Sri Lanka Institute of Information Technology

Faculty of Computing

Year 2 – Semester 1 (2025)

IT2140 - Database Design and Development

Assignment 01 - Part 02: Relational Schema Design, SQL Implementation, Queries, and Advanced SQL Features

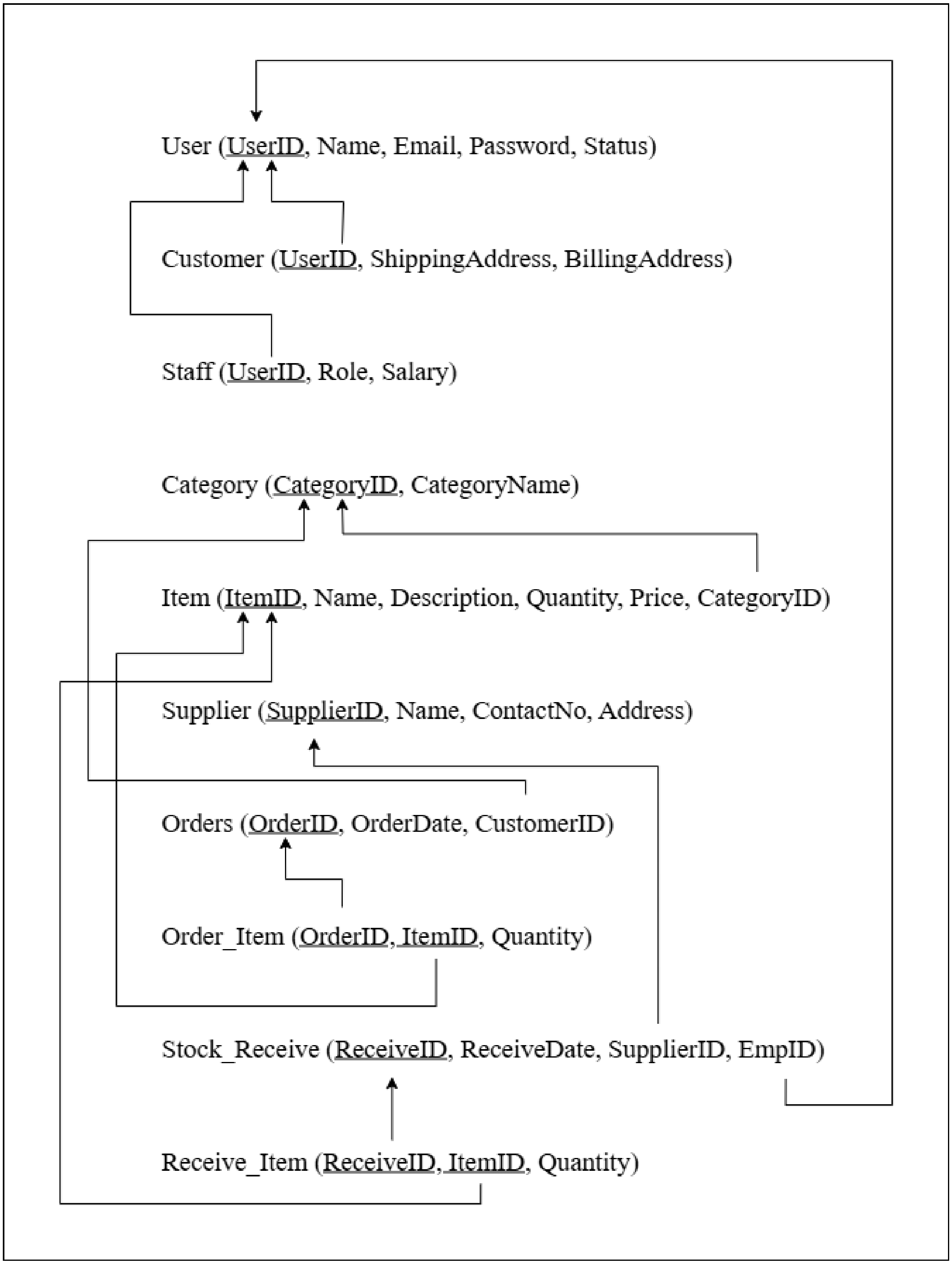
Project Title: Web-based Inventory Control System

Group ID: 2025-Y2-S1-MLB-B10G2-06

Registration Number	**Name**
IT24102699	Mummullage B.U.T
IT24102758	Priyamalka W D N
IT24102784	Panagodage N.M.H
IT24102773	Siriwardane K.D.D.D
IT24102795	Alahakoon A. M. J. P
IT24102798	Sooriyabandara U.R.G.W.K

Part A – Mapping EER to Relational Schema

1. Converting EER Diagram into a Relational Schema →



User (UserID, Name, Email, Password, Status)

Customer (UserID, ShippingAddress, BillingAddress)

Staff (UserID, Role, Salary)

Category (CategoryID, CategoryName)

Item (ItemID, Name, Description, Quantity, Price, CategoryID)

Supplier (SupplierID, Name, ContactNo, Address)

Orders (OrderID, OrderDate, CustomerID)

Order_Item (OrderID, ItemID, Quantity)

Stock_Receive (ReceiveID, ReceiveDate, SupplierID, EmpID)

Receive_Item (ReceiveID, ItemID, Quantity)

2. Showing PKs, FKs, and Constraints →

Table Name	Primary Key	Foreign Key	Logical Constraints
User	UserID	-	-
Customer	UserID	UserID	-
Staff	UserID	UserID	CHECK (Salary >= 0)
Category	CategoryID	-	CategoryName UNIQUE NOT NULL
Item	ItemID	CategoryID	Name NOT NULL, CHECK (Quantity >= 0), CHECK (Price >= 0)
Supplier	SupplierID	-	Name NOT NULL
Orders	OrderID	CustomerID	OrderDate NOT NULL
Order_Item	OrderID, ItemID	OrderID, ItemID	Quantity NOT NULL, CHECK (Quantity > 0)
Stock_Receive	ReceiveID	SupplierID, EmpID	ReceiveDate NOT NULL
Receive_Item	ReceiveID, ItemID	ReceiveID, ItemID	Quantity NOT NULL, CHECK (Quantity > 0)

3. Mapping choices for ISA →

For the ISA (is-a) relationship between the **User** superclass and its **Customer** and **Staff** subclasses, we are going to apply Option 1.

This option creates a separate relation for the superclass and for each subclass.

4. Refining the schema →

Schema refinement is the process of analyzing and improving a schema to minimize problems like data redundancy and update anomalies.

The key refinements we used here are:

Removing ISA Hierarchy:

The User generalization was removed to get distinct Customer and Employee tables. This is the most significant refinement, done to improve the clarity and efficiency of the schema.

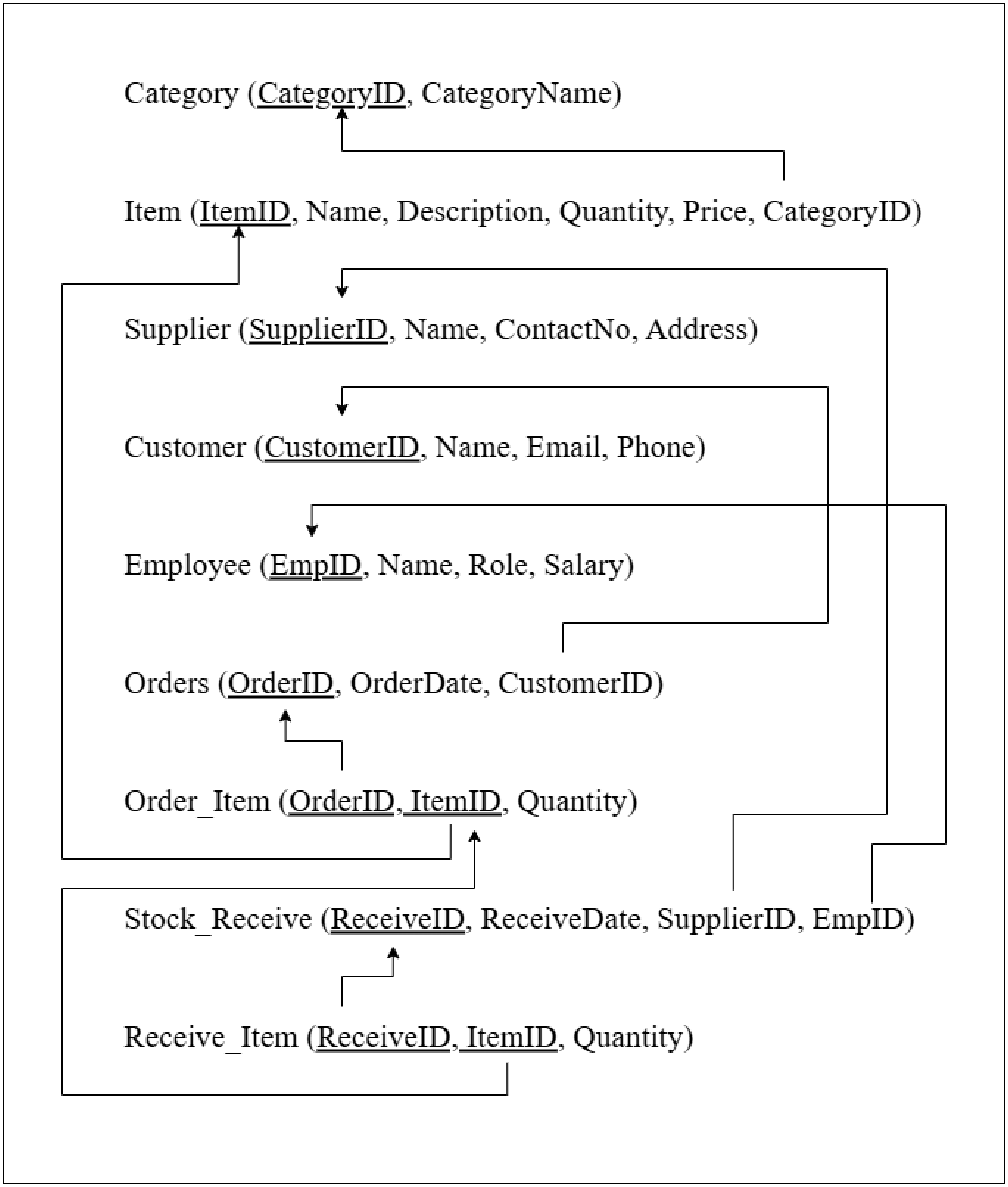
Correct Decomposition:

The final schema correctly separates distinct real-world concepts into different tables like Item, Supplier, Employee. This avoids the problematic attribute grouping that leads to insertion, deletion, and update anomalies, which is the main reason for schema refinement.

Resolution of M:N Relationships:

The many-to-many relationship between Orders and Item was correctly resolved by creating an associative table named Order_Item.

- **Final Relational Schema after applying Schema Refinement →**



Category (CategoryID, CategoryName)

Item (ItemID, Name, Description, Quantity, Price, CategoryID)

Supplier (SupplierID, Name, ContactNo, Address)

Customer (CustomerID, Name, Email, Phone)

Employee (EmpID, Name, Role, Salary)

Orders (OrderID, OrderDate, CustomerID)

Order_Item (OrderID, ItemID, Quantity)

Stock_Receive (ReceiveID, ReceiveDate, SupplierID, EmpID)

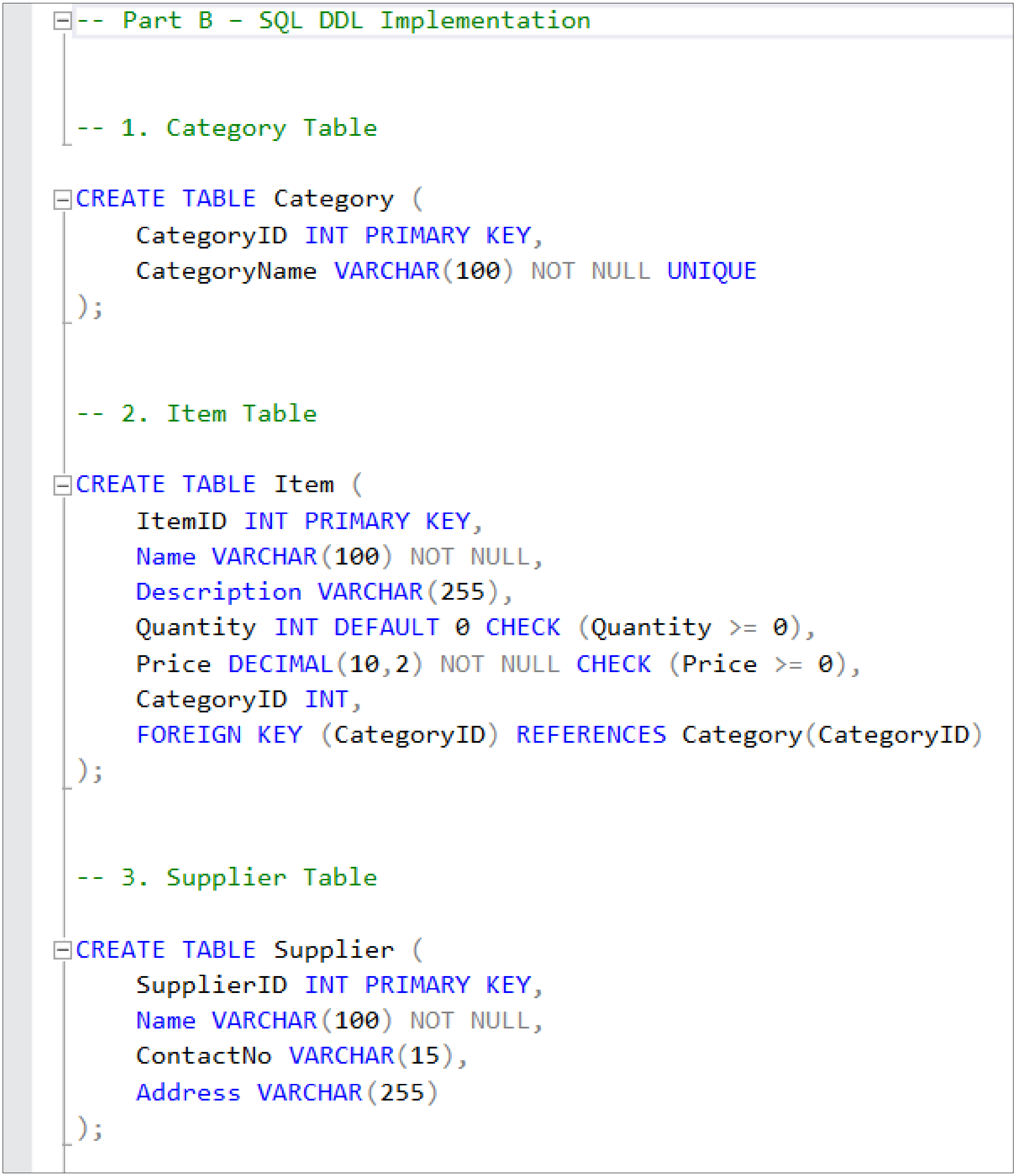
Receive_Item (ReceiveID, ItemID, Quantity)

- **Showing PKs, FKs, and Constraints after applying Schema Refinement →**

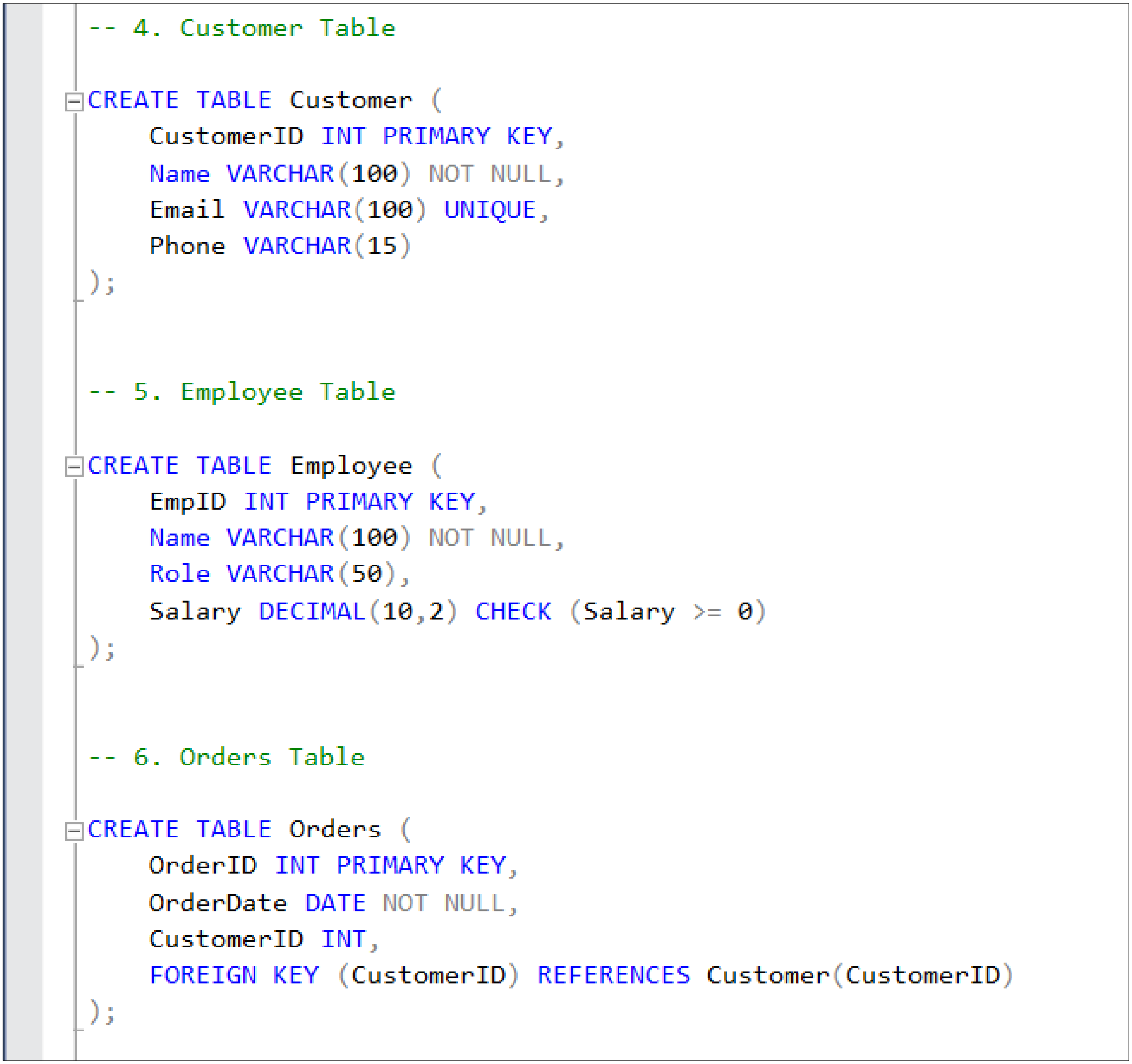
Table Name	**Primary Key**	**Foreign Key**	**Logical Constraints**
Category	CategoryID	-	CategoryName UNIQUE NOT NULL
Item	ItemID	CategoryID	Name NOT NULL, CHECK (Quantity >= 0), Price DECIMAL(10,2), CHECK (Price >= 0)
Supplier	SupplierID	-	Name NOT NULL
Customer	CustomerID	-	Name NOT NULL, Email UNIQUE
Employee	EmpID	-	Name NOT NULL, Salary DECIMAL(10,2), CHECK (Salary >= 0)
Orders	OrderID	CustomerID	OrderDate NOT NUL
Order_Item	OrderID, ItemID	OrderID, ItemID	Quantity NOT NULL, CHECK (Quantity > 0))
Stock_Receive	ReceiveID	SupplierID, EmpID	ReceiveDate NOT NULL
Receive_Item	ReceiveID, ItemID	ReceiveID, ItemID	Quantity NOT NULL, CHECK (Quantity > 0)

Part B – SQL DDL Implementation

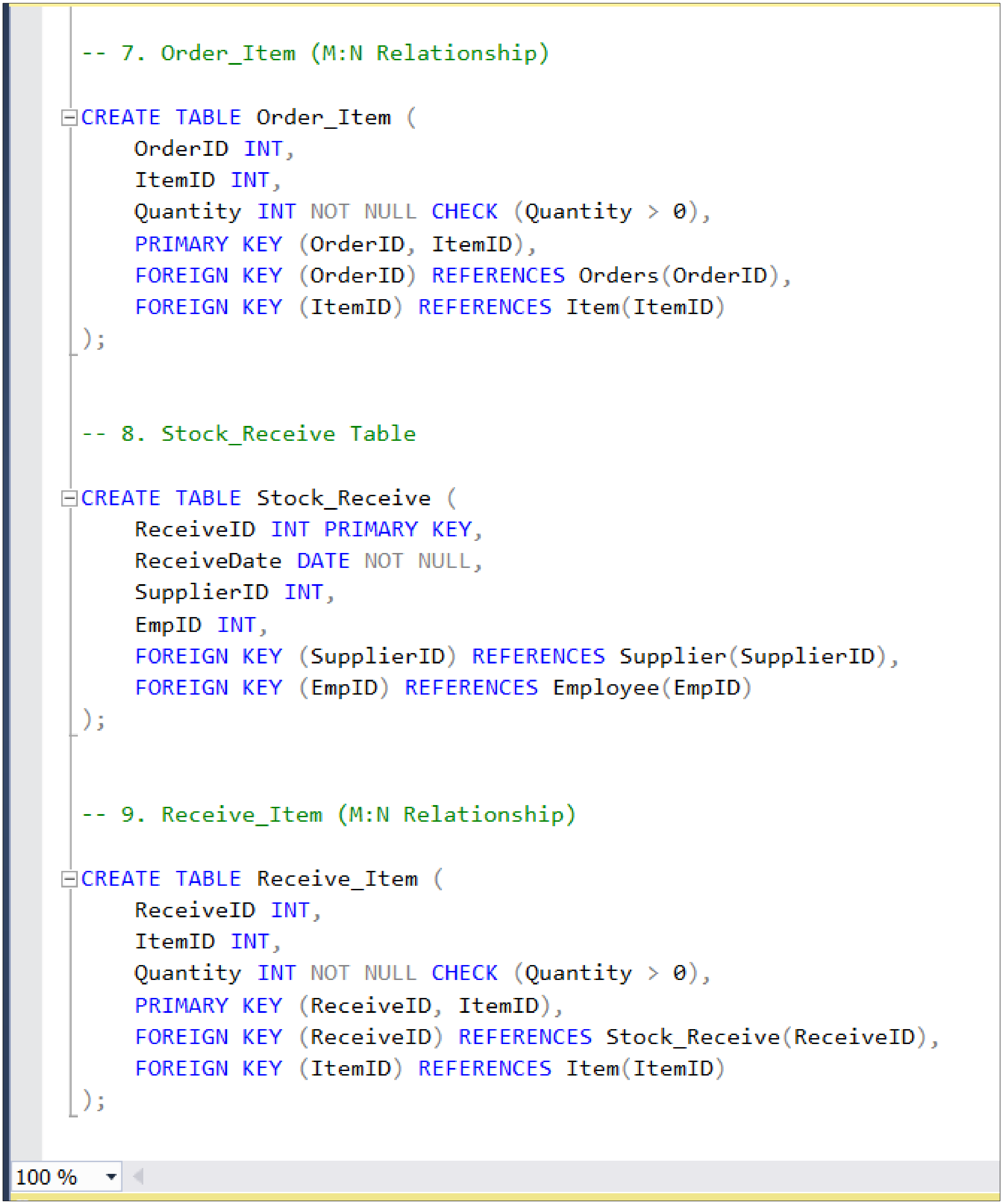
Screenshots of creating tables:



```
-- Part B - SQL DDL Implementation


-- 1. Category Table

CREATE TABLE Category (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(100) NOT NULL UNIQUE
);


-- 2. Item Table

CREATE TABLE Item (
    ItemID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Description VARCHAR(255),
    Quantity INT DEFAULT 0 CHECK (Quantity >= 0),
    Price DECIMAL(10,2) NOT NULL CHECK (Price >= 0),
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);


-- 3. Supplier Table

CREATE TABLE Supplier (
    SupplierID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    ContactNo VARCHAR(15),
    Address VARCHAR(255)
);
```



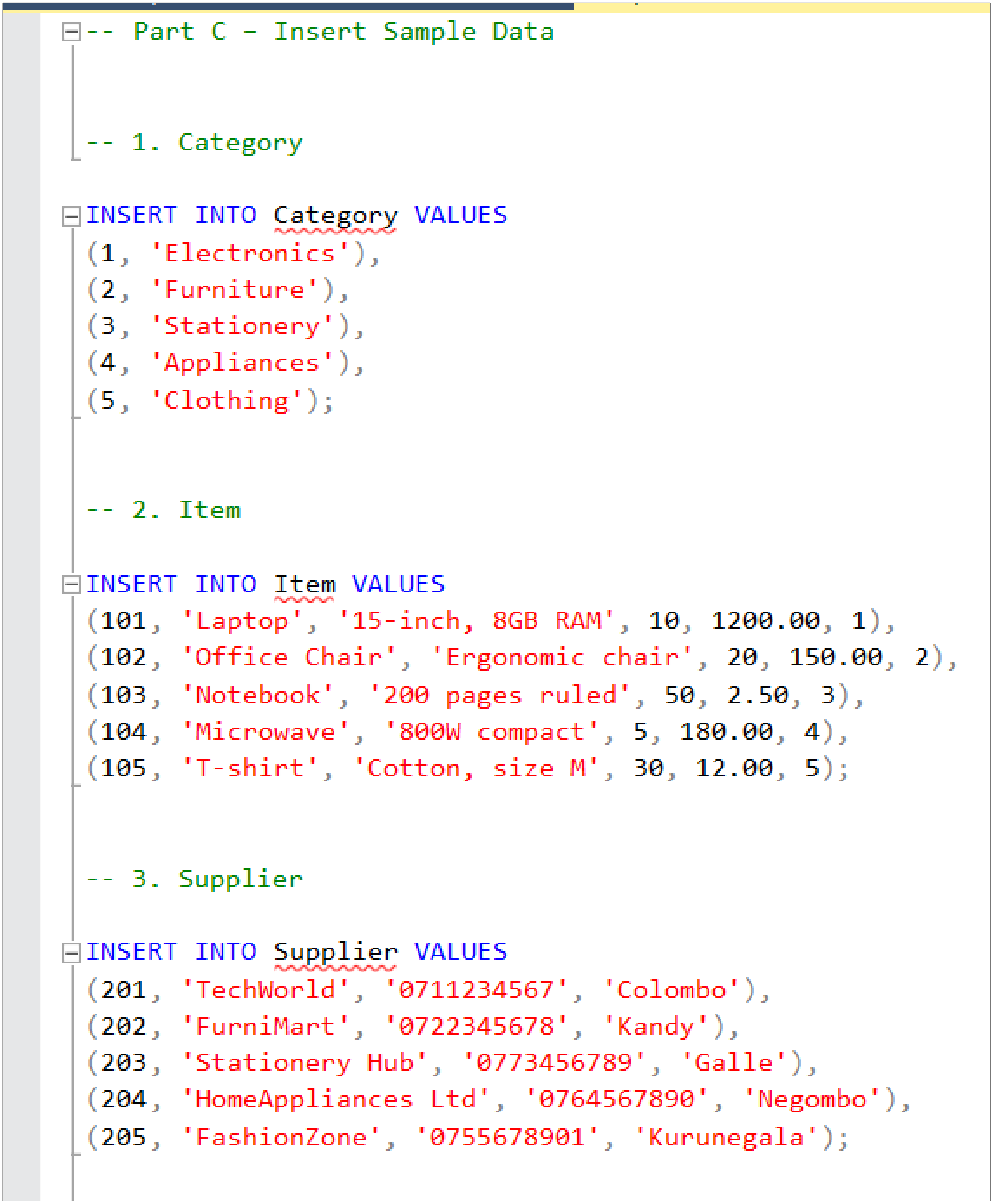
```sql
-- 4. Customer Table

CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Phone VARCHAR(15)
);


-- 5. Employee Table

CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Role VARCHAR(50),
    Salary DECIMAL(10,2) CHECK (Salary >= 0)
);


-- 6. Orders Table

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE NOT NULL,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```



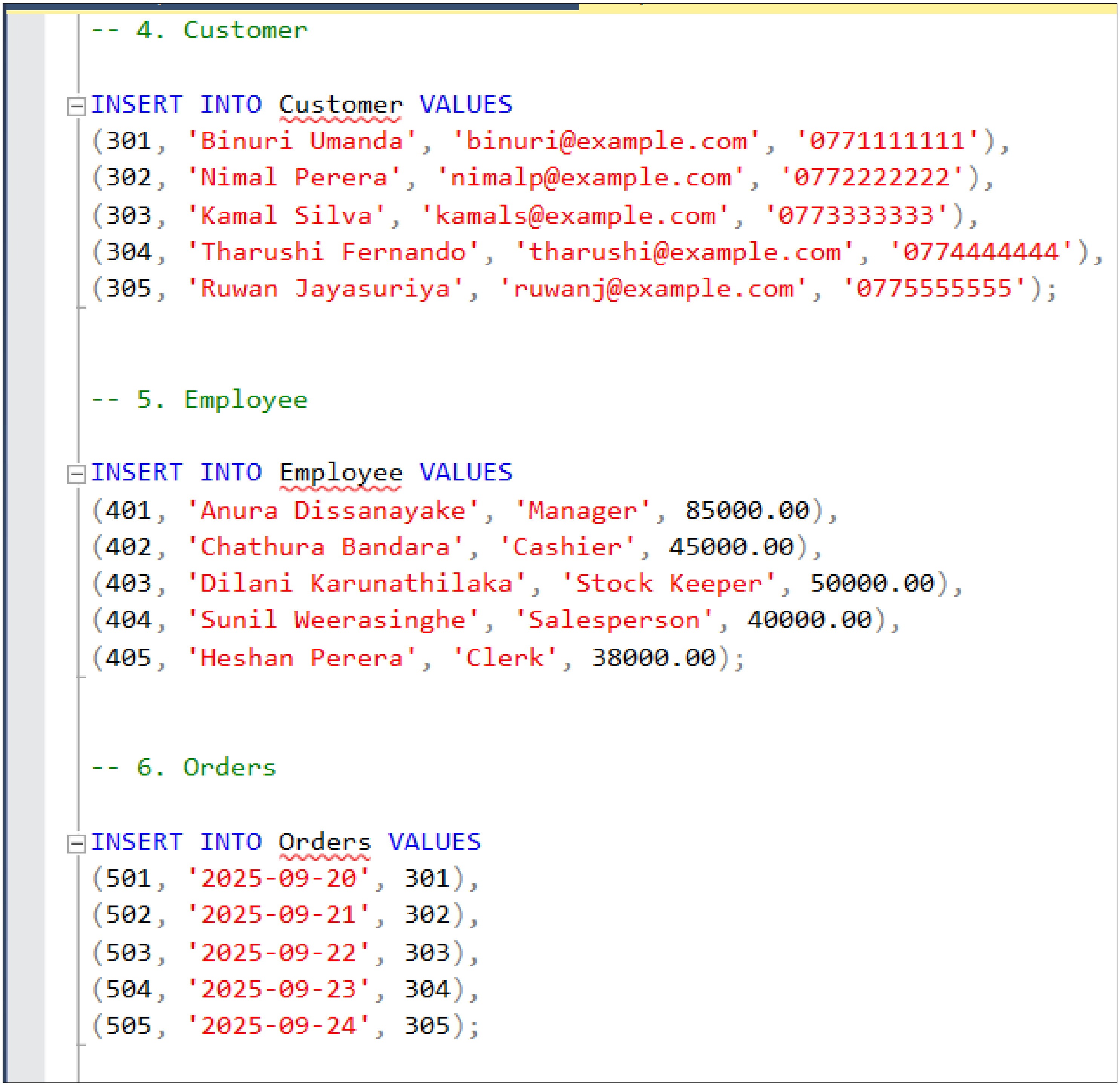
```sql
-- 7. Order_Item (M:N Relationship)

CREATE TABLE Order_Item (
    OrderID INT,
    ItemID INT,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    PRIMARY KEY (OrderID, ItemID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID)
);


-- 8. Stock_Receive Table

CREATE TABLE Stock_Receive (
    ReceiveID INT PRIMARY KEY,
    ReceiveDate DATE NOT NULL,
    SupplierID INT,
    EmpID INT,
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID),
    FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)
);


-- 9. Receive_Item (M:N Relationship)

CREATE TABLE Receive_Item (
    ReceiveID INT,
    ItemID INT,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    PRIMARY KEY (ReceiveID, ItemID),
    FOREIGN KEY (ReceiveID) REFERENCES Stock_Receive(ReceiveID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID)
);
```

Part C – Insert Sample Data

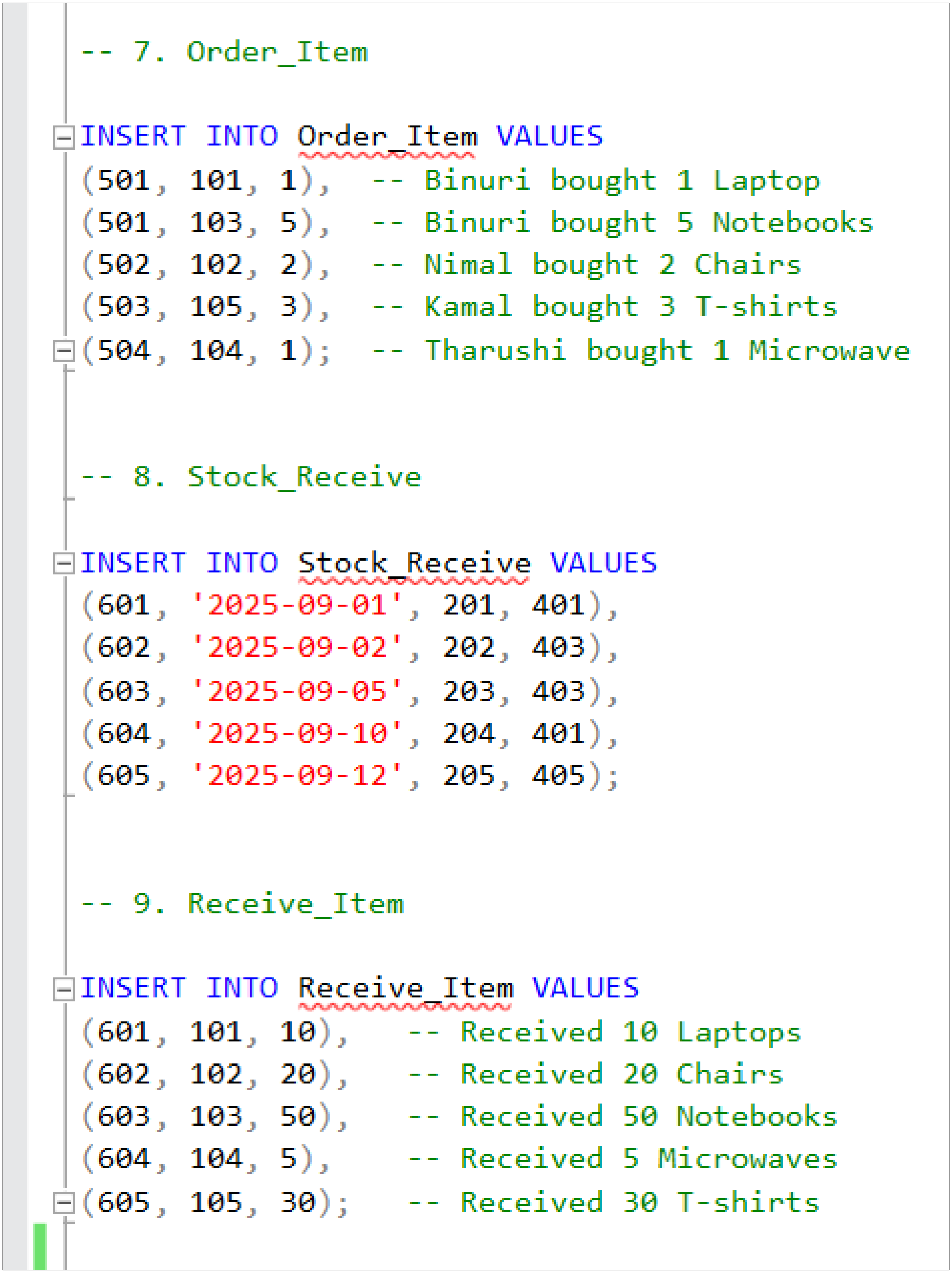
Screenshots of inserting data:



```sql
-- Part C - Insert Sample Data


-- 1. Category

INSERT INTO Category VALUES
(1, 'Electronics'),
(2, 'Furniture'),
(3, 'Stationery'),
(4, 'Appliances'),
(5, 'Clothing');


-- 2. Item

INSERT INTO Item VALUES
(101, 'Laptop', '15-inch, 8GB RAM', 10, 1200.00, 1),
(102, 'Office Chair', 'Ergonomic chair', 20, 150.00, 2),
(103, 'Notebook', '200 pages ruled', 50, 2.50, 3),
(104, 'Microwave', '800W compact', 5, 180.00, 4),
(105, 'T-shirt', 'Cotton, size M', 30, 12.00, 5);


-- 3. Supplier

INSERT INTO Supplier VALUES
(201, 'TechWorld', '0711234567', 'Colombo'),
(202, 'FurniMart', '0722345678', 'Kandy'),
(203, 'Stationery Hub', '0773456789', 'Galle'),
(204, 'HomeAppliances Ltd', '0764567890', 'Negombo'),
(205, 'FashionZone', '0755678901', 'Kurunegala');
```

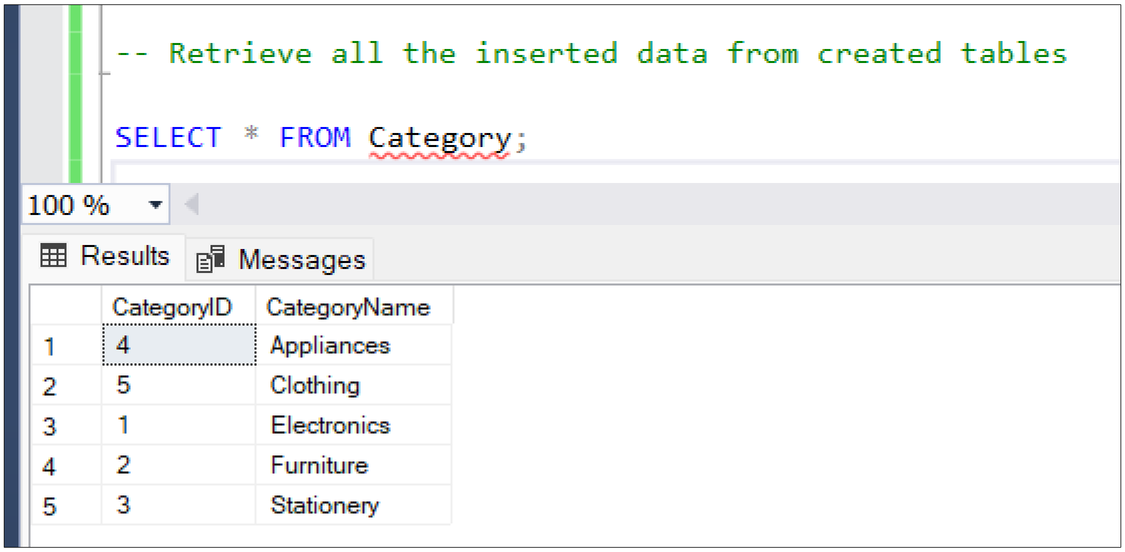


```sql
-- 4. Customer

INSERT INTO Customer VALUES
(301, 'Binuri Umanda', 'binuri@example.com', '0771111111'),
(302, 'Nimal Perera', 'nimalp@example.com', '0772222222'),
(303, 'Kamal Silva', 'kamals@example.com', '0773333333'),
(304, 'Tharushi Fernando', 'tharushi@example.com', '0774444444'),
(305, 'Ruwan Jayasuriya', 'ruwanj@example.com', '0775555555');


-- 5. Employee

INSERT INTO Employee VALUES
(401, 'Anura Dissanayake', 'Manager', 85000.00),
(402, 'Chathura Bandara', 'Cashier', 45000.00),
(403, 'Dilani Karunathilaka', 'Stock Keeper', 50000.00),
(404, 'Sunil Weerasinghe', 'Salesperson', 40000.00),
(405, 'Heshan Perera', 'Clerk', 38000.00);


-- 6. Orders

INSERT INTO Orders VALUES
(501, '2025-09-20', 301),
(502, '2025-09-21', 302),
(503, '2025-09-22', 303),
(504, '2025-09-23', 304),
(505, '2025-09-24', 305);
```



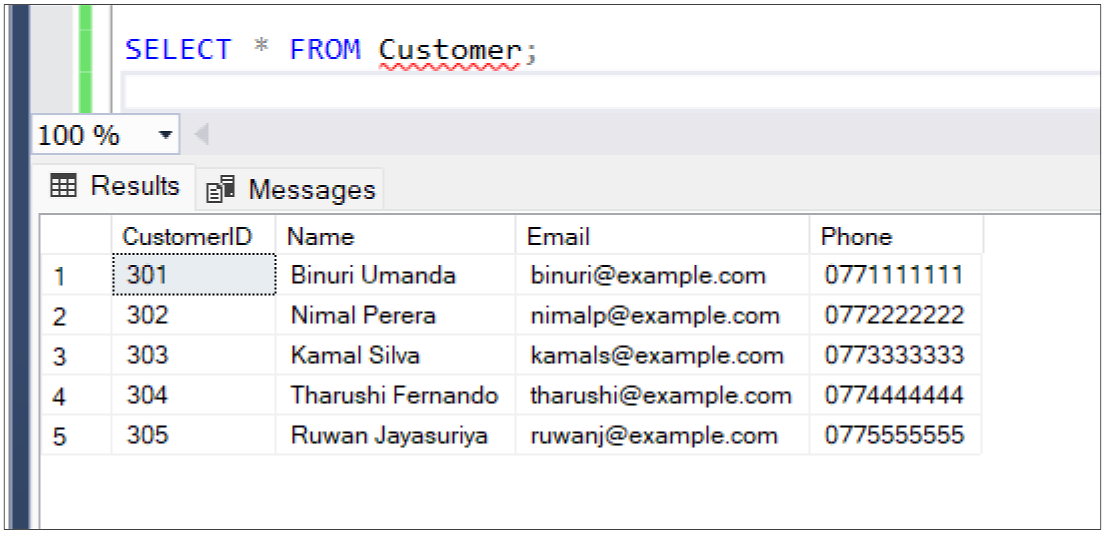
```sql
-- 7. Order_Item

INSERT INTO Order_Item VALUES
(501, 101, 1),  -- Binuri bought 1 Laptop
(501, 103, 5),  -- Binuri bought 5 Notebooks
(502, 102, 2),  -- Nimal bought 2 Chairs
(503, 105, 3),  -- Kamal bought 3 T-shirts
(504, 104, 1);  -- Tharushi bought 1 Microwave


-- 8. Stock_Receive

INSERT INTO Stock_Receive VALUES
(601, '2025-09-01', 201, 401),
(602, '2025-09-02', 202, 403),
(603, '2025-09-05', 203, 403),
(604, '2025-09-10', 204, 401),
(605, '2025-09-12', 205, 405);


-- 9. Receive_Item

INSERT INTO Receive_Item VALUES
(601, 101, 10),   -- Received 10 Laptops
(602, 102, 20),   -- Received 20 Chairs
(603, 103, 50),   -- Received 50 Notebooks
(604, 104, 5),    -- Received 5 Microwaves
(605, 105, 30);   -- Received 30 T-shirts
```

Screenshots of inserted data:



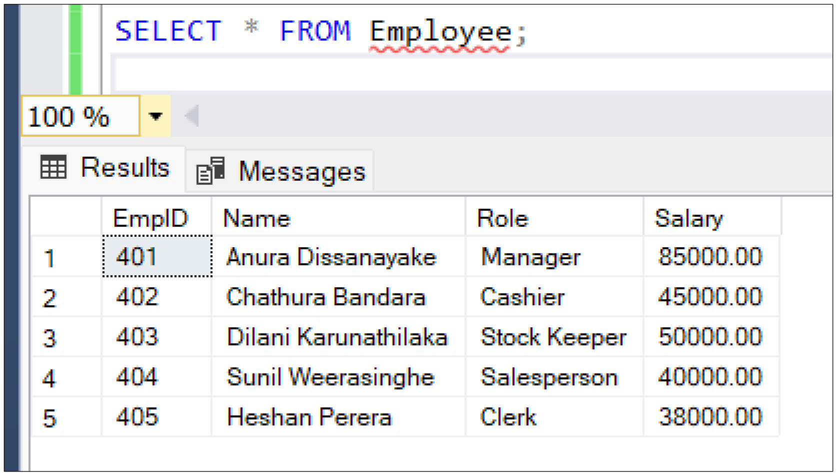
```
-- Retrieve all the inserted data from created tables

SELECT * FROM Category;
```

	CategoryID	CategoryName
1	4	Appliances
2	5	Clothing
3	1	Electronics
4	2	Furniture
5	3	Stationery



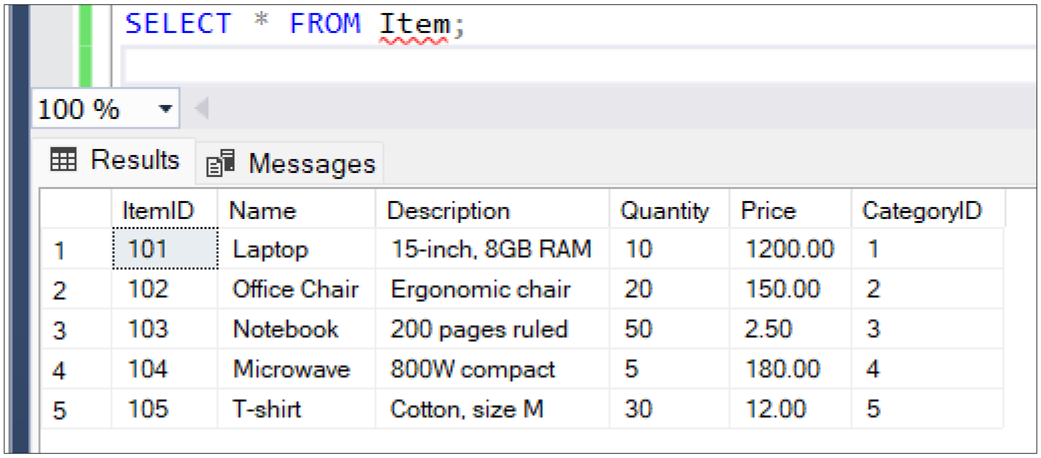
```
SELECT * FROM Customer;
```

	CustomerID	Name	Email	Phone
1	301	Binuri Umanda	binuri@example.com	0771111111
2	302	Nimal Perera	nimalp@example.com	0772222222
3	303	Kamal Silva	kamals@example.com	0773333333
4	304	Tharushi Fernando	tharushi@example.com	0774444444
5	305	Ruwan Jayasuriya	ruwanj@example.com	0775555555



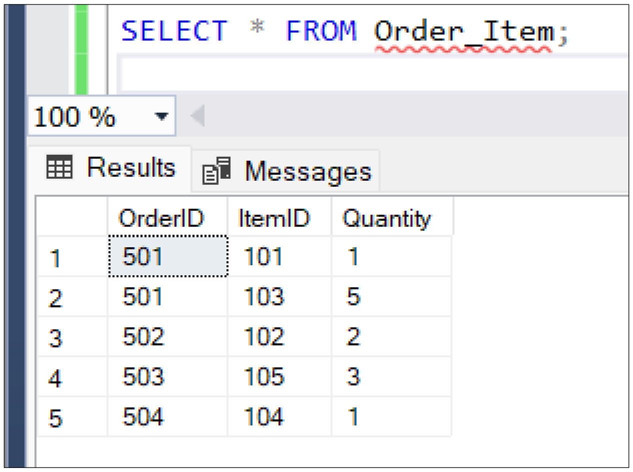
```
SELECT * FROM Employee;
```

	EmpID	Name	Role	Salary
1	401	Anura Dissanayake	Manager	85000.00
2	402	Chathura Bandara	Cashier	45000.00
3	403	Dilani Karunathilaka	Stock Keeper	50000.00
4	404	Sunil Weerasinghe	Salesperson	40000.00
5	405	Heshan Perera	Clerk	38000.00



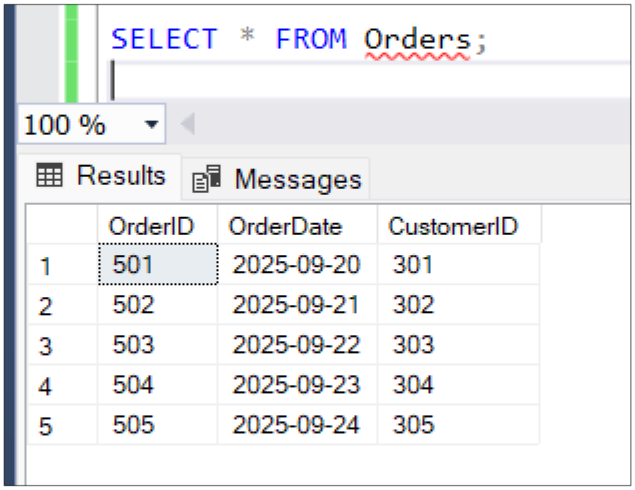
```sql
SELECT * FROM Item;
```

	ItemID	Name	Description	Quantity	Price	CategoryID
1	101	Laptop	15-inch, 8GB RAM	10	1200.00	1
2	102	Office Chair	Ergonomic chair	20	150.00	2
3	103	Notebook	200 pages ruled	50	2.50	3
4	104	Microwave	800W compact	5	180.00	4
5	105	T-shirt	Cotton, size M	30	12.00	5



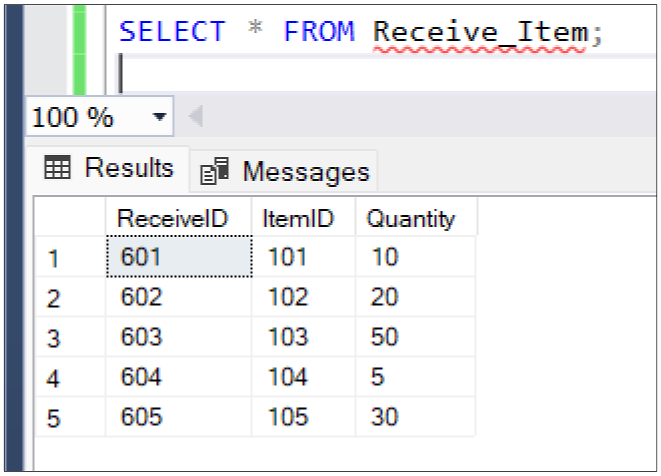
```sql
SELECT * FROM Order_Item;
```

	OrderID	ItemID	Quantity
1	501	101	1
2	501	103	5
3	502	102	2
4	503	105	3
5	504	104	1



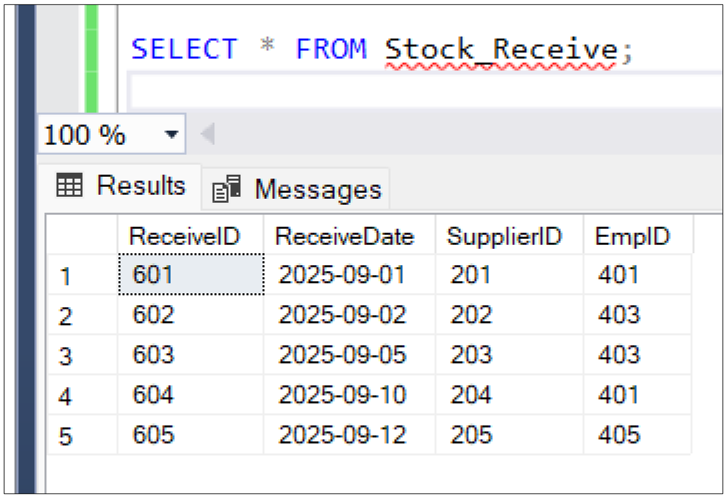
```sql
SELECT * FROM Orders;
```

	OrderID	OrderDate	CustomerID
1	501	2025-09-20	301
2	502	2025-09-21	302
3	503	2025-09-22	303
4	504	2025-09-23	304
5	505	2025-09-24	305



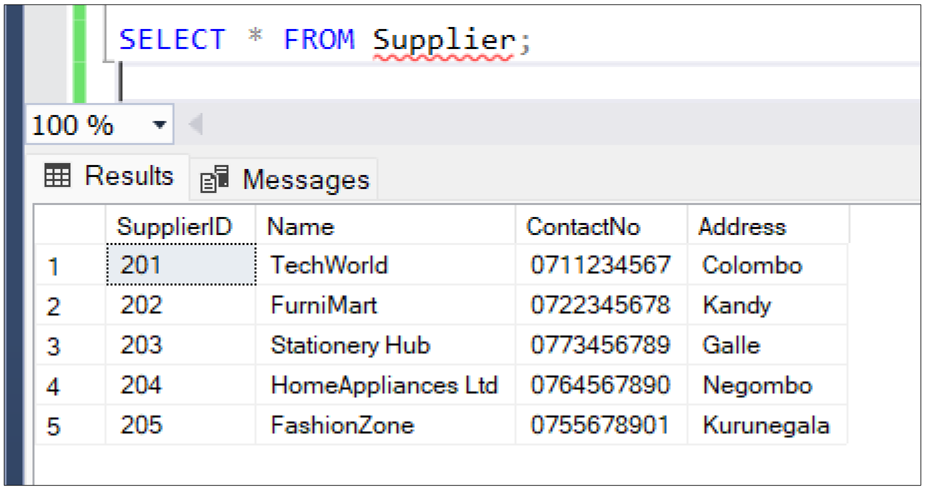
```sql
SELECT * FROM Receive_Item;
```

	ReceiveID	ItemID	Quantity
1	601	101	10
2	602	102	20
3	603	103	50
4	604	104	5
5	605	105	30



```sql
SELECT * FROM Stock_Receive;
```

	ReceiveID	ReceiveDate	SupplierID	EmpID
1	601	2025-09-01	201	401
2	602	2025-09-02	202	403
3	603	2025-09-05	203	403
4	604	2025-09-10	204	401
5	605	2025-09-12	205	405



```sql
SELECT * FROM Supplier;
```

	SupplierID	Name	ContactNo	Address
1	201	TechWorld	0711234567	Colombo
2	202	FurniMart	0722345678	Kandy
3	203	Stationery Hub	0773456789	Galle
4	204	HomeAppliances Ltd	0764567890	Negombo
5	205	FashionZone	0755678901	Kurunegala

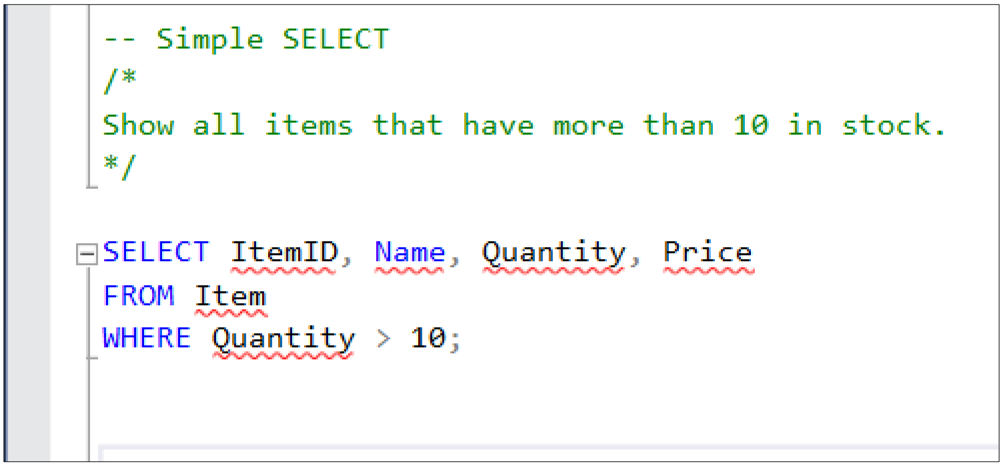
Part D – SQL Queries & Outputs

Write at least 5 queries:

- Simple SELECT
- JOIN
- Aggregation
- GROUP BY / HAVING
- Subquery

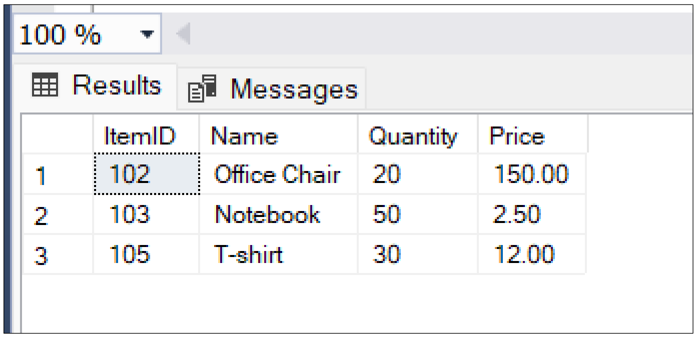
01. Simple SELECT

- **Query →**



```
-- Simple SELECT
/*
Show all items that have more than 10 in stock.
*/

SELECT ItemID, Name, Quantity, Price
FROM Item
WHERE Quantity > 10;
```

- **Output →**



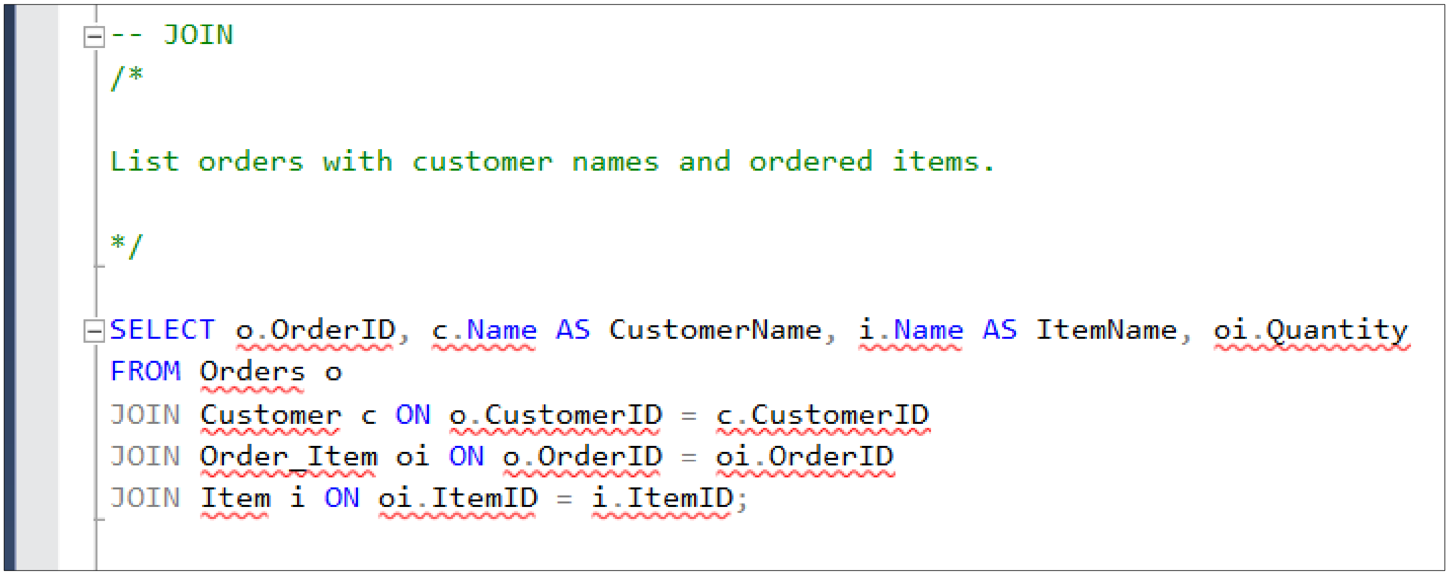
	ItemID	Name	Quantity	Price
1	102	Office Chair	20	150.00
2	103	Notebook	50	2.50
3	105	T-shirt	30	12.00

- **Explanation →**

This query looks at the Item table and retrieves a list of all items. However, it doesn't show every item. The WHERE clause filters the results to only include items where the Quantity in stock is greater than 10.

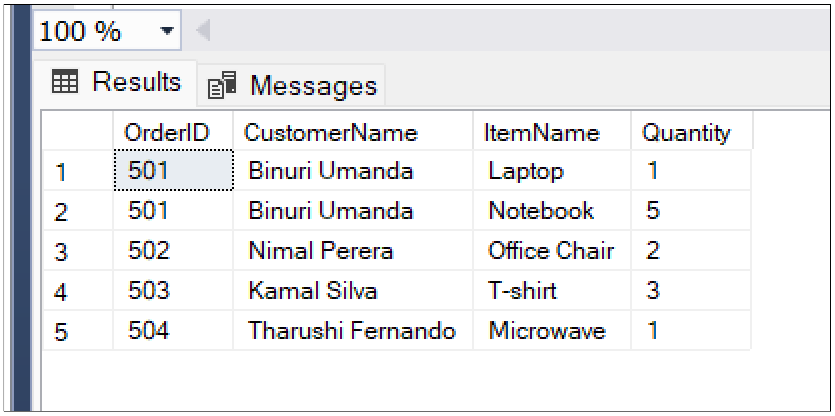
02. JOIN

- **Query →**



```
-- JOIN
/*

List orders with customer names and ordered items.

*/

SELECT o.OrderID, c.Name AS CustomerName, i.Name AS ItemName, oi.Quantity
FROM Orders o
JOIN Customer c ON o.CustomerID = c.CustomerID
JOIN Order_Item oi ON o.OrderID = oi.OrderID
JOIN Item i ON oi.ItemID = i.ItemID;
```

- **Output →**



	OrderID	CustomerName	ItemName	Quantity
1	501	Binuri Umanda	Laptop	1
2	501	Binuri Umanda	Notebook	5
3	502	Nimal Perera	Office Chair	2
4	503	Kamal Silva	T-shirt	3
5	504	Tharushi Fernando	Microwave	1

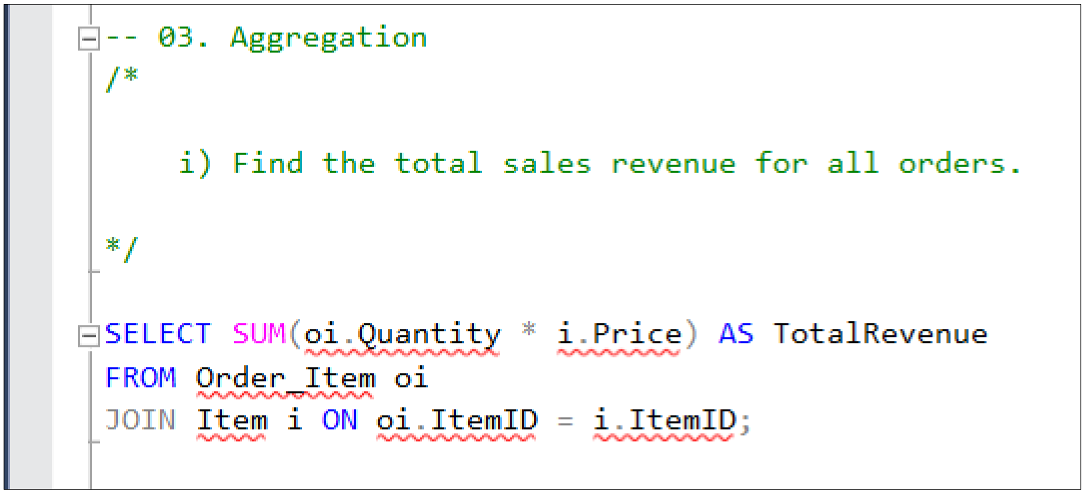
- **Explanation →**

This query gathers information from four different tables (Orders, Customer, Order_Item, and Item) and presents it together in a single view. It uses JOIN to link each order to the customer who made it, the specific items purchased, and the item's name, creating a neat list showing the order ID, customer name, item name, and quantity ordered.

03. Aggregation

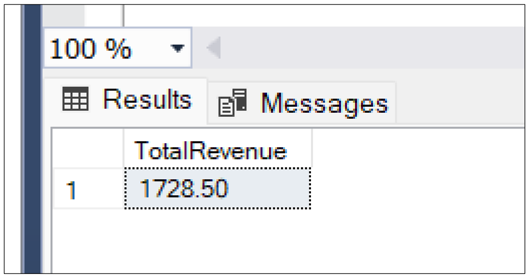
i.

- **Query →**



```
-- 03. Aggregation
/*

    i) Find the total sales revenue for all orders.

*/

SELECT SUM(oi.Quantity * i.Price) AS TotalRevenue
FROM Order_Item oi
JOIN Item i ON oi.ItemID = i.ItemID;
```

- **Output →**



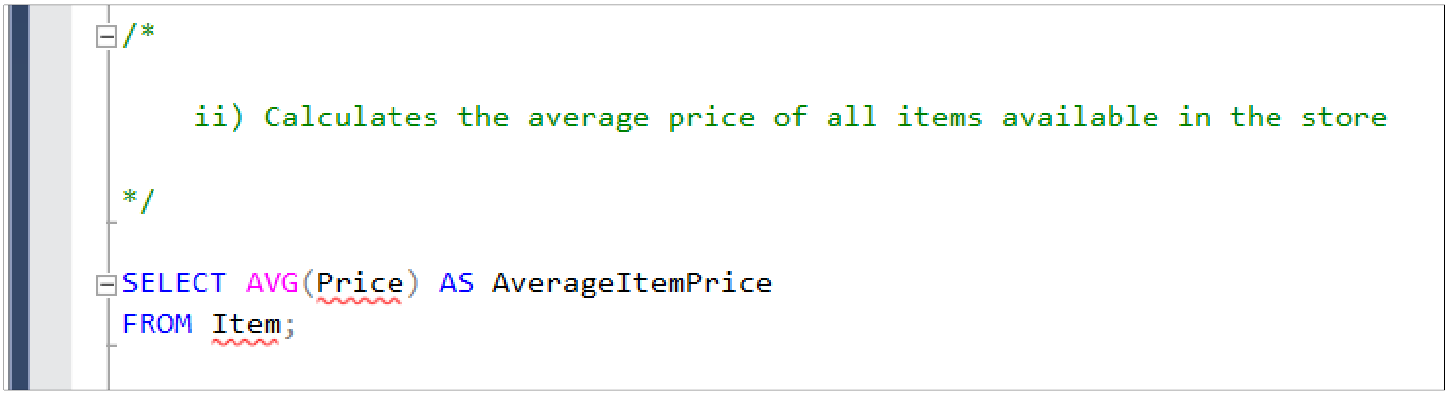
	TotalRevenue
1	1728.50

- **Explanation →**

This query calculates the total sales revenue from all orders. It first multiplies the Quantity of each item sold by its Price to find the total for that line item. Then, the SUM() function adds up all these individual totals to give you one final number the TotalRevenue.

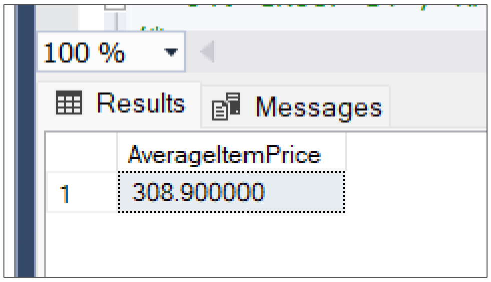
ii.

- **Query →**



```
/*

    ii) Calculates the average price of all items available in the store

*/

SELECT AVG(Price) AS AverageItemPrice
FROM Item;
```

- **Output →**



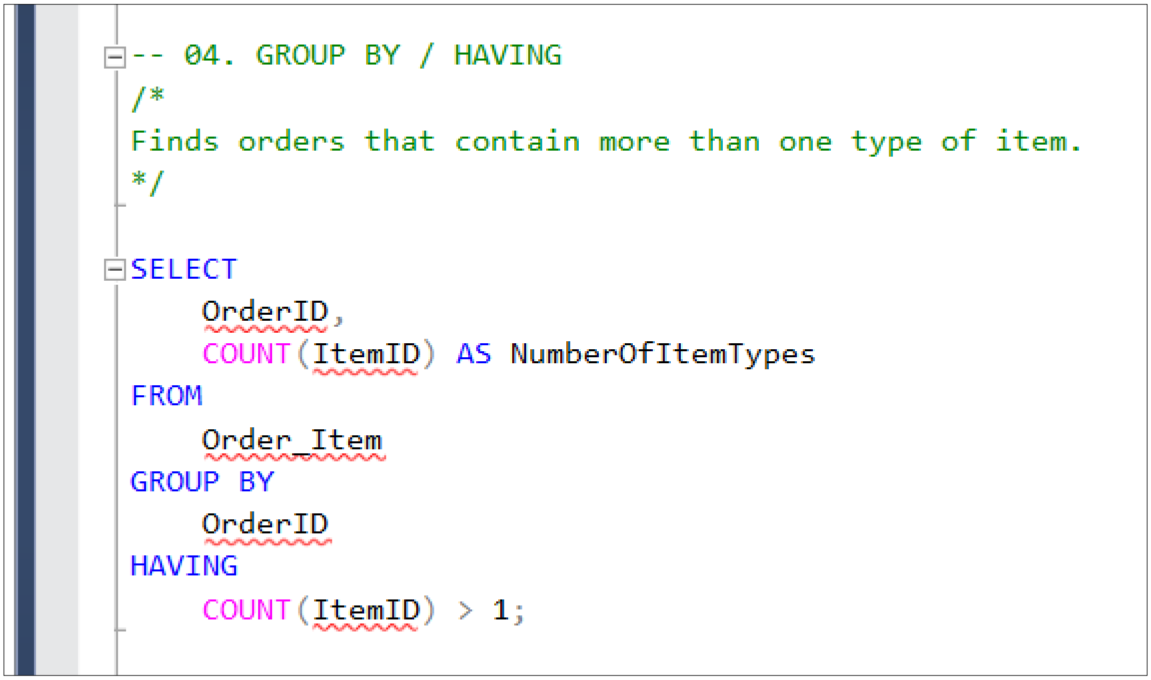
	AverageItemPrice
1	308.900000

- **Explanation →**

This query calculates the average price of all items in the Item table. The AVG() function takes up all the values in the Price column, adds them together, and then divides by the number of items to find the average.

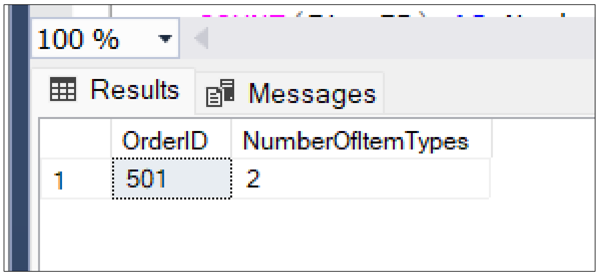
04. GROUP BY / HAVING

- **Query →**



```
-- 04. GROUP BY / HAVING
/*
Finds orders that contain more than one type of item.
*/

SELECT
    OrderID,
    COUNT(ItemID) AS NumberOfItemTypes
FROM
    Order_Item
GROUP BY
    OrderID
HAVING
    COUNT(ItemID) > 1;
```

- **Output →**



```
     OrderID   NumberOfItemTypes
1    501       2
```

- **Explanation →**

This query identifies orders that contain more than one type of item.

The GROUP BY clause first groups all rows by their OrderID.

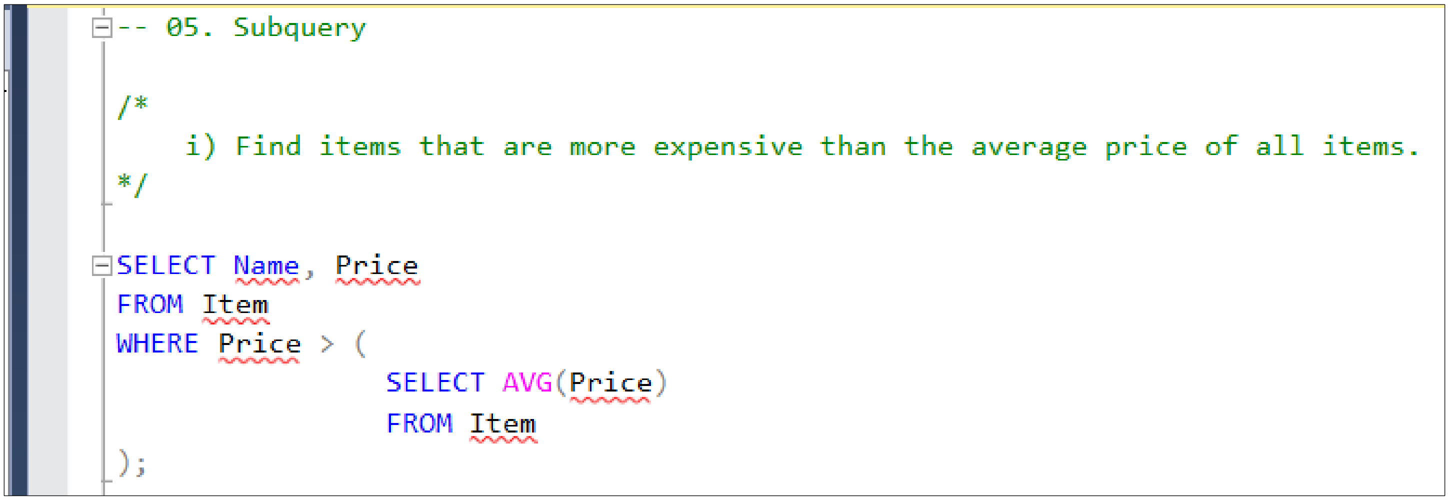
Then, the COUNT(ItemID) function counts how many distinct items are in each group.

Finally, the HAVING clause filters these groups, only showing the OrderIDs where the item count is greater than 1.

05. Subquery

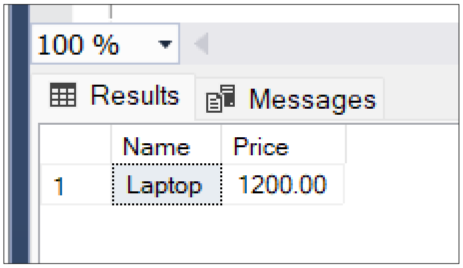
i.

- **Query →**



```
-- 05. Subquery

/*
    i) Find items that are more expensive than the average price of all items.
*/

SELECT Name, Price
FROM Item
WHERE Price > (
                SELECT AVG(Price)
                FROM Item
);
```

- **Output →**



	Name	Price
1	Laptop	1200.00

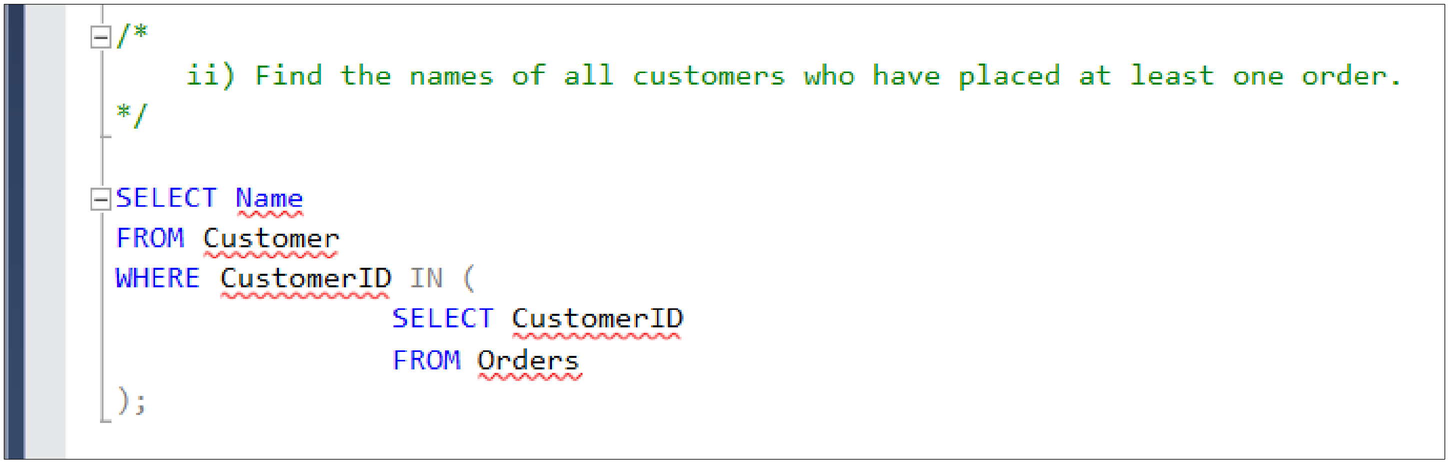
- **Explanation →**

Here the inner query (SELECT AVG(Price) FROM Item) first calculates the average price of all items.

The outer query then selects the Name and Price of all items from the Item table where the Price is greater than this calculated average.

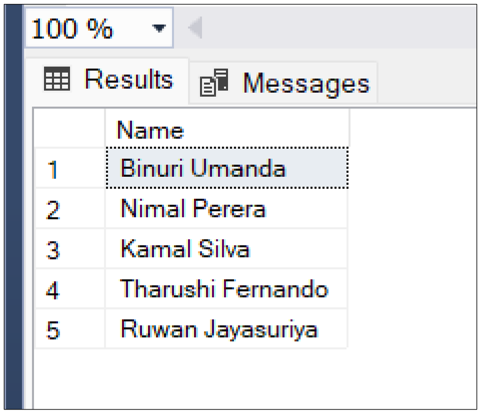
ii.

- **Query →**



```
/*
    ii) Find the names of all customers who have placed at least one order.
*/

SELECT Name
FROM Customer
WHERE CustomerID IN (
                    SELECT CustomerID
                    FROM Orders
);
```

- **Output →**



	Name
1	Binuri Umanda
2	Nimal Perera
3	Kamal Silva
4	Tharushi Fernando
5	Ruwan Jayasuriya

- **Explanation →**

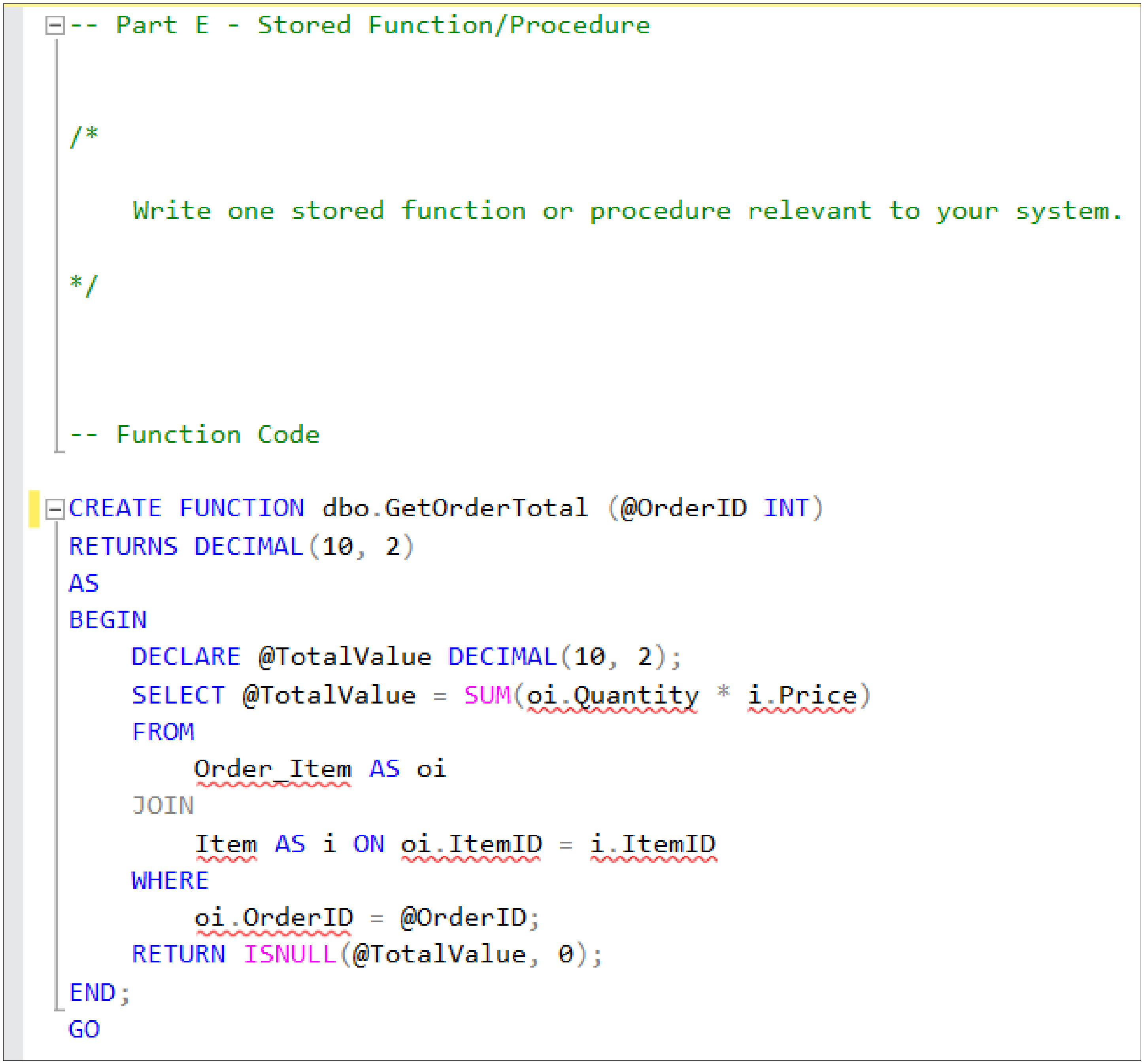
Here the inner query (SELECT CustomerID FROM Orders) creates a list of all unique customer IDs found in the Orders table.

The outer query then selects the Name from the Customer table for every customer whose CustomerID is present in that list.

Part E – Stored Function/Procedure

- **Function Code →**

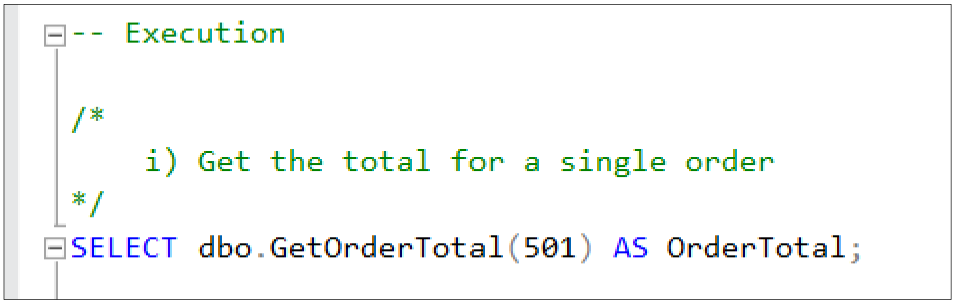
This function, dbo.GetOrderTotal, calculates the total price for any given OrderID. It takes an OrderID as input and returns the total value as a decimal.



```sql
-- Part E - Stored Function/Procedure


/*

    Write one stored function or procedure relevant to your system.

*/



-- Function Code

CREATE FUNCTION dbo.GetOrderTotal (@OrderID INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalValue DECIMAL(10, 2);
    SELECT @TotalValue = SUM(oi.Quantity * i.Price)
    FROM
        Order_Item AS oi
    JOIN
        Item AS i ON oi.ItemID = i.ItemID
    WHERE
        oi.OrderID = @OrderID;
    RETURN ISNULL(@TotalValue, 0);
END;
GO
```

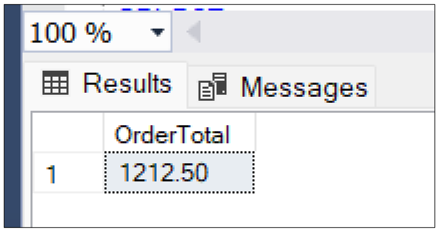
i. Get the total for a single order:

- **Execution →**



```
-- Execution

/*
    i) Get the total for a single order
*/
SELECT dbo.GetOrderTotal(501) AS OrderTotal;
```

- **Output →**



```
   OrderTotal
1  1212.50
```

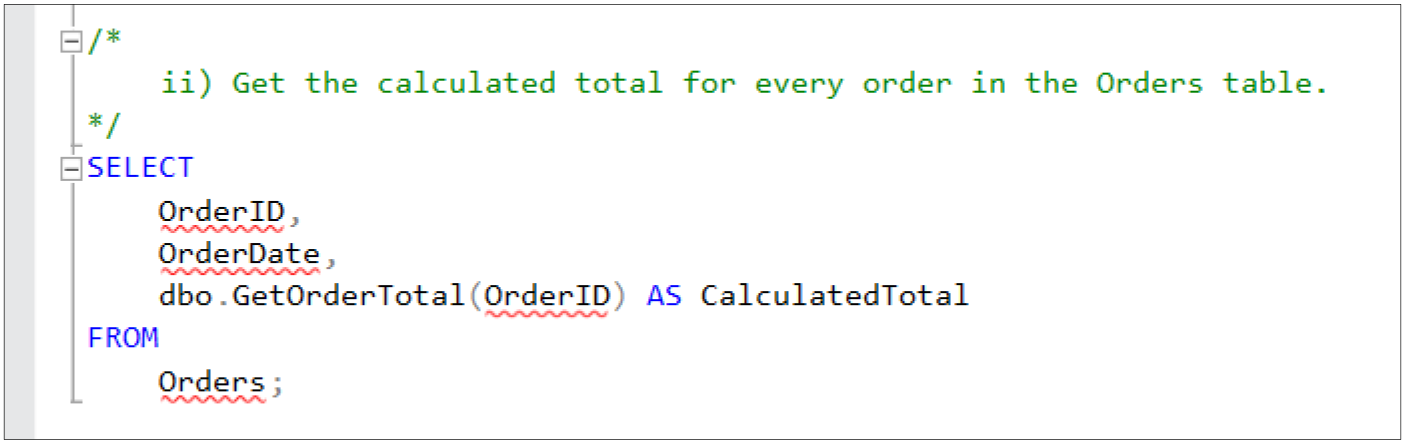
- **Explanation →**

After creating this function now, we can use this function just like any built-in SQL function inside a SELECT query.

Here, we find the total value for Order ID 501.

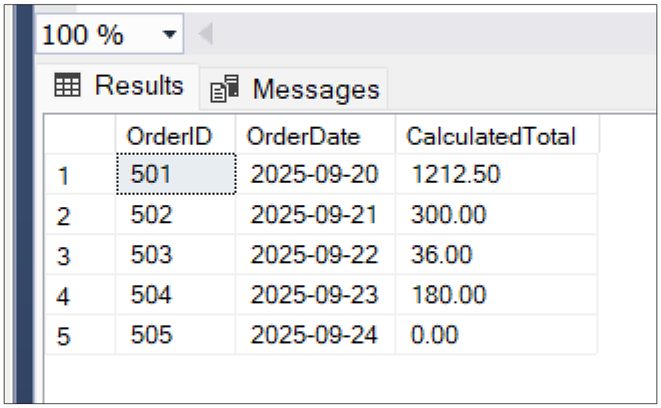
ii. Get the calculated total for every order in the Orders table.

- **Execution →**



```
/*
    ii) Get the calculated total for every order in the Orders table.
*/
SELECT
    OrderID,
    OrderDate,
    dbo.GetOrderTotal(OrderID) AS CalculatedTotal
FROM
    Orders;
```

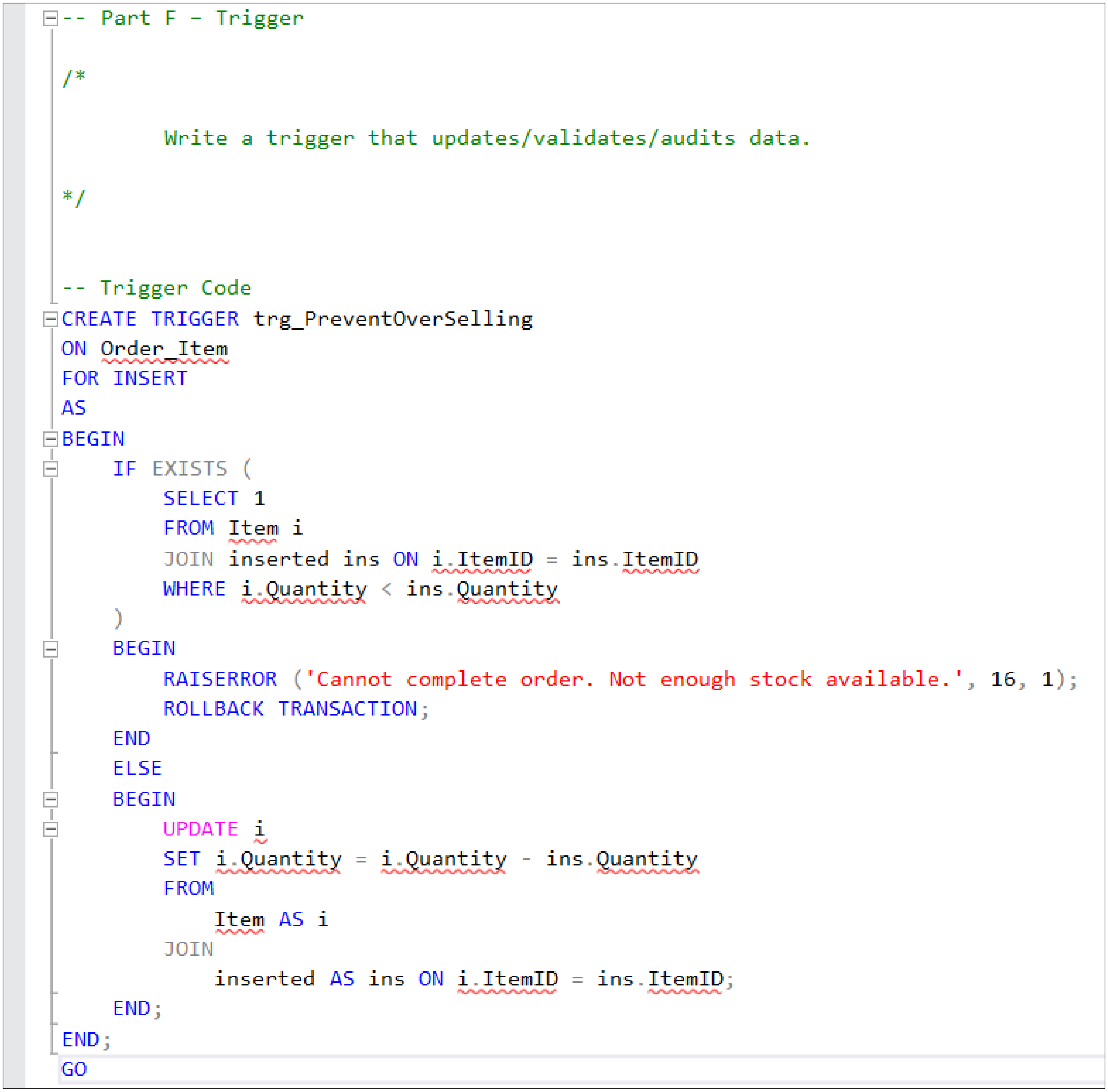
- **Output →**



	OrderID	OrderDate	CalculatedTotal
1	501	2025-09-20	1212.50
2	502	2025-09-21	300.00
3	503	2025-09-22	36.00
4	504	2025-09-23	180.00
5	505	2025-09-24	0.00

Part F – Trigger

- **Trigger Code →**

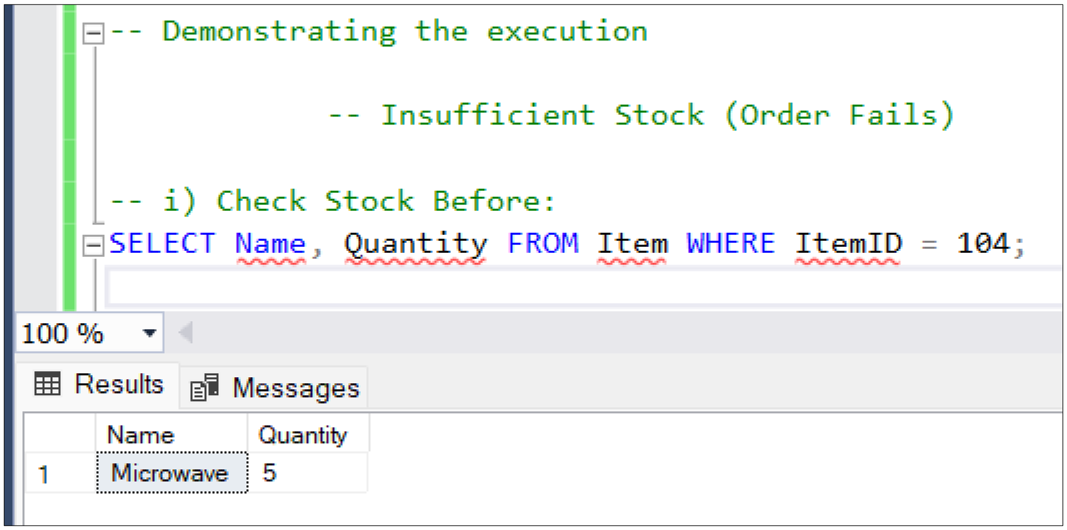


```
-- Part F - Trigger

/*

        Write a trigger that updates/validates/audits data.

*/


-- Trigger Code
CREATE TRIGGER trg_PreventOverSelling
ON Order_Item
FOR INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Item i
        JOIN inserted ins ON i.ItemID = ins.ItemID
        WHERE i.Quantity < ins.Quantity
    )
    BEGIN
        RAISERROR ('Cannot complete order. Not enough stock available.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        UPDATE i
        SET i.Quantity = i.Quantity - ins.Quantity
        FROM
            Item AS i
        JOIN
            inserted AS ins ON i.ItemID = ins.ItemID;
    END;
END;
GO
```

- **Demonstrating the execution →**

a. Insufficient Stock (Order Fails)

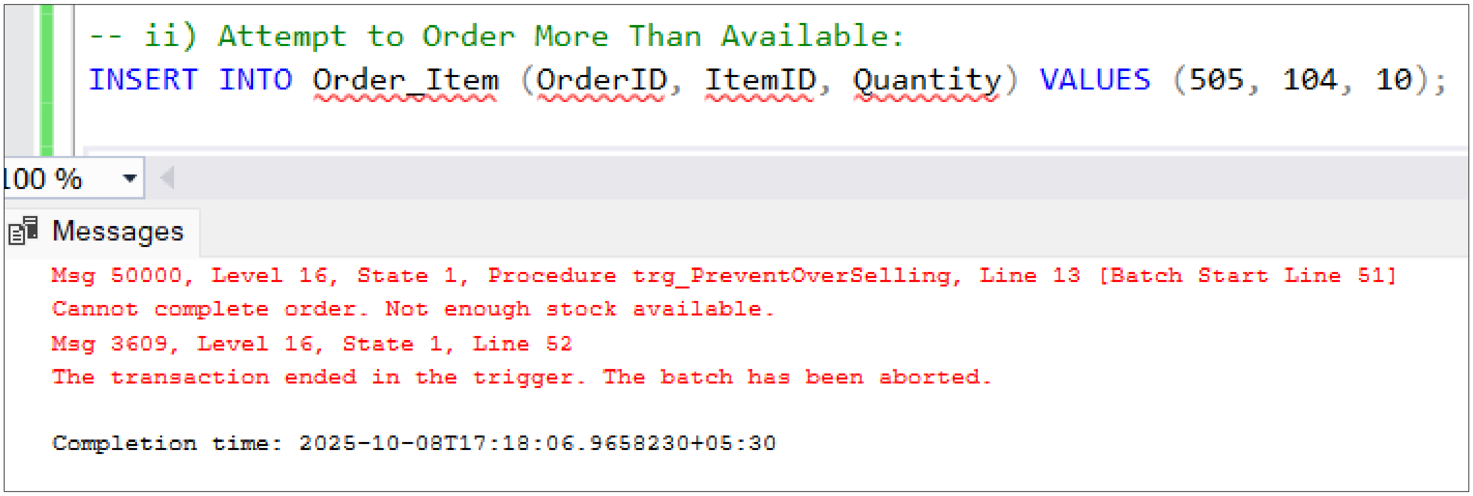
i. Check Stock Before:



```
-- Demonstrating the execution

            -- Insufficient Stock (Order Fails)

-- i) Check Stock Before:
SELECT Name, Quantity FROM Item WHERE ItemID = 104;
```

	Name	Quantity
1	Microwave	5

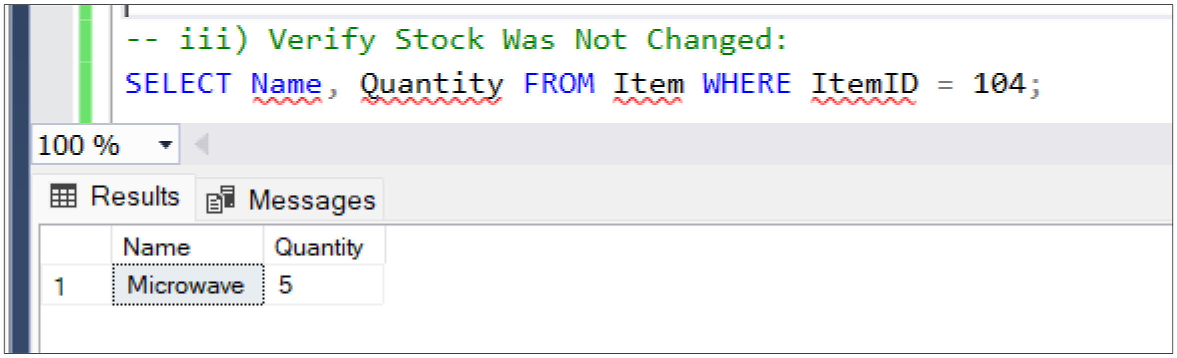
ii. Attempt to Order More Than Available:



```
-- ii) Attempt to Order More Than Available:
INSERT INTO Order_Item (OrderID, ItemID, Quantity) VALUES (505, 104, 10);
```

```
Msg 50000, Level 16, State 1, Procedure trg_PreventOverSelling, Line 13 [Batch Start Line 51]
Cannot complete order. Not enough stock available.
Msg 3609, Level 16, State 1, Line 52
The transaction ended in the trigger. The batch has been aborted.

Completion time: 2025-10-08T17:18:06.9658230+05:30
```

iii. Verify Stock Was Not Changed:

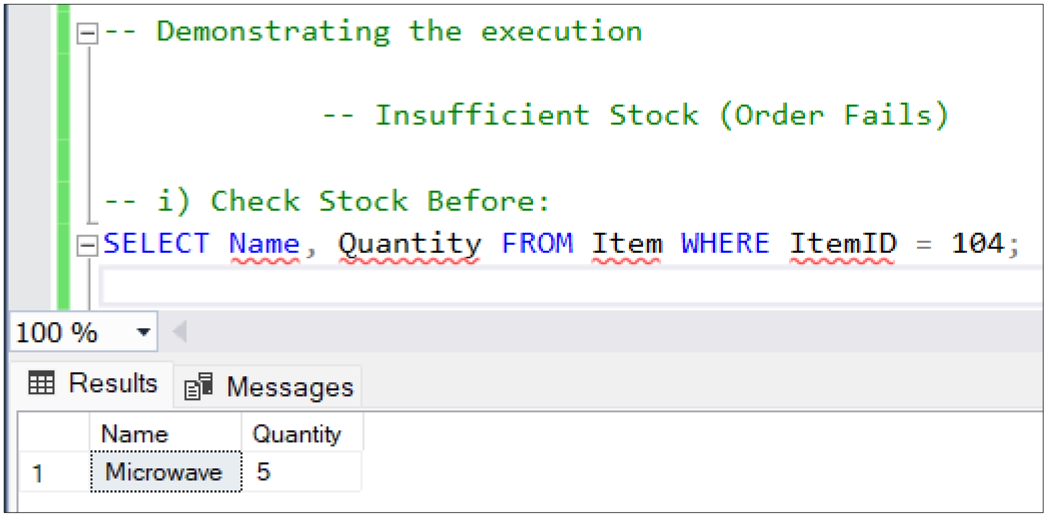


```
-- iii) Verify Stock Was Not Changed:
SELECT Name, Quantity FROM Item WHERE ItemID = 104;
```

	Name	Quantity
1	Microwave	5

b. Sufficient Stock (Order Succeeds)

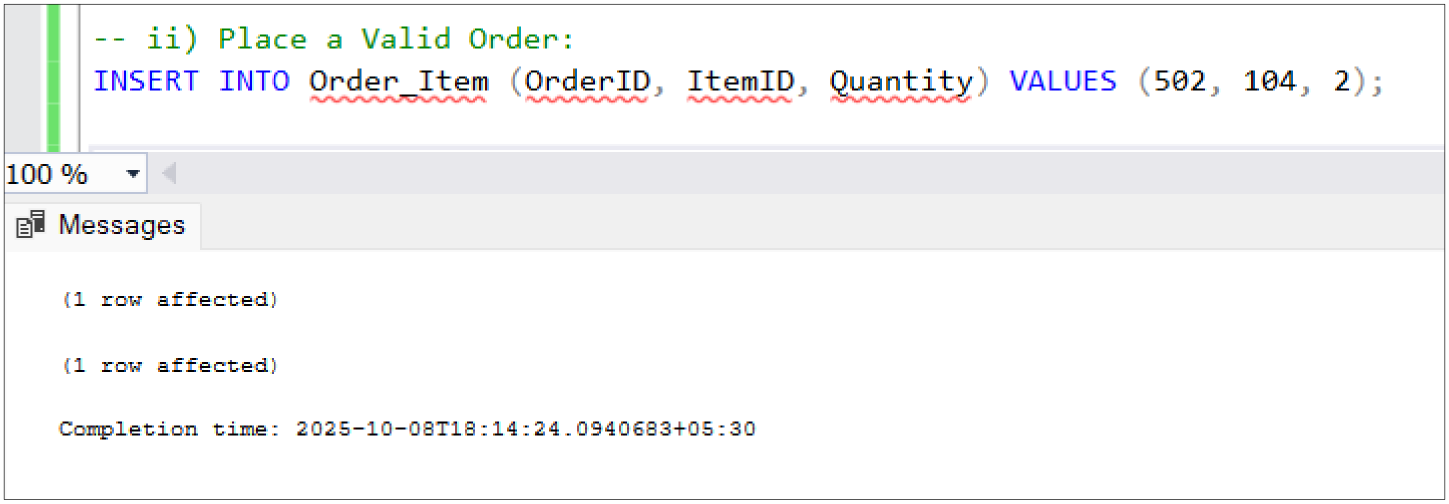
i. Check Stock Before:



```
-- Demonstrating the execution

            -- Insufficient Stock (Order Fails)

-- i) Check Stock Before:
SELECT Name, Quantity FROM Item WHERE ItemID = 104;
```

	Name	Quantity
1	Microwave	5

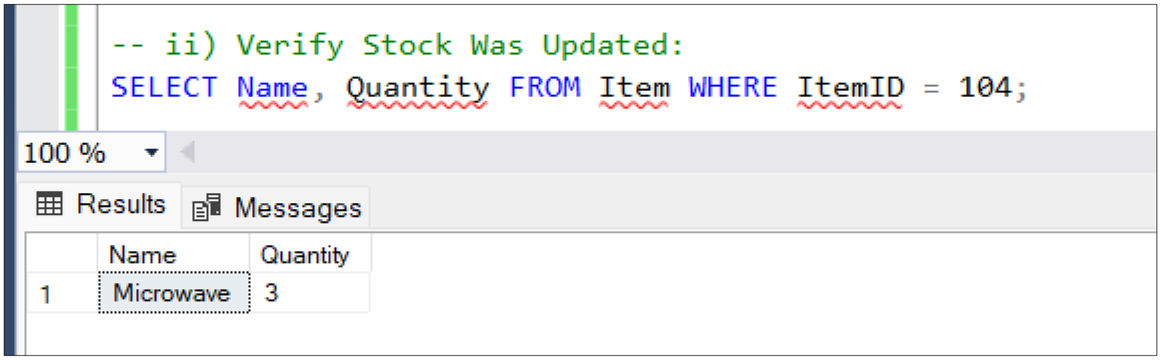
ii. Place a Valid Order:



```
-- ii) Place a Valid Order:
INSERT INTO Order_Item (OrderID, ItemID, Quantity) VALUES (502, 104, 2);
```

```
(1 row affected)

(1 row affected)

Completion time: 2025-10-08T18:14:24.0940683+05:30
```

iii. Verify Stock Was Updated:



```
-- ii) Verify Stock Was Updated:
SELECT Name, Quantity FROM Item WHERE ItemID = 104;
```

	Name	Quantity
1	Microwave	3