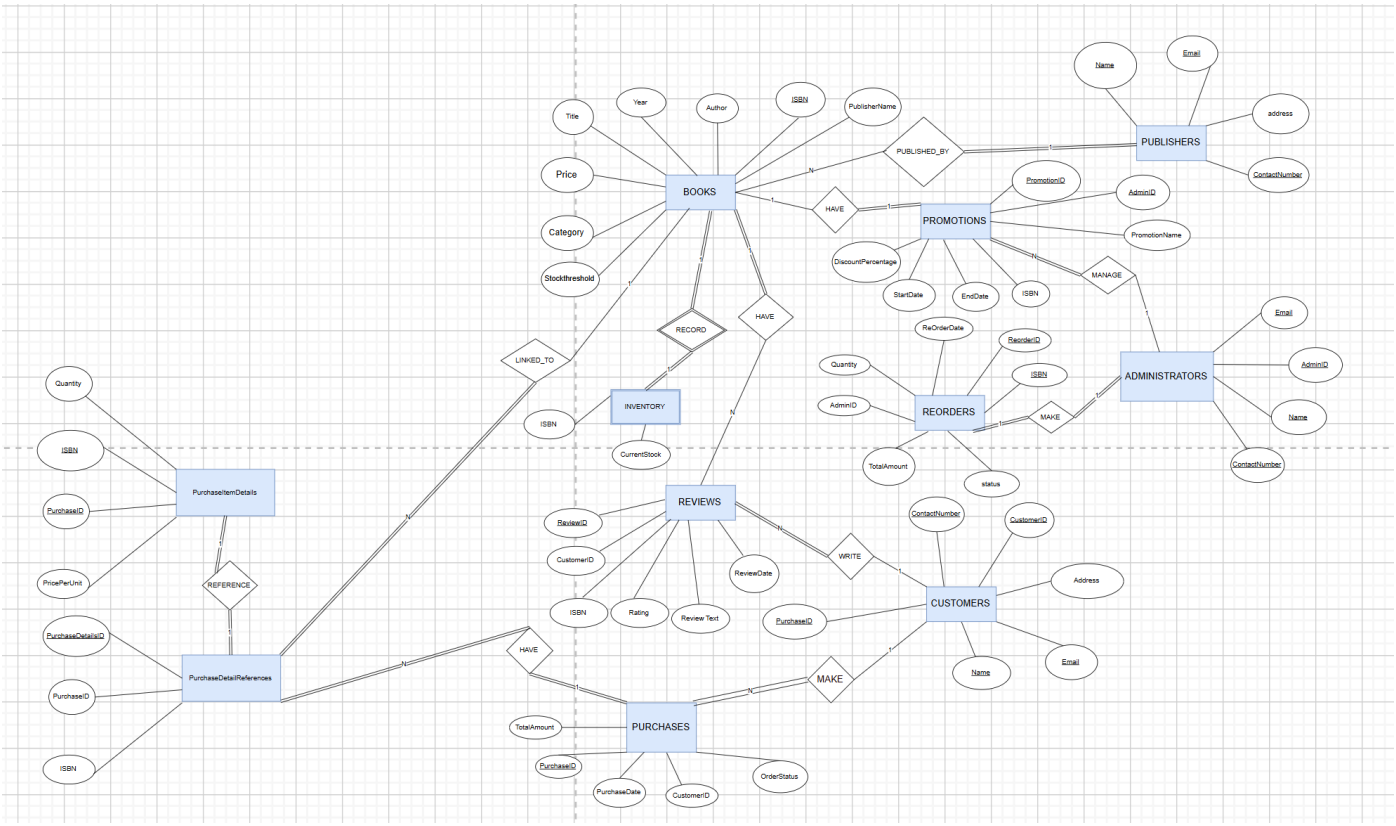
CSE 3241 Project Checkpoint 04 – Functional Dependencies and Normal Forms

Names	Date
Yujie Yang, Yuang Li, zhengyang peng	04/03/2025

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.**



https://app.diagrams.net/#G1t1o36p-UUX8Euo_DwwTht0PH2SnF9Lej#%7B%22pageId%22%3A%22ekqiZcGwjMcHRnvRa6mv%22%7D

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).

1. Administrators(AdminID, Email, Name, ContactNumber)

   { AdminID } → { Email, Name, ContactNumber }

2. Publishers(PublisherName, Email, Address, ContactNumber)
   { PublisherName } → { Email, Address, ContactNumber }

3. Books(ISBN, Title, Author, PublisherName, Year, Price, Category, StockThreshold)
   { ISBN } → { Title, Author, PublisherName, Year, Price, Category, StockThreshold }

4. Promotions(PromotionID, PromotionName, DiscountPercentage, StartDate, EndDate, ISBN, AdminID)
   { PromotionID } → { PromotionName, DiscountPercentage, StartDate, EndDate, ISBN, AdminID }

5. Customers(CustomerID, ContactNumber, Name, Address, Email)
   { CustomerID } → { ContactNumber, Name, Address, Email }

6. Purchases(PurchaseDetailsID, PurchaseID, CustomerID, OrderStatus, PurchaseDate, TotalAmount)
   { PurchaseID } → { PurchaseDetailsID, CustomerID, OrderStatus, PurchaseDate, TotalAmount }

7. PurchaseDetailReferences (PurchaseDetailsID, PurchaseID, ISBN
   { PurchaseDetailsID } → { PurchaseID, ISBN, Quantity, PricePerUnit }

8. PurchaseItemDetails (PurchaseID, ISBN, Quantity, PricePerUnit)
   {PurchaseID, ISBN } → { Quantity, PricePerUnit }

9. Reviews(ReviewID, CustomerID, ISBN, Rating, ReviewText, ReviewDate)
   { ReviewID } → { CustomerID, ISBN, Rating, ReviewText, ReviewDate }

10. Reorders(ReorderID, ISBN, Quantity, Status, TotalAmount, AdminID, ReOrderDate)
    { ReorderID } → { ISBN, Quantity, Status, TotalAmount, AdminID, ReOrderDate }

11. Inventory(ISBN, CurrentStock)
    { ISBN } → { CurrentStock }

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.

   Administrators: BCNF
   Publishers BCNF
   Books BCNF
   Promotions BCNF
   Customers BCNF
   Purchases BCNF

PurchaseDetailReferences BCNF
PurchaseItemDetails BCNF
Reviews BCNF
Reorders BCNF
Inventory BCNF

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.

There's no need for further decomposition as all relations are already in BCNF, which indicates that the schema is free from redundancies.

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.

   Views 1: discover the top selling books by revenue

   Description: This view lists each book's title, author, total revenue generated from sales, and the number of copies sold. It collects sales data to show which books generate the most revenue.

   SQL CODES:

```
CREATE VIEW TopBooksByRevenue AS
SELECT B.Title, B.Author, SUM(PD.Quantity * PD.PricePerUnit) AS TotalRevenue, SUM(PD.Quantity) AS
TotalCopiesSold
FROM Books B
JOIN PurchaseDetails PD ON B.ISBN = PD.ISBN
GROUP BY B.Title, B.Author
ORDER BY TotalRevenue DESC;
```

   Explanations of codes:
   1. Joins Books and PurchaseDetails tables using ISBN.
   2. Using SUM to calculate
   3. Groups by book **Title** and **Author**.
   4. Orders by **TotalRevenue** to rank the most profitable books.

   Views 2: Customer Purchase Summary

Description:
This view provides a summary of each customer's purchase activity, including the total amount spent and the total number of purchases made. It helps identify high-value customers.

SQL codes:

```
CREATE VIEW HighValueCustomers AS

SELECT C.Name, C.Email, COUNT(P.PurchaseID) AS TotalPurchases, SUM(P.TotalAmount) AS TotalSpent

FROM Customers C

JOIN Purchases P ON C.CustomerID = P.CustomerID

GROUP BY C.Name, C.Email

ORDER BY TotalSpent DESC;
```

Explanation of codes:

Joins **Customers** and **Purchases** using **CustomerID**.

Uses COUNT to find the number of purchases and SUM to calculate total spending.

Groups by customer **Name** and **Email**.

Orders by **TotalSpent** to highlight the top spenders.