GROUP ASSIGNMENT

APD1F2303

CT042-3-1-IDB

INTRODUCTION TO DATABASES (PART 2)

HANDOUT DATE: 2023

HAND-IN DATE: 20th NOVEMBER 2023

GROUP: 24

NO	STUDENT NAME	TP NUMBER
1	CHURILOV MIKHAIL	TP072847
2	VOORISHTA GOPAUL	TP073620
3	NASTARAN ESMAEIL ZADEH	TP073210
4	RAVIN A/L KANAGARAJAN	TP068019

Contents

Database Schema....................3
Entity Relationship Diagram....................3
Database Diagram....................4
SQL-Data Definition Language (DDL)....................5
Address Publisher Table....................5
Publishers Table....................6
Book Table....................7
Genres Table....................8
Authors Table....................9
Discounts Table....................10
Address Member Table....................11
Members Table....................12
Order Stock Table....................13
Reviews Table....................14
Carts Table....................15
Cart Item Table....................16
Payments Table....................17
Member Order Table....................18
SQL-Data Manipulation Language (DML)....................19
1. Total Number of Books Published by each publisher....................19
2. Books present in shopping carts without completed payments.....................20
3. Average Rating for Each Book from Reviews....................21
4. Total Feedbacks Received by Each Member....................22
5. Total Books Ordered by Each Publisher....................23
6. Total Quantity of Books Ordered by Each Publisher....................24
7. Genre of Most Stocked Book....................25
8. Total Sold Quantity for Each Book....................26
9. Total Spent by Each Member:....................27
10. Members with No Orders....................28
12. Members with Two or More Orders....................30
Workload Matrix....................31

Database Schema

Entity Relationship Diagram

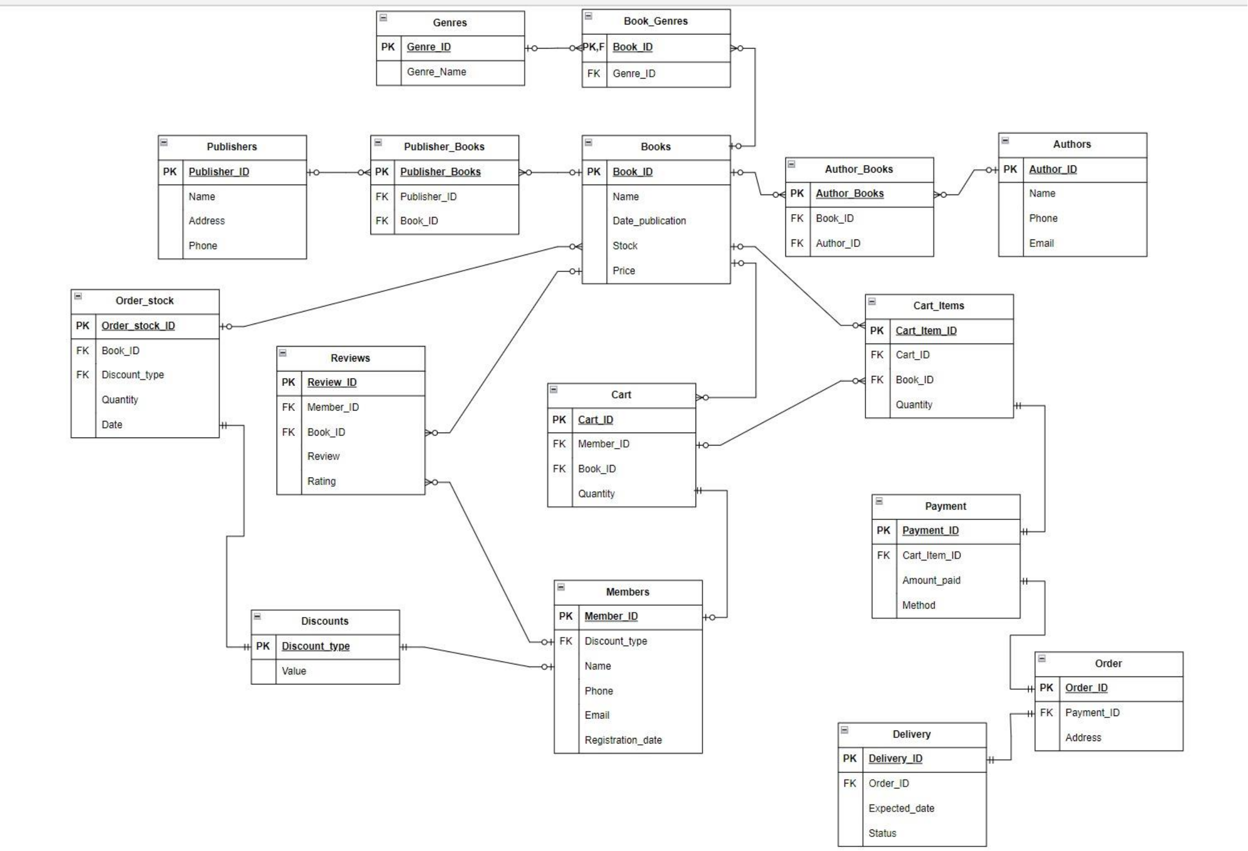


Figure 1: Crow's Foot Notation

Database Diagram

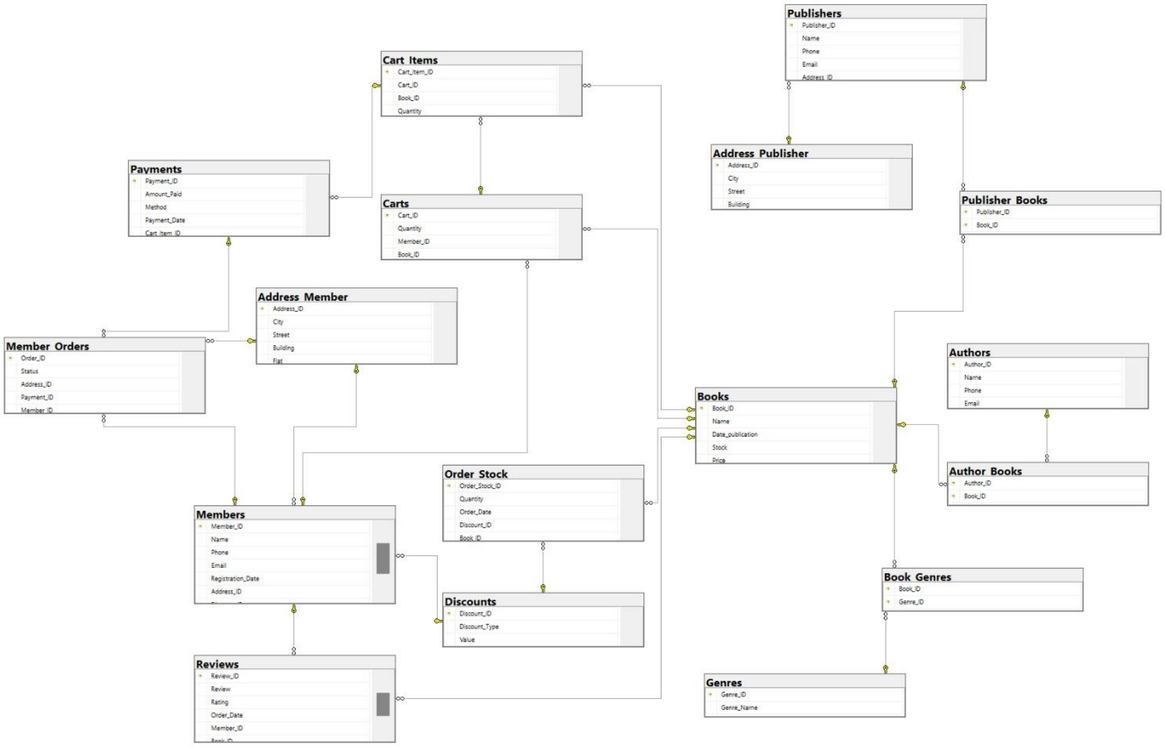
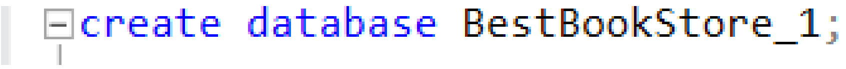


Figure 2: Database Diagram of E-Bookstore Database System

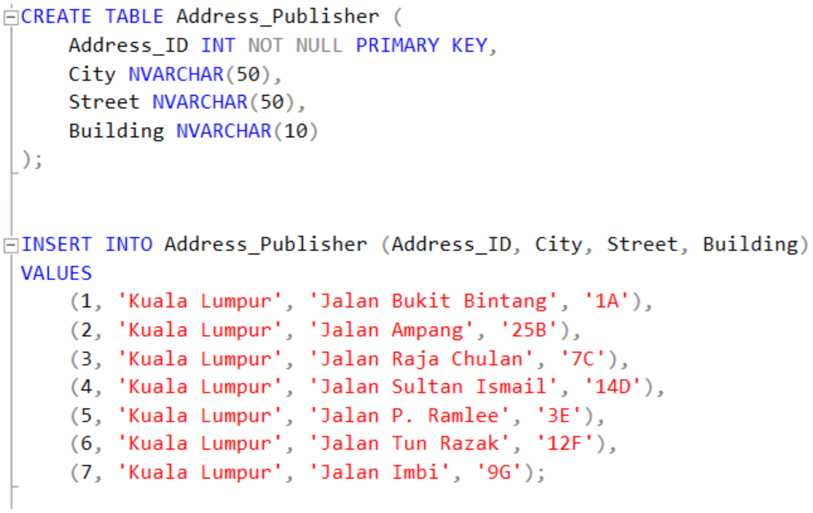
SQL-Data Definition Language (DDL)



```sql
create database BestBookStore_1;
```

Figure 3: Database Syntax to Create E_Bookstore_Database_System Database

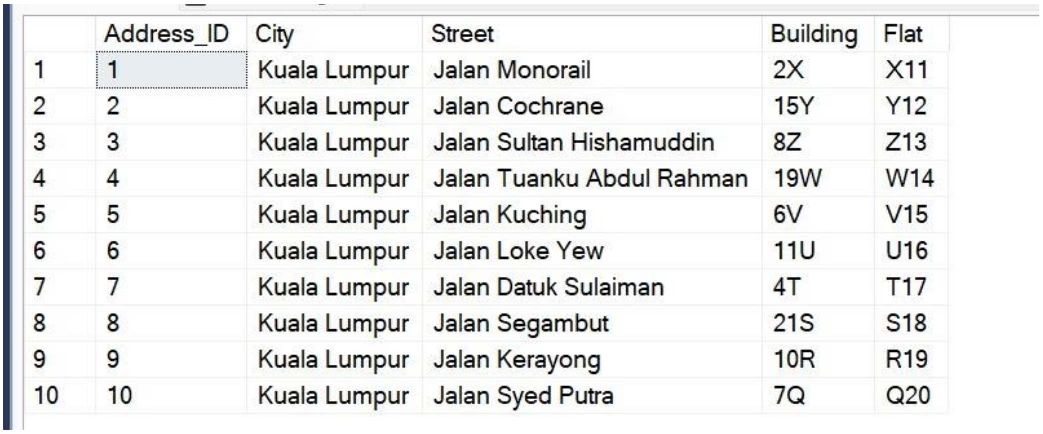
Address Publisher Table



```sql
CREATE TABLE Address_Publisher (
    Address_ID INT NOT NULL PRIMARY KEY,
    City NVARCHAR(50),
    Street NVARCHAR(50),
    Building NVARCHAR(10)
);


INSERT INTO Address_Publisher (Address_ID, City, Street, Building)
VALUES
    (1, 'Kuala Lumpur', 'Jalan Bukit Bintang', '1A'),
    (2, 'Kuala Lumpur', 'Jalan Ampang', '25B'),
    (3, 'Kuala Lumpur', 'Jalan Raja Chulan', '7C'),
    (4, 'Kuala Lumpur', 'Jalan Sultan Ismail', '14D'),
    (5, 'Kuala Lumpur', 'Jalan P. Ramlee', '3E'),
    (6, 'Kuala Lumpur', 'Jalan Tun Razak', '12F'),
    (7, 'Kuala Lumpur', 'Jalan Imbi', '9G');
```

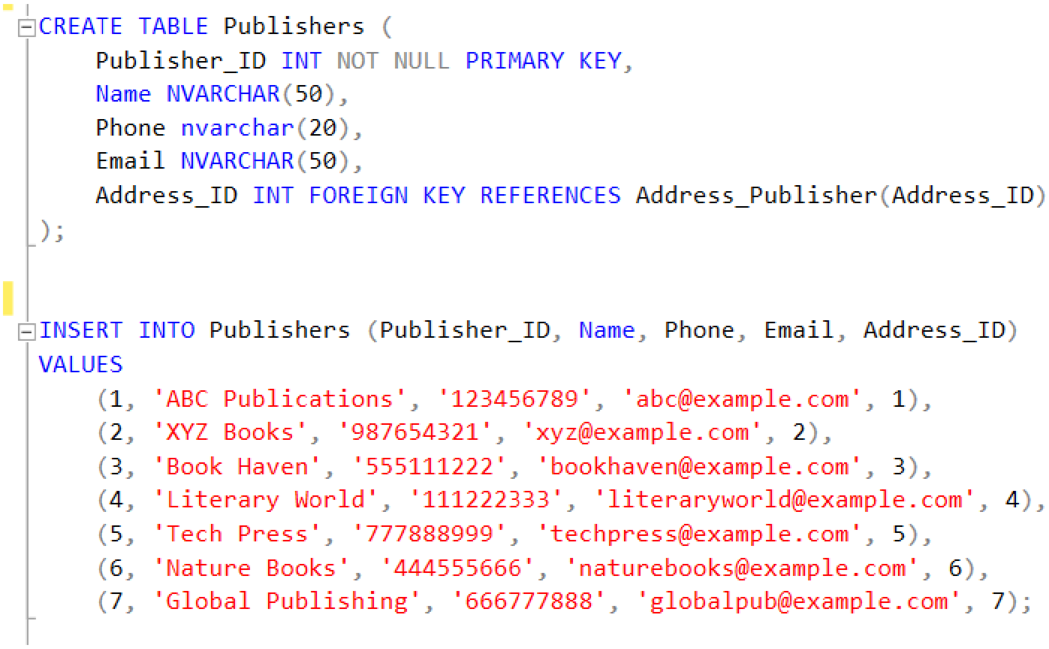
Figure 4: Address Publisher Table



	Address_ID	City	Street	Building	Flat
1	1	Kuala Lumpur	Jalan Monorail	2X	X11
2	2	Kuala Lumpur	Jalan Cochrane	15Y	Y12
3	3	Kuala Lumpur	Jalan Sultan Hishamuddin	8Z	Z13
4	4	Kuala Lumpur	Jalan Tuanku Abdul Rahman	19W	W14
5	5	Kuala Lumpur	Jalan Kuching	6V	V15
6	6	Kuala Lumpur	Jalan Loke Yew	11U	U16
7	7	Kuala Lumpur	Jalan Datuk Sulaiman	4T	T17
8	8	Kuala Lumpur	Jalan Segambut	21S	S18
9	9	Kuala Lumpur	Jalan Kerayong	10R	R19
10	10	Kuala Lumpur	Jalan Syed Putra	7Q	Q20

Figure 5: Result of Address Publisher Table

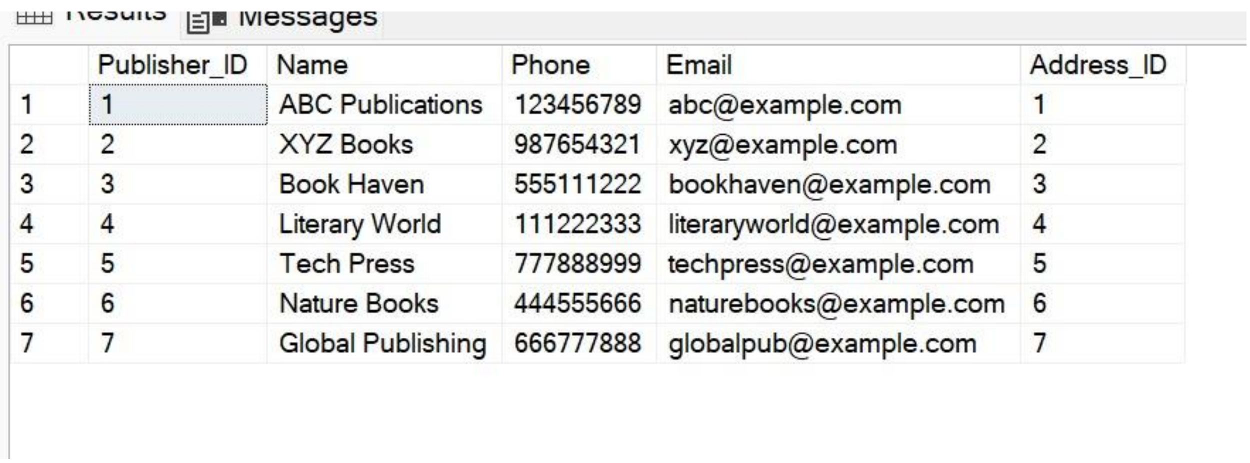
Publishers Table



```
CREATE TABLE Publishers (
    Publisher_ID INT NOT NULL PRIMARY KEY,
    Name NVARCHAR(50),
    Phone nvarchar(20),
    Email NVARCHAR(50),
    Address_ID INT FOREIGN KEY REFERENCES Address_Publisher(Address_ID)
);


INSERT INTO Publishers (Publisher_ID, Name, Phone, Email, Address_ID)
VALUES
    (1, 'ABC Publications', '123456789', 'abc@example.com', 1),
    (2, 'XYZ Books', '987654321', 'xyz@example.com', 2),
    (3, 'Book Haven', '555111222', 'bookhaven@example.com', 3),
    (4, 'Literary World', '111222333', 'literaryworld@example.com', 4),
    (5, 'Tech Press', '777888999', 'techpress@example.com', 5),
    (6, 'Nature Books', '444555666', 'naturebooks@example.com', 6),
    (7, 'Global Publishing', '666777888', 'globalpub@example.com', 7);
```

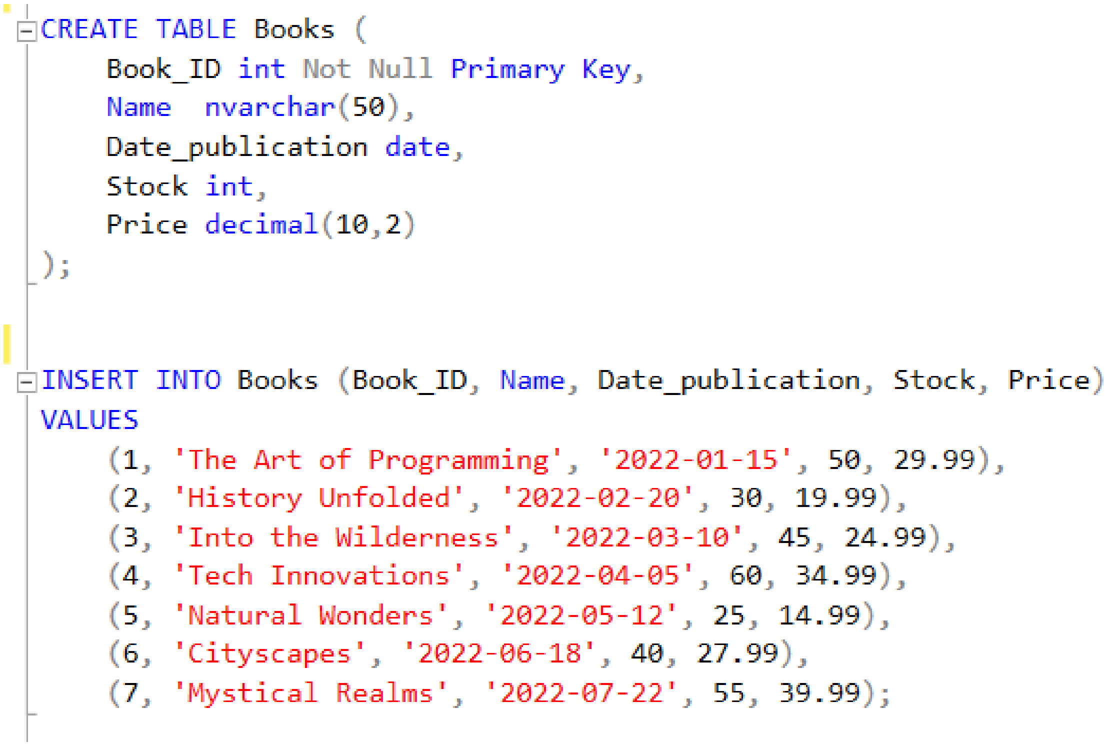
Figure 6: Publisher Table



	Publisher_ID	Name	Phone	Email	Address_ID
1	1	ABC Publications	123456789	abc@example.com	1
2	2	XYZ Books	987654321	xyz@example.com	2
3	3	Book Haven	555111222	bookhaven@example.com	3
4	4	Literary World	111222333	literaryworld@example.com	4
5	5	Tech Press	777888999	techpress@example.com	5
6	6	Nature Books	444555666	naturebooks@example.com	6
7	7	Global Publishing	666777888	globalpub@example.com	7

Figure 7: Result of Publisher Table

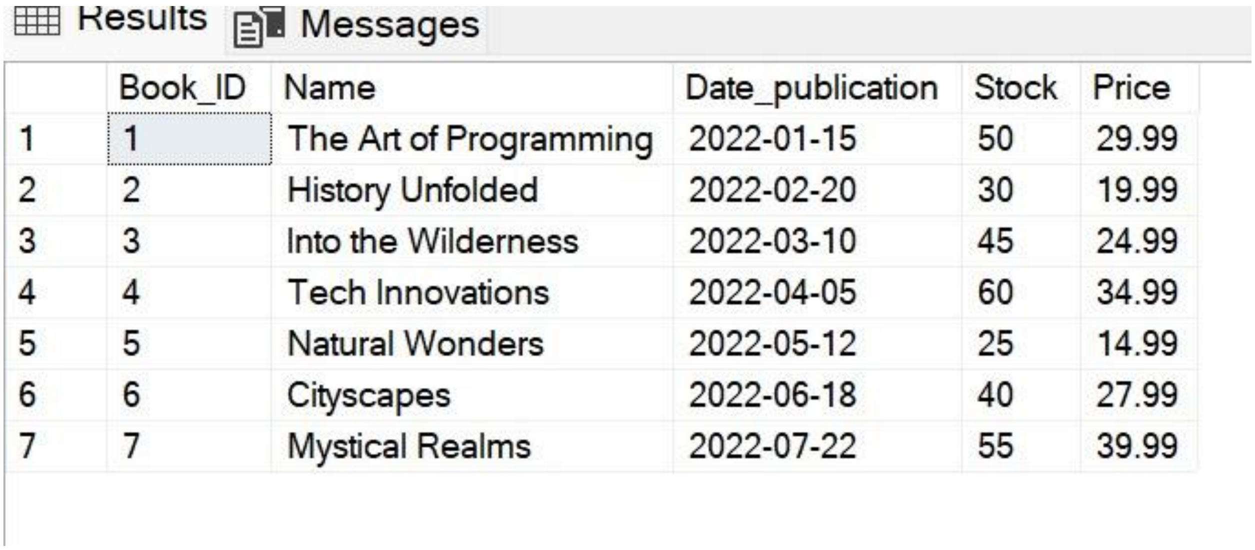
Book Table



```
CREATE TABLE Books (
    Book_ID int Not Null Primary Key,
    Name  nvarchar(50),
    Date_publication date,
    Stock int,
    Price decimal(10,2)
);


INSERT INTO Books (Book_ID, Name, Date_publication, Stock, Price)
VALUES
    (1, 'The Art of Programming', '2022-01-15', 50, 29.99),
    (2, 'History Unfolded', '2022-02-20', 30, 19.99),
    (3, 'Into the Wilderness', '2022-03-10', 45, 24.99),
    (4, 'Tech Innovations', '2022-04-05', 60, 34.99),
    (5, 'Natural Wonders', '2022-05-12', 25, 14.99),
    (6, 'Cityscapes', '2022-06-18', 40, 27.99),
    (7, 'Mystical Realms', '2022-07-22', 55, 39.99);
```

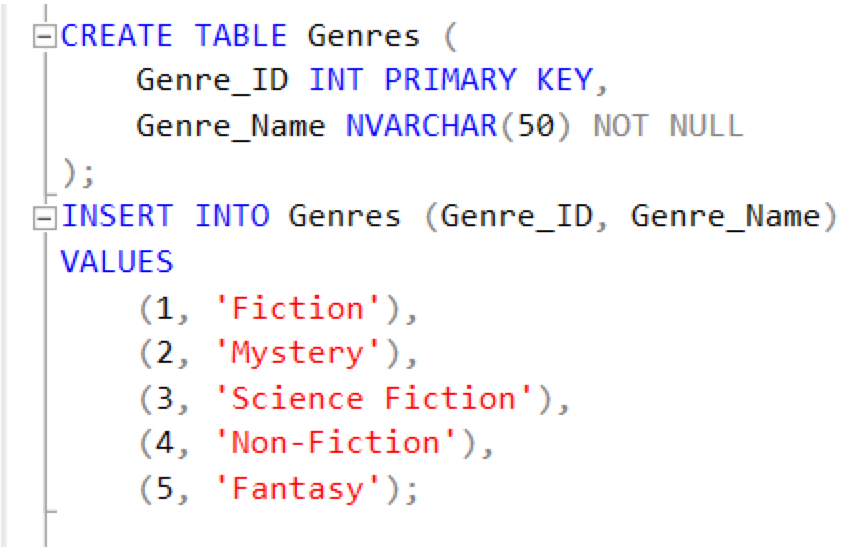
Figure 8: Books Table



	Book_ID	Name	Date_publication	Stock	Price
1	1	The Art of Programming	2022-01-15	50	29.99
2	2	History Unfolded	2022-02-20	30	19.99
3	3	Into the Wilderness	2022-03-10	45	24.99
4	4	Tech Innovations	2022-04-05	60	34.99
5	5	Natural Wonders	2022-05-12	25	14.99
6	6	Cityscapes	2022-06-18	40	27.99
7	7	Mystical Realms	2022-07-22	55	39.99

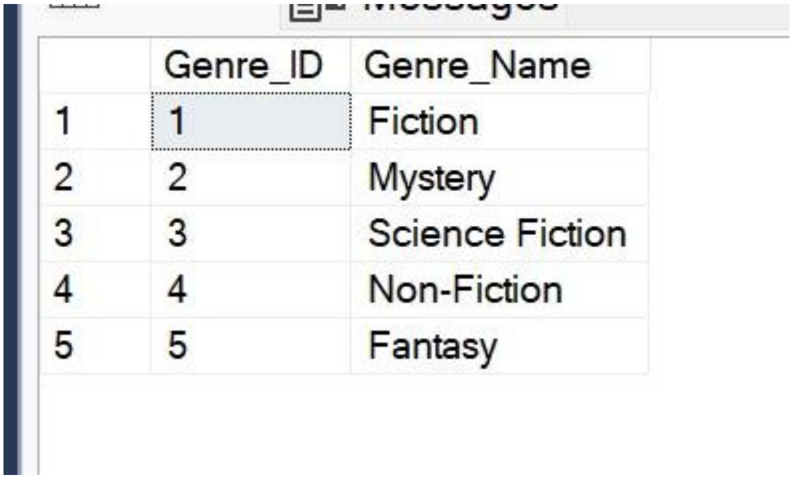
Figure 9: Results of Book Table

Genres Table



```
CREATE TABLE Genres (
    Genre_ID INT PRIMARY KEY,
    Genre_Name NVARCHAR(50) NOT NULL
);
INSERT INTO Genres (Genre_ID, Genre_Name)
VALUES
    (1, 'Fiction'),
    (2, 'Mystery'),
    (3, 'Science Fiction'),
    (4, 'Non-Fiction'),
    (5, 'Fantasy');
```

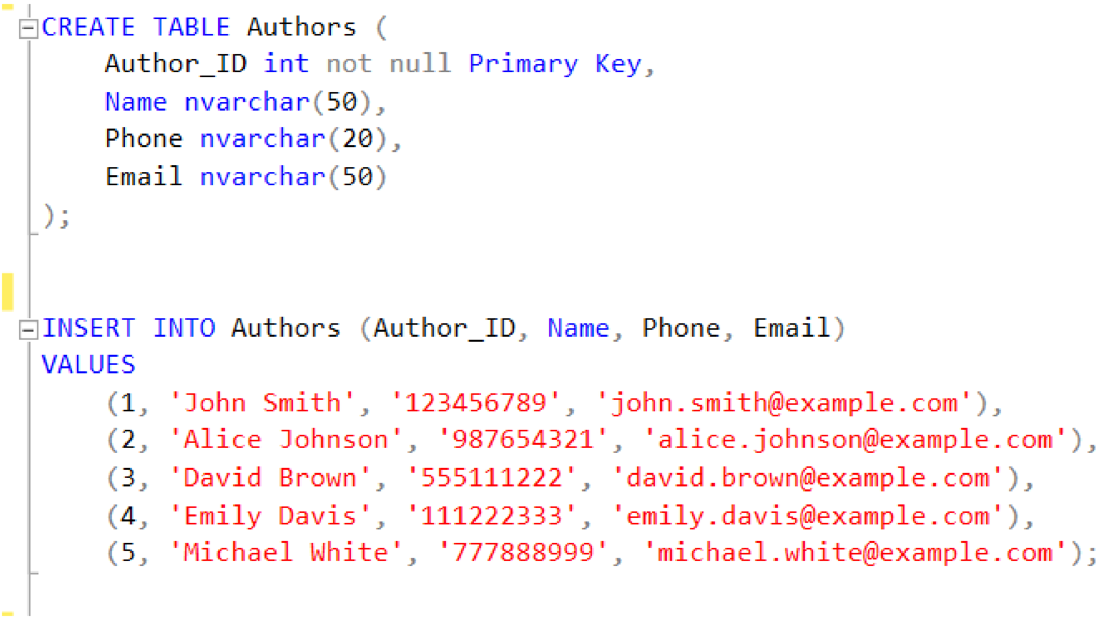
Figure 10: Genres Table



	Genre_ID	Genre_Name
1	1	Fiction
2	2	Mystery
3	3	Science Fiction
4	4	Non-Fiction
5	5	Fantasy

Figure 11: Results of Genre Tables

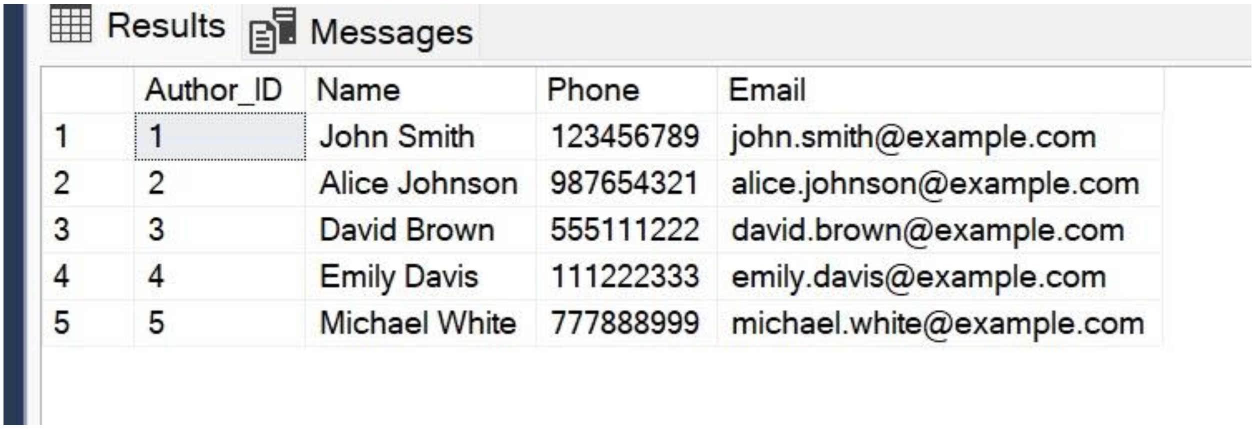
Authors Table



```sql
CREATE TABLE Authors (
    Author_ID int not null Primary Key,
    Name nvarchar(50),
    Phone nvarchar(20),
    Email nvarchar(50)
);


INSERT INTO Authors (Author_ID, Name, Phone, Email)
VALUES
    (1, 'John Smith', '123456789', 'john.smith@example.com'),
    (2, 'Alice Johnson', '987654321', 'alice.johnson@example.com'),
    (3, 'David Brown', '555111222', 'david.brown@example.com'),
    (4, 'Emily Davis', '111222333', 'emily.davis@example.com'),
    (5, 'Michael White', '777888999', 'michael.white@example.com');
```

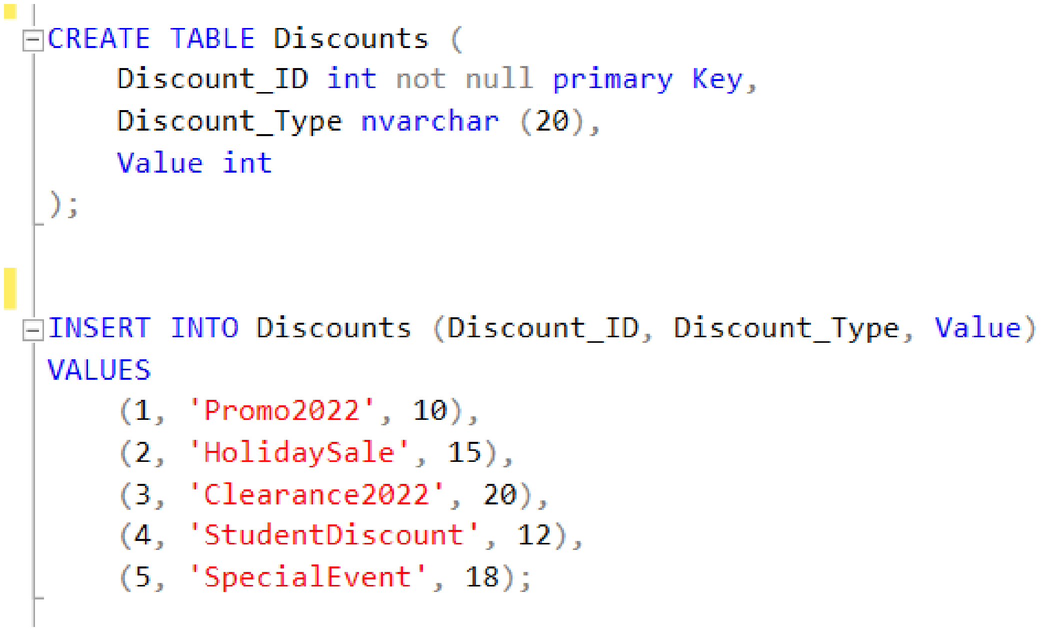
Figure 12: Authors Table



	Author_ID	Name	Phone	Email
1	1	John Smith	123456789	john.smith@example.com
2	2	Alice Johnson	987654321	alice.johnson@example.com
3	3	David Brown	555111222	david.brown@example.com
4	4	Emily Davis	111222333	emily.davis@example.com
5	5	Michael White	777888999	michael.white@example.com

Figure 13: Results of Authors Table

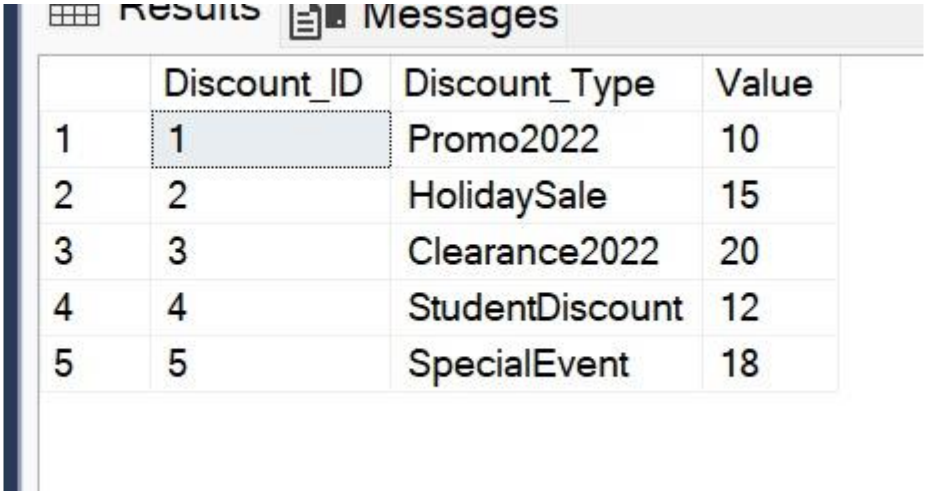
Discounts Table



```
CREATE TABLE Discounts (
	Discount_ID int not null primary Key,
	Discount_Type nvarchar (20),
	Value int
);


INSERT INTO Discounts (Discount_ID, Discount_Type, Value)
VALUES
	(1, 'Promo2022', 10),
	(2, 'HolidaySale', 15),
	(3, 'Clearance2022', 20),
	(4, 'StudentDiscount', 12),
	(5, 'SpecialEvent', 18);
```

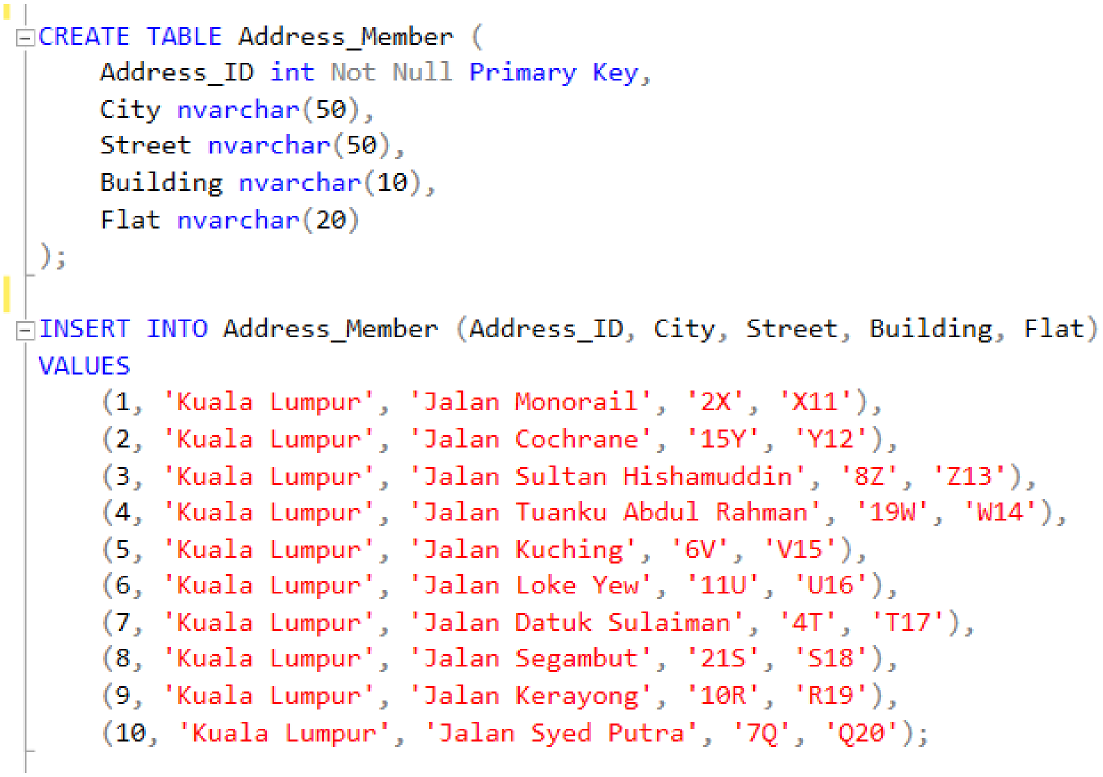
Figure 14: Discount Table



	Discount_ID	Discount_Type	Value
1	1	Promo2022	10
2	2	HolidaySale	15
3	3	Clearance2022	20
4	4	StudentDiscount	12
5	5	SpecialEvent	18

Figure 15: Results of Discount Table

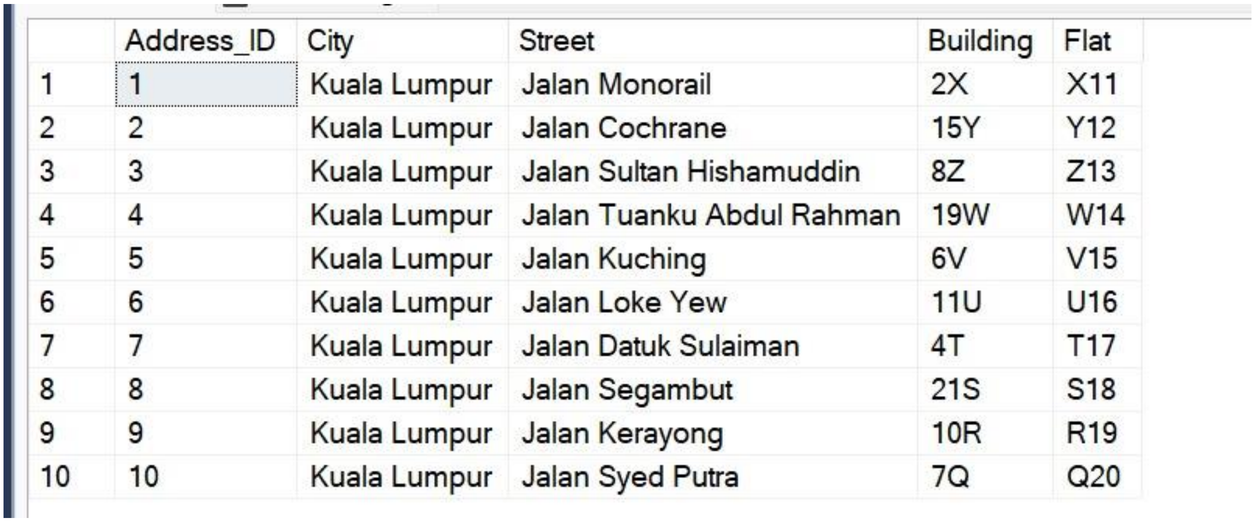
Address Member Table



```
CREATE TABLE Address_Member (
    Address_ID int Not Null Primary Key,
    City nvarchar(50),
    Street nvarchar(50),
    Building nvarchar(10),
    Flat nvarchar(20)
);

INSERT INTO Address_Member (Address_ID, City, Street, Building, Flat)
VALUES
    (1, 'Kuala Lumpur', 'Jalan Monorail', '2X', 'X11'),
    (2, 'Kuala Lumpur', 'Jalan Cochrane', '15Y', 'Y12'),
    (3, 'Kuala Lumpur', 'Jalan Sultan Hishamuddin', '8Z', 'Z13'),
    (4, 'Kuala Lumpur', 'Jalan Tuanku Abdul Rahman', '19W', 'W14'),
    (5, 'Kuala Lumpur', 'Jalan Kuching', '6V', 'V15'),
    (6, 'Kuala Lumpur', 'Jalan Loke Yew', '11U', 'U16'),
    (7, 'Kuala Lumpur', 'Jalan Datuk Sulaiman', '4T', 'T17'),
    (8, 'Kuala Lumpur', 'Jalan Segambut', '21S', 'S18'),
    (9, 'Kuala Lumpur', 'Jalan Kerayong', '10R', 'R19'),
    (10, 'Kuala Lumpur', 'Jalan Syed Putra', '7Q', 'Q20');
```

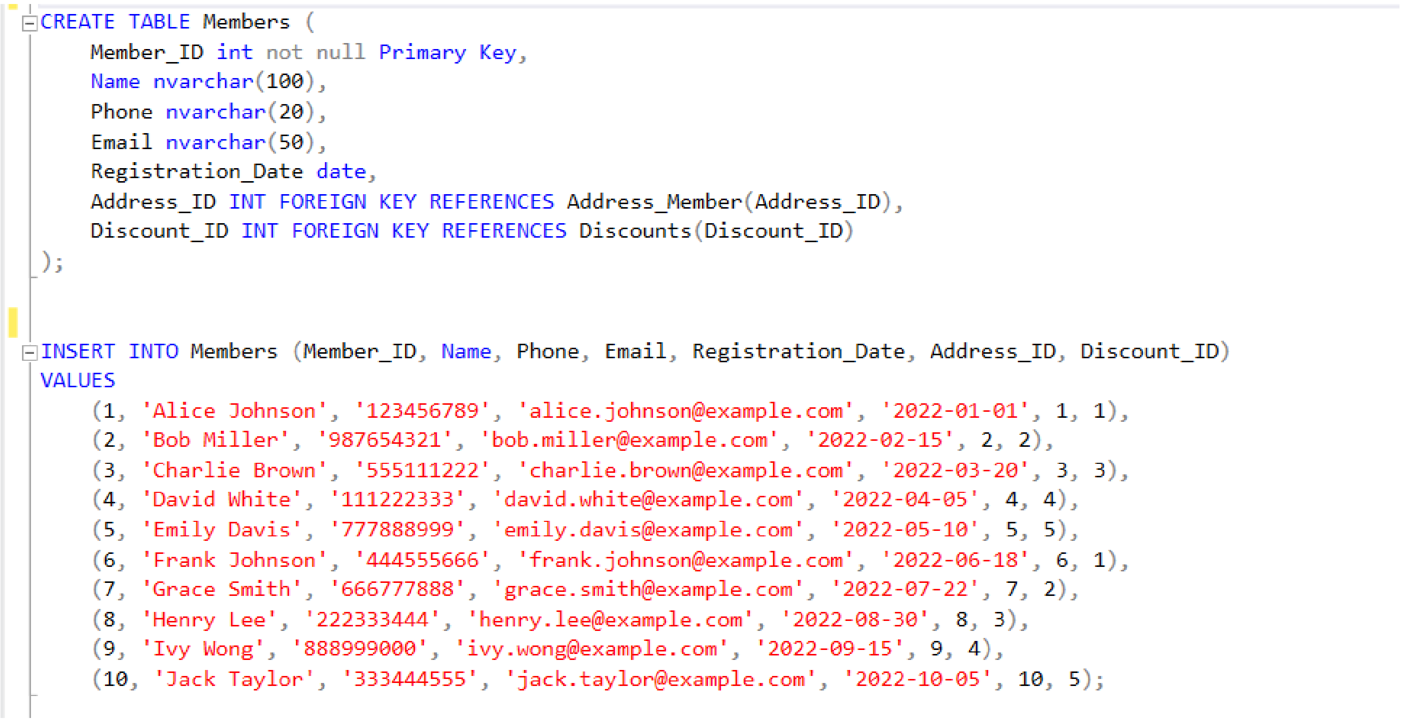
Figure 16: Address Member Table



	Address_ID	City	Street	Building	Flat
1	1	Kuala Lumpur	Jalan Monorail	2X	X11
2	2	Kuala Lumpur	Jalan Cochrane	15Y	Y12
3	3	Kuala Lumpur	Jalan Sultan Hishamuddin	8Z	Z13
4	4	Kuala Lumpur	Jalan Tuanku Abdul Rahman	19W	W14
5	5	Kuala Lumpur	Jalan Kuching	6V	V15
6	6	Kuala Lumpur	Jalan Loke Yew	11U	U16
7	7	Kuala Lumpur	Jalan Datuk Sulaiman	4T	T17
8	8	Kuala Lumpur	Jalan Segambut	21S	S18
9	9	Kuala Lumpur	Jalan Kerayong	10R	R19
10	10	Kuala Lumpur	Jalan Syed Putra	7Q	Q20

Figure 17: Results of Address Members Table

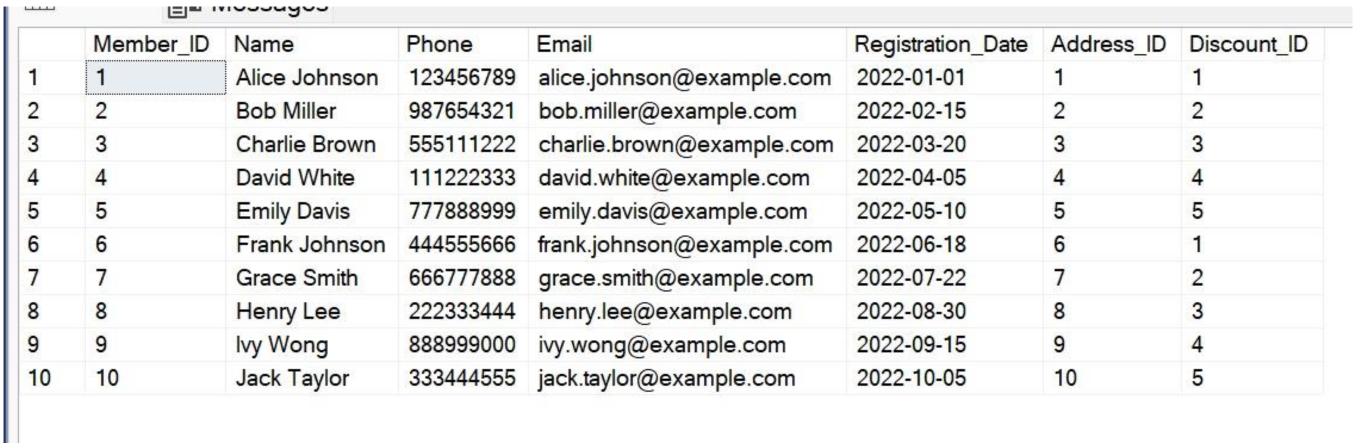
Members Table



```sql
CREATE TABLE Members (
    Member_ID int not null Primary Key,
    Name nvarchar(100),
    Phone nvarchar(20),
    Email nvarchar(50),
    Registration_Date date,
    Address_ID INT FOREIGN KEY REFERENCES Address_Member(Address_ID),
    Discount_ID INT FOREIGN KEY REFERENCES Discounts(Discount_ID)
);


INSERT INTO Members (Member_ID, Name, Phone, Email, Registration_Date, Address_ID, Discount_ID)
VALUES
    (1, 'Alice Johnson', '123456789', 'alice.johnson@example.com', '2022-01-01', 1, 1),
    (2, 'Bob Miller', '987654321', 'bob.miller@example.com', '2022-02-15', 2, 2),
    (3, 'Charlie Brown', '555111222', 'charlie.brown@example.com', '2022-03-20', 3, 3),
    (4, 'David White', '111222333', 'david.white@example.com', '2022-04-05', 4, 4),
    (5, 'Emily Davis', '777888999', 'emily.davis@example.com', '2022-05-10', 5, 5),
    (6, 'Frank Johnson', '444555666', 'frank.johnson@example.com', '2022-06-18', 6, 1),
    (7, 'Grace Smith', '666777888', 'grace.smith@example.com', '2022-07-22', 7, 2),
    (8, 'Henry Lee', '222333444', 'henry.lee@example.com', '2022-08-30', 8, 3),
    (9, 'Ivy Wong', '888999000', 'ivy.wong@example.com', '2022-09-15', 9, 4),
    (10, 'Jack Taylor', '333444555', 'jack.taylor@example.com', '2022-10-05', 10, 5);
```

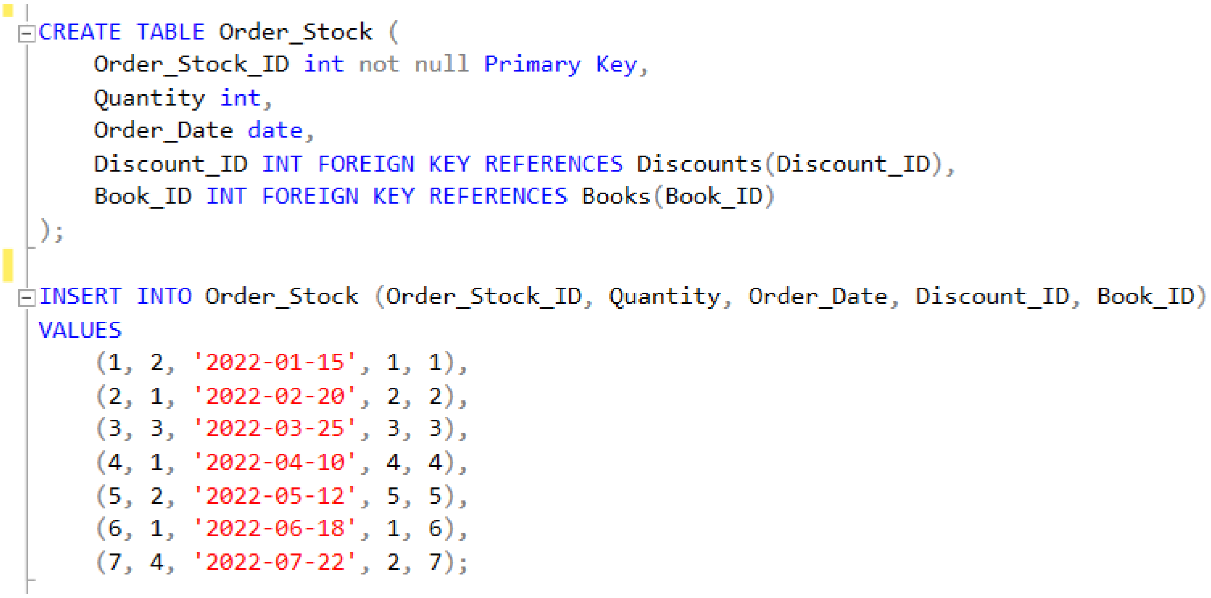
Figure 18: Members Table



	Member_ID	Name	Phone	Email	Registration_Date	Address_ID	Discount_ID
1	1	Alice Johnson	123456789	alice.johnson@example.com	2022-01-01	1	1
2	2	Bob Miller	987654321	bob.miller@example.com	2022-02-15	2	2
3	3	Charlie Brown	555111222	charlie.brown@example.com	2022-03-20	3	3
4	4	David White	111222333	david.white@example.com	2022-04-05	4	4
5	5	Emily Davis	777888999	emily.davis@example.com	2022-05-10	5	5
6	6	Frank Johnson	444555666	frank.johnson@example.com	2022-06-18	6	1
7	7	Grace Smith	666777888	grace.smith@example.com	2022-07-22	7	2
8	8	Henry Lee	222333444	henry.lee@example.com	2022-08-30	8	3
9	9	Ivy Wong	888999000	ivy.wong@example.com	2022-09-15	9	4
10	10	Jack Taylor	333444555	jack.taylor@example.com	2022-10-05	10	5

Figure 19: Results of Members Table

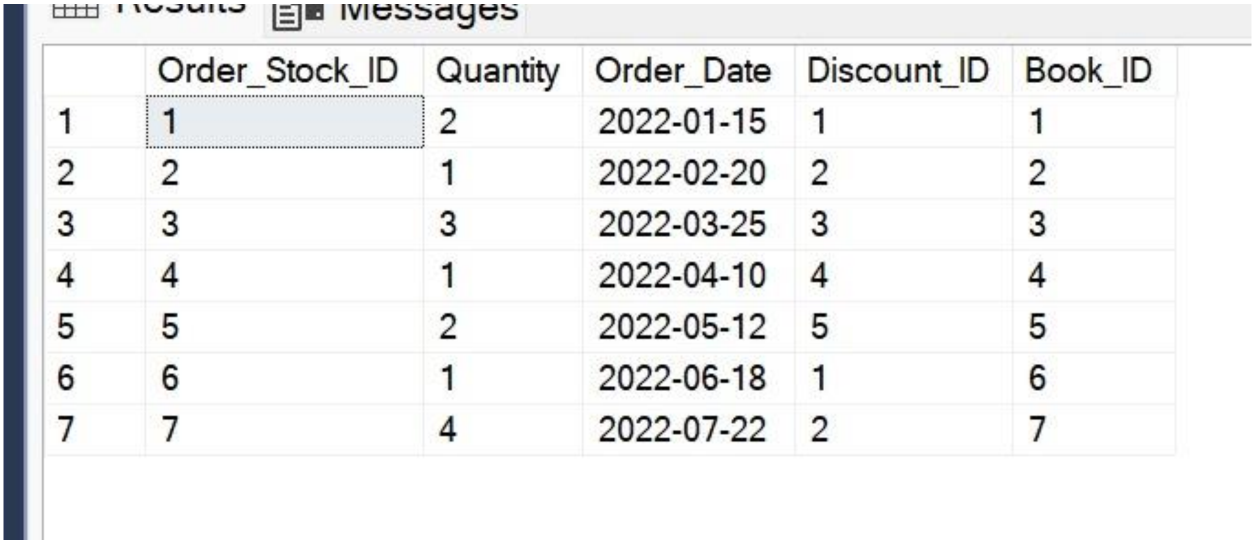
Order Stock Table



```
CREATE TABLE Order_Stock (
    Order_Stock_ID int not null Primary Key,
    Quantity int,
    Order_Date date,
    Discount_ID INT FOREIGN KEY REFERENCES Discounts(Discount_ID),
    Book_ID INT FOREIGN KEY REFERENCES Books(Book_ID)
);

INSERT INTO Order_Stock (Order_Stock_ID, Quantity, Order_Date, Discount_ID, Book_ID)
VALUES
    (1, 2, '2022-01-15', 1, 1),
    (2, 1, '2022-02-20', 2, 2),
    (3, 3, '2022-03-25', 3, 3),
    (4, 1, '2022-04-10', 4, 4),
    (5, 2, '2022-05-12', 5, 5),
    (6, 1, '2022-06-18', 1, 6),
    (7, 4, '2022-07-22', 2, 7);
```

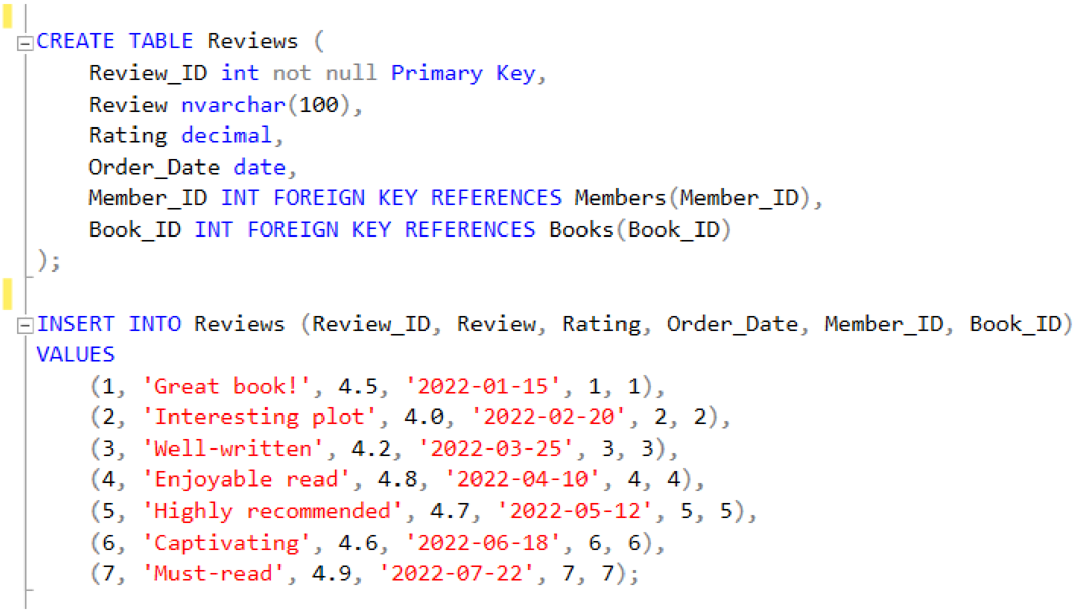
Figure 20: Order Stock Table



	Order_Stock_ID	Quantity	Order_Date	Discount_ID	Book_ID
1	1	2	2022-01-15	1	1
2	2	1	2022-02-20	2	2
3	3	3	2022-03-25	3	3
4	4	1	2022-04-10	4	4
5	5	2	2022-05-12	5	5
6	6	1	2022-06-18	1	6
7	7	4	2022-07-22	2	7

Figure 21: Results of Order Stock Table

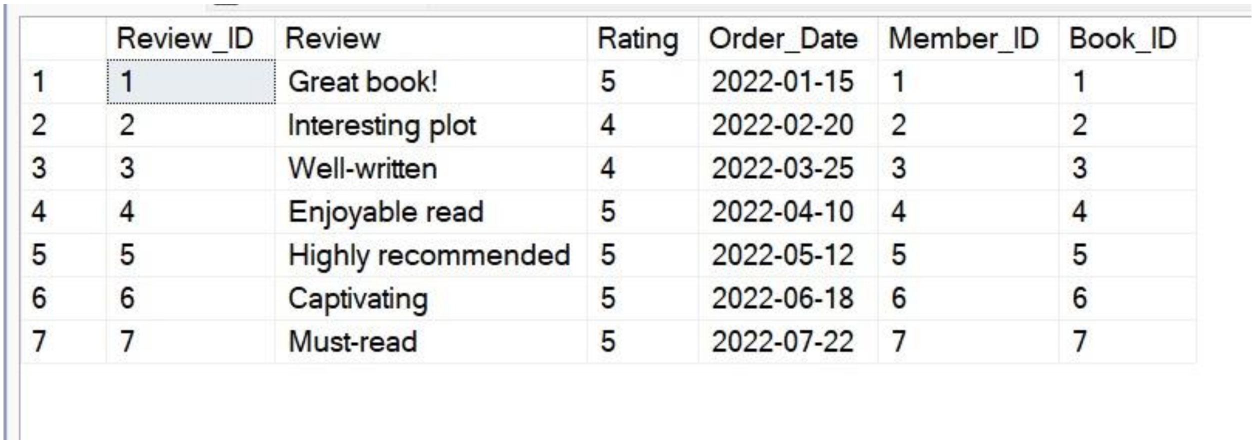
Reviews Table



```
CREATE TABLE Reviews (
    Review_ID int not null Primary Key,
    Review nvarchar(100),
    Rating decimal,
    Order_Date date,
    Member_ID INT FOREIGN KEY REFERENCES Members(Member_ID),
    Book_ID INT FOREIGN KEY REFERENCES Books(Book_ID)
);

INSERT INTO Reviews (Review_ID, Review, Rating, Order_Date, Member_ID, Book_ID)
VALUES
    (1, 'Great book!', 4.5, '2022-01-15', 1, 1),
    (2, 'Interesting plot', 4.0, '2022-02-20', 2, 2),
    (3, 'Well-written', 4.2, '2022-03-25', 3, 3),
    (4, 'Enjoyable read', 4.8, '2022-04-10', 4, 4),
    (5, 'Highly recommended', 4.7, '2022-05-12', 5, 5),
    (6, 'Captivating', 4.6, '2022-06-18', 6, 6),
    (7, 'Must-read', 4.9, '2022-07-22', 7, 7);
```

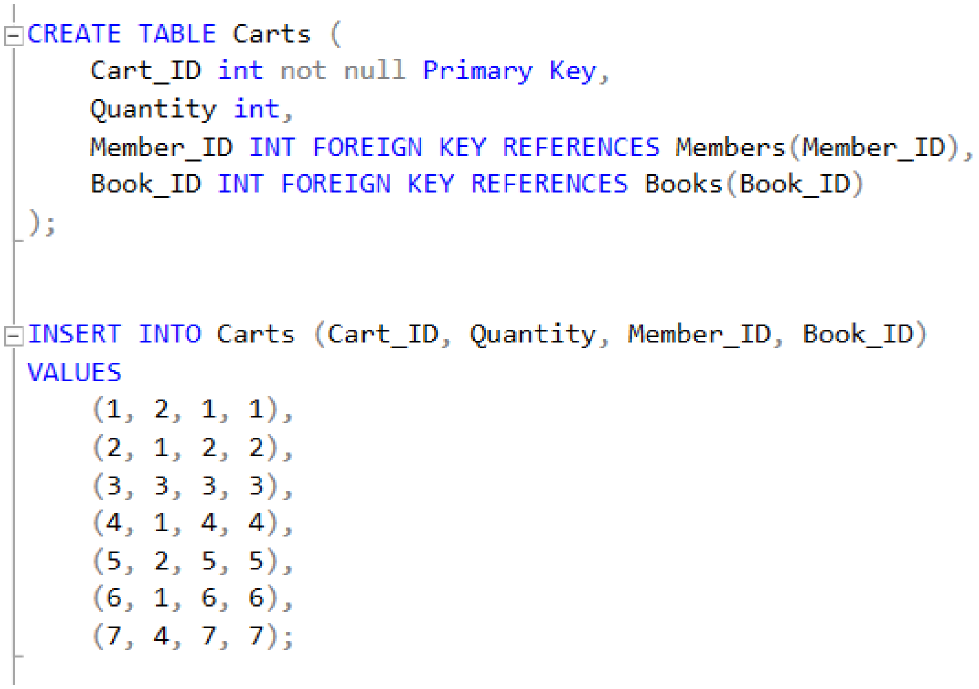
Figure 22: Reviews Table



	Review_ID	Review	Rating	Order_Date	Member_ID	Book_ID
1	1	Great book!	5	2022-01-15	1	1
2	2	Interesting plot	4	2022-02-20	2	2
3	3	Well-written	4	2022-03-25	3	3
4	4	Enjoyable read	5	2022-04-10	4	4
5	5	Highly recommended	5	2022-05-12	5	5
6	6	Captivating	5	2022-06-18	6	6
7	7	Must-read	5	2022-07-22	7	7

Figure 23: Results of Reviews Table

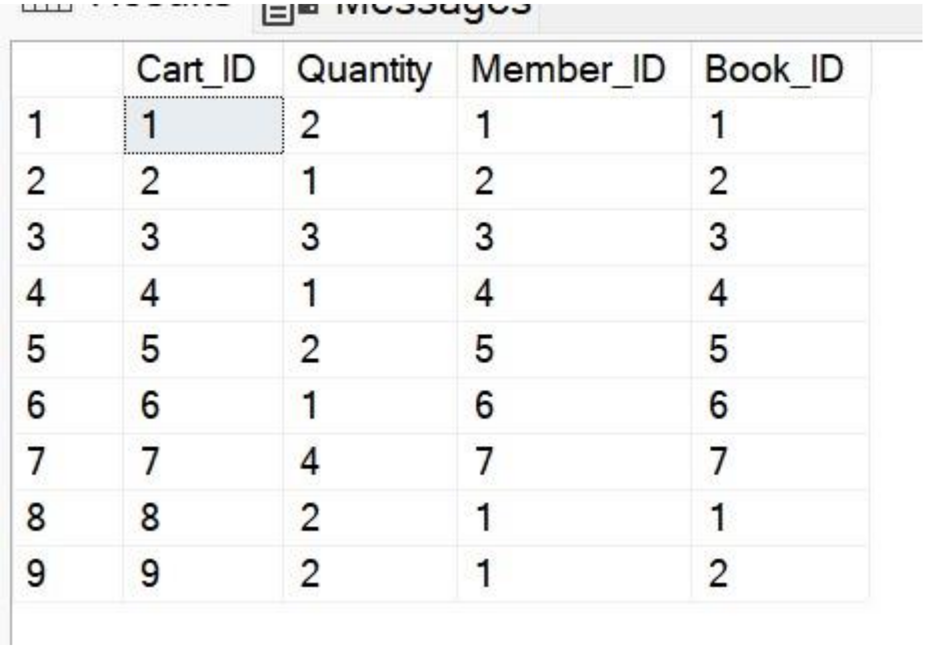
Carts Table



```
CREATE TABLE Carts (
    Cart_ID int not null Primary Key,
    Quantity int,
    Member_ID INT FOREIGN KEY REFERENCES Members(Member_ID),
    Book_ID INT FOREIGN KEY REFERENCES Books(Book_ID)
);


INSERT INTO Carts (Cart_ID, Quantity, Member_ID, Book_ID)
VALUES
    (1, 2, 1, 1),
    (2, 1, 2, 2),
    (3, 3, 3, 3),
    (4, 1, 4, 4),
    (5, 2, 5, 5),
    (6, 1, 6, 6),
    (7, 4, 7, 7);
```

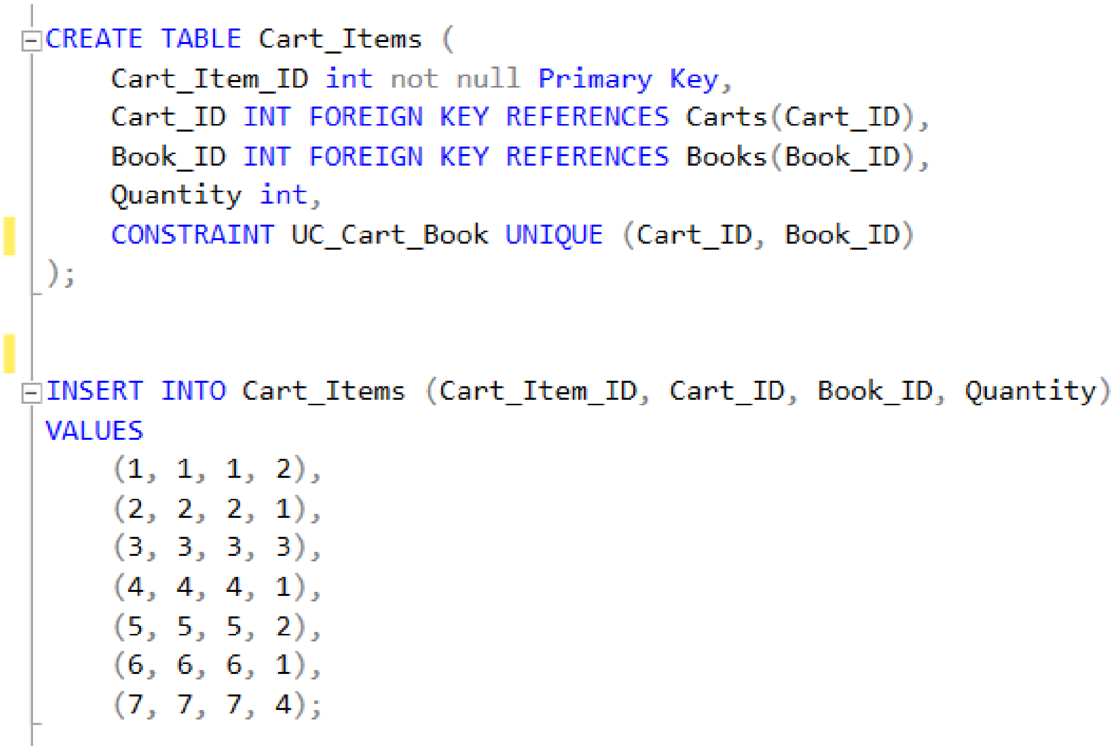
Figure 24: Carts Table



	Cart_ID	Quantity	Member_ID	Book_ID
1	1	2	1	1
2	2	1	2	2
3	3	3	3	3
4	4	1	4	4
5	5	2	5	5
6	6	1	6	6
7	7	4	7	7
8	8	2	1	1
9	9	2	1	2

Figure 25: Results of Carts Table

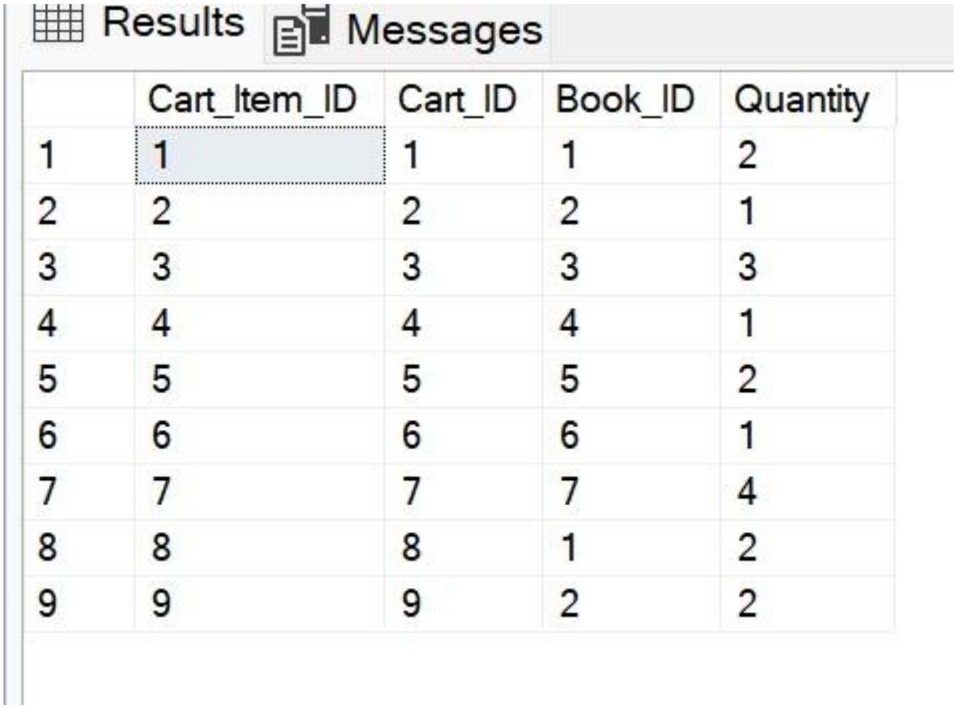
Cart Item Table



```
CREATE TABLE Cart_Items (
    Cart_Item_ID int not null Primary Key,
    Cart_ID INT FOREIGN KEY REFERENCES Carts(Cart_ID),
    Book_ID INT FOREIGN KEY REFERENCES Books(Book_ID),
    Quantity int,
    CONSTRAINT UC_Cart_Book UNIQUE (Cart_ID, Book_ID)
);


INSERT INTO Cart_Items (Cart_Item_ID, Cart_ID, Book_ID, Quantity)
VALUES
    (1, 1, 1, 2),
    (2, 2, 2, 1),
    (3, 3, 3, 3),
    (4, 4, 4, 1),
    (5, 5, 5, 2),
    (6, 6, 6, 1),
    (7, 7, 7, 4);
```

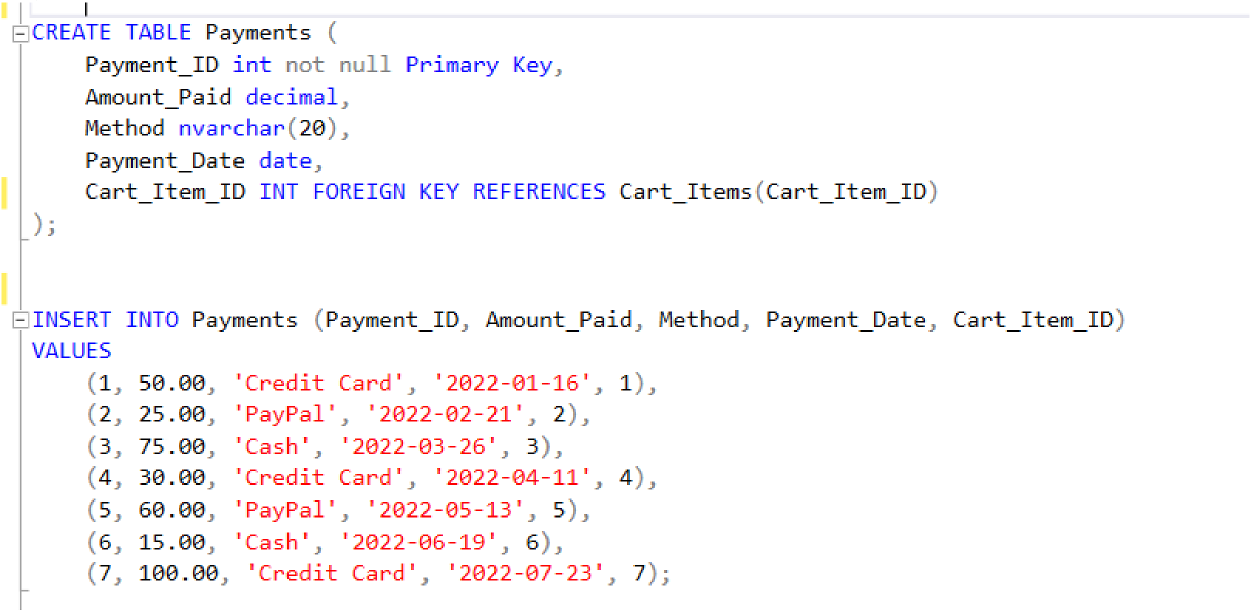
Figure 26: Cart Item Table



	Cart_Item_ID	Cart_ID	Book_ID	Quantity
1	1	1	1	2
2	2	2	2	1
3	3	3	3	3
4	4	4	4	1
5	5	5	5	2
6	6	6	6	1
7	7	7	7	4
8	8	8	1	2
9	9	9	2	2

Figure 27: Results of Cart Item Table

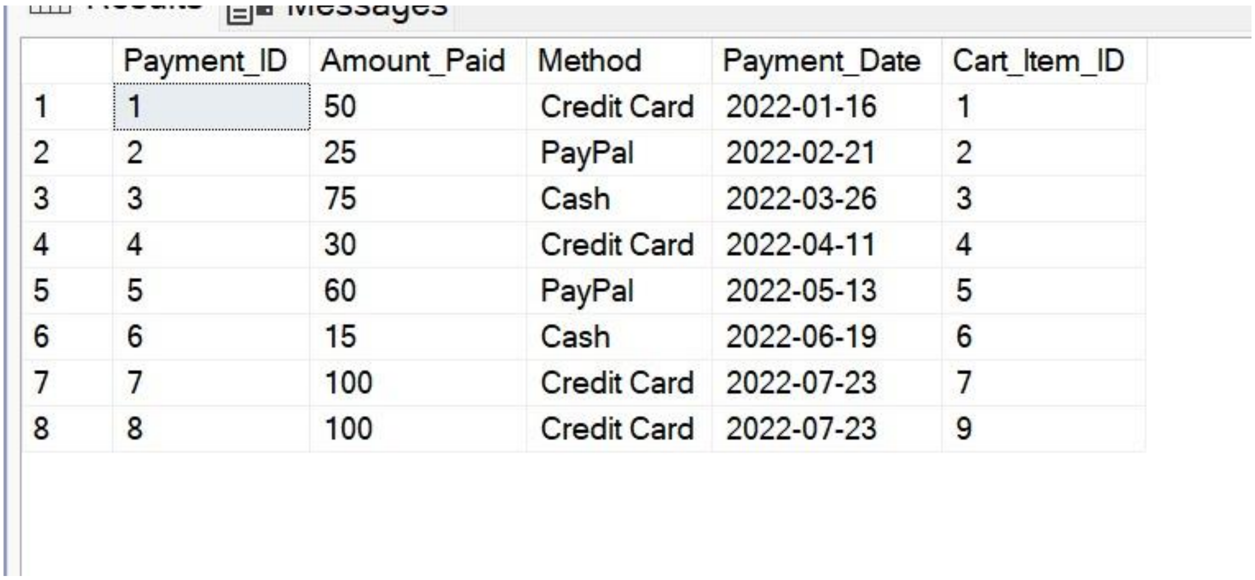
Payments Table



```
CREATE TABLE Payments (
    Payment_ID int not null Primary Key,
    Amount_Paid decimal,
    Method nvarchar(20),
    Payment_Date date,
    Cart_Item_ID INT FOREIGN KEY REFERENCES Cart_Items(Cart_Item_ID)
);


INSERT INTO Payments (Payment_ID, Amount_Paid, Method, Payment_Date, Cart_Item_ID)
VALUES
    (1, 50.00, 'Credit Card', '2022-01-16', 1),
    (2, 25.00, 'PayPal', '2022-02-21', 2),
    (3, 75.00, 'Cash', '2022-03-26', 3),
    (4, 30.00, 'Credit Card', '2022-04-11', 4),
    (5, 60.00, 'PayPal', '2022-05-13', 5),
    (6, 15.00, 'Cash', '2022-06-19', 6),
    (7, 100.00, 'Credit Card', '2022-07-23', 7);
```

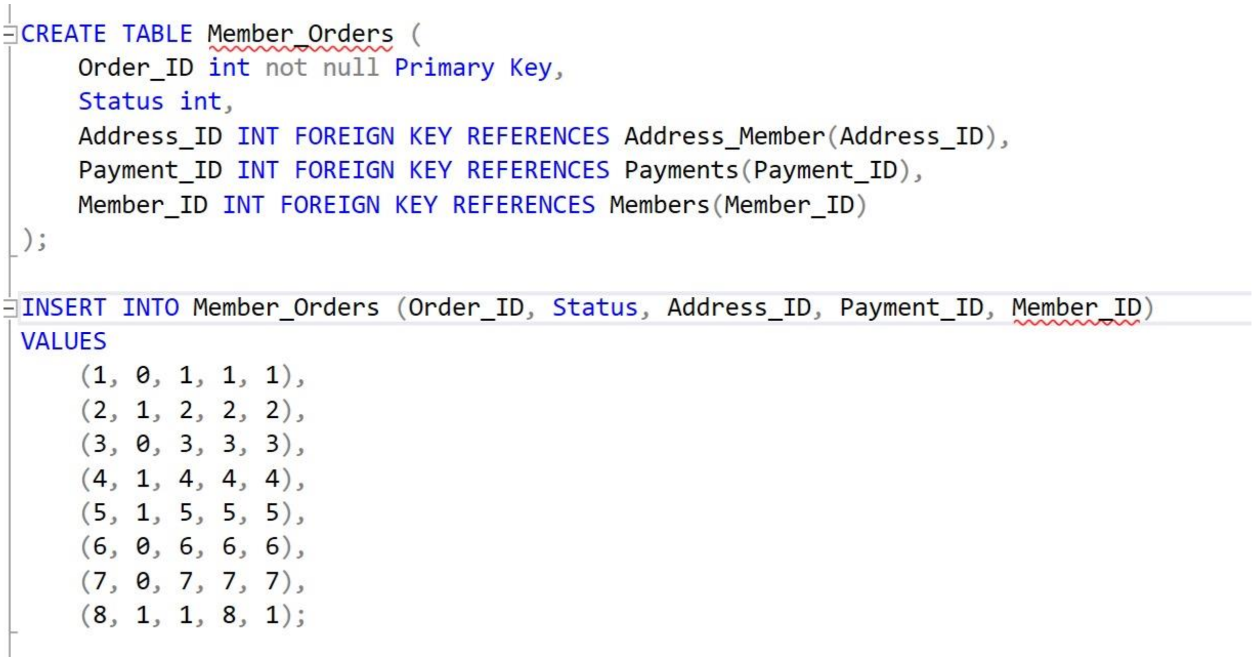
Figure 28: Payments Table



	Payment_ID	Amount_Paid	Method	Payment_Date	Cart_Item_ID
1	1	50	Credit Card	2022-01-16	1
2	2	25	PayPal	2022-02-21	2
3	3	75	Cash	2022-03-26	3
4	4	30	Credit Card	2022-04-11	4
5	5	60	PayPal	2022-05-13	5
6	6	15	Cash	2022-06-19	6
7	7	100	Credit Card	2022-07-23	7
8	8	100	Credit Card	2022-07-23	9

Figure 29: Results of Payment Table

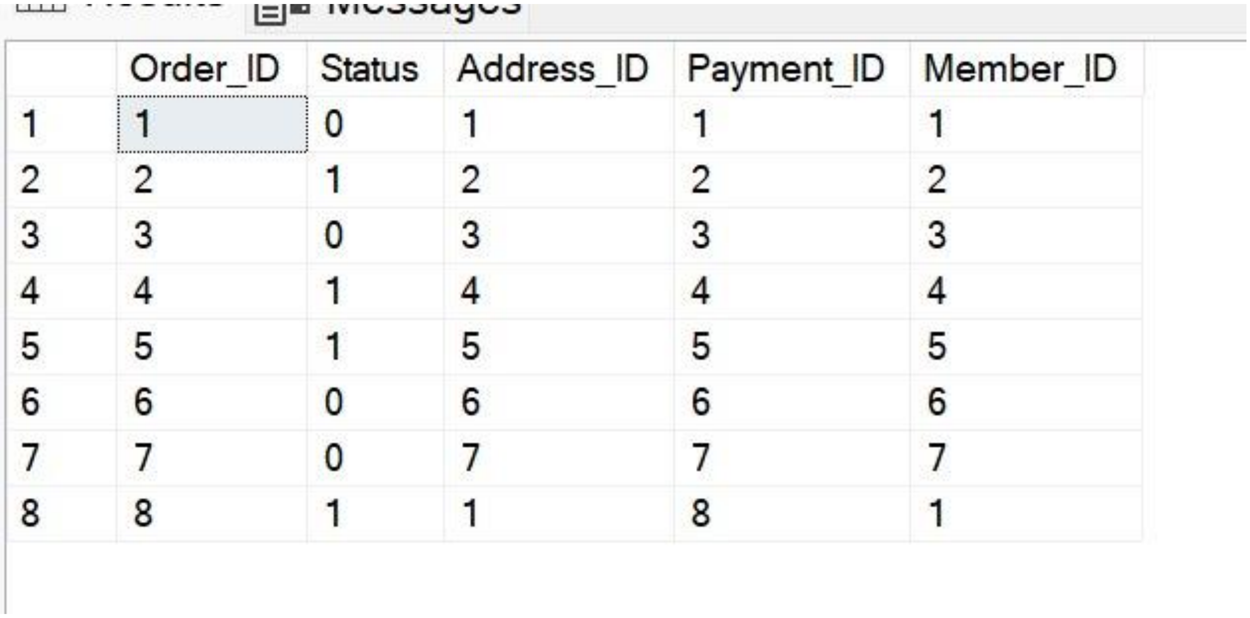
<u>Member Order Table</u>



```
CREATE TABLE Member_Orders (
    Order_ID int not null Primary Key,
    Status int,
    Address_ID INT FOREIGN KEY REFERENCES Address_Member(Address_ID),
    Payment_ID INT FOREIGN KEY REFERENCES Payments(Payment_ID),
    Member_ID INT FOREIGN KEY REFERENCES Members(Member_ID)
);

INSERT INTO Member_Orders (Order_ID, Status, Address_ID, Payment_ID, Member_ID)
VALUES
    (1, 0, 1, 1, 1),
    (2, 1, 2, 2, 2),
    (3, 0, 3, 3, 3),
    (4, 1, 4, 4, 4),
    (5, 1, 5, 5, 5),
    (6, 0, 6, 6, 6),
    (7, 0, 7, 7, 7),
    (8, 1, 1, 8, 1);
```

Figure 30: Member Order Table

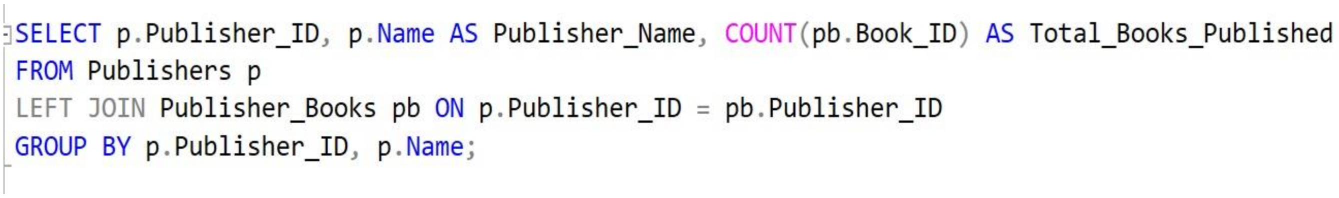


	Order_ID	Status	Address_ID	Payment_ID	Member_ID
1	1	0	1	1	1
2	2	1	2	2	2
3	3	0	3	3	3
4	4	1	4	4	4
5	5	1	5	5	5
6	6	0	6	6	6
7	7	0	7	7	7
8	8	1	1	8	1

Figure 31: Results of Member Order Table

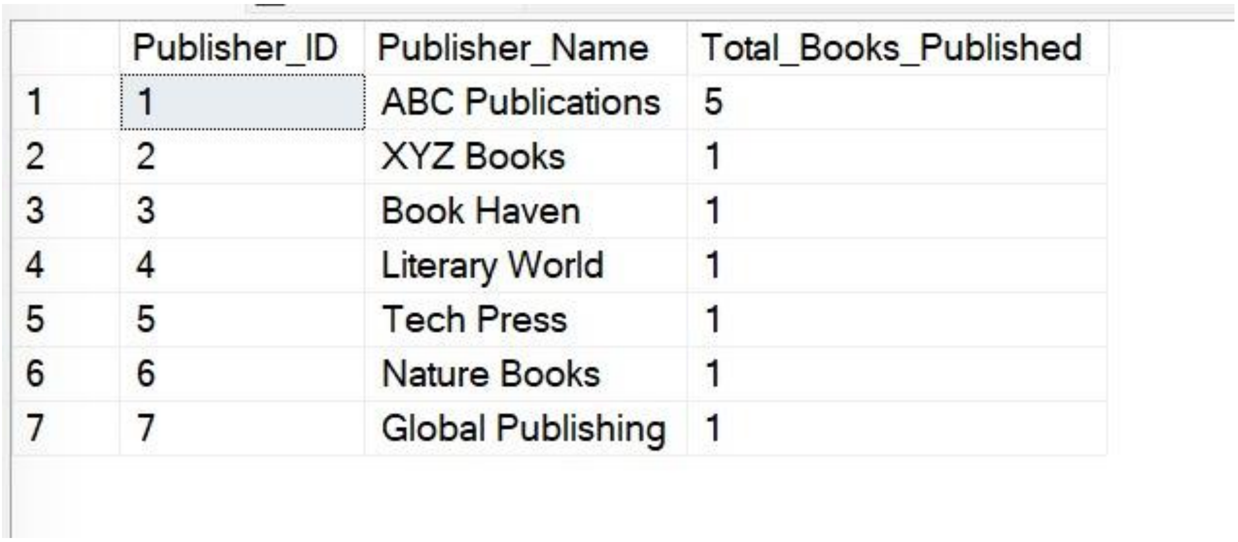
SQL-Data Manipulation Language (DML)

1. Total Number of Books Published by each publisher



```
SELECT p.Publisher_ID, p.Name AS Publisher_Name, COUNT(pb.Book_ID) AS Total_Books_Published
FROM Publishers p
LEFT JOIN Publisher_Books pb ON p.Publisher_ID = pb.Publisher_ID
GROUP BY p.Publisher_ID, p.Name;
```

Figure 32: DML for total number of books published by each publisher.

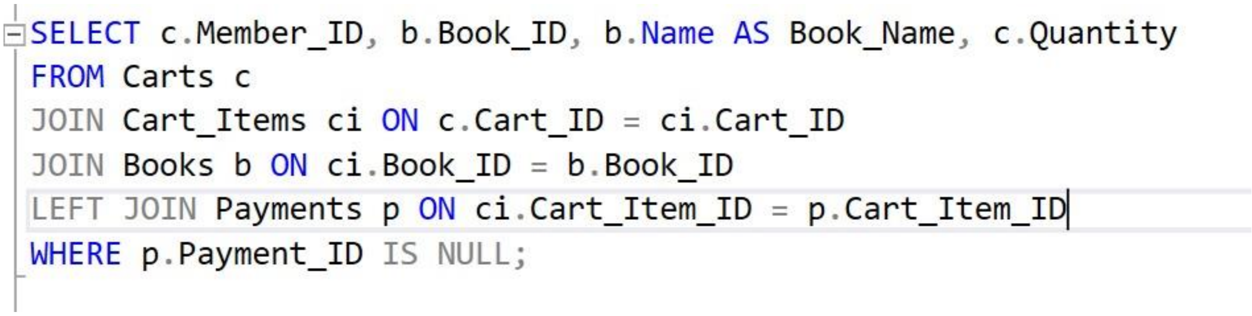


	Publisher_ID	Publisher_Name	Total_Books_Published
1	1	ABC Publications	5
2	2	XYZ Books	1
3	3	Book Haven	1
4	4	Literary World	1
5	5	Tech Press	1
6	6	Nature Books	1
7	7	Global Publishing	1

Figure 33: Results of total number of books published by each publisher

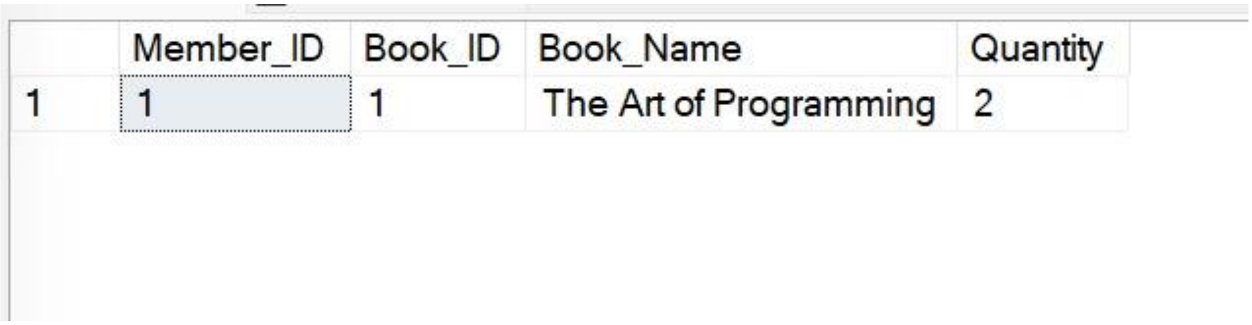
This query uses a JOIN to join the Order_Stock, Books, and Publisher_Books tables and counts the total number of books ordered for each publisher. The result includes Publisher_ID, publisher name, and total number of books ordered. This query is useful for assessing the sales success of a publisher's books.

2. Books present in shopping carts without completed payments.



```
SELECT c.Member_ID, b.Book_ID, b.Name AS Book_Name, c.Quantity
FROM Carts c
JOIN Cart_Items ci ON c.Cart_ID = ci.Cart_ID
JOIN Books b ON ci.Book_ID = b.Book_ID
LEFT JOIN Payments p ON ci.Cart_Item_ID = p.Cart_Item_ID
WHERE p.Payment_ID IS NULL;
```

Figure 34: DML for Books present in shopping cart without completed payments

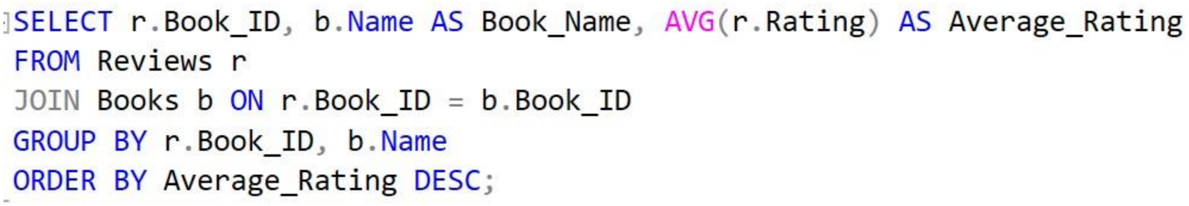


	Member_ID	Book_ID	Book_Name	Quantity
1	1	1	The Art of Programming	2

Figure 35: Results for Books present in shopping cart without completed payments

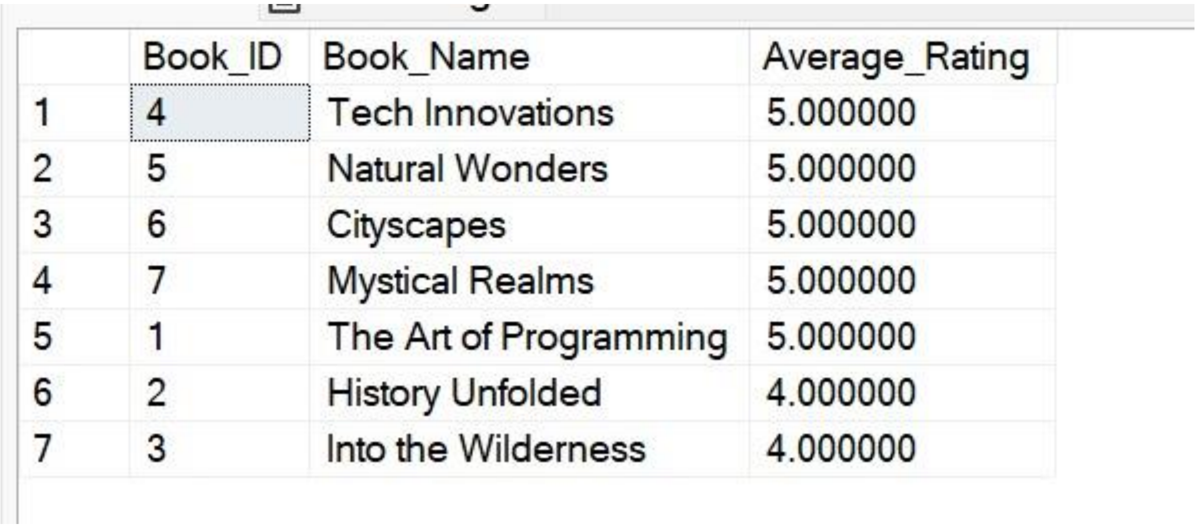
This query identifies books present in shopping carts without completed payments. It combines data from the Carts, Cart_Items, Books, and Payments tables, using JOIN operations to link relevant information. The WHERE clause filters out entries where payment is already completed, showcasing Member_ID, Book_ID, Book_Name, and Quantity for potential issues in the payment process. This information proves crucial for monitoring and resolving incomplete transactions, ensuring a seamless customer experience.

3. Average Rating for Each Book from Reviews



```
SELECT r.Book_ID, b.Name AS Book_Name, AVG(r.Rating) AS Average_Rating
FROM Reviews r
JOIN Books b ON r.Book_ID = b.Book_ID
GROUP BY r.Book_ID, b.Name
ORDER BY Average_Rating DESC;
```

Figure 36: DML for average rating for Each Book from reviews

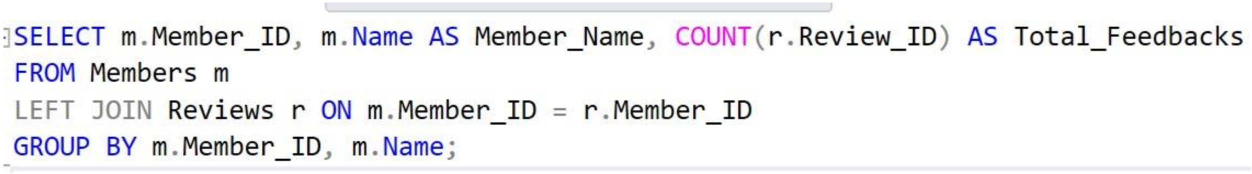


	Book_ID	Book_Name	Average_Rating
1	4	Tech Innovations	5.000000
2	5	Natural Wonders	5.000000
3	6	Cityscapes	5.000000
4	7	Mystical Realms	5.000000
5	1	The Art of Programming	5.000000
6	2	History Unfolded	4.000000
7	3	Into the Wilderness	4.000000

Figure 37: Results for average rating for Each Book from reviews

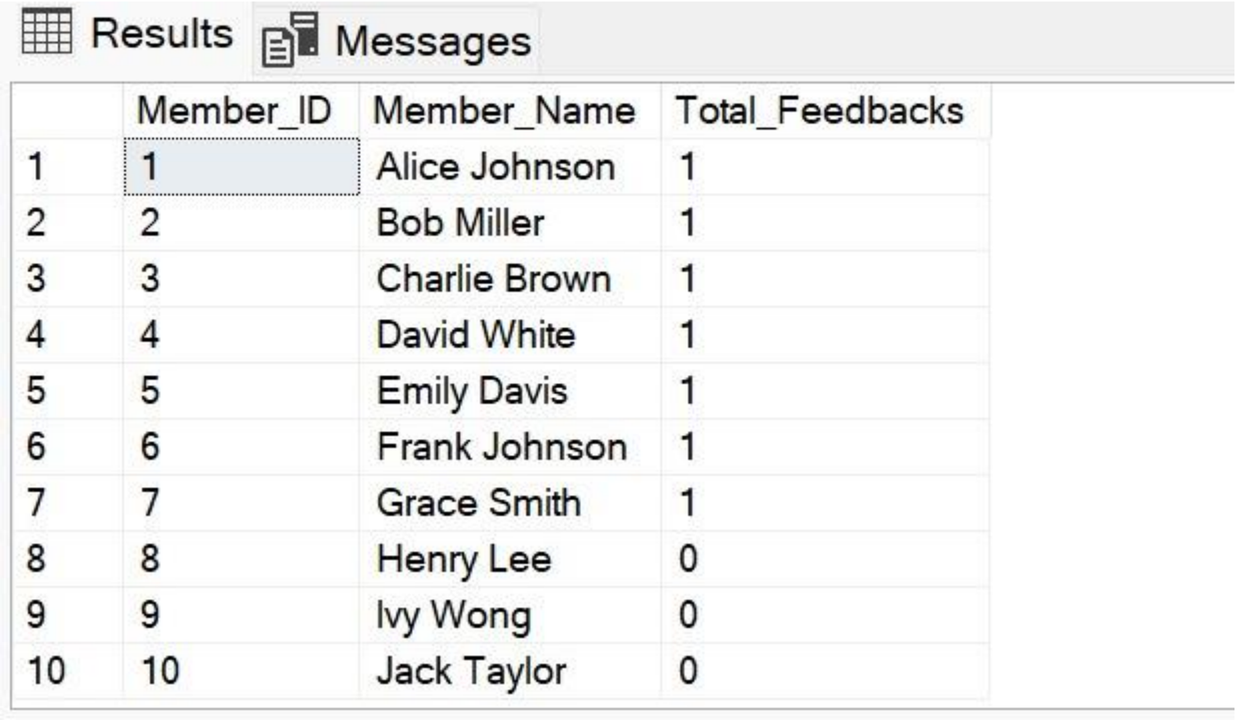
This query calculates the average rating for each book by Joining the Reviews and Books tables, grouping the results by Book_ID and Book_Name. The AVG() function computes the average rating, providing valuable insights into the overall reception of each book. Sorting the results in descending order by Average_Rating enables easy identification of the highest-rated books, aiding in promotional efforts and inventory management.

4. Total Feedbacks Received by Each Member



```sql
SELECT m.Member_ID, m.Name AS Member_Name, COUNT(r.Review_ID) AS Total_Feedbacks
FROM Members m
LEFT JOIN Reviews r ON m.Member_ID = r.Member_ID
GROUP BY m.Member_ID, m.Name;
```

Figure 38: DML for Total feedbacks received by each member

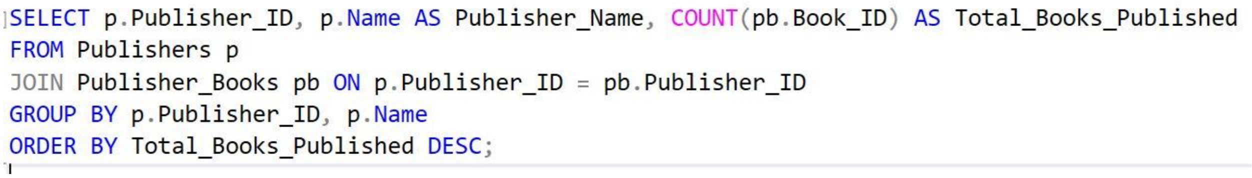


	Member_ID	Member_Name	Total_Feedbacks
1	1	Alice Johnson	1
2	2	Bob Miller	1
3	3	Charlie Brown	1
4	4	David White	1
5	5	Emily Davis	1
6	6	Frank Johnson	1
7	7	Grace Smith	1
8	8	Henry Lee	0
9	9	Ivy Wong	0
10	10	Jack Taylor	0

Figure 39: Results for Total feedbacks received by each member

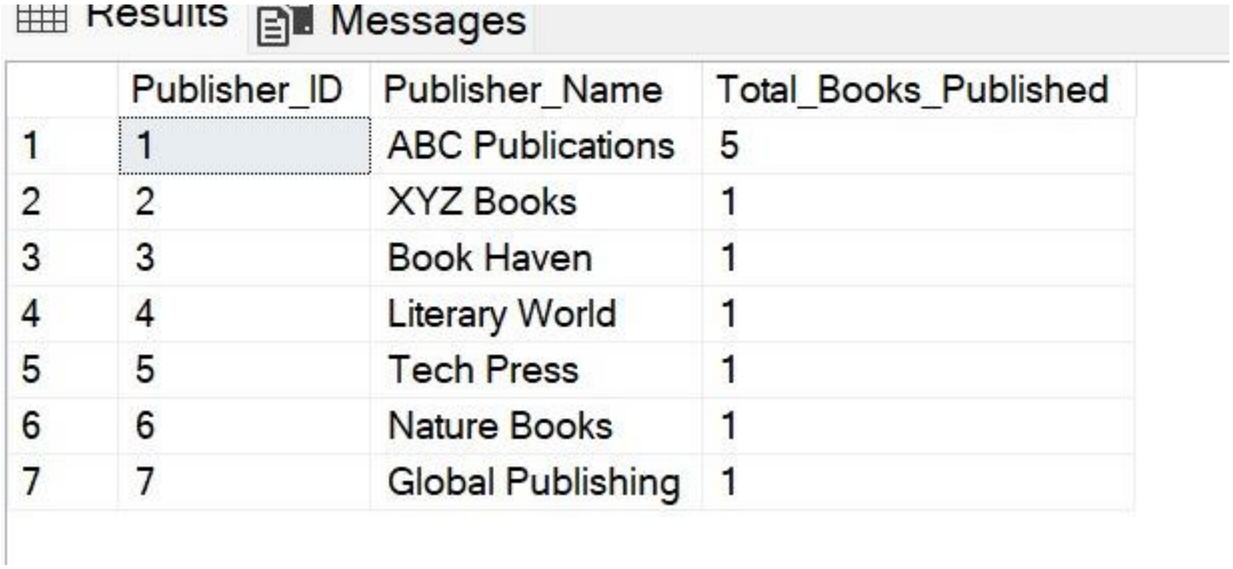
This query utilizes a LEFT JOIN between Members and Reviews, grouping the results by Member_ID and Member_Name. The COUNT() function tallies the total number of feedback entries (Review_ID) for each member. The result offers a comprehensive overview of member engagement and feedback participation. This information is valuable for recognizing active members and tailoring engagement strategies based on their feedback history.

5. Total Books Ordered by Each Publisher



```sql
SELECT p.Publisher_ID, p.Name AS Publisher_Name, COUNT(pb.Book_ID) AS Total_Books_Published
FROM Publishers p
JOIN Publisher_Books pb ON p.Publisher_ID = pb.Publisher_ID
GROUP BY p.Publisher_ID, p.Name
ORDER BY Total_Books_Published DESC;
```

Figure 40: DML for Total Books ordered by Each Publisher

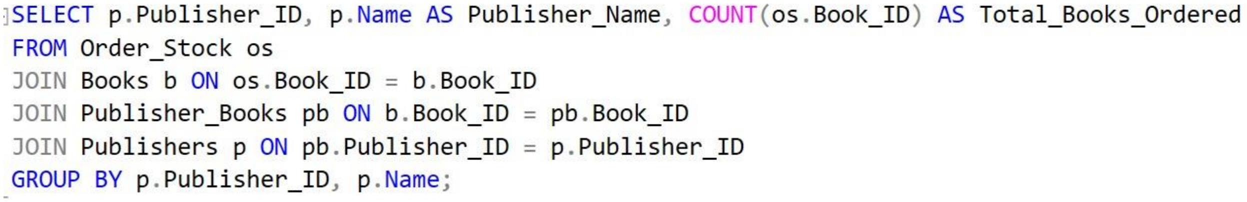


	Publisher_ID	Publisher_Name	Total_Books_Published
1	1	ABC Publications	5
2	2	XYZ Books	1
3	3	Book Haven	1
4	4	Literary World	1
5	5	Tech Press	1
6	6	Nature Books	1
7	7	Global Publishing	1

Figure 41: Results for Total Books ordered by Each Publisher

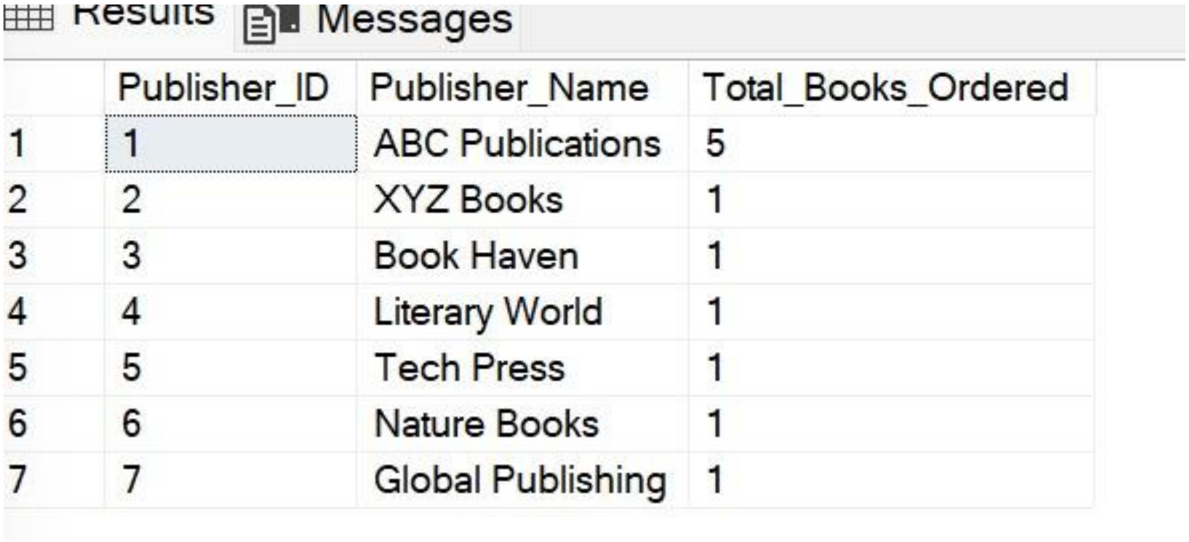
In this query, the integration of Publishers and Publisher_Books is achieved through a JOIN operation based on Publisher_ID. The COUNT() function is then utilized to meticulously calculate the total number of books published by each individual publisher. The outcome is meticulously arranged in descending order based on Total_Books_Published, showcasing Publisher_ID, Publisher_Name, and the corresponding count. This inquiry offers an exhaustive panorama of the publishing landscape, providing invaluable insights for assessing the prolificacy of each publisher. It serves as a compass for stakeholders to pinpoint key contributors in the dynamic book market.

6. Total Quantity of Books Ordered by Each Publisher



```
SELECT p.Publisher_ID, p.Name AS Publisher_Name, COUNT(os.Book_ID) AS Total_Books_Ordered
FROM Order_Stock os
JOIN Books b ON os.Book_ID = b.Book_ID
JOIN Publisher_Books pb ON b.Book_ID = pb.Book_ID
JOIN Publishers p ON pb.Publisher_ID = p.Publisher_ID
GROUP BY p.Publisher_ID, p.Name;
```

Figure 42: DML for Total Quantity of Books ordered by each Publisher

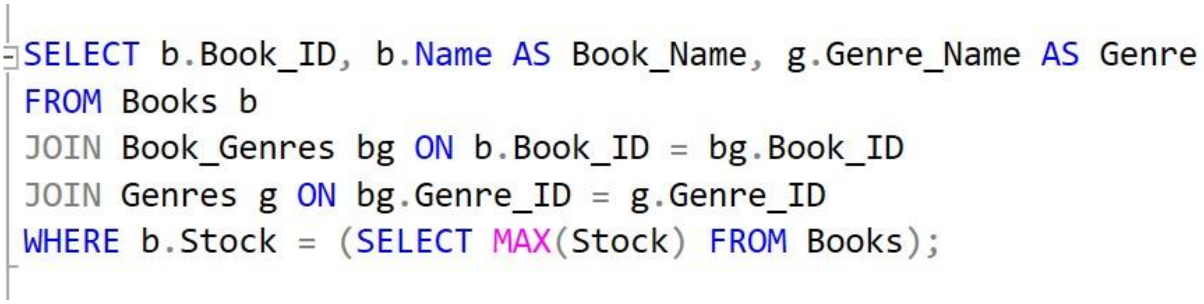


	Publisher_ID	Publisher_Name	Total_Books_Ordered
1	1	ABC Publications	5
2	2	XYZ Books	1
3	3	Book Haven	1
4	4	Literary World	1
5	5	Tech Press	1
6	6	Nature Books	1
7	7	Global Publishing	1

Figure 43: Results for Total Quantity of Books ordered by each Publisher

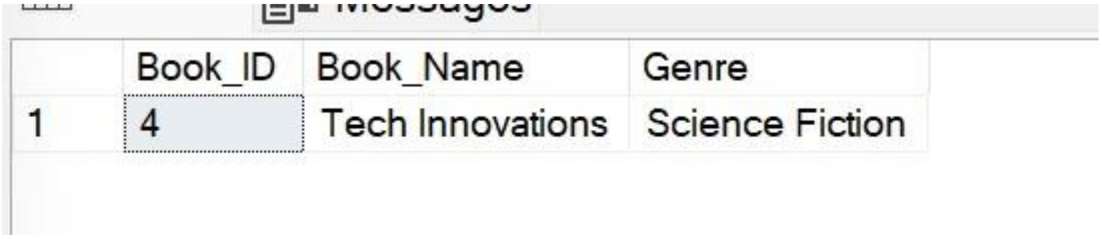
Expanding on the previous query's groundwork, this iteration introduces the SUM() function to meticulously calculate the comprehensive quantity of books ordered from each publisher. The synergy of Order_Stock, Books, Publisher_Books, and Publishers facilitates an intricate understanding of the sales performance landscape for publishers. The meticulously organized output, grouped by Publisher_ID and Publisher_Name, serves as a robust resource for publishers to gauge their market impact. Armed with this information, publishers can make judicious and informed decisions, shaping their strategies for future success.

7. Genre of Most Stocked Book



```sql
SELECT b.Book_ID, b.Name AS Book_Name, g.Genre_Name AS Genre
FROM Books b
JOIN Book_Genres bg ON b.Book_ID = bg.Book_ID
JOIN Genres g ON bg.Genre_ID = g.Genre_ID
WHERE b.Stock = (SELECT MAX(Stock) FROM Books);
```

Figure 44: DML for Genre of Most Stocked Book

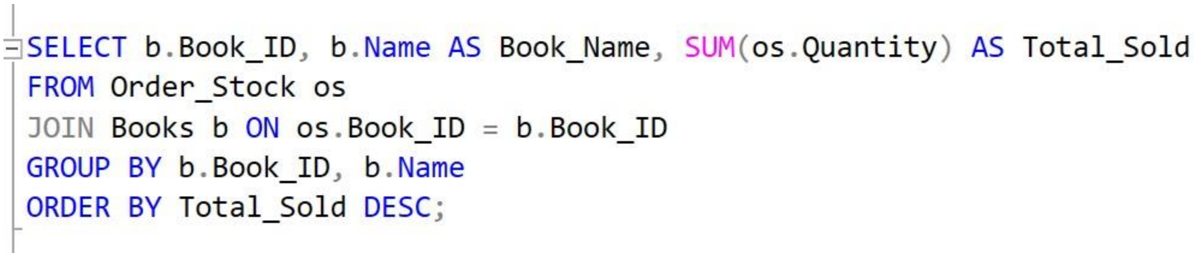


	Book_ID	Book_Name	Genre
1	4	Tech Innovations	Science Fiction

Figure 45: Results for Genre of Most Stocked Book

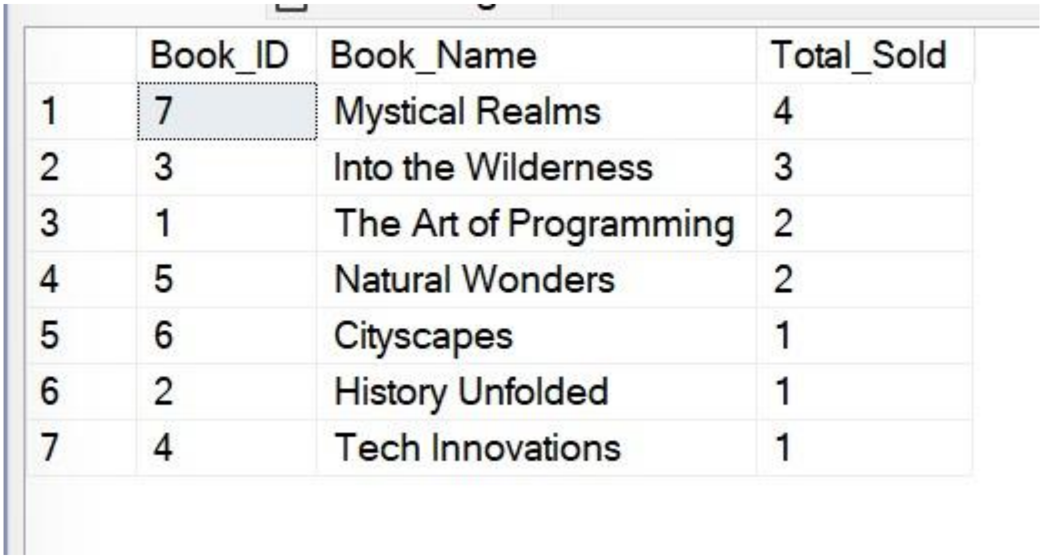
This query embarks on a journey to unveil the dominant genre within the book inventory. By skillfully joining the tables Books, Book_Genres, and Genres, it navigates through the data landscape. The WHERE clause, complemented by a subquery selecting MAX(Stock) from Books, acts as a compass to pinpoint the genre reigning supreme in terms of stock. The result is an illuminating ensemble of Book_ID, Book_Name, and Genre_Name, providing a crucial lens for inventory management. Publishers can leverage this insight to streamline their catalog and align it with prevailing market trends, fostering strategic decision-making.

8. Total Sold Quantity for Each Book



```
SELECT b.Book_ID, b.Name AS Book_Name, SUM(os.Quantity) AS Total_Sold
FROM Order_Stock os
JOIN Books b ON os.Book_ID = b.Book_ID
GROUP BY b.Book_ID, b.Name
ORDER BY Total_Sold DESC;
```

Figure 46:DML for Total Sold Quantity for each book

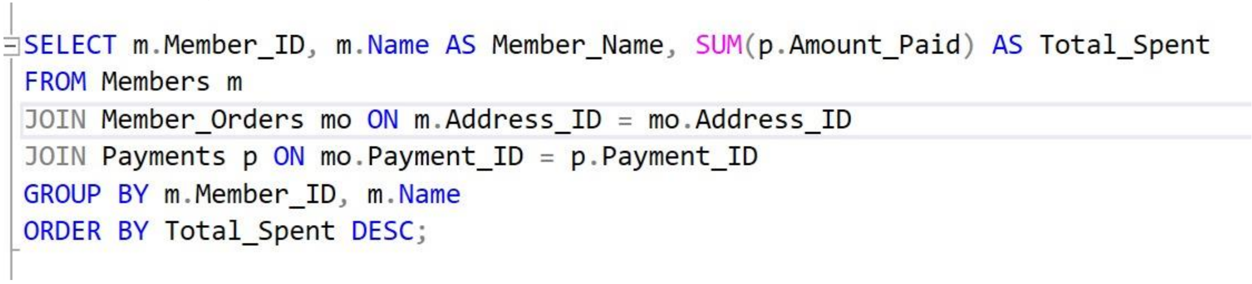


	Book_ID	Book_Name	Total_Sold
1	7	Mystical Realms	4
2	3	Into the Wilderness	3
3	1	The Art of Programming	2
4	5	Natural Wonders	2
5	6	Cityscapes	1
6	2	History Unfolded	1
7	4	Tech Innovations	1

Figure 47: Results for Total Sold Quantity for each book

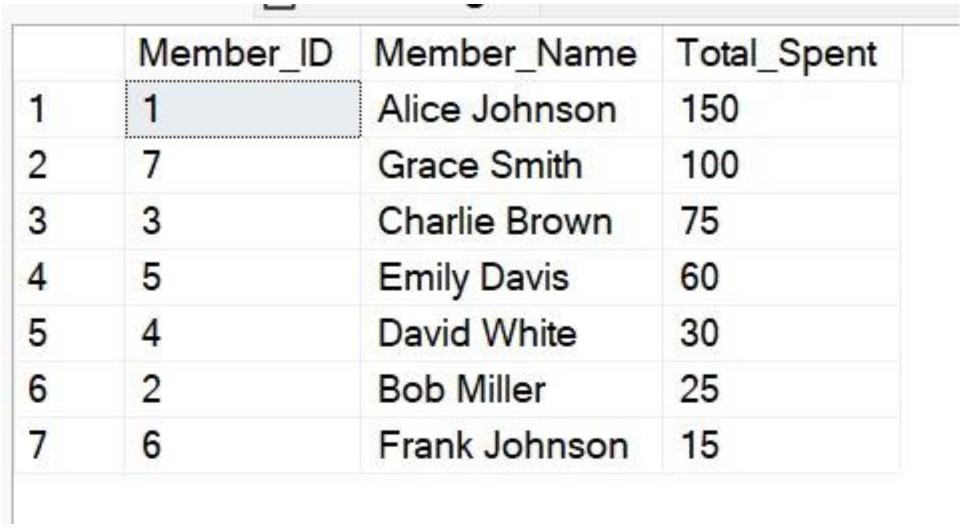
This intricate SQL query meticulously analyzes the intersection of Order_Stock and Books, intricately woven together through a JOIN operation on Book_ID. The SUM() function meticulously tallies the quantity of each book sold. The grouping by Book_ID and Book_Name ensures a detailed breakdown of sales for individual books. By ordering the results in descending order based on Total_Sold, the query provides a comprehensive overview of the most popular books, guiding inventory management and marketing efforts. This data-driven approach aids in understanding customer preferences, optimizing stock levels, and strategizing promotions for high-demand books, ultimately enhancing the bookstore's operational efficiency and profitability.

9. Total Spent by Each Member:



```
SELECT m.Member_ID, m.Name AS Member_Name, SUM(p.Amount_Paid) AS Total_Spent
FROM Members m
JOIN Member_Orders mo ON m.Address_ID = mo.Address_ID
JOIN Payments p ON mo.Payment_ID = p.Payment_ID
GROUP BY m.Member_ID, m.Name
ORDER BY Total_Spent DESC;
```

Figure 48: DML for Total Spent by each member

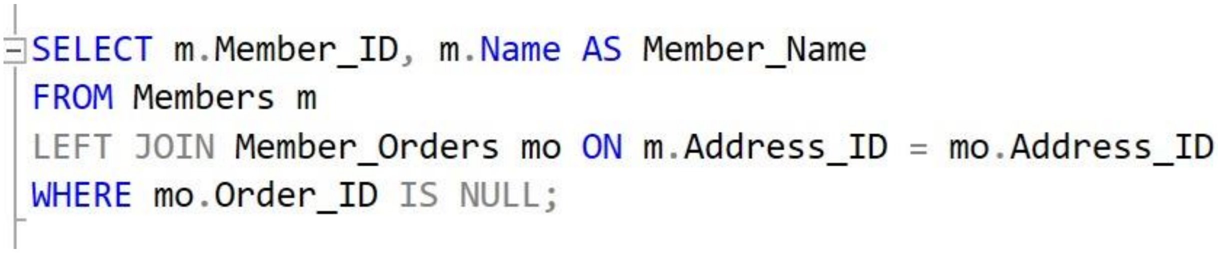


	Member_ID	Member_Name	Total_Spent
1	1	Alice Johnson	150
2	7	Grace Smith	100
3	3	Charlie Brown	75
4	5	Emily Davis	60
5	4	David White	30
6	2	Bob Miller	25
7	6	Frank Johnson	15

Figure 49: Results for Total Spent by each member

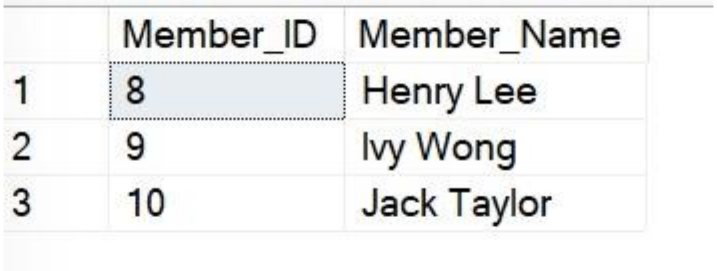
This intricate query orchestrates a synergy between Members, Member_Orders, and Payments through JOIN operations based on shared Address_ID and Payment_ID. The SUM() function diligently computes the total amount paid by each member, resulting in a comprehensive overview. The output, structured by Member_ID, Member_Name, and Total_Spent, is meticulously ordered in descending fashion by Total_Spent. This query serves as a financial compass for stakeholders, shedding light on the most financially engaged members and aiding in strategic decision-making regarding member engagement and loyalty programs.

10. Members with No Orders



```
SELECT m.Member_ID, m.Name AS Member_Name
FROM Members m
LEFT JOIN Member_Orders mo ON m.Address_ID = mo.Address_ID
WHERE mo.Order_ID IS NULL;
```

Figure 50: DML for members with no orders

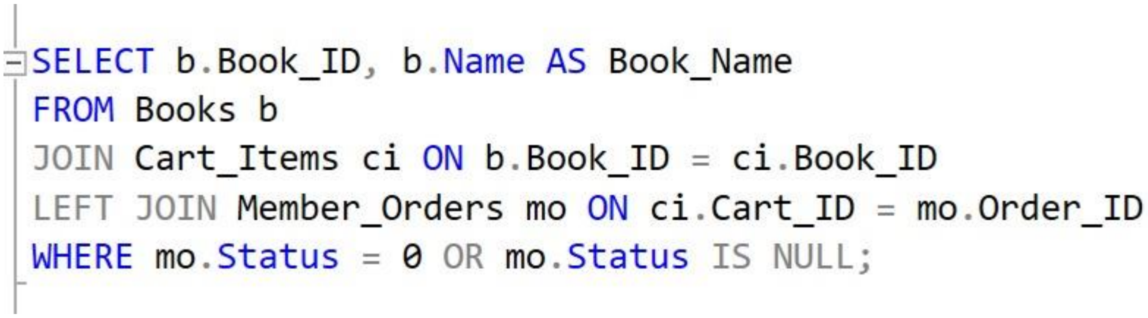


	Member_ID	Member_Name
1	8	Henry Lee
2	9	Ivy Wong
3	10	Jack Taylor

Figure 51: Results for member with no order

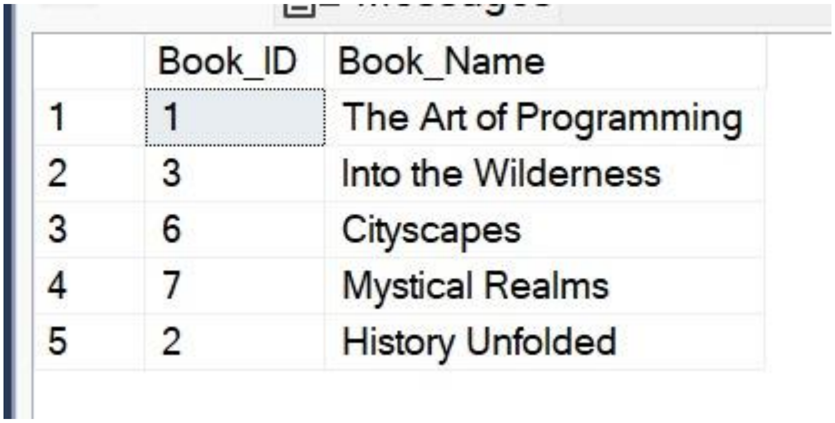
In this insightful query, Members are scrutinized alongside Member_Orders in a LEFT JOIN operation based on Address_ID. The WHERE clause is then employed to filter members with no associated orders, specifically those where Order_ID is NULL. The result is a list of Member_IDs and Member_Names, providing valuable insights into members who have not initiated any orders. This information is pivotal for targeted outreach and engagement strategies to encourage order placements and enhance overall member activity.

11. Books in Carts with Uncompleted Orders:



```
SELECT b.Book_ID, b.Name AS Book_Name
FROM Books b
JOIN Cart_Items ci ON b.Book_ID = ci.Book_ID
LEFT JOIN Member_Orders mo ON ci.Cart_ID = mo.Order_ID
WHERE mo.Status = 0 OR mo.Status IS NULL;
```

Figure 52: DML of Books in carts with uncompleted orders

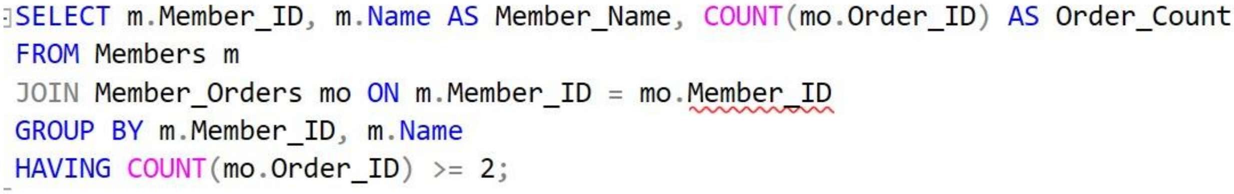


	Book_ID	Book_Name
1	1	The Art of Programming
2	3	Into the Wilderness
3	6	Cityscapes
4	7	Mystical Realms
5	2	History Unfolded

Figure 53: Results Books in carts with uncompleted orders

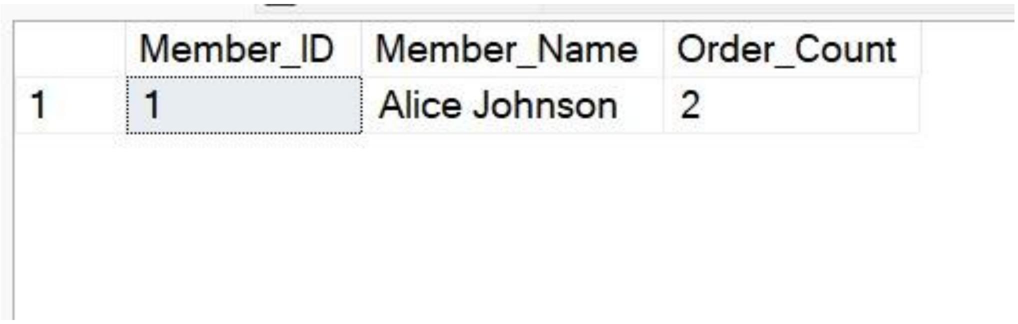
This query intricately intertwines Books, Cart_Items, and Member_Orders, navigating through their relationships via JOIN operations. The LEFT JOIN and WHERE clauses filter out books associated with carts lacking completed orders (Status = 0 or NULL). The result includes Book_IDs and Book_Names, spotlighting books in limbo between carts and uncompleted orders. This information proves invaluable for inventory management, helping identify books that may have piqued interest but haven't transitioned into completed transactions.

12. Members with Two or More Orders



```
SELECT m.Member_ID, m.Name AS Member_Name, COUNT(mo.Order_ID) AS Order_Count
FROM Members m
JOIN Member_Orders mo ON m.Member_ID = mo.Member_ID
GROUP BY m.Member_ID, m.Name
HAVING COUNT(mo.Order_ID) >= 2;
```

Figure 54: DML for Members with two or more orders



	Member_ID	Member_Name	Order_Count
1	1	Alice Johnson	2

Figure 55: Results for members with two or more orders

In this multifaceted query, Members and Member_Orders collaborate through JOIN operations on Member_ID. The COUNT() function plays a pivotal role in determining the number of orders each member has made. The HAVING clause filters the results to spotlight members with two or more orders. The output, structured by Member_ID, Member_Name, and Order_Count, acts as a beacon for identifying highly engaged members, offering crucial insights into recurring customer behavior for personalized engagement strategies.

Workload Matrix

Part	Component	Student Name: CHURILOV MIKHAIL	Student Name: RAVIN A/L KANAGARAJAN	Student Name: NASTARAN ESMAEIL ZADEH	Student Name: VOORISHTA GOPAUL	Total
2	a) Database Schema	**25%**	**25%**	**25%**	**25%**	**100%**
2	b) SQL-Data Definition Language (DDL)	**0%**	**50%**	**0%**	**50%**	**100%**
2	c) SQL-Data Manipulation Language (DML)	**50%**	**0%**	**50%**	**0%**	**100%**