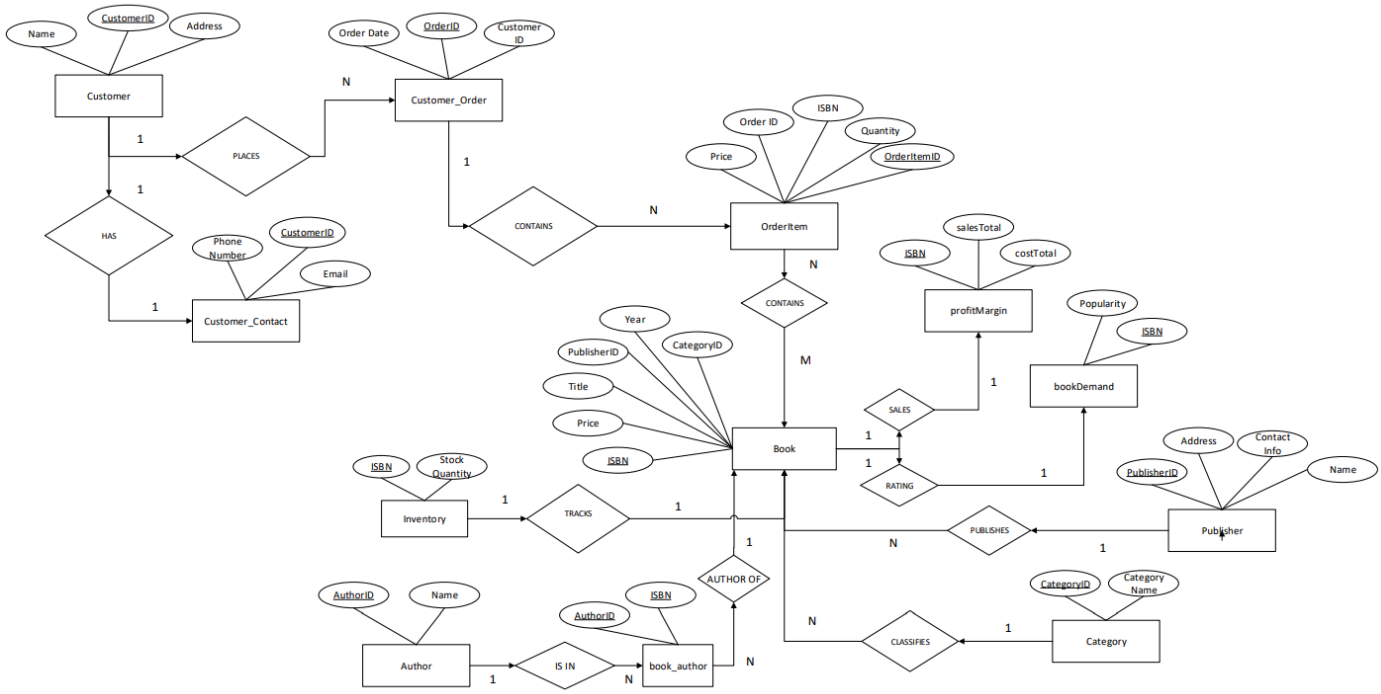
Database Description

CSE 3241

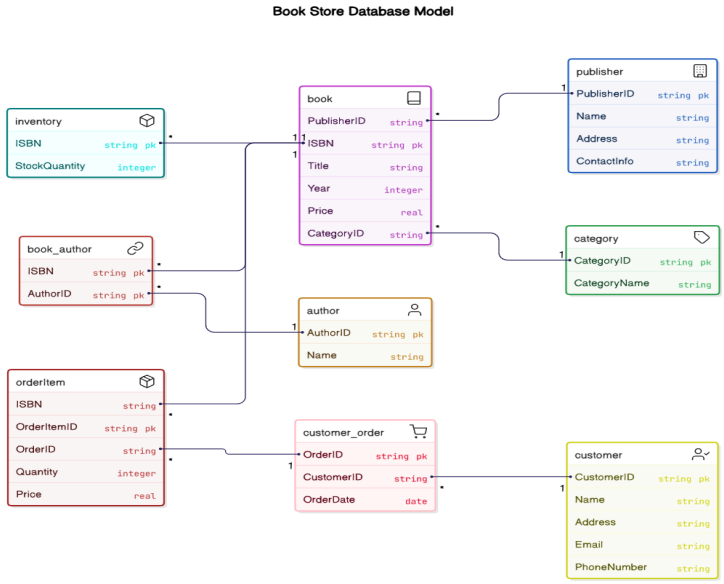
Elijah Paulman, Christian Coulibaly, Rohit Navaneetha, Kyle Roessle

Table of Contents

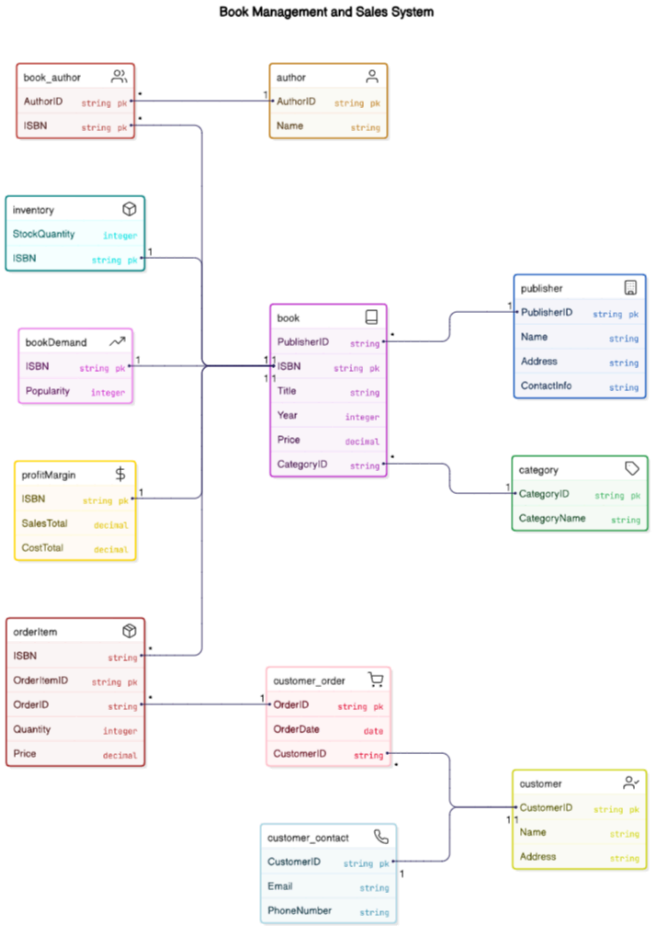
ER Model


Name
CustomerID
Address
Customer
PLACES
Order Date
OrderID
Customer ID
Customer_Order
HAS
Phone Number
CustomerID
Email
Customer_Contact
CONTAINS
Order ID
ISBN
Quantity
Price
OrderItemID
OrderItem
CONTAINS
salesTotal
ISBN
costTotal
profitMargin
Popularity
ISBN
bookDemand
Year
PublisherID
CategoryID
Title
Price
ISBN
Book
SALES
RATING
Address
Contact Info
PublisherID
Name
Publisher
ISBN
Stock Quantity
Inventory
TRACKS
PUBLISHES
AUTHOR OF
AuthorID
Name
Author
IS IN
AuthorID
ISBN
book_author
CategoryID
Category Name
Category
CLASSIFIES

Schema Diagrams



Schema based on project requirements only, not including “extra entities”



Schema including “extra entities”

Normalization Description

Table: category

- **Normalization Level:** BCNF

- **Justification:**
  The CategoryID attribute serves as the primary key and uniquely determines the CategoryName. There are no partial, transitive, or non-superkey dependencies, thus satisfying BCNF.

Table: publisher

- **Normalization Level:** BCNF

- **Justification:**
  The table's primary key is PublisherID, which functionally determines all other attributes (Name, Address, ContactInfo). All attributes are atomic, and no anomalies or redundant data are present.

Table: author

- **Normalization Level:** BCNF

- **Justification:**
  The AuthorID attribute is the primary key and fully determines the Name. The table contains only atomic fields and has no transitive dependencies.

Table: book

- **Normalization Level:** BCNF

- **Justification:**
  ISBN is the primary key and determines all other non-key attributes. The foreign keys PublisherID and CategoryID reference other BCNF-compliant tables, and no transitive dependencies exist.

Table: book_author

- **Normalization Level:** BCNF

- **Justification:**
  The composite primary key (ISBN, AuthorID) ensures that each pair is unique. There are no additional attributes, and no partial or transitive dependencies are present.

Table: customer

- **Normalization Level:** BCNF

- **Justification:**
  The table uses CustomerID as the primary key, which determines Name and Address.

Contact information such as Email and PhoneNumber is separated into its own table to eliminate alternate key dependencies.

Table: customer_contact

- **Normalization Level:** BCNF

- **Justification:**
  CustomerID is the primary key and functionally determines Email and PhoneNumber. Although both contact fields are unique, they are not candidate keys. All dependencies are on the primary key only.

Table: customer_order

- **Normalization Level:** BCNF

- **Justification:**
  The primary key OrderID determines both CustomerID and OrderDate. The table contains no redundant data or transitive dependencies.

Table: orderItem

- **Normalization Level:** BCNF

- **Justification:**
  The primary key OrderItemID uniquely identifies each record and determines the associated OrderID, ISBN, Quantity, and Price. All non-key attributes are fully functionally dependent on the primary key.

Table: inventory

- **Normalization Level:** BCNF

- **Justification:**
  ISBN is the primary key and determines StockQuantity. No anomalies or non-superkey dependencies exist.

Table: bookDemand

- **Normalization Level:** BCNF

- **Justification:**
  The ISBN attribute is the primary key and determines the Popularity score. The design avoids surrogate keys and ensures all dependencies are on the primary key.

Table: profitMargin

- **Normalization Level:** BCNF

- **Justification:**
  ISBN is the primary key and functionally determines SalesTotal and CostTotal. All attributes are atomic, and no transitive or partial dependencies exist.

Functional Dependencies

category Table
FDs:

- CategoryID → CategoryName *(Primary Key)*

---

publisher Table
FDs:

- PublisherID → {Name, Address, ContactInfo} *(Primary Key)*

---

author Table
FDs:

- AuthorID → Name *(Primary Key)*

---

book Table
FDs:

- ISBN → {Title, Year, Price, PublisherID, CategoryID} *(Primary Key)*

---

book_author Table
FDs:

- {ISBN, AuthorID} → {ISBN, AuthorID} *(Composite Primary Key)*

---

customer Table
FDs:

- CustomerID → {Name, Address} *(Primary Key)*

---

customer_contact Table
FDs:

- CustomerID → {Email, PhoneNumber} *(Primary Key, also Foreign Key to customer)*
- Email → CustomerID *(Email is UNIQUE)*
- PhoneNumber → CustomerID *(PhoneNumber is UNIQUE)*

---

customer_order Table
FDs:

- OrderID → {CustomerID, OrderDate} *(Primary Key)*

---

orderItem Table
FDs:

- OrderItemID → {OrderID, ISBN, Quantity, Price} *(Primary Key)*

---

inventory Table
FDs:

- ISBN → StockQuantity *(Primary Key and Foreign Key to book)*

---

bookDemand Table
FDs:

- ISBN → Popularity *(Primary Key and Foreign Key to book)*

---

profitMargin Table
FDs:

- ISBN → {SalesTotal, CostTotal} *(Primary Key and Foreign Key to book)*

Database Indexes

1. Primary Key Indexes

Each table with a PRIMARY KEY automatically benefits from a unique index on the key column(s). These indexes ensure that each row is uniquely identifiable and enable fast lookups.

- **Tables with Primary Key Indexes**:
  category(CategoryID), publisher(PublisherID), author(AuthorID), book(ISBN), book_author(ISBN, AuthorID), customer(CustomerID), customer_contact(CustomerID), customer_order(OrderID), orderItem(OrderItemID), inventory(ISBN), bookDemand(ISBN), profitMargin(ISBN)
- **Rationale**:
  These indexes are essential for maintaining entity integrity and supporting fast retrieval by unique identifiers.

---

2. Foreign Key Support Indexes

Although not created automatically, indexes on foreign key columns are added where join operations or frequent filtering are likely.

- **Added Indexes**:
  - book(PublisherID)
  - book(CategoryID)
  - book_author(AuthorID)
  - customer_order(CustomerID)
  - orderItem(OrderID)
  - orderItem(ISBN)
- **Rationale**:
  These indexes enhance the performance of common join operations such as:
  - Retrieving books by category or publisher
  - Listing all orders for a customer
  - Viewing all items in a particular order
  - Fetching all books written by a specific author

---

3. Unique Indexes

The customer_contact table defines Email and PhoneNumber as unique fields.

- **Enforced Unique Indexes**:
  - customer_contact(Email)

    - customer_contact(PhoneNumber)
- **Rationale**:
  These indexes ensure no duplicate contact information exists and support fast lookups or validations when inserting or updating contact details.

---

4. Optional Indexes Not Implemented

To maintain simplicity, the following potential indexes were **not implemented** but may be considered for future scalability:

- book(Title) – for full-text search or partial matches on book names
- author(Name) – for alphabetical searches or auto-complete features
- bookDemand(Popularity) – if sorting/filtering by popularity becomes frequent

These were omitted because the dataset is small and such queries are unlikely to cause significant performance issues in a classroom environment.

Database Views

View 1: CategorySales

Description:
This view calculates the total sales revenue for each book category. It joins the category, book, and orderItem tables, and aggregates the revenue based on the product of quantity and unit price for each order item. This view is useful for identifying which categories contribute most to overall revenue.

Relational Algebra Expression:
Let:
C = category
B = book
OI = orderItem

CategorySales := π_{CategoryID, CategoryName, SUM(Quantity × Price) → TotalSales}
(γ_{CategoryID, CategoryName; SUM(Quantity × Price)}
(σ_{B.CategoryID = C.CategoryID ∧ B.ISBN = OI.ISBN} (C × B × OI)))

SQL Statement to Construct the View:
```
CREATE VIEW CategorySales AS
SELECT c.CategoryID, c.CategoryName, SUM(oi.Quantity * oi.Price) AS TotalSales
FROM category c
JOIN book b ON c.CategoryID = b.CategoryID
JOIN orderItem oi ON b.ISBN = oi.ISBN
GROUP BY c.CategoryID, c.CategoryName;
```

Sample Output:

CategoryID	CategoryName	TotalSales
CAT001	Computer	1451.68
CAT002	Literature & Fiction	173.01
CAT003	Accounting & Finance	56.849999999999994
CAT008	Fantasy	23.97
CAT009	Reference	27.5

View 2: AuthorPopularity

Description:
This view ranks authors by the total popularity of the books they have written. It joins the

author, book_author, and bookDemand tables and aggregates the popularity scores from the bookDemand table. This is useful for identifying high-demand authors based on cumulative interest in their books.

Relational Algebra Expression:
Let:
A = author
BA = book_author
BD = bookDemand

AuthorPopularity := π_{AuthorID, Name, SUM(Popularity) → TotalPopularity}
(γ_{AuthorID, Name; SUM(Popularity)}
(σ_{A.AuthorID = BA.AuthorID ∧ BA.ISBN = BD.ISBN} (A × BA × BD)))

SQL Statement to Construct the View:

```
CREATE VIEW AuthorPopularity AS
SELECT a.AuthorID, a.Name, SUM(bd.Popularity) AS TotalPopularity
FROM author a
JOIN book_author ba ON a.AuthorID = ba.AuthorID
JOIN bookDemand bd ON ba.ISBN = bd.ISBN
GROUP BY a.AuthorID, a.Name;
```

Sample Output (Limited to 10 Results):

AuthorID	Name	TotalPopularity
AUTH001	Chip Dawes	100
AUTH002	Biju Thomas	100
AUTH003	Doug Stuns	100
AUTH004	Matthew Weishan	100
AUTH005	Joseph C. Johnson	100
AUTH006	Brian Knight	95
AUTH007	Ralph Kimball	90
AUTH008	Margy Ross	90
AUTH009	Bill Mann	85
AUTH010	Ian H. Witten	80

Sample Transactions

Transaction 1: Customer Places a New Order

Description:
This transaction handles the creation of a new customer order. It inserts a new order record, adds associated order items, and updates inventory to reflect the sold quantities. This ensures that if any part of the order fails (e.g., due to a constraint or data issue), none of the changes will be committed.

SQL Code:

```
BEGIN TRANSACTION;

INSERT INTO customer_order (OrderID, CustomerID, OrderDate)
VALUES ('ORD100', 'CUST005', CURRENT_DATE);

INSERT INTO orderItem (OrderItemID, OrderID, ISBN, Quantity, Price)
VALUES
('ITEM100', 'ORD100', '0072227885', 2, 34.99),
('ITEM101', 'ORD100', '0471200247', 1, 50.00);

UPDATE inventory
SET StockQuantity = StockQuantity - 2
WHERE ISBN = '0072227885';

UPDATE inventory
SET StockQuantity = StockQuantity - 1
WHERE ISBN = '0471200247';

COMMIT;
```

Transaction 2: Canceling an Existing Order

Description:
When a customer cancels an order, this transaction deletes the order and its associated items and restores the inventory levels. All updates are made within the same transaction to prevent data inconsistencies.

SQL Code:

```
BEGIN TRANSACTION;
```

```
UPDATE inventory
SET StockQuantity = StockQuantity + 2
WHERE ISBN = '0072227885';

UPDATE inventory
SET StockQuantity = StockQuantity + 1
WHERE ISBN = '0471200247';

DELETE FROM orderItem
WHERE OrderID = 'ORD100';

DELETE FROM customer_order
WHERE OrderID = 'ORD100';

COMMIT;
```

Transaction 3: Adding a New Book with a New Author

Description:
This transaction adds a new book to the catalog, introduces a new author, establishes the author-book relationship, and initializes inventory, demand, and profit tracking for the book. This ensures all related data is inserted together without violating referential integrity.

SQL Code:

```
BEGIN TRANSACTION;

INSERT INTO author (AuthorID, Name)
VALUES ('AUTH200', 'Lena Clarke');

INSERT INTO book (ISBN, Title, Year, Price, PublisherID, CategoryID)
VALUES ('9999999999', 'Climate Systems Engineering', 2024, 49.99, 'PUB001', 'CAT005');

INSERT INTO book_author (ISBN, AuthorID)
VALUES ('9999999999', 'AUTH200');

INSERT INTO inventory (ISBN, StockQuantity)
VALUES ('9999999999', 20);

INSERT INTO bookDemand (ISBN, Popularity)
VALUES ('9999999999', 0);

INSERT INTO profitMargin (ISBN, SalesTotal, CostTotal)
VALUES ('9999999999', 0, 0);

COMMIT;
```