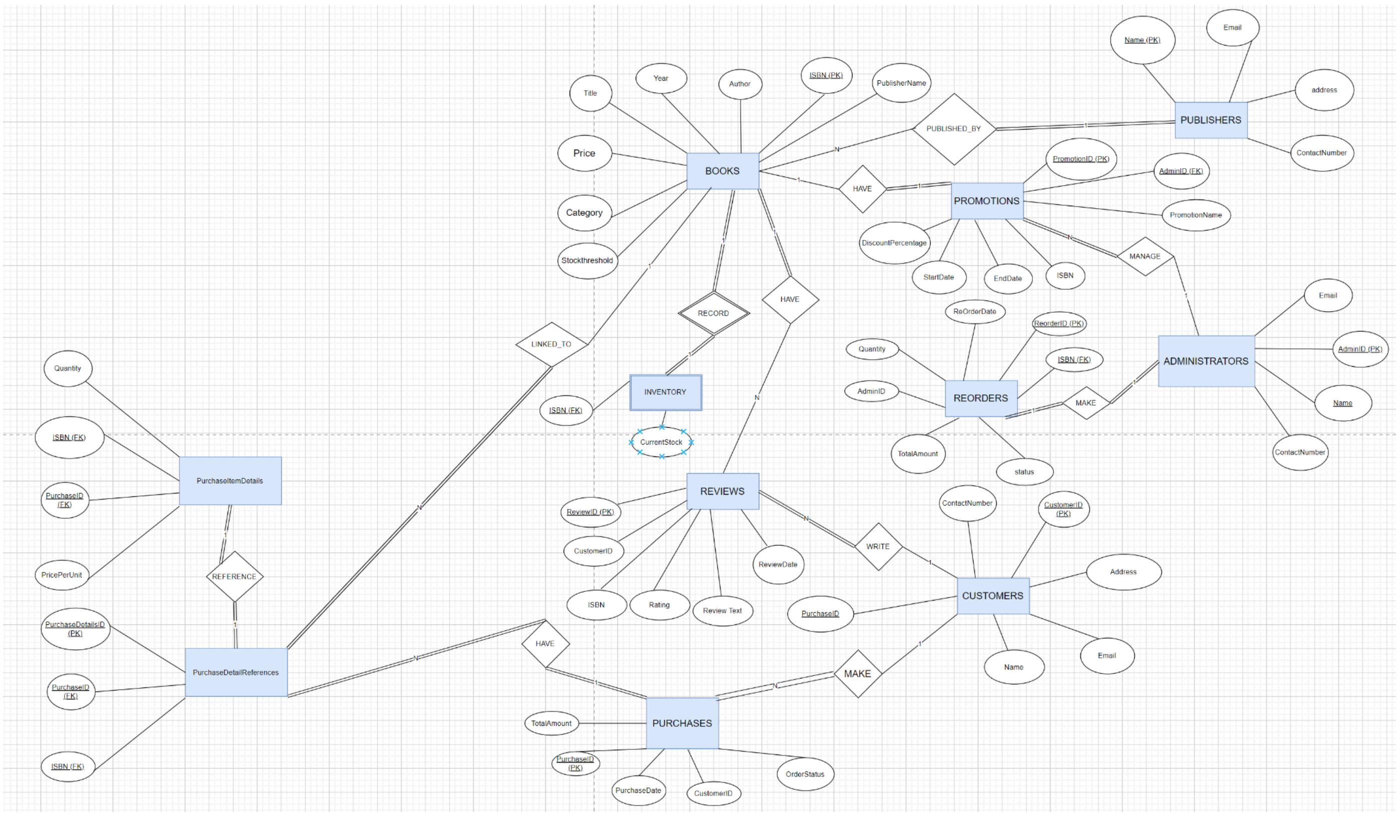
CSE 3241 FINAL PROJECT

Team Member: Yujie Yang, Zhengyang Peng, Yuang Li

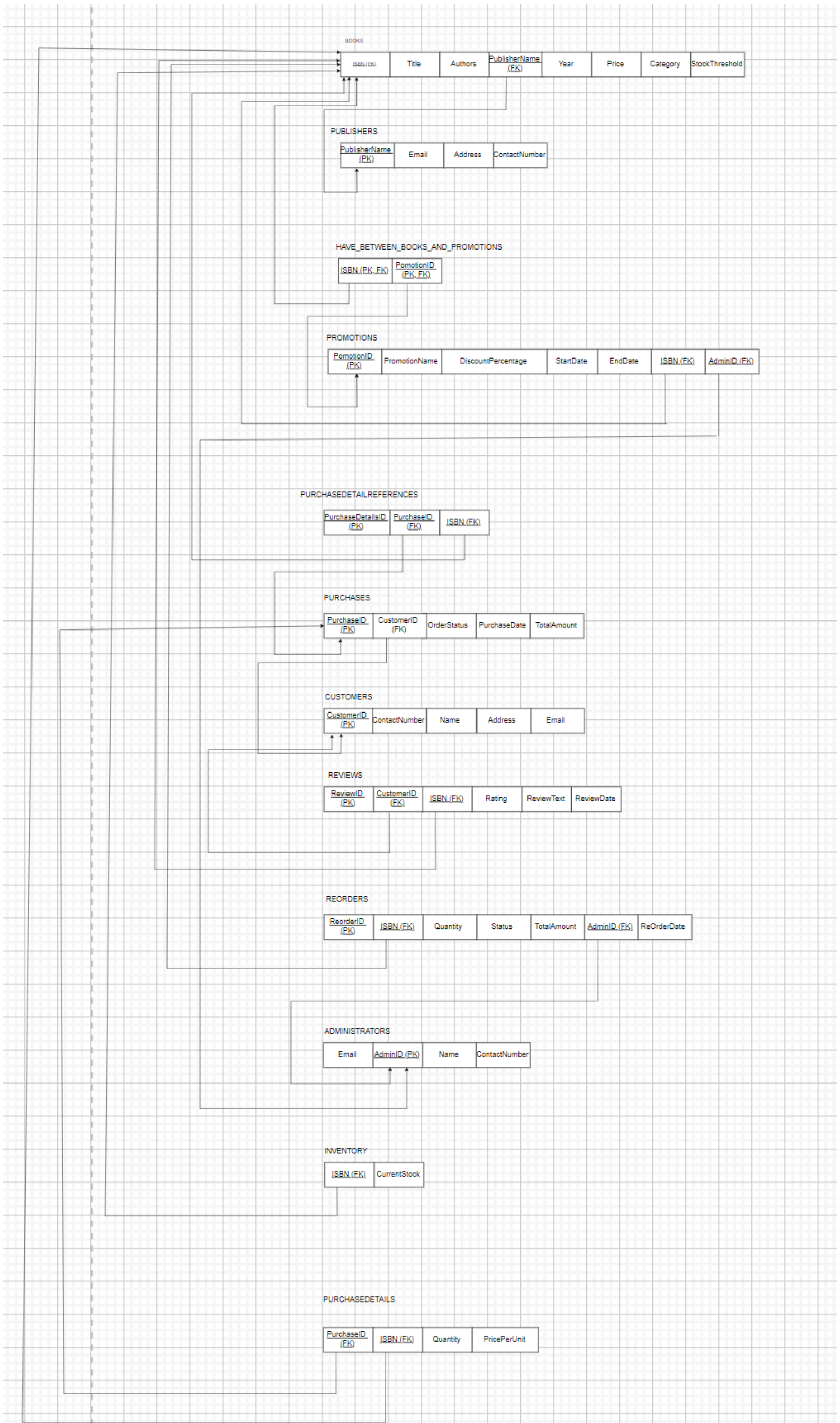
Part I – The Final Report

Section 1 - Database Description

1. ER-model



2. Relational schema


BOOKS
ISBN (PK)
Title
Authors
PublisherName (FK)
Year
Price
Category
StockThreshold
PUBLISHERS
PublisherName (PK)
Email
Address
ContactNumber
HAVE_BETWEEN_BOOKS_AND_PROMOTIONS
ISBN (PK, FK)
PomotionID (PK, FK)
PROMOTIONS
PomotionID (PK)
PromotionName
DiscountPercentage
StartDate
EndDate
ISBN (FK)
AdminID (FK)
PURCHASEDETAILREFERENCES
PurchaseDetailsID (PK)
PurchaseID (FK)
ISBN (FK)
PURCHASES
PurchaseID (PK)
CustomerID (FK)
OrderStatus
PurchaseDate
TotalAmount
CUSTOMERS
CustomerID (PK)
ContactNumber
Name
Address
Email
REVIEWS
ReviewID (PK)
CustomerID (FK)
ISBN (FK)
Rating
ReviewText
ReviewDate
REORDERS
ReorderID (PK)
ISBN (FK)
Quantity
Status
TotalAmount
AdminID (FK)
ReOrderDate
ADMINISTRATORS
Email
AdminID (PK)
Name
ContactNumber
INVENTORY
ISBN (FK)
CurrentStock
PURCHASEDETAILS
PurchaseID (FK)
ISBN (FK)
Quantity
PricePerUnit

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

{ AdminID } → { Email, Name, ContactNumber }

{ Email} → { AdminID, Name, ContactNumber }

2. Publishers(PublisherName, Email, Address, ContactNumber)

{ PublisherName } → { Email, Address, ContactNumber }

{ Email } → { PublisherName, 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 }

{ Email } → { CustomerID, ContactNumber, Name, Address }

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. Description of tables

BOOKS: BCNF
PUBLISHERS: 3NF.
Justification: Decomposing PUBLISHERS into, e.g., R1(Email, PublisherName) and R2(Email, Address, ContactNumber) creates two tables, requiring joins for common queries (e.g., retrieving all publisher details). Since PUBLISHERS is likely small and queried frequently with BOOKS, keeping it as one table avoids join overhead, improving performance.

PROMOTIONS: BCNF
PurchaseDetailReferences: BCNF
PurchaseItemDetails: BCNF
PURCHASES: BCNF
CUSTOMERS: 3NF
Justification: Customers are typically identified by CustomerID in transactions (PURCHASES, REVIEWS). Using Email as a key or decomposing the table (e.g.,

R1(Email, CustomerID), R2(Email, ContactNumber, Name, Address)) complicates queries for customer details, requiring joins for common operations like displaying a customer’s profile. A single table ensures all attributes are accessible without joins, enhancing user experience.

REVIEWS: BCNF
REORDERS: BCNF
ADMINISTRATORS: 3NF
Justification: The ADMINISTRATORS table is likely very small (few admins in a bookstore system), so the overhead of decomposition (e.g., R1(Email, AdminID), R2(Email, Name, ContactNumber)) outweighs benefits. A single table minimizes maintenance and query complexity.

INVENTORY: BCNF

4. Description of indexes

We created indexes to improve query performance, especially on columns that appear frequently in joins or WHERE clauses. Below is a list of our indexes and the rationale for each:

1. Customers(CustomerID): Used to join with Purchases and Reviews tables to retrieve customer-related data quickly.
2. Books(ISBN): Appears in many relationships (e.g., PurchaseDetails, Promotions, Reviews) and filters, making it crucial for joins.
3. Purchases(PurchaseID, CustomerID): These keys support join operations with PurchaseDetails and user history queries.
4. PurchaseDetails(PurchaseID, ISBN): Indexes on these columns enhance join speed for order breakdowns and reporting.
5. Promotions(ISBN): Used to filter books that are currently on sale.
6. Reviews(CustomerID, ISBN): These indexes help find reviews per user and per book, improving filtering and update operations.
7. Reorders(ISBN, AdminID): Supports efficient access to reorder history based on books or administrators.

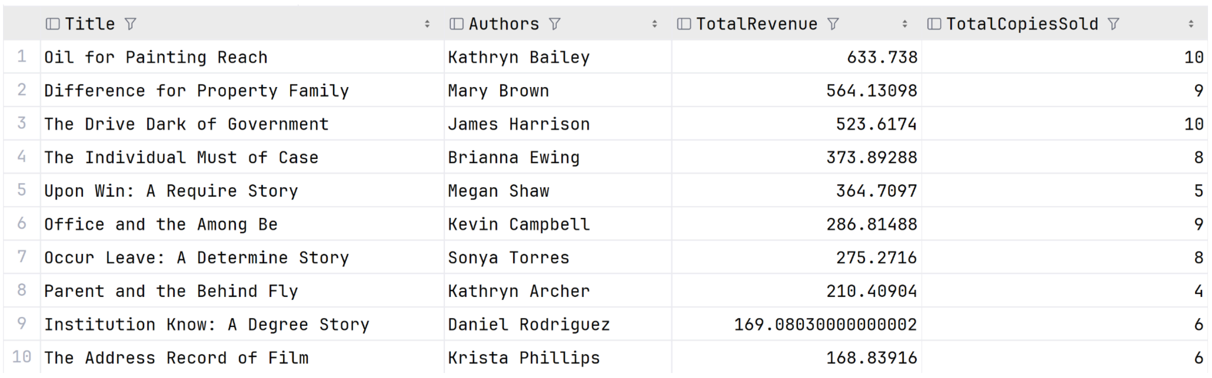
5. View

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.

```
CREATE VIEW TopBooksByRevenue AS
SELECT B.Title, B.Authors, SUM(PID.Quantity *
PID.PricePerUnit) AS TotalRevenue, SUM(PID.Quantity) AS
TotalCopiesSold
FROM Books B
JOIN PurchaseDetailReferences ON B.ISBN =
PurchaseDetailReferences.ISBN
JOIN PurchaseItemDetails PID ON
PurchaseDetailReferences.PurchaseID = PID.PurchaseID and
PurchaseDetailReferences.ISBN = PID.ISBN
GROUP BY B.Title, B.Authors
ORDER BY TotalRevenue DESC;
```

Sample Output:



	Title	Authors	TotalRevenue	TotalCopiesSold
1	Oil for Painting Reach	Kathryn Bailey	633.738	10
2	Difference for Property Family	Mary Brown	564.13098	9
3	The Drive Dark of Government	James Harrison	523.6174	10
4	The Individual Must of Case	Brianna Ewing	373.89288	8
5	Upon Win: A Require Story	Megan Shaw	364.7097	5
6	Office and the Among Be	Kevin Campbell	286.81488	9
7	Occur Leave: A Determine Story	Sonya Torres	275.2716	8
8	Parent and the Behind Fly	Kathryn Archer	210.40904	4
9	Institution Know: A Degree Story	Daniel Rodriguez	169.08030000000002	6
10	The Address Record of Film	Krista Phillips	168.83916	6

Relational Algebra

T1=BOOKS ⋈ BOOKS.ISBN = PurchaseDetailReferences.ISBN PurchaseDetailReferences

T2=T1⋈T1.PurchaseID = PurchaseItemDetails.PurchaseID ∧ T1.PurchaseDetailReferences.ISBN = PurchaseItemDetails.ISBN PurchaseItemDetails

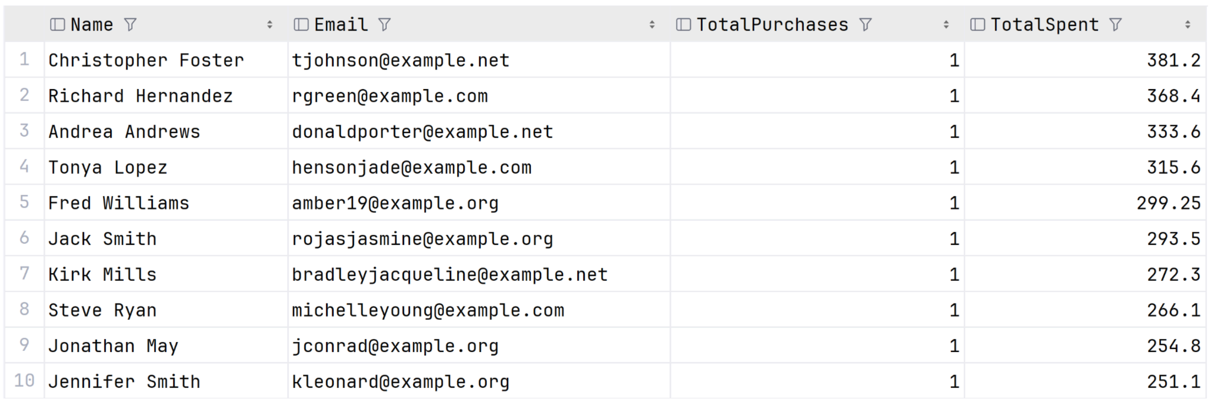
T3=BOOKS.Title, BOOKS.Authors F SUM(Quantity×PricePerUnit)→TotalRevenue,SUM(Quantity)→TotalCopiesSold(T2)

T4=π BOOKS.Title, BOOKS.Authors, TotalRevenue,TotalCopiesSold(T3)

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.

```
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;
```



	Name	Email	TotalPurchases	TotalSpent
1	Christopher Foster	tjohnson@example.net	1	381.2
2	Richard Hernandez	rgreen@example.com	1	368.4
3	Andrea Andrews	donaldporter@example.net	1	333.6
4	Tonya Lopez	hensonjade@example.com	1	315.6
5	Fred Williams	amber19@example.org	1	299.25
6	Jack Smith	rojasjasmine@example.org	1	293.5
7	Kirk Mills	bradleyjacqueline@example.net	1	272.3
8	Steve Ryan	michelleyoung@example.com	1	266.1
9	Jonathan May	jconrad@example.org	1	254.8
10	Jennifer Smith	kleonard@example.org	1	251.1

Relational Algebra

T1 = Customers ⋈ Customers.CustomerID=Purchases.CustomerID Purchases

T2 = Name, Email F COUNT(PurchaseID) as TotalPurchases, SUM(TotalAmount) as TotalSpent T1

T3 = π Name, Email, TotalPurchases, TotalSpent T2

6. Description of three sample transactions

Transaction 1: Processing a New Customer Purchase

```
INSERT INTO PurchaseDetailReferences (PurchaseDetailsID,
PurchaseID, ISBN)
VALUES (2001, 1001, '7625400096352');
```

```
INSERT INTO PurchaseItemDetails (PurchaseID, ISBN,
Quantity, PricePerUnit)
VALUES (1001, '7625400096352', 2, 28.14);

SELECT CurrentStock
FROM INVENTORY
WHERE ISBN = '7625400096352' AND CurrentStock >= 2;

UPDATE INVENTORY
SET CurrentStock = CurrentStock - 2
WHERE ISBN = '7625400096352' AND CurrentStock >= 2;



COMMIT TRANSACTION;
```

Description: This transaction represents the "unit of work" for a customer placing a new order for books. It involves inserting a new purchase record into the PURCHASES table, adding the purchased items into PurchaseDetailReferences and PurchaseItemDetails, showing the current stock of the specific books and updating the inventory stock in the INVENTORY table.

Transaction 2: Modifying an Existing Order

```
BEGIN TRANSACTION;

-- Check if sufficient stock exists for the additional
quantity (1 unit)
SELECT CurrentStock
FROM INVENTORY
WHERE ISBN = '7625400096352' AND CurrentStock >= 1;

-- Update inventory to reserve additional stock (reduce by
1)
UPDATE INVENTORY
SET CurrentStock = CurrentStock - 1
```

```
WHERE ISBN = '7625400096352' AND CurrentStock >= 1;

-- Update PurchaseItemDetails with new quantity and price
UPDATE PurchaseItemDetails
SET Quantity = 3,
   PricePerUnit = 28.14
WHERE PurchaseID = 1001 AND ISBN = '7625400096352';

-- Update PURCHASES total amount based on all items
UPDATE PURCHASES
SET TotalAmount = (
   SELECT SUM(Quantity * PricePerUnit)
   FROM PurchaseItemDetails
   WHERE PurchaseID = 1001
)
WHERE PurchaseID = 1001;

COMMIT TRANSACTION;
```

Description: This transaction represents the "unit of work" for a customer modifying an existing order before it is shipped (e.g., changing the quantity of a book or adding a new book). It involves updating the PurchaseItemDetails table to adjust quantities, adding new items if needed, updating the PURCHASES table's TotalAmount, and adjusting the INVENTORY table to reflect stock changes. This ensures that the order and inventory remain consistent, preventing issues like incorrect billing or stock discrepancies.

Transaction 3: Applying a New Promotion to a Book

```
BEGIN TRANSACTION;

-- Insert a new promotion (relies on foreign key constraint
to validate ISBN)
```

```sql
INSERT INTO PROMOTIONS (PromotionID, PromotionName,
DiscountPercentage, StartDate, EndDate, ISBN, AdminID)
VALUES (3001, 'Spring Sale', 15.00, '2025-04-15',
'2025-04-30', '5091014685363', 1);

COMMIT TRANSACTION;
```

Description: This transaction represents the "unit of work" for an administrator creating a new promotion for a specific book. It involves inserting a new record into the PROMOTIONS table and ensuring the book exists in the BOOKS table. The transaction ensures that the promotion is only applied if the book is valid, preventing orphaned promotion records.