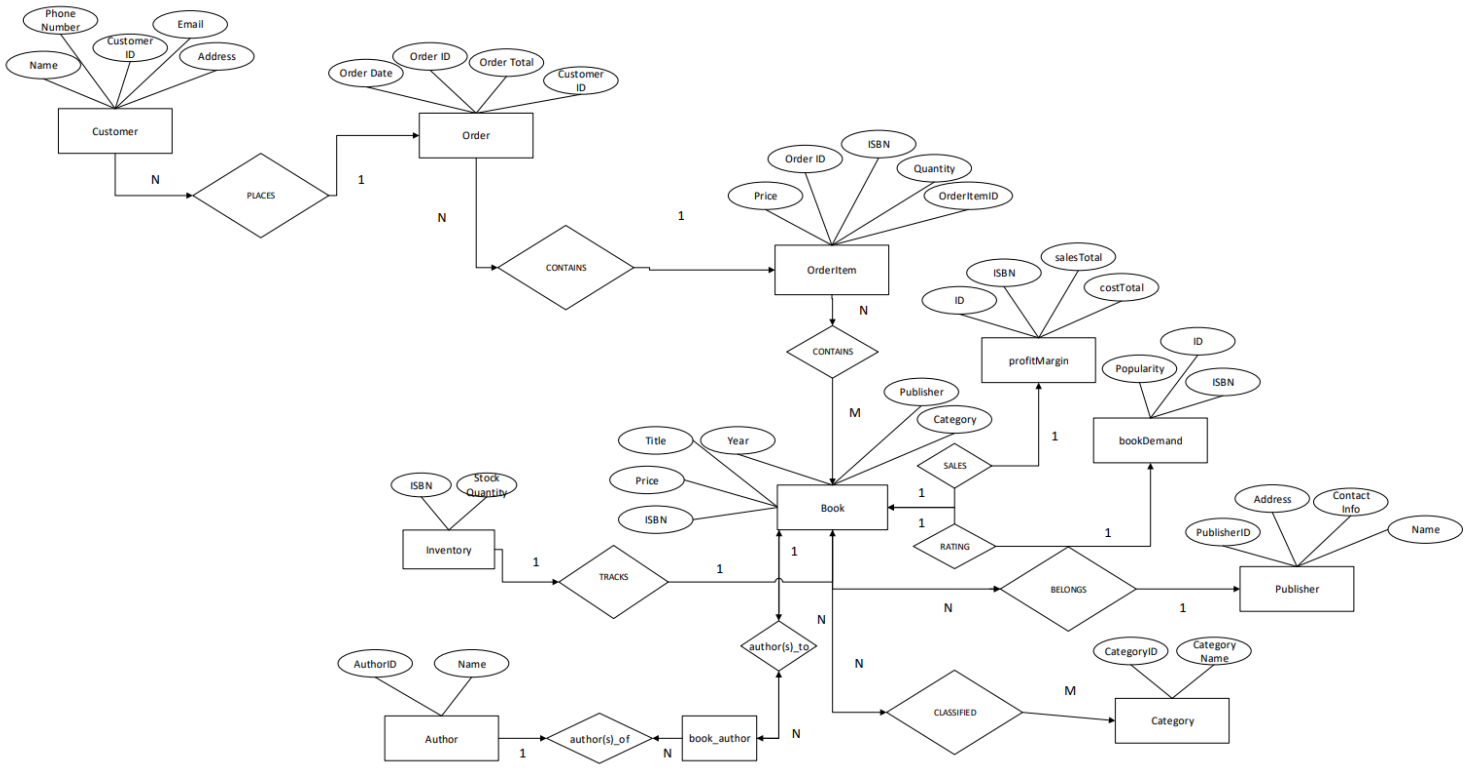
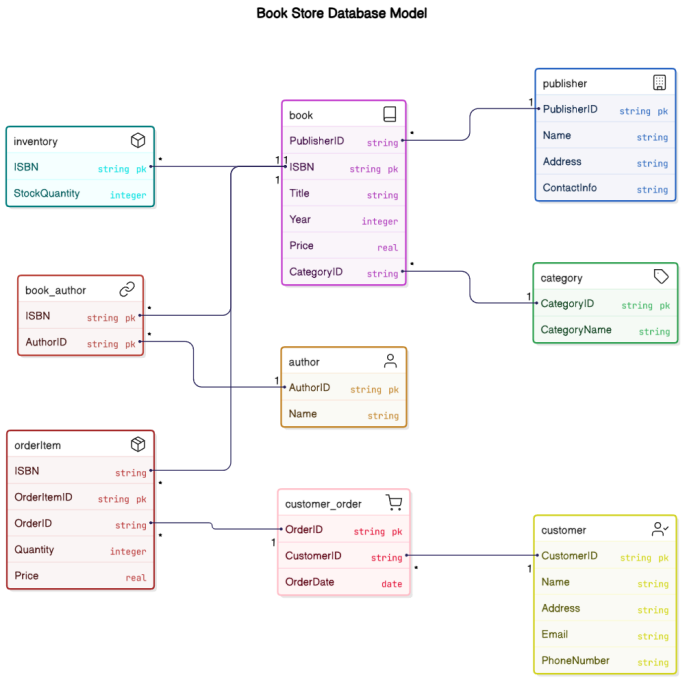
CSE 3241 Project Checkpoint 04 – Functional Dependencies and Normal Forms

Names: Christian Coulibaly, Elijah Paulman, Kyle Roessle, Rohan Navaneetha	**Date**: 4/2/25

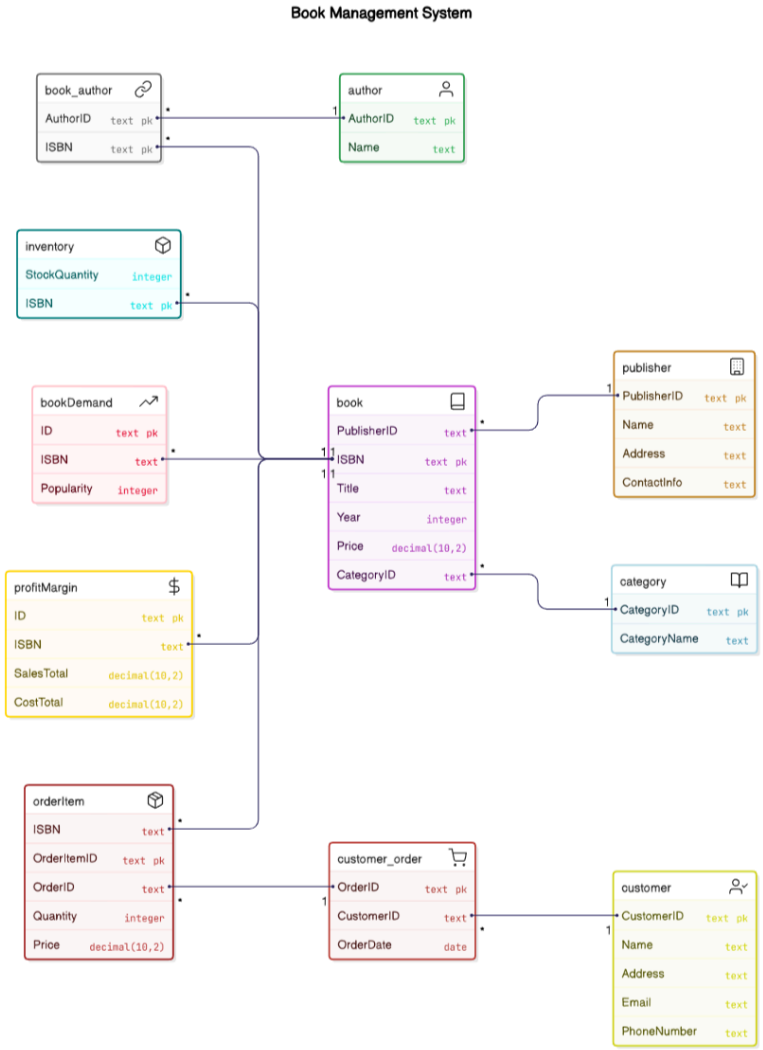
In a **NEATLY TYPED** document, provide the following:

1. Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 03. **If you were instructed to change the model for Project Checkpoint 03, make sure you use the revised versions of your models.**





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



Schema including “extra entities”

2. For each relation schema in your model, indicate the functional dependencies. Think carefully about what you are modeling here - make sure you consider all the possible dependencies in each relation and not just the ones from your primary keys. For example, a customer's credit card number is unique, and so will uniquely identify a customer even if you have another key in the same table (in fact, if the customer can have multiple credit card numbers, the dependencies can get even more involved).

- **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, Email, PhoneNumber} (Primary Key).
    - Email → CustomerID (Unique).
    - PhoneNumber → CustomerID (Unique).
- **customer_order Table (Original)**
  - FDs:
    - OrderID → {CustomerID, OrderDate, OrderTotal} (Primary Key).
    - Issue: OrderTotal is redundant and creates a transitive dependency (derived from orderItem).
- **orderItem Table**
  - FDs:
    - OrderItemID → {OrderID, ISBN, Quantity, Price} (Primary Key).
- **inventory Table**
  - FDs:
    - ISBN → StockQuantity (Primary Key).
- **bookDemand Table**
  - FDs:
    - ID → {ISBN, Popularity} (Primary Key).
- **profitMargin Table**
  - FDs:
    - ID → {ISBN, SalesTotal, CostTotal} (Primary Key).2.1 category Table

3. For each relation schema in your model, determine the highest normal form of the relation. If the relation is not in 3NF, rewrite your relation schema so that it is in at least 3NF.

   The following tables are in **BCNF:**
   - **category**
   - **publisher**
   - **author**
   - **book**
   - **book_author**
   - **customer**
   - **orderItem**
   - **inventory**
   - **bookDemand**
   - **profitMargin**

   Reasoning:

   All determinants in these tables are superkeys, ensuring there are no partial dependencies (as in 2NF) and no transitive dependencies (as in 3NF).

   customer_order Table (Original)
   - Original Normal Form: 2NF (violates 3NF due to transitive dependency via OrderTotal).
   - Revised Schema (3NF):

```
CREATE TABLE customer_order (
    OrderID TEXT PRIMARY KEY,
    CustomerID TEXT NOT NULL,
    OrderDate DATE NOT NULL DEFAULT CURRENT_DATE,
    FOREIGN KEY (CustomerID) REFERENCES customer(CustomerID) ON DELETE
CASCADE
);
```

   - Revised Normal Form: BCNF (after removing OrderTotal).
     - *Changes also reflected in second schema diagram for part 2

   Revised Schema (3NF):

4. For each relation schema in your model that is in 3NF but not in BCNF, either rewrite the relation schema to BCNF or provide a short justification for why this relation should be an exception to the rule of putting relations into BCNF.

   No exceptions needed. All relations are in BCNF after fixing customer_order.

5. For your database, propose at least two interesting views that can be built from your relations. These views must involve joining at least two tables together each and must include some kind of aggregation in the view. Each view must also be able to be described by a one or two sentence description in plain English. Provide the code for constructing your views along with the English language description of what the view is supposed to be providing.

CategorySales View

- Description: Shows total sales revenue per book category by aggregating order item prices and quantities.
- Code:

```
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;
```

AuthorPopularity View

- Description: Ranks authors by the total popularity of their books, aggregated from demand data.
- Code:

```
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;
```