User Manual

CSE 3241

Elijah Paulman, Christian Coulibaly, Rohit Navaneetha, Kyle Roessle

This manual is intended for developers and technical stakeholders integrating with the Bits & Books database system. It documents the structure, behavior, and sample interactions with all entities within the system.

Table of Contents

Table Description

1. category

Represents: Book classification categories (e.g., Fiction, Science, Romance).

Attributes:

- CategoryID – Text, Primary Key. Unique identifier for each category.
- CategoryName – Text, Not Null. Descriptive name of the category.

---

2. publisher

Represents: Publishing companies that produce books.

Attributes:

- PublisherID – Text, Primary Key. Unique ID for the publisher.
- Name – Text, Not Null. Full name of the publisher.
- Address – Text. Optional physical or mailing address.
- ContactInfo – Text. Optional contact details such as email or phone.

---

3. author

Represents: Writers or contributors of books.

Attributes:

- AuthorID – Text, Primary Key. Unique identifier for each author.
- Name – Text, Not Null. Full name of the author.

---

4. book

Represents: Individual books listed in the catalog.

Attributes:

- ISBN – Text, Primary Key. International Standard Book Number.
- Title – Text, Not Null. Title of the book.
- Year – Integer. Year of publication.

- Price – Decimal(10,2). Retail price of the book.
- PublisherID – Text, Foreign Key to publisher. Nullable.
- CategoryID – Text, Foreign Key to category. Nullable.

Constraints:

- On deletion of a referenced publisher or category, the field is set to NULL.

---

5. book_author

Represents: Many-to-many relationship linking books with their authors.

Attributes:

- ISBN – Text, Foreign Key to book.
- AuthorID – Text, Foreign Key to author.

Constraints:

- Composite primary key (ISBN, AuthorID).
- On deletion of a book or author, the corresponding relationship entry is also deleted (cascade).

---

6. customer

Represents: Registered customers who place orders.

Attributes:

- CustomerID – Text, Primary Key. Unique identifier for the customer.
- Name – Text, Not Null. Full name of the customer.
- Address – Text. Optional mailing or billing address.

---

7. customer_contact

Represents: Contact details for each customer.

Attributes:

- CustomerID – Text, Primary Key and Foreign Key to customer.
- Email – Text, Unique. Customer’s email address.

- PhoneNumber – Text, Unique. Customer’s phone number.

---

8. customer_order

Represents: Orders placed by customers.

Attributes:

- OrderID – Text, Primary Key. Unique order identifier.
- CustomerID – Text, Not Null. Foreign Key to customer.
- OrderDate – Date, Not Null. Defaults to current date.

Constraints:

- On deletion of a customer, all their orders are also deleted (cascade).

---

9. orderItem

Represents: Items (books) included in customer orders.

Attributes:

- OrderItemID – Text, Primary Key. Unique identifier for the order item.
- OrderID – Text, Not Null. Foreign Key to customer_order.
- ISBN – Text, Not Null. Foreign Key to book.
- Quantity – Integer, Not Null. Must be greater than 0.
- Price – Decimal(10,2), Not Null. Price at the time of purchase.

---

10. inventory

Represents: Tracking of available stock for each book.

Attributes:

- ISBN – Text, Primary Key and Foreign Key to book.
- StockQuantity – Integer, Not Null. Must be zero or positive.

---

11. bookDemand

Represents: Popularity or demand of a book based on sales or other metrics.

Attributes:

- ISBN – Text, Primary Key and Foreign Key to book.
- Popularity – Integer, Not Null. Must be zero or greater.

---

12. profitMargin

Represents: Financial metrics for each book including revenue and costs.

Attributes:

- ISBN – Text, Primary Key and Foreign Key to book.
- SalesTotal – Decimal(10,2). Defaults to 0. Represents total revenue.
- CostTotal – Decimal(10,2). Defaults to 0. Represents total cost.

Sample Checkpoint SQL Queries

1. Find titles of all books by Pratchett under $10

- **Description**: Retrieve all book titles written by authors containing "Pratchett" in their name that are priced below $10.
- **Relational Algebra**:
  π Title (σ Name='Pratchett' $\wedge$ Price < 10 (book $\bowtie$ book_author $\bowtie$ author))
- **SQL**:
  ```
  SELECT b.Title
  FROM book b
  JOIN book_author ba ON b.ISBN = ba.ISBN
  JOIN author a ON ba.AuthorID = a.AuthorID
  WHERE a.Name LIKE '%Pratchett%' AND b.Price < 10;
  ```

---

2. Titles and purchase dates for a specific customer

- **Description**: Retrieve all book titles and their order dates purchased by customer 'CUST001'.
- **Relational Algebra**:
  π Title, OrderDate (σ CustomerID='CUST001' (customer_order $\bowtie$ orderItem $\bowtie$ book))
- **SQL**:
  ```
  SELECT b.Title, co.OrderDate
  FROM customer_order co
  JOIN orderItem oi ON co.OrderID = oi.OrderID
  JOIN book b ON oi.ISBN = b.ISBN
  WHERE co.CustomerID = 'CUST001';
  ```

---

3. Books with low stock (<5 copies)

- **Description**: Find all book titles and ISBNs that have fewer than 5 copies in inventory.

- **Relational Algebra**:
  π Title, ISBN (σ StockQuantity < 5 (inventory ⋈ book))
- **SQL**:
  SELECT b.Title, b.ISBN
  FROM book b
  JOIN inventory i ON b.ISBN = i.ISBN
  WHERE i.StockQuantity < 5;

---

4. Customers who purchased Pratchett books

- **Description**: List all customer names along with the titles of Pratchett books they purchased.
- **Relational Algebra**:
  π Name, Title (σ Name='Pratchett' (customer ⋈ customer_order ⋈ orderItem ⋈ book ⋈ book_author ⋈ author))
- **SQL**:
  SELECT c.Name, b.Title
  FROM customer c
  JOIN customer_order co ON c.CustomerID = co.CustomerID
  JOIN orderItem oi ON co.OrderID = oi.OrderID
  JOIN book b ON oi.ISBN = b.ISBN
  JOIN book_author ba ON b.ISBN = ba.ISBN
  JOIN author a ON ba.AuthorID = a.AuthorID
  WHERE a.Name LIKE '%Pratchett%';

---

5. Total books purchased by a specific customer

- **Description**: Calculate the total number of books purchased by customer 'CUST001'.

- **Relational Algebra**:
  γ CustomerID, SUM(Quantity)→Total (σ CustomerID='CUST001' (customer_order ⋈ orderItem))
- **SQL**:
  SELECT SUM(oi.Quantity) AS TotalBooks
  FROM customer_order co
  JOIN orderItem oi ON co.OrderID = oi.OrderID
  WHERE co.CustomerID = 'CUST001';

---

6. Customer who purchased the most books

- **Description**: Identify the customer who purchased the highest quantity of books and their total purchases.
- **Relational Algebra**:
  π Name, MAX(TotalBooks) (γ CustomerID, SUM(Quantity)→TotalBooks (customer_order ⋈ orderItem) ⋈ customer)
- **SQL**:
  SELECT c.Name, SUM(oi.Quantity) AS TotalBooks
  FROM customer c
  JOIN customer_order co ON c.CustomerID = co.CustomerID
  JOIN orderItem oi ON co.OrderID = oi.OrderID
  GROUP BY c.CustomerID, c.Name
  ORDER BY TotalBooks DESC
  LIMIT 1;

---

7. Customer spending report

- **Description**: List all customers with the total dollar amount each has spent.

- **Relational Algebra**:

  π Name, SUM(Price×Quantity)→TotalSpent (customer ⋈ customer_order ⋈ orderItem)
- **SQL**:

  SELECT c.Name, SUM(oi.Quantity * oi.Price) AS TotalSpent

  FROM customer c

  JOIN customer_order co ON c.CustomerID = co.CustomerID

  JOIN orderItem oi ON co.OrderID = oi.OrderID

  GROUP BY c.CustomerID, c.Name;

---

8. Customers spending above average

- **Description**: List customers and their email addresses who spent more than the average customer.
- **Relational Algebra**:

  π Name, Email (σ TotalSpent > AVG(TotalSpent) (γ CustomerID, SUM(Price×Quantity)→TotalSpent (customer ⋈ customer_order ⋈ orderItem) ⋈ customer_contact))
- **SQL**:

  WITH CustomerTotals AS (

  SELECT c.CustomerID, SUM(oi.Quantity * oi.Price) AS TotalSpent

  FROM customer c

  JOIN customer_order co ON c.CustomerID = co.CustomerID

  JOIN orderItem oi ON co.OrderID = oi.OrderID

  GROUP BY c.CustomerID

)

  SELECT c.Name, cc.Email

  FROM customer c

  JOIN customer_contact cc ON c.CustomerID = cc.CustomerID

```
JOIN CustomerTotals ct ON c.CustomerID = ct.CustomerID
WHERE ct.TotalSpent > (SELECT AVG(TotalSpent) FROM CustomerTotals);
```

---

9. Books by copies sold (descending)

- **Description**: List all book titles with total copies sold, sorted from highest to lowest.
- **Relational Algebra**:

  π Title, SUM(Quantity)→TotalSold (book ⋈ orderItem)

  ORDER BY TotalSold DESC
- **SQL**:

```
SELECT b.Title, SUM(oi.Quantity) AS TotalSold
FROM book b
JOIN orderItem oi ON b.ISBN = oi.ISBN
GROUP BY b.ISBN, b.Title
ORDER BY TotalSold DESC;
```

---

10. Books by sales revenue (descending)

- **Description**: List all book titles with total sales revenue, sorted from highest to lowest.
- **Relational Algebra**:

  π Title, SUM(Price×Quantity)→TotalRevenue (book ⋈ orderItem)

  ORDER BY TotalRevenue DESC
- **SQL**:

```
SELECT b.Title, SUM(oi.Quantity * oi.Price) AS TotalRevenue
FROM book b
JOIN orderItem oi ON b.ISBN = oi.ISBN
GROUP BY b.ISBN, b.Title
ORDER BY TotalRevenue DESC;
```

---

11. Most popular author by copies sold

- **Description**: Identify the author whose books have sold the most copies.
- **Relational Algebra**:

  π Name, SUM(Quantity)→TotalSold (author ⋈ book_author ⋈ orderItem)

  ORDER BY TotalSold DESC

  LIMIT 1
- **SQL**:

  SELECT a.Name, SUM(oi.Quantity) AS TotalSold

  FROM author a

  JOIN book_author ba ON a.AuthorID = ba.AuthorID

  JOIN orderItem oi ON ba.ISBN = oi.ISBN

  GROUP BY a.AuthorID, a.Name

  ORDER BY TotalSold DESC

  LIMIT 1;

---

12. Most profitable author by revenue

- **Description**: Identify the author whose books have generated the most revenue.
- **Relational Algebra**:

  π Name, SUM(Price×Quantity)→TotalRevenue (author ⋈ book_author ⋈ orderItem)

  ORDER BY TotalRevenue DESC

  LIMIT 1
- **SQL**:

  SELECT a.Name, SUM(oi.Quantity * oi.Price) AS TotalRevenue

  FROM author a

  JOIN book_author ba ON a.AuthorID = ba.AuthorID

  JOIN orderItem oi ON ba.ISBN = oi.ISBN

GROUP BY a.AuthorID, a.Name

ORDER BY TotalRevenue DESC

LIMIT 1;

---

13. Customers of most profitable author

- **Description**: List all customers who purchased books by the most profitable author.
- **Relational Algebra**:

  π Name, Email, PhoneNumber (customer ⋈ customer_contact ⋈ customer_order ⋈ orderItem ⋈ book_author ⋈

  (π AuthorID (γ AuthorID, SUM(Price×Quantity)→TotalRevenue (author ⋈ book_author ⋈ orderItem)

  ORDER BY TotalRevenue DESC

  LIMIT 1))
- **SQL**:

  WITH TopAuthor AS (

  SELECT ba.AuthorID

  FROM book_author ba

  JOIN orderItem oi ON ba.ISBN = oi.ISBN

  GROUP BY ba.AuthorID

  ORDER BY SUM(oi.Quantity * oi.Price) DESC

  LIMIT 1

)

  SELECT DISTINCT c.Name, cc.Email, cc.PhoneNumber

  FROM customer c

  JOIN customer_contact cc ON c.CustomerID = cc.CustomerID

  JOIN customer_order co ON c.CustomerID = co.CustomerID

  JOIN orderItem oi ON co.OrderID = oi.OrderID

```
JOIN book_author ba ON oi.ISBN = ba.ISBN
WHERE ba.AuthorID = (SELECT AuthorID FROM TopAuthor);
```

---

14. Authors popular among big spenders

- **Description**: List authors whose books were purchased by customers who spent more than average.
- **Relational Algebra**:
  π DISTINCT Name (author ⋈ book_author ⋈ orderItem ⋈ customer_order ⋈
  (σ TotalSpent > AVG(TotalSpent) (γ CustomerID, SUM(Price×Quantity)→TotalSpent
  (customer_order ⋈ orderItem))))
- **SQL**:

```
WITH HighSpenders AS (
SELECT co.CustomerID
FROM customer_order co
JOIN orderItem oi ON co.OrderID = oi.OrderID
GROUP BY co.CustomerID
HAVING SUM(oi.Quantity * oi.Price) > (SELECT AVG(Total) FROM (
SELECT SUM(oi.Quantity * oi.Price) AS Total
FROM customer_order co
JOIN orderItem oi ON co.OrderID = oi.OrderID
GROUP BY co.CustomerID
))
)
SELECT DISTINCT a.Name
FROM author a
JOIN book_author ba ON a.AuthorID = ba.AuthorID
JOIN orderItem oi ON ba.ISBN = oi.ISBN
JOIN customer_order co ON oi.OrderID = co.OrderID
JOIN HighSpenders hs ON co.CustomerID = hs.CustomerID;
```

---

15. Most popular book and sales total

- **Description**: Retrieve the title and total sales of the most popular book (popularity = 10).
- **Relational Algebra**:

  π Title, SalesTotal (σ Popularity=10 (bookDemand ⋈ profitMargin ⋈ book))
- **SQL**:

```
SELECT b.Title, pm.SalesTotal
FROM book b
JOIN bookDemand bd ON b.ISBN = bd.ISBN
JOIN profitMargin pm ON b.ISBN = pm.ISBN
WHERE bd.Popularity = 10;
```

---

16. Author of most purchased book

- **Description**: Identify the author of the book with the highest total quantity sold.
- **Relational Algebra**:

  π ISBN, Title, TotalQuantity, Name (σ TotalQuantity=MAX(TotalQuantity) (γ ISBN, SUM(Quantity)→TotalQuantity (orderItem)) ⋈ book ⋈ book_author ⋈ author)
- **SQL**:

```
WITH TotalSales AS (
SELECT ISBN, SUM(Quantity) AS TotalQuantity
FROM orderItem
GROUP BY ISBN
)
SELECT b.ISBN, b.Title, ts.TotalQuantity, a.Name
FROM TotalSales ts
```

```
JOIN book b ON ts.ISBN = b.ISBN
JOIN book_author ba ON b.ISBN = ba.ISBN
JOIN author a ON ba.AuthorID = a.AuthorID
ORDER BY ts.TotalQuantity DESC
LIMIT 1;
```

---

17. Least popular book's profit margin

- **Description**: Find the title and profit margin (sales - cost) of the least popular book.
- **Relational Algebra**:
  π Title, (SalesTotal - CostTotal) (σ Popularity=MIN(Popularity) (bookDemand ⋈ book ⋈ profitMargin))
- **SQL**:

```
SELECT b.Title, (pm.SalesTotal - pm.CostTotal) AS ProfitMargin
FROM book b
JOIN bookDemand bd ON b.ISBN = bd.ISBN
JOIN profitMargin pm ON b.ISBN = pm.ISBN
WHERE bd.Popularity = (SELECT MIN(Popularity) FROM bookDemand);
```

Insert Syntax

Data must be inserted in this order (from new database creation). If data already exists within the table and new data must be added, check the order below to ensure that all constraints are added to ensure no errors.

Insertion Order (Due to Dependencies)

1. category
2. publisher
3. author
4. customer
5. customer_contact
6. book
7. book_author
8. inventory
9. customer_order
10. orderItem
11. bookDemand
12. profitMargin

---

1. Insert into category

Adds five entries representing different book categories.

- CategoryID = 'CAT001', CategoryName = 'Fiction'
- CategoryID = 'CAT002', CategoryName = 'Non-Fiction'
- CategoryID = 'CAT003', CategoryName = 'Science Fiction'
- CategoryID = 'CAT004', CategoryName = 'Biography'
- CategoryID = 'CAT005', CategoryName = 'Self-Help'

SQL:
```
INSERT INTO category (CategoryID, CategoryName)
VALUES
('CAT001', 'Fiction'),
('CAT002', 'Non-Fiction'),
('CAT003', 'Science Fiction'),
('CAT004', 'Biography'),
('CAT005', 'Self-Help');
```

---

2. Insert into publisher

Adds three publishers with name, address, and contact info.

- PUB001: Penguin Random House, 1745 Broadway, NY
- PUB002: HarperCollins, 195 Broadway, NY
- PUB003: Simon & Schuster, 1230 Avenue of the Americas, NY

SQL:

```
 INSERT INTO publisher (PublisherID, Name, Address, ContactInfo)
 VALUES
 ('PUB001', 'Penguin Random House', '1745 Broadway, New York, NY',
'contact@penguinrandomhouse.com'),
 ('PUB002', 'HarperCollins', '195 Broadway, New York, NY', 'info@harpercollins.com'),
 ('PUB003', 'Simon & Schuster', '1230 Avenue of the Americas, New York, NY',
'contact@simonandschuster.com');
```

---

3. Insert into author

Adds five authors.

- AUT001: J.K. Rowling
- AUT002: Stephen King
- AUT003: Michelle Obama
- AUT004: Yuval Noah Harari
- AUT005: George Orwell

SQL:

```
 INSERT INTO author (AuthorID, Name)
 VALUES
 ('AUT001', 'J.K. Rowling'),
 ('AUT002', 'Stephen King'),
 ('AUT003', 'Michelle Obama'),
 ('AUT004', 'Yuval Noah Harari'),
 ('AUT005', 'George Orwell');
```

---

4. Insert into customer

Adds three customers (basic info only).

- CUST001: John Smith, 123 Main St
- CUST002: Jane Doe, 456 Oak Ave

- CUST003: Robert Johnson, NULL address

SQL:
```
INSERT INTO customer (CustomerID, Name, Address)
VALUES
('CUST001', 'John Smith', '123 Main St, Anytown, USA'),
('CUST002', 'Jane Doe', '456 Oak Ave, Somewhere, USA'),
('CUST003', 'Robert Johnson', NULL);
```

---

5. Insert into customer_contact

Links emails and phone numbers to existing customers.

- CUST001: john.smith@email.com, 555-123-4567
- CUST002: jane.doe@email.com, 555-987-6543
- CUST003: robert.j@email.com, 555-456-7890

SQL:
```
INSERT INTO customer_contact (CustomerID, Email, PhoneNumber)
VALUES
('CUST001', 'john.smith@email.com', '555-123-4567'),
('CUST002', 'jane.doe@email.com', '555-987-6543'),
('CUST003', 'robert.j@email.com', '555-456-7890');
```

---

6. Insert into book

Adds five books, referencing publisher and category.

- ISBNs range from Harry Potter to Becoming
- Publishers and categories must already exist

SQL:
```
INSERT INTO book (ISBN, Title, Year, Price, PublisherID, CategoryID)
VALUES
('9780747532743', 'Harry Potter and the Philosopher''s Stone', 1997, 12.99, 'PUB001',
'CAT001'),
('9780061120084', 'To Kill a Mockingbird', 1960, 9.99, 'PUB002', 'CAT001'),
('9780307474278', 'Sapiens: A Brief History of Humankind', 2011, 15.99, 'PUB003', 'CAT002'),
('9780451524935', '1984', 1949, 8.99, 'PUB001', 'CAT003'),
('9781524763138', 'Becoming', 2018, 22.99, 'PUB002', 'CAT004');
```

---

7. Insert into book_author

Creates the many-to-many relationships between books and authors.

- Each book linked to 1 author in this example

SQL:
```
INSERT INTO book_author (ISBN, AuthorID)
VALUES
('9780747532743', 'AUT001'),
('9780061120084', 'AUT002'),
('9780307474278', 'AUT004'),
('9780451524935', 'AUT005'),
('9781524763138', 'AUT003');
```

---

8. Insert into inventory

Tracks stock levels for each book.

SQL:
```
INSERT INTO inventory (ISBN, StockQuantity)
VALUES
('9780747532743', 50),
('9780061120084', 30),
('9780307474278', 25),
('9780451524935', 40),
('9781524763138', 35);
```

---

9. Insert into customer_order

Adds customer orders with fixed dates.

SQL:
```
INSERT INTO customer_order (OrderID, CustomerID, OrderDate)
VALUES
('ORD001', 'CUST001', '2023-05-15'),
('ORD002', 'CUST002', '2023-05-16'),
('ORD003', 'CUST001', '2023-05-17');
```

---

10. Insert into orderItem

Adds specific books to specific orders.

SQL:

```
INSERT INTO orderItem (OrderItemID, OrderID, ISBN, Quantity, Price)
VALUES
('ITEM001', 'ORD001', '9780747532743', 2, 12.99),
('ITEM002', 'ORD001', '9780307474278', 1, 15.99),
('ITEM003', 'ORD002', '9781524763138', 1, 22.99),
('ITEM004', 'ORD003', '9780451524935', 3, 8.99);
```

---

11. Insert into bookDemand

Tracks popularity scores for each book.

SQL:

```
INSERT INTO bookDemand (ISBN, Popularity)
VALUES
('9780747532743', 95),
('9780061120084', 80),
('9780307474278', 85),
('9780451524935', 75),
('9781524763138', 90);
```

---

12. Insert into profitMargin

Tracks total sales and cost for each book.

SQL:

```
INSERT INTO profitMargin (ISBN, SalesTotal, CostTotal)
VALUES
('9780747532743', 1299.00, 800.00),
('9780061120084', 799.20, 500.00),
('9780307474278', 1279.20, 900.00),
('9780451524935', 899.00, 600.00),
('9781524763138', 1149.50, 800.00);
```

Delete Syntax

Deletions must follow a specific order to avoid integrity violations.

Deletion Order (Due to Dependencies)

1. orderItem
2. customer_order
3. customer_contact
4. customer
5. inventory
6. book_author
7. bookDemand
8. profitMargin
9. book
10. author
11. publisher
12. category

---

1. Delete from orderItem

Removes specific items in orders. Must be deleted before customer_order.

Example:
DELETE FROM orderItem WHERE OrderItemID = 'ITEM001';

---

2. Delete from customer_order

Removes full orders. Must be deleted before customer.

Example:
DELETE FROM customer_order WHERE OrderID = 'ORD001';

---

3. Delete from customer_contact

Removes contact details. Must be deleted before customer.

Example:
DELETE FROM customer_contact WHERE CustomerID = 'CUST001';

---

4. Delete from customer

Removes the customer record. Must be after deleting orders and contact info.

Example:
DELETE FROM customer WHERE CustomerID = 'CUST001';

---

5. Delete from inventory

Removes inventory info linked to a book.

Example:
DELETE FROM inventory WHERE ISBN = '9780747532743';

---

6. Delete from book_author

Removes relationships between books and authors.

Example:
DELETE FROM book_author WHERE ISBN = '9780747532743';

---

7. Delete from bookDemand

Removes popularity tracking. Must be removed before deleting books.

Example:
DELETE FROM bookDemand WHERE ISBN = '9780747532743';

---

8. Delete from profitMargin

Removes profit tracking. Must be removed before deleting books.

Example:
DELETE FROM profitMargin WHERE ISBN = '9780747532743';

---

9. Delete from book

Deletes the book record itself. All dependencies must be cleared first.

Example:
DELETE FROM book WHERE ISBN = '9780747532743';

---

10. Delete from author

Deletes the author. Must delete any references in book_author first.

Example:
DELETE FROM author WHERE AuthorID = 'AUT001';

---

11. Delete from publisher

Deletes publisher info. Must be done after removing all associated books.

Example:
DELETE FROM publisher WHERE PublisherID = 'PUB001';

---

12. Delete from category

Deletes the category. Must be done after all associated books are removed.

Example:
DELETE FROM category WHERE CategoryID = 'CAT001';