Create three tables: Book, Customer, and Order, with appropriate attributes and insert some records to each.

-- Create the Book table

CREATE TABLE Book (

BookID INT PRIMARY KEY,

Title VARCHAR(100),

Author VARCHAR(100),

Genre VARCHAR(50),

Price DECIMAL(10, 2)

);

-- Create the Customer table

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20)

);

-- Create the Order1 table

CREATE TABLE Order1 (

OrderID INT PRIMARY KEY,

CustomerID INT,

BookID INT,

Quantity INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (BookID) REFERENCES Book(BookID)

);

-- Insert sample data into the Book table

INSERT INTO Book (BookID, Title, Author, Genre, Price)

VALUES

(1, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 12.99),

(2, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 10.99),

(3, '1984', 'George Orwell', 'Science Fiction', 9.99),

(4, 'Pride and Prejudice', 'Jane Austen', 'Romance', 11.99);

-- Insert sample data into the Customer table

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone)

VALUES

(1, 'John', 'Doe', 'john.doe@example.com', '123-456-7890'),

(2, 'Jane', 'Smith', 'jane.smith@example.com', '234-567-8901'),

(3, 'Michael', 'Johnson', 'michael.johnson@example.com', '345-678-9012');

-- Insert sample data into the Order table

INSERT INTO Order1 (OrderID, CustomerID, BookID, Quantity, OrderDate)

VALUES

(1, 1, 1, 2, '2024-04-01'),

(2, 2, 2, 1, '2024-04-02'),

(3, 3, 3, 3, '2024-04-03'),

(4, 1, 4, 1, '2024-04-04');

1create a scalar user-defined function that calculates the total price of an order based on the quantity and the price of an ordered book.

-- Scalar function to calculate total order price

CREATE FUNCTION CalculateTotalPrice (@quantity INT, @price DECIMAL(10, 2))

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @totalPrice DECIMAL(10, 2);

SET @totalPrice = @quantity \* @price;

RETURN @totalPrice;

END;

Now, using the created function, write a query to Calculate the total price of the order with OrderID = 2.

SELECT dbo.CalculateTotalPrice(Quantity, Price) AS TotalPrice

FROM Order1

JOIN Book ON Order1.BookID = Book.BookID

WHERE OrderID = 2;

2Create a scalar function that calculates the total number of characters in a book title.

-- Scalar function to calculate the total number of characters in a book title

CREATE FUNCTION CalculateTitleLength (@title VARCHAR(100))

RETURNS INT

AS

BEGIN

DECLARE @titleLength INT;

SET @titleLength = LEN(@title);

RETURN @titleLength;

END;

Now, using the created function, write a query to calculate the length of a book titled , for example, "To Kill a Mockingbird".

SELECT dbo.CalculateTitleLength('Pride and Prejudice') AS TitleLength;

3Create a scalar function that calculates the total number of orders placed by a customer.

-- Scalar function to calculate the total number of orders placed by a customer

CREATE FUNCTION CountOrdersByCustomer (@customerID INT)

RETURNS INT

AS

BEGIN

DECLARE @orderCount INT;

SELECT @orderCount = COUNT(\*) FROM [Order] WHERE CustomerID = @customerID;

RETURN @orderCount;

END;

Now, using the created function, write a query to find out the number of orders placed by the customer with CustomerID = 1.

SELECT dbo.CountOrdersByCustomer(1) AS TotalOrders;

4Create a scalar function to calculate the total price of all orders placed on a specific date.

-- Scalar function to calculate the total price of all orders placed on a specific date

CREATE FUNCTION CalculateTotalPriceByDate (@orderDate DATE)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @totalPrice DECIMAL(10, 2);

SELECT @totalPrice = SUM(Quantity \* Price)

FROM Order1

JOIN Book ON Order1.BookID = Book.BookID

WHERE OrderDate = @orderDate;

RETURN @totalPrice;

END;

Now, using the created function, write a query to find out the total price of all orders placed on April 3rd, 2024.

SELECT dbo.CalculateTotalPriceByDate('2024-04-03') AS TotalPriceOnDate;

5Create an inline table-valued function to retrieve all orders placed by a specific customer.

-- Inline table-valued function to retrieve all orders placed by a specific customer

CREATE FUNCTION GetOrdersByCustomer (@customerID INT)

RETURNS TABLE

AS

RETURN

(

SELECT \* FROM Order1 WHERE CustomerID = @customerID

);

Now, using the created function, write a query to retrieve all orders placed by the customer with CustomerID = 1.

SELECT \* FROM dbo.GetOrdersByCustomer(1);

6Create an inline table-valued function to retrieve all customers who placed orders on a specific date.

-- Inline table-valued function to retrieve all customers who placed orders on a specific date

CREATE FUNCTION GetCustomersByOrderDate (@orderDate DATE)

RETURNS TABLE

AS

RETURN

(

SELECT DISTINCT Customer.\*

FROM Customer

JOIN Order1 ON Customer.CustomerID = Order1.CustomerID

WHERE Order1.OrderDate = @orderDate

);

Now, using the created function, write a query to retrieve all customers who placed orders on April 3rd, 2024.

SELECT \* FROM dbo.GetCustomersByOrderDate('2024-04-03');

7Create a multi-statement table-valued function to retrieve all orders with their corresponding book details.

-- Multi-statement table-valued function to retrieve all orders with book details

CREATE FUNCTION GetOrdersWithBookDetails ()

RETURNS @OrderDetails TABLE (

OrderID INT,

CustomerID INT,

BookID INT,

Quantity INT,

OrderDate DATE,

Title VARCHAR(100),

Author VARCHAR(100),

Genre VARCHAR(50),

Price DECIMAL(10, 2)

)

AS

BEGIN

INSERT INTO @OrderDetails (OrderID, CustomerID, BookID, Quantity, OrderDate, Title, Author, Genre, Price)

SELECT Order1.OrderID, Order1.CustomerID, Order1.BookID, Order1.Quantity, Order1.OrderDate, Book.Title, Book.Author, Book.Genre, Book.Price

FROM Order1

JOIN Book ON Order1.BookID = Book.BookID;

RETURN;

END;

Now, using the created function, write a query to retrieve all orders with their corresponding book details.

SELECT \* FROM dbo.GetOrdersWithBookDetails();

8Create a multi-statement table-valued function to retrieve all customers along with the total number of orders they have placed.

-- Multi-statement table-valued function to retrieve all customers along with the total number of orders

CREATE FUNCTION GetCustomersWithOrderCount ()

RETURNS @CustomerOrderCount TABLE (

CustomerID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20),

OrderCount INT

)

AS

BEGIN

INSERT INTO @CustomerOrderCount (CustomerID, FirstName, LastName, Email, Phone, OrderCount)

SELECT Customer.CustomerID, Customer.FirstName, Customer.LastName, Customer.Email, Customer.Phone, COUNT(Order1.OrderID) AS OrderCount

FROM Customer

LEFT JOIN Order1 ON Customer.CustomerID = Order1.CustomerID

GROUP BY Customer.CustomerID, Customer.FirstName, Customer.LastName, Customer.Email, Customer.Phone;

RETURN;

END;

Now, using the created function, write a query to retrieve all customers along with the total number of orders they have placed.

SELECT \* FROM dbo.GetCustomersWithOrderCount();

9Alter the Scalar function created to calculate total order price and include a discount percentage as an additional parameter.

ALTER FUNCTION CalculateTotalPrice

(

@quantity INT,

@price DECIMAL(10, 2),

@discountPercentage DECIMAL(5, 2)

)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @totalPrice DECIMAL(10, 2);

SET @totalPrice = (@quantity \* @price) \* (1 - (@discountPercentage / 100));

RETURN @totalPrice;

END;

Now, use this altered function to find out the “Total Price With Discount” for OrderID=1 applying 10% discount.

-- Calculate the total price of the order with OrderID = 1 with a 10% discount

SELECT dbo.CalculateTotalPrice(Quantity, Price, 10) AS TotalPriceWithDiscount

FROM Order1

JOIN Book ON Order1.BookID = Book.BookID

WHERE OrderID = 1;

10Alter the inline table-valued function created above (for retrieving all customers who placed orders on a specific date) by adding another parameter for the minimum number of orders.

-- Alter the inline table-valued function to include an additional parameter for the minimum number of orders

ALTER FUNCTION GetCustomersByOrderDate

(

@orderDate DATE,

@minOrders INT

)

RETURNS TABLE

AS

RETURN

(

SELECT Customer.\*, COUNT(Order1.OrderID) AS OrderCount

FROM Customer

JOIN Order1 ON Customer.CustomerID = Order1.CustomerID

WHERE Order1.OrderDate = @orderDate

GROUP BY Customer.CustomerID, Customer.FirstName, Customer.LastName, Customer.Email, Customer.Phone

HAVING COUNT(Order1.OrderID) >= @minOrders

);

Now, use this altered function to find out customers who placed at least 1 order on April 3rd, 2024.

-- Retrieve all customers who placed at least 1 order on April 3rd, 2024 using the altered function

SELECT \* FROM dbo.GetCustomersByOrderDate('2024-04-03', 2);

11Alter the Multi-statement table-valued function created above (for retrieving all orders with their corresponding book details) by including an additional parameter for filtering orders by genre.

-- Alter the multi-statement table-valued function to include an additional parameter for filtering orders by genre.

ALTER FUNCTION GetOrdersWithBookDetails

(

@genreFilter VARCHAR(50) = NULL

)

RETURNS @OrderDetails TABLE (

OrderID INT,

CustomerID INT,

BookID INT,

Quantity INT,

OrderDate DATE,

Title VARCHAR(100),

Author VARCHAR(100),

Genre VARCHAR(50),

Price DECIMAL(10, 2)

)

AS

BEGIN

INSERT INTO @OrderDetails (OrderID, CustomerID, BookID, Quantity, OrderDate, Title, Author, Genre, Price)

SELECT Order1.OrderID, Order1.CustomerID, Order1.BookID, Order1.Quantity, Order1.OrderDate, Book.Title, Book.Author, Book.Genre, Book.Price

FROM Order1

JOIN Book ON Order1.BookID = Book.BookID

WHERE @genreFilter IS NULL OR Book.Genre = @genreFilter;

RETURN;

END;

Now, use this altered function to retrieve all orders with book details for the "Fiction" genre.

-- Retrieve all orders with book details for the "Fiction" genre using the altered function

SELECT \* FROM dbo.GetOrdersWithBookDetails('Fiction');

12At last, drop created functions.

DROP FUNCTION IF EXISTS CalculateTotalPrice;

DROP FUNCTION IF EXISTS ScalarGetCustomersByOrderDate;

DROP FUNCTION IF EXISTS GetOrdersWithBookDetails;