Section 2 - User Manual

UserManual.pdf

Section 1: Table Descriptions

1. BOOKS
Description:
Stores all information about books sold in the store. Each book has a unique ISBN and is associated with a publisher. It includes metadata like title, authors, publication year, price, and inventory-related attributes.

Key Attributes:

ISBN – VARCHAR(13), Primary Key

PublisherName – VARCHAR(100), Foreign Key → PUBLISHERS

Title, Authors – NOT NULL

Price – DECIMAL(10, 2)

StockThreshold – used for triggering reorder alerts

2. PUBLISHERS
Description:
Represents publishing companies responsible for the books sold. Contains contact and location information.

Key Attributes:

PublisherName – VARCHAR(100), Primary Key

Email, Address, ContactNumber – optional contact fields

3. PROMOTIONS
Description:
Represents discount campaigns associated with specific books and created by administrators. Tracks active date ranges and discount percentage.

Key Attributes:

PromotionID – INT, Primary Key

ISBN – Foreign Key → BOOKS

AdminID – Foreign Key → ADMINISTRATORS

DiscountPercentage, StartDate, EndDate – NOT NULL

4. HAVE_BETWEEN_BOOKS_AND_PROMOTION
Description:
A bridge table for a many-to-many relationship between books and promotions.

Key Attributes:

ISBN, PromotionID – Composite Primary Key

Both are Foreign Keys

5. PURCHASEDETAILS
Description:
Stores line-item breakdowns for each purchase, including quantity and unit price of books purchased.

Key Attributes:

PurchaseDetailsID – INT, Primary Key

PurchaseID – FK → PURCHASES

ISBN – FK → BOOKS

Quantity, PricePerUnit – NOT NULL

6. PURCHASES
Description:
Records purchases made by customers, including date, status, and total amount spent.

Key Attributes:

PurchaseID – INT, Primary Key

CustomerID – FK → CUSTOMERS

OrderStatus, PurchaseDate, TotalAmount – NOT NULL

7. CUSTOMERS
Description:
Stores customer information including contact info and delivery address.

Key Attributes:

CustomerID – INT, Primary Key

Name, Email – NOT NULL

ContactNumber, Address – optional

8. REVIEWS
Description:
Captures customer feedback and ratings on purchased books.

Key Attributes:

ReviewID – INT, Primary Key

CustomerID – FK → CUSTOMERS

ISBN – FK → BOOKS

Rating – INT, CHECK (1 ≤ Rating ≤ 5)

ReviewDate – NOT NULL

9. REORDERS
Description:
Tracks inventory restocking requests submitted by administrators. Includes reorder quantity, amount, and status.

Key Attributes:

ReorderID – INT, Primary Key

ISBN – FK → BOOKS

AdminID – FK → ADMINISTRATORS

Quantity, TotalAmount, Status, ReOrderDate – NOT NULL

10. ADMINISTRATORS
Description:
Stores admin staff who manage orders, inventory, and promotions.

Key Attributes:

AdminID – INT, Primary Key

Email, Name – NOT NULL

ContactNumber – optional

11. INVENTORY
Description:
Tracks current stock level for each book. Updated by purchases and reorders.

Key Attributes:

ISBN – VARCHAR(13), Primary Key, FK → BOOKS

CurrentStock – INT, CHECK (CurrentStock ≥ 0), NOT NULL

//////////////////
Section 2: Sample SQL Queries

I.basic queries

Query A – Books by Pratchett under $10
English:
Find the titles of all books written by "Pratchett" that cost less than $10.

Relational Algebra:
π_Title(σ_Authors = 'Pratchett' ∧ Price < 10 (BOOKS))

SQL:
```
SELECT Title
FROM BOOKS
WHERE Authors = 'Pratchett' AND Price < 10;
```

Query B – Titles and Purchase Dates by Customer 1
English:
Show the titles and their purchase dates for purchases made by Customer ID = 1.

Relational Algebra:
π_Title, PurchaseDate(σ_CustomerID = 1(CUSTOMERS ⋈ PURCHASES ⋈ PURCHASEDETAILS ⋈ BOOKS))

SQL:
```
SELECT Title, PurchaseDate
FROM CUSTOMERS
JOIN PURCHASES P ON CUSTOMERS.CustomerID = P.CustomerID
JOIN PURCHASEDETAILS P2 ON P.PurchaseID = P2.PurchaseID
JOIN BOOKS B ON P2.ISBN = B.ISBN
WHERE P.CustomerID = 1;
```

Query C – Books with Less Than 5 in Stock
English:
Find the titles and ISBNs for all books that have fewer than 5 copies in inventory.

Relational Algebra:
π_Title, ISBN(σ_CurrentStock < 5(BOOKS ⋈ INVENTORY))

SQL:
```
SELECT Title, BOOKS.ISBN
FROM BOOKS
JOIN INVENTORY I ON BOOKS.ISBN = I.ISBN
WHERE CurrentStock < 5;
```

Query D – Customers Who Purchased Pratchett Books
English:
Get names of all customers who purchased a book by Pratchett and the titles of those books.

Relational Algebra:
π_Name, Title(σ_Authors = 'Pratchett'(CUSTOMERS ⋈ PURCHASES ⋈ PURCHASEDETAILS ⋈ BOOKS))

SQL:
```
SELECT CUSTOMERS.Name, B.Title
FROM CUSTOMERS
JOIN PURCHASES P ON CUSTOMERS.CustomerID = P.CustomerID
JOIN PURCHASEDETAILS P2 ON P.PurchaseID = P2.PurchaseID
JOIN BOOKS B ON P2.ISBN = B.ISBN
WHERE B.Authors = 'Pratchett';
```

Query E – Total Number of Books Purchased by Customer 1
English:
Calculate how many books in total were purchased by Customer ID = 1.

Relational Algebra:
γ_CustomerID; SUM(Quantity)(σ_CustomerID = 1(CUSTOMERS ⋈ PURCHASES ⋈ PURCHASEDETAILS))

SQL:
```
SELECT SUM(Quantity), CUSTOMERS.Name
FROM CUSTOMERS
JOIN PURCHASES P ON CUSTOMERS.CustomerID = P.CustomerID
JOIN PURCHASEDETAILS P2 ON P.PurchaseID = P2.PurchaseID
WHERE CUSTOMERS.CustomerID = 1;
```

Query F – Customer Who Purchased the Most Books
English:
Find the customer who purchased the highest number of books and display the total.

Relational Algebra:
γ_CustomerID; SUM(Quantity) → TotalBooks(CUSTOMERS ⋈ PURCHASES ⋈ PURCHASEDETAILS)
→ sort by TotalBooks descending
→ top 1

SQL:
```
SELECT CUSTOMERS.Name, SUM(P2.Quantity) AS TotalBooks
FROM CUSTOMERS
JOIN PURCHASES P ON CUSTOMERS.CustomerID = P.CustomerID
JOIN PURCHASEDETAILS P2 ON P.PurchaseID = P2.PurchaseID
GROUP BY CUSTOMERS.CustomerID
ORDER BY TotalBooks DESC
LIMIT 1;
```

II. extra queries

Query 1 – Customers Who Gave 5-Star Reviews
English:
Find the names of all customers who have given a 5-star rating for any book.

Relational Algebra:

π_Name (σ_Rating = 5 (CUSTOMERS ⋈ REVIEWS))

SQL:
```
SELECT DISTINCT C.Name
FROM CUSTOMERS C
JOIN REVIEWS R ON C.CustomerID = R.CustomerID
WHERE R.Rating = 5;
```

Query 2 – Number of Purchases per Customer
English:
Calculate the total number of purchases made by each customer.

Relational Algebra:
γ_CustomerID, COUNT(PurchaseID) → TotalPurchases (PURCHASES)

SQL:
```
SELECT C.Name, COUNT(P.PurchaseID) AS TotalPurchases
FROM CUSTOMERS C
JOIN PURCHASES P ON C.CustomerID = P.CustomerID
GROUP BY C.CustomerID;
```

Query 3 – Average Rating for Books with ≥ 2 Reviews
English:
Retrieve the average rating for each book, but only include books that have at least 2 reviews.

Relational Algebra:
γ_ISBN; AVG(Rating)(σ_COUNT(ReviewID) ≥ 2 (REVIEWS))

SQL:
```
SELECT B.Title, AVG(R.Rating) AS AvgRating
FROM BOOKS B
JOIN REVIEWS R ON B.ISBN = R.ISBN
GROUP BY B.ISBN
HAVING COUNT(R.ReviewID) >= 2;
```

III. Advanced queries

Query A – Total Amount Spent by Each Customer
English:
List each customer along with the total dollar amount they have spent on purchases.

Relational Algebra:
γ_CustomerID; SUM(TotalAmount) (CUSTOMERS ⋈ PURCHASES)

SQL:

```
SELECT C.Name, SUM(P.TotalAmount) AS TotalSpent
FROM CUSTOMERS C
JOIN PURCHASES P ON C.CustomerID = P.CustomerID
GROUP BY C.CustomerID;
```

Query B – Customers Who Spent More Than the Average
English:
Retrieve names and emails of customers who have spent more than the average customer.

SQL:

```
SELECT C.Name, C.Email
FROM CUSTOMERS C
JOIN PURCHASES P ON C.CustomerID = P.CustomerID
GROUP BY C.CustomerID
HAVING SUM(P.TotalAmount) > (
  SELECT AVG(TotalSpent)
  FROM (
    SELECT SUM(TotalAmount) AS TotalSpent
    FROM PURCHASES
    GROUP BY CustomerID
  )
);
```

Query C – Total Copies Sold Per Book (Descending)
English:
List each book title and the total number of copies sold, ordered from most to least sold.

Relational Algebra:
γ_ISBN; SUM(Quantity)(BOOKS ⋈ PURCHASEDETAILS)

SQL:

```
SELECT B.Title, SUM(PD.Quantity) AS TotalSold
FROM BOOKS B
JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
GROUP BY B.ISBN
```

ORDER BY TotalSold DESC;

Query D – Total Revenue by Book Title
English:
Show each book title and total revenue generated, ordered from highest to lowest.

SQL:
SELECT B.Title, SUM(PD.Quantity * PD.PricePerUnit) AS TotalRevenue
FROM BOOKS B
JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
GROUP BY B.ISBN
ORDER BY TotalRevenue DESC;

Query E – Most Popular Author (by Quantity Sold)
English:
Find the author who has sold the most books in terms of quantity.

SQL:
SELECT B.Authors, SUM(PD.Quantity) AS TotalSold
FROM BOOKS B
JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
GROUP BY B.Authors
ORDER BY TotalSold DESC
LIMIT 1;

Query F – Most Profitable Author
English:
Identify the author who generated the most revenue from sales.

SQL:
SELECT B.Authors, SUM(PD.Quantity * PD.PricePerUnit) AS TotalProfit
FROM BOOKS B
JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
GROUP BY B.Authors
ORDER BY TotalProfit DESC
LIMIT 1;
Query G – Customers Who Bought from the Most Profitable Author
English:
Show all customers who purchased books written by the most profitable author.

SQL:

```
WITH MostProfitable AS (
  SELECT B.Authors
  FROM BOOKS B
  JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
  GROUP BY B.Authors
  ORDER BY SUM(PD.Quantity * PD.PricePerUnit) DESC
  LIMIT 1
)
SELECT DISTINCT C.*
FROM CUSTOMERS C
JOIN PURCHASES P ON C.CustomerID = P.CustomerID
JOIN PURCHASEDETAILS PD ON P.PurchaseID = PD.PurchaseID
JOIN BOOKS B ON PD.ISBN = B.ISBN
JOIN MostProfitable M ON B.Authors = M.Authors;
```

Query H – Authors Purchased by High-Spending Customers
English:
List authors whose books were purchased by customers who spent more than the average total.

SQL:

```
WITH HighSpenders AS (
  SELECT CustomerID
  FROM PURCHASES
  GROUP BY CustomerID
  HAVING SUM(TotalAmount) > (
    SELECT AVG(TotalAmount)
    FROM PURCHASES
    GROUP BY CustomerID
  )
)
SELECT DISTINCT B.Authors
FROM BOOKS B
JOIN PURCHASEDETAILS PD ON B.ISBN = PD.ISBN
JOIN PURCHASES P ON PD.PurchaseID = P.PurchaseID
WHERE P.CustomerID IN (SELECT CustomerID FROM HighSpenders);
```

////////
Section 3: Insert and Delete Syntax

(note: ! used to explain foreign key dependencies)

I. insert examples:

1. Insert a Customer and a Purchase

❗ Insert Customer before Purchase due to FK constraint.

INSERT INTO CUSTOMERS (CustomerID, ContactNumber, Name, Address, Email)
VALUES (1, '206-880-5742', 'Rebecca Mills', 'PSC 6341, Box 4009, APO AE 75217', 'jonathanjohnson@example.com');

INSERT INTO PURCHASES (PurchaseID, CustomerID, OrderStatus, PurchaseDate, TotalAmount)
VALUES (1, 1, 'Delivered', '2026-09-30', 227.5);

2. Insert a Purchase Item (Detail)

❗ Purchase must exist before inserting details.

INSERT INTO PURCHASEDETAILS (PurchaseDetailsID, PurchaseID, ISBN, Quantity, PricePerUnit)
VALUES (1, 1, '3626767700882', 8, 46.73661);

3. Insert a Review

❗ Book and Customer must exist before inserting a review.

INSERT INTO REVIEWS (ReviewID, CustomerID, ISBN, Rating, ReviewText, ReviewDate)
VALUES (1, 1, '3626767700882', 5, 'Fantastic book! Must read.', '2026-01-31');

II. DELETE examples:

1. Delete a Customer and their Reviews

❗ Must delete child records (Reviews) before deleting parent (Customer) unless ON DELETE CASCADE is defined.

DELETE FROM REVIEWS WHERE CustomerID = 1;
DELETE FROM CUSTOMERS WHERE CustomerID = 1;

2. Delete a Book with Dependencies

❗ Delete from PURCHASEDETAILS and REVIEWS before deleting the Book.

DELETE FROM PURCHASEDETAILS WHERE ISBN = '3626767700882';
DELETE FROM REVIEWS WHERE ISBN = '3626767700882';

```
DELETE FROM BOOKS WHERE ISBN = '3626767700882';
```

3. Delete a Publisher and All Related Books

❗ All related books must be deleted first due to FK in BOOKS.

```
DELETE FROM BOOKS WHERE PublisherName = 'Perez LLC';
DELETE FROM PUBLISHERS WHERE PublisherName = 'Perez LLC';
```

4. Delete a Promotion

❗ Ensure no FK dependency exists in linking tables.

```
DELETE FROM PROMOTIONS WHERE PromotionID = 1;
```

Section 3 - Graded Checkpoint Documents