StudentId:CT_CSI_SQ_4948

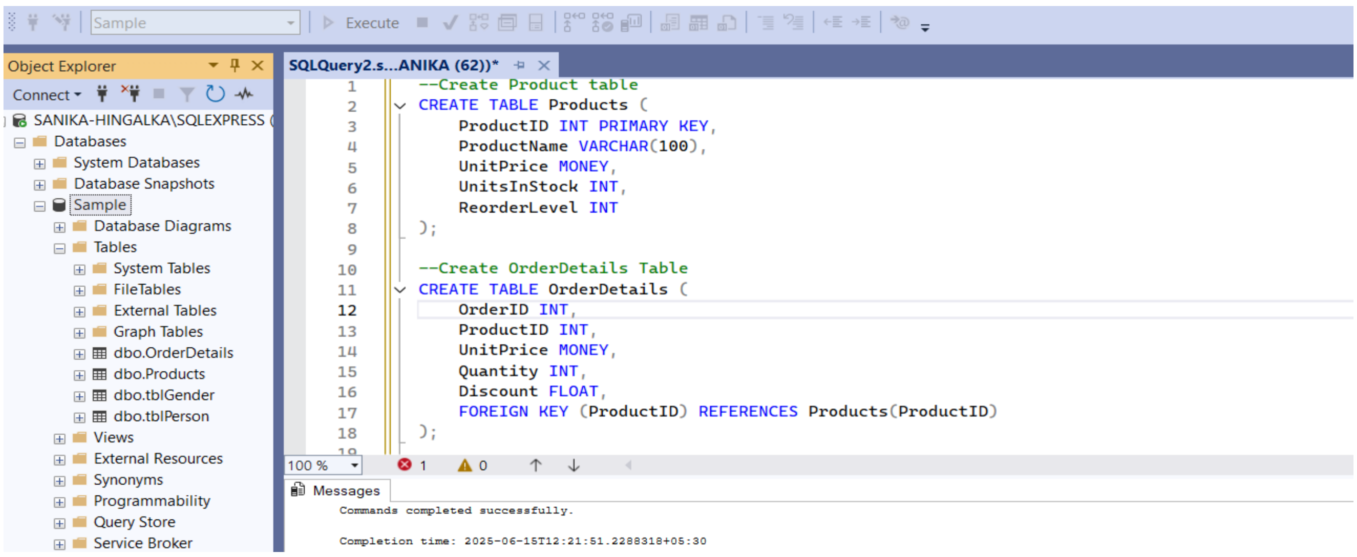
Student Name – Sanika Popat Hingalkar

Domain : SQL

Weekly Assignment 2

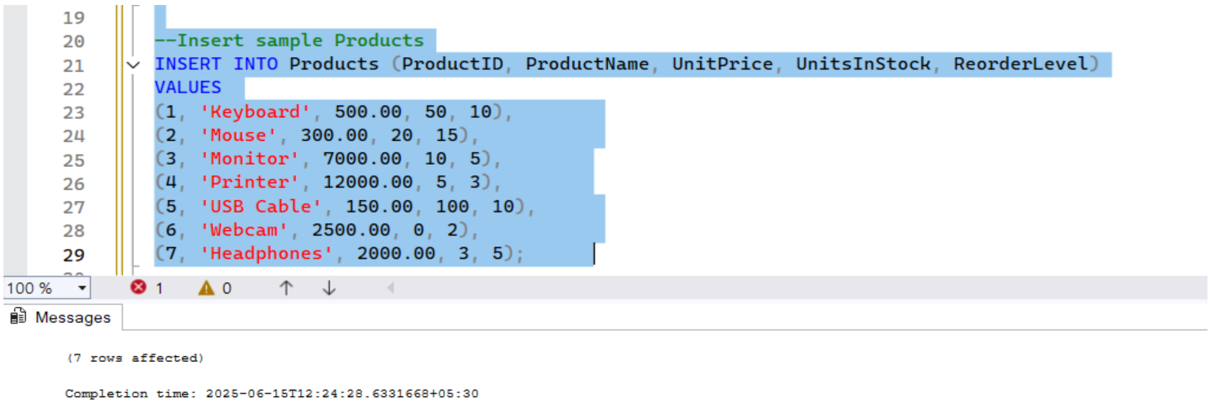
<u>STORED PROCEDURES -</u>

Task 1 : Create a procedure InsertOrderDetails that takes OrderID, ProductID, UnitPrice, Quantiy, Discount as input parameters and inserts that order information in the Order Details table. After each order inserted, check the @@rowcount value to make sure that order was inserted properly. If for any reason the order was not inserted, print the message: Failed to place the order. Please try again. Also yourprocedure should have these functionalities Make the UnitPrice and Discount parameters optionalIf no UnitPrice is given, then use the UnitPrice value from the product table.If no Discount is given, then use a discount of 0.Adjust the quantity in stock (UnitsInStock) for the product by subtracting the quantity sold from inventory.However, if there is not enough of a product in stock, then abort the stored procedure without making any changes to the database.Print a message if the quantity in stock of a product drops below its Reorder Level as a result of the update.



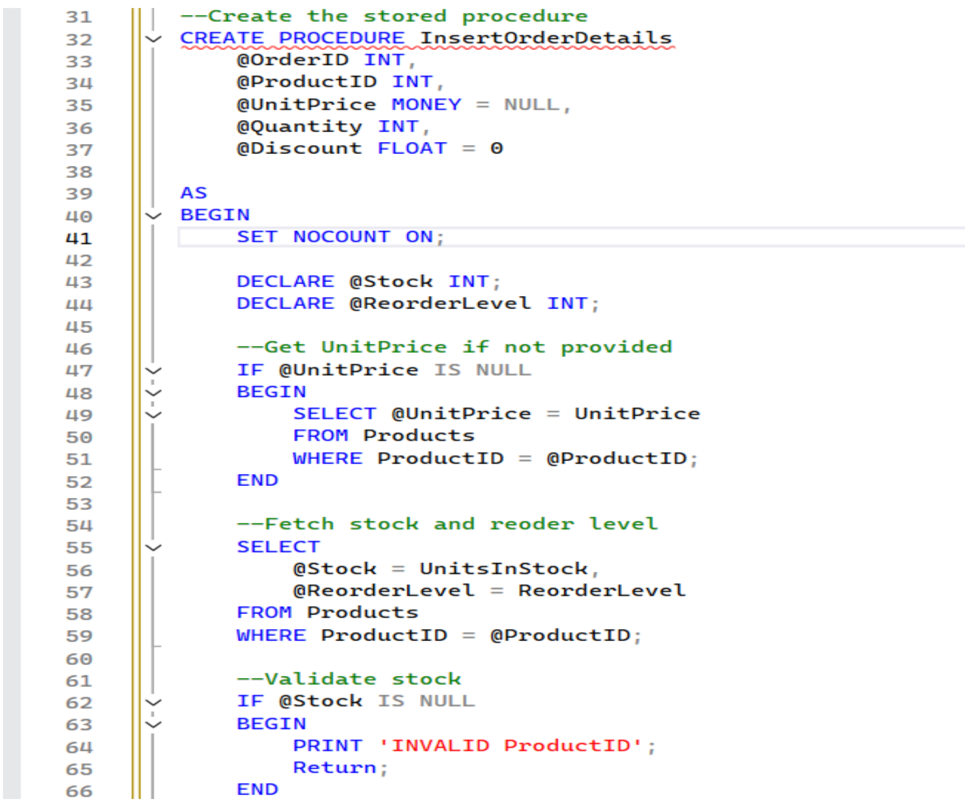
```sql
--Create Product table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    UnitPrice MONEY,
    UnitsInStock INT,
    ReorderLevel INT
);

--Create OrderDetails Table
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    UnitPrice MONEY,
    Quantity INT,
    Discount FLOAT,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

```
Commands completed successfully.

Completion time: 2025-06-15T12:21:51.2288318+05:30
```

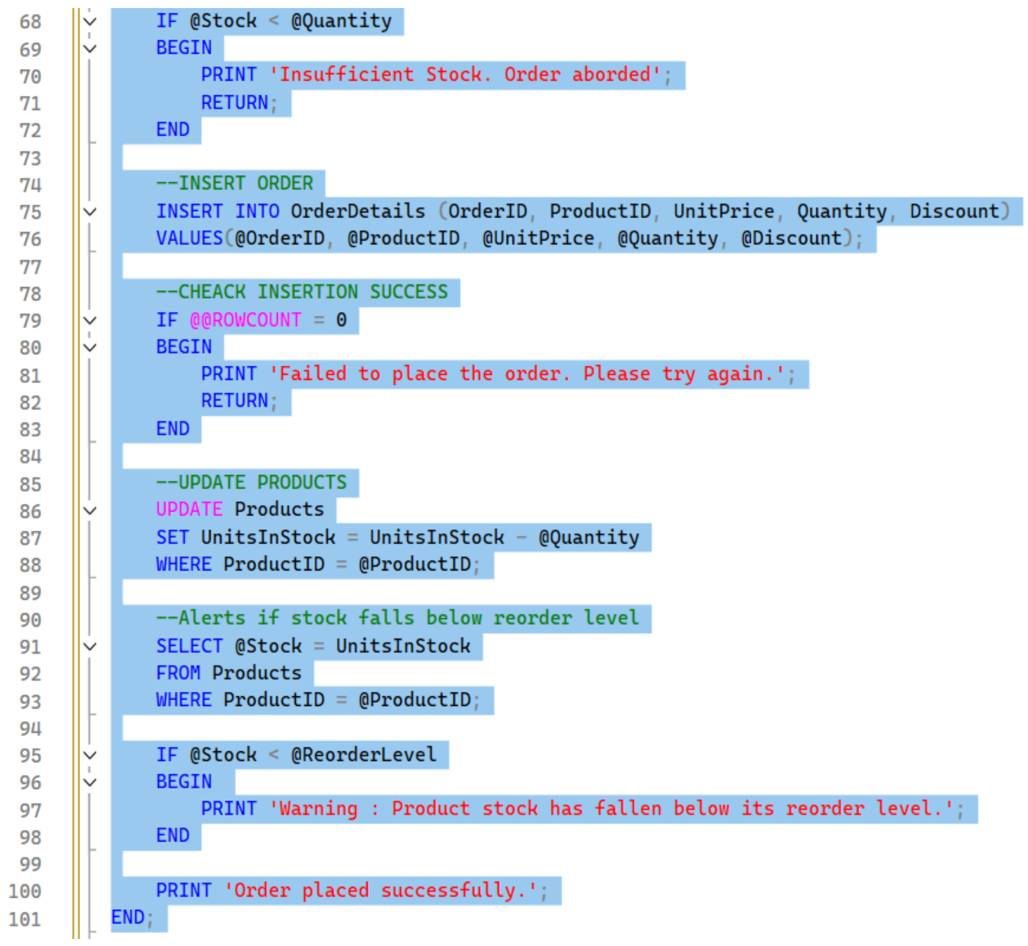


```sql
--Insert sample Products
INSERT INTO Products (ProductID, ProductName, UnitPrice, UnitsInStock, ReorderLevel)
VALUES
(1, 'Keyboard', 500.00, 50, 10),
(2, 'Mouse', 300.00, 20, 15),
(3, 'Monitor', 7000.00, 10, 5),
(4, 'Printer', 12000.00, 5, 3),
(5, 'USB Cable', 150.00, 100, 10),
(6, 'Webcam', 2500.00, 0, 2),
(7, 'Headphones', 2000.00, 3, 5);
```

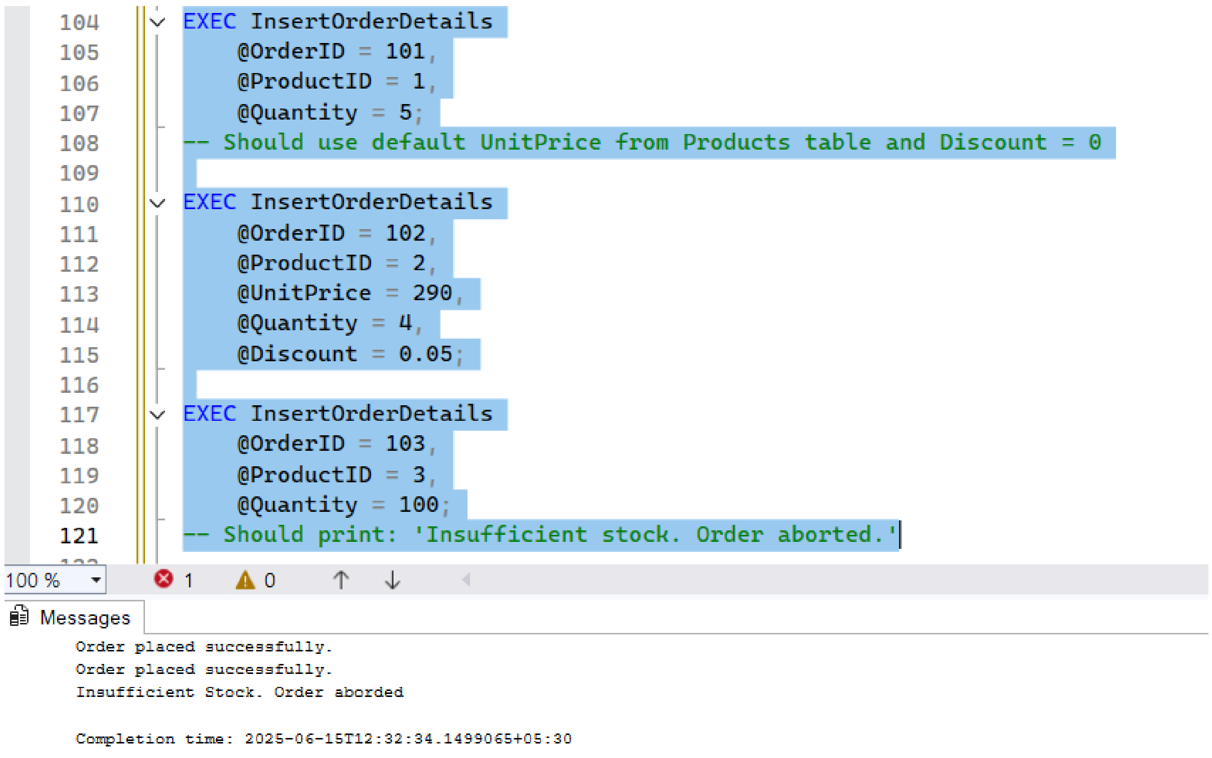
```
(7 rows affected)

Completion time: 2025-06-15T12:24:28.6331668+05:30
```



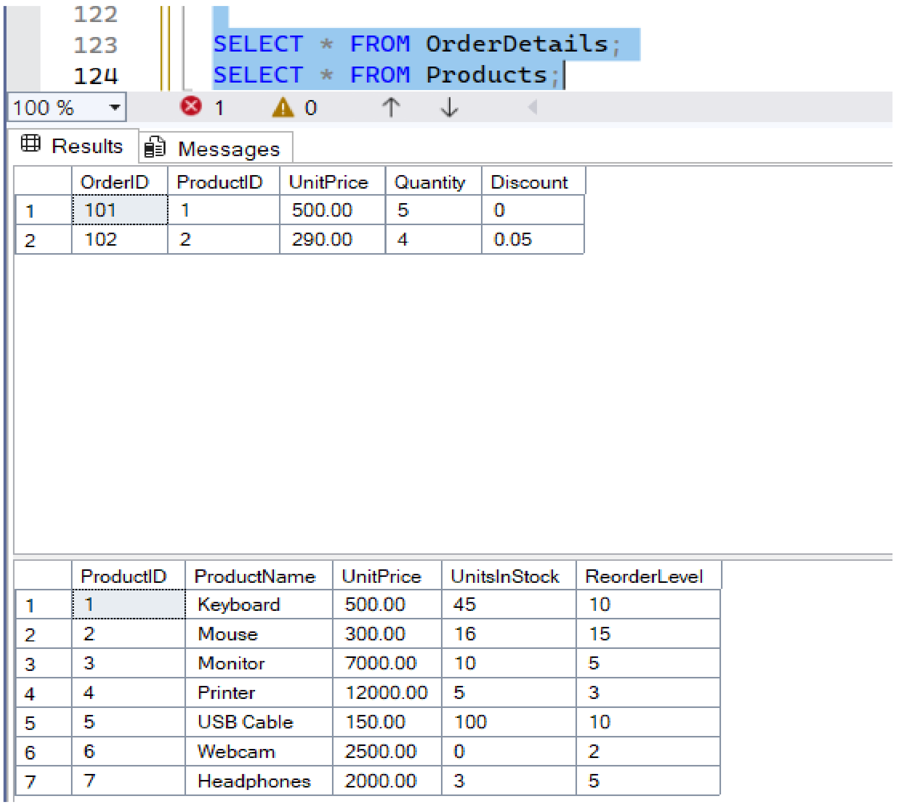
```sql
--Create the stored procedure
CREATE PROCEDURE InsertOrderDetails
    @OrderID INT,
    @ProductID INT,
    @UnitPrice MONEY = NULL,
    @Quantity INT,
    @Discount FLOAT = 0

AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Stock INT;
    DECLARE @ReorderLevel INT;

    --Get UnitPrice if not provided
    IF @UnitPrice IS NULL
    BEGIN
        SELECT @UnitPrice = UnitPrice
        FROM Products
        WHERE ProductID = @ProductID;
    END

    --Fetch stock and reoder level
    SELECT
        @Stock = UnitsInStock,
        @ReorderLevel = ReorderLevel
    FROM Products
    WHERE ProductID = @ProductID;

    --Validate stock
    IF @Stock IS NULL
    BEGIN
        PRINT 'INVALID ProductID';
        Return;
    END
```



```sql
    IF @Stock < @Quantity
    BEGIN
        PRINT 'Insufficient Stock. Order aborded';
        RETURN;
    END

    --INSERT ORDER
    INSERT INTO OrderDetails (OrderID, ProductID, UnitPrice, Quantity, Discount)
    VALUES(@OrderID, @ProductID, @UnitPrice, @Quantity, @Discount);

    --CHEACK INSERTION SUCCESS
    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Failed to place the order. Please try again.';
        RETURN;
    END

    --UPDATE PRODUCTS
    UPDATE Products
    SET UnitsInStock = UnitsInStock - @Quantity
    WHERE ProductID = @ProductID;

    --Alerts if stock falls below reorder level
    SELECT @Stock = UnitsInStock
    FROM Products
    WHERE ProductID = @ProductID;

    IF @Stock < @ReorderLevel
    BEGIN
        PRINT 'Warning : Product stock has fallen below its reorder level.';
    END

    PRINT 'Order placed successfully.';
END;
```


```sql
EXEC InsertOrderDetails
     @OrderID = 101,
     @ProductID = 1,
     @Quantity = 5;
-- Should use default UnitPrice from Products table and Discount = 0

EXEC InsertOrderDetails
     @OrderID = 102,
     @ProductID = 2,
     @UnitPrice = 290,
     @Quantity = 4,
     @Discount = 0.05;

EXEC InsertOrderDetails
     @OrderID = 103,
     @ProductID = 3,
     @Quantity = 100;
-- Should print: 'Insufficient stock. Order aborted.'
```

```
Order placed successfully.
Order placed successfully.
Insufficient Stock. Order aborded

Completion time: 2025-06-15T12:32:34.1499065+05:30
```


```sql
SELECT * FROM OrderDetails;
SELECT * FROM Products;
```

	OrderID	ProductID	UnitPrice	Quantity	Discount
1	101	1	500.00	5	0
2	102	2	290.00	4	0.05

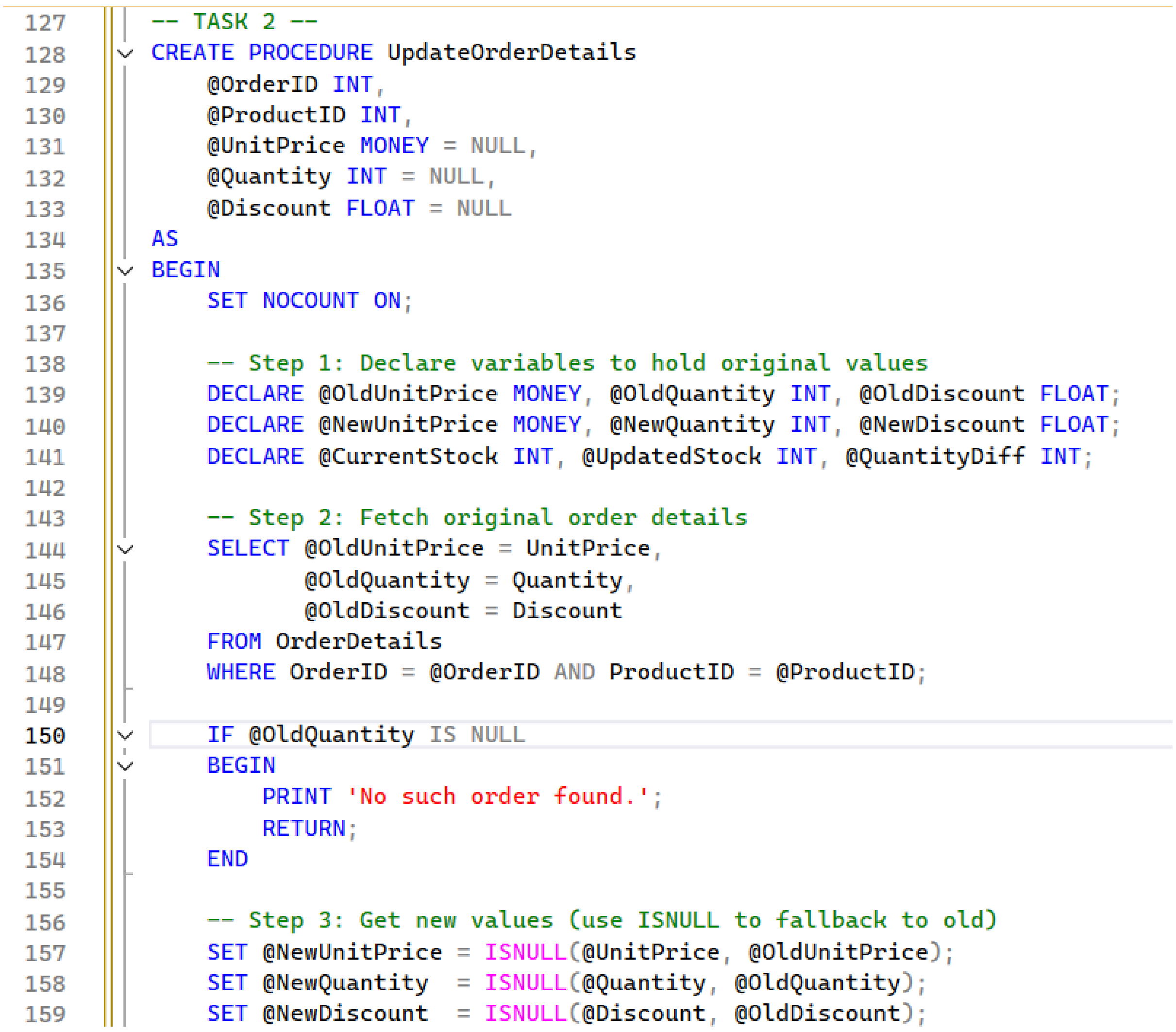
	ProductID	ProductName	UnitPrice	UnitsInStock	ReorderLevel
1	1	Keyboard	500.00	45	10
2	2	Mouse	300.00	16	15
3	3	Monitor	7000.00	10	5
4	4	Printer	12000.00	5	3
5	5	USB Cable	150.00	100	10
6	6	Webcam	2500.00	0	2
7	7	Headphones	2000.00	3	5

****

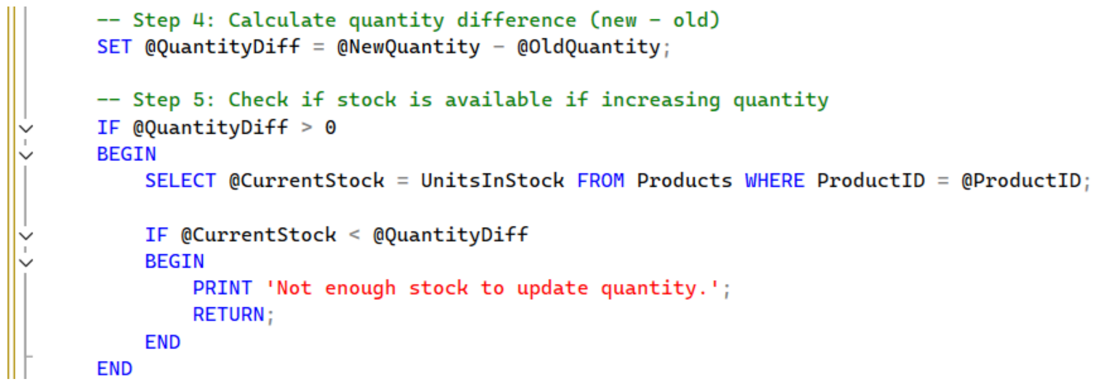
Task 2 :-

now "Create a procedure UpdateOrderDetails that takes OrderID, ProductID, UnitPrice, Quantity, and discount, and updates these values for that ProductID in that Order

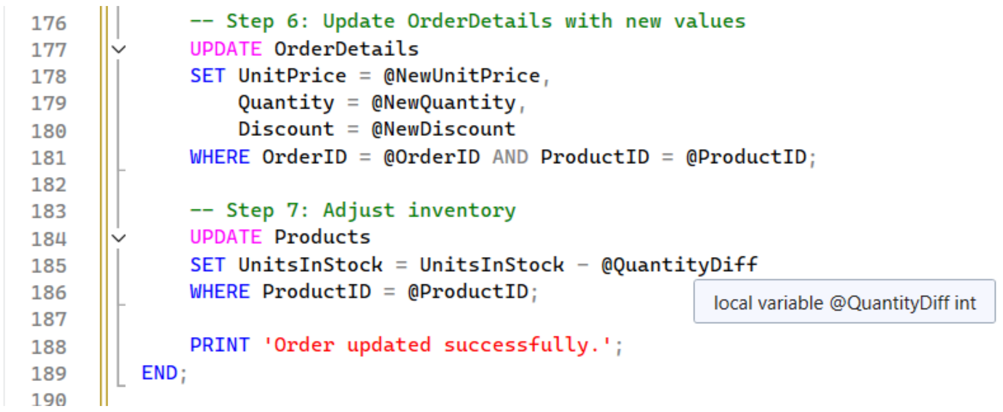
All the parameters except the OrderID and ProductID should be optional so that if the user wants to only update Quantity she should be able to do so without providing the rest of the values. You need to also make sure that if any of the values are being passed in as NULL, then you want to retain the original value instead of overwriting it with NULL. To accomplish this, look for the ISNULL() function in google or sql server books online. Adjust the UnitsInStock value in products table accordingly." complete this task



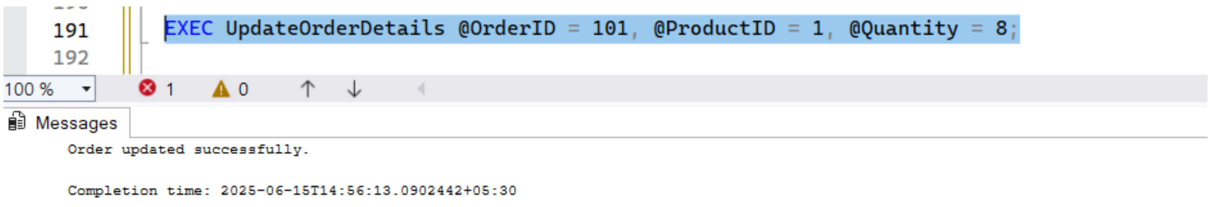
```sql
-- TASK 2 --
CREATE PROCEDURE UpdateOrderDetails
    @OrderID INT,
    @ProductID INT,
    @UnitPrice MONEY = NULL,
    @Quantity INT = NULL,
    @Discount FLOAT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Step 1: Declare variables to hold original values
    DECLARE @OldUnitPrice MONEY, @OldQuantity INT, @OldDiscount FLOAT;
    DECLARE @NewUnitPrice MONEY, @NewQuantity INT, @NewDiscount FLOAT;
    DECLARE @CurrentStock INT, @UpdatedStock INT, @QuantityDiff INT;

    -- Step 2: Fetch original order details
    SELECT @OldUnitPrice = UnitPrice,
           @OldQuantity = Quantity,
           @OldDiscount = Discount
    FROM OrderDetails
    WHERE OrderID = @OrderID AND ProductID = @ProductID;

    IF @OldQuantity IS NULL
    BEGIN
        PRINT 'No such order found.';
        RETURN;
    END

    -- Step 3: Get new values (use ISNULL to fallback to old)
    SET @NewUnitPrice = ISNULL(@UnitPrice, @OldUnitPrice);
    SET @NewQuantity  = ISNULL(@Quantity, @OldQuantity);
    SET @NewDiscount  = ISNULL(@Discount, @OldDiscount);
```



```sql
    -- Step 4: Calculate quantity difference (new - old)
    SET @QuantityDiff = @NewQuantity - @OldQuantity;

    -- Step 5: Check if stock is available if increasing quantity
    IF @QuantityDiff > 0
    BEGIN
        SELECT @CurrentStock = UnitsInStock FROM Products WHERE ProductID = @ProductID;

        IF @CurrentStock < @QuantityDiff
        BEGIN
            PRINT 'Not enough stock to update quantity.';
            RETURN;
        END
    END
```

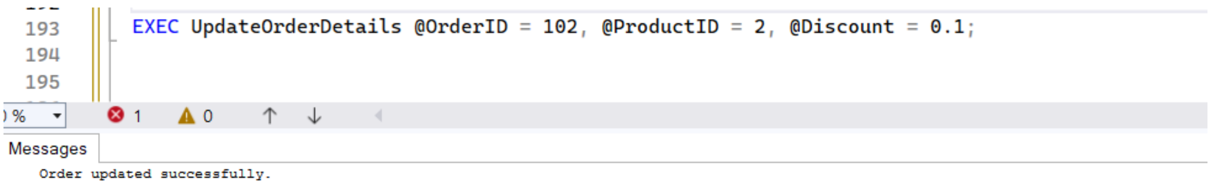


```sql
    -- Step 6: Update OrderDetails with new values
    UPDATE OrderDetails
    SET UnitPrice = @NewUnitPrice,
        Quantity = @NewQuantity,
        Discount = @NewDiscount
    WHERE OrderID = @OrderID AND ProductID = @ProductID;

    -- Step 7: Adjust inventory
    UPDATE Products
    SET UnitsInStock = UnitsInStock - @QuantityDiff
    WHERE ProductID = @ProductID;

    PRINT 'Order updated successfully.';
END;
```



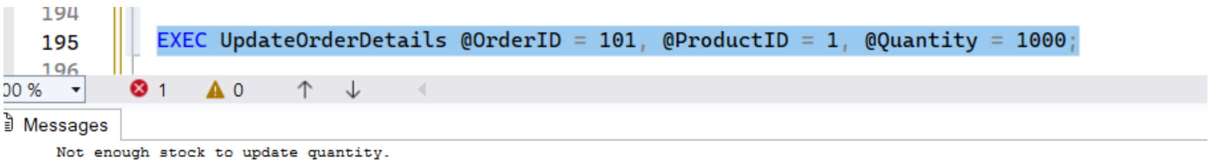
```sql
EXEC UpdateOrderDetails @OrderID = 101, @ProductID = 1, @Quantity = 8;
```

```
Order updated successfully.

Completion time: 2025-06-15T14:56:13.0902442+05:30
```



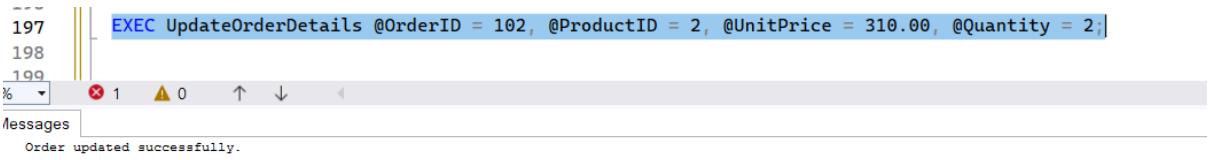
```sql
EXEC UpdateOrderDetails @OrderID = 102, @ProductID = 2, @Discount = 0.1;
```

```
Order updated successfully.
```



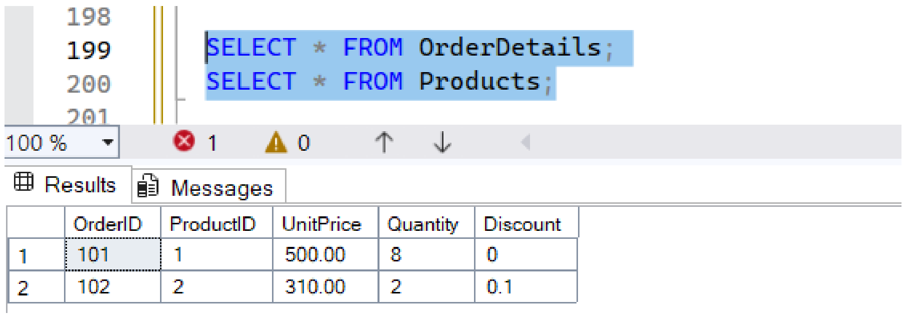
```sql
EXEC UpdateOrderDetails @OrderID = 101, @ProductID = 1, @Quantity = 1000;
```

```
Not enough stock to update quantity.
```



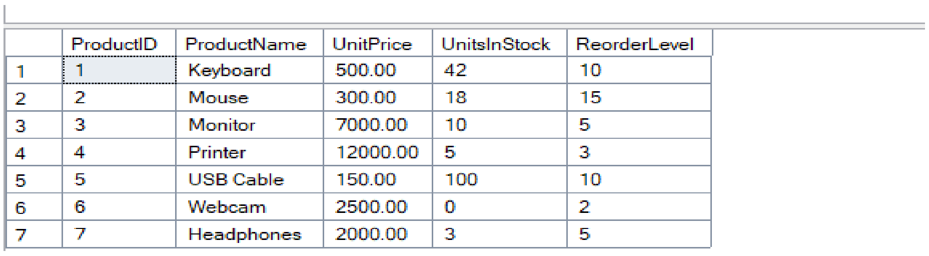
```sql
EXEC UpdateOrderDetails @OrderID = 102, @ProductID = 2, @UnitPrice = 310.00, @Quantity = 2;
```

```
Order updated successfully.
```



```sql
SELECT * FROM OrderDetails;
SELECT * FROM Products;
```

	OrderID	ProductID	UnitPrice	Quantity	Discount
1	101	1	500.00	8	0
2	102	2	310.00	2	0.1

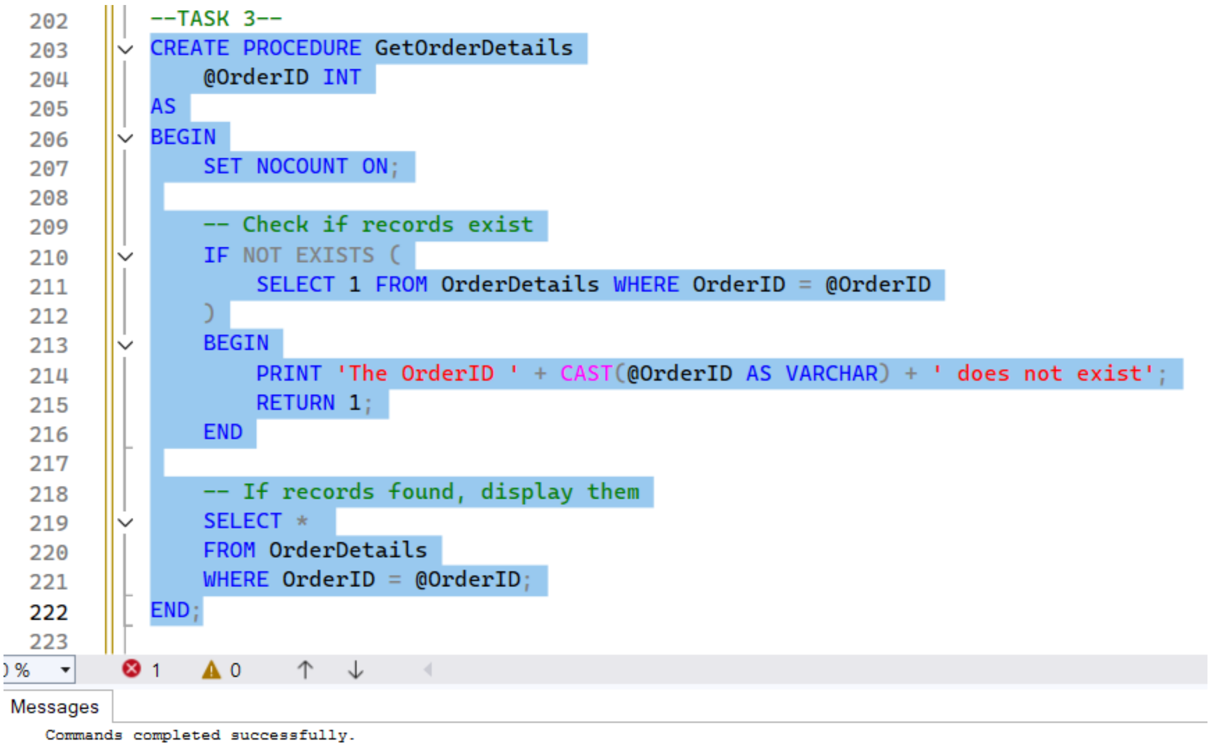


	ProductID	ProductName	UnitPrice	UnitsInStock	ReorderLevel
1	1	Keyboard	500.00	42	10
2	2	Mouse	300.00	18	15
3	3	Monitor	7000.00	10	5
4	4	Printer	12000.00	5	3
5	5	USB Cable	150.00	100	10
6	6	Webcam	2500.00	0	2
7	7	Headphones	2000.00	3	5

****

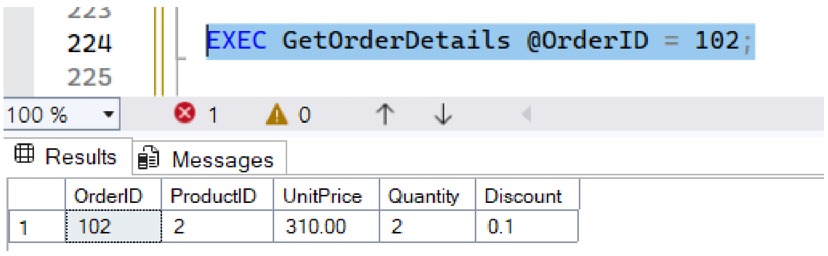
Task 3 :-

Create a procedure GetOrderDetails that takes OrderID as input parameter and retums all the records for that OrderID. If no records are found in Order Details table, then it should print the line: "The OrderID XXXX does not exits", where XXX should be the OrderID entered by user and the procedure should RETURN the value 1.



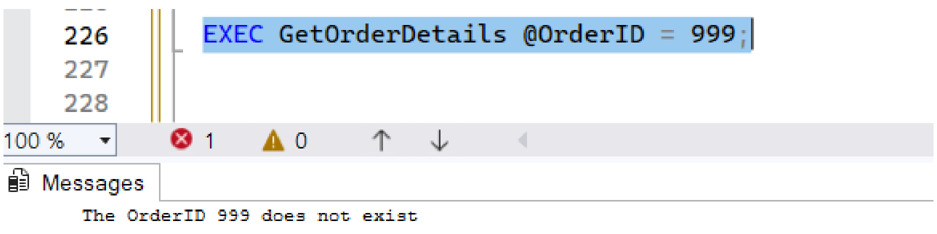
```
--TASK 3--
CREATE PROCEDURE GetOrderDetails
    @OrderID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if records exist
    IF NOT EXISTS (
        SELECT 1 FROM OrderDetails WHERE OrderID = @OrderID
    )
    BEGIN
        PRINT 'The OrderID ' + CAST(@OrderID AS VARCHAR) + ' does not exist';
        RETURN 1;
    END

    -- If records found, display them
    SELECT *
    FROM OrderDetails
    WHERE OrderID = @OrderID;
END;
```

```
Commands completed successfully.
```



```
EXEC GetOrderDetails @OrderID = 102;
```

	OrderID	ProductID	UnitPrice	Quantity	Discount
1	102	2	310.00	2	0.1



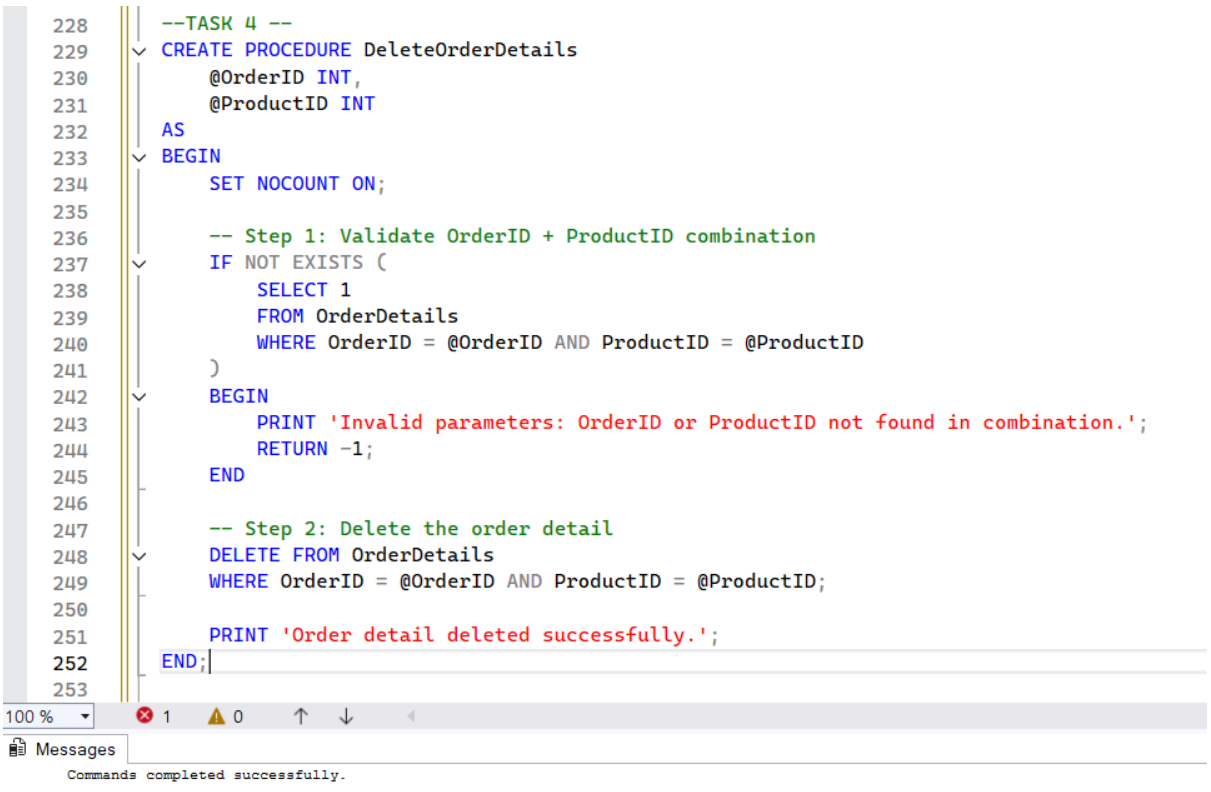
```
EXEC GetOrderDetails @OrderID = 999;
```

```
The OrderID 999 does not exist
```

****

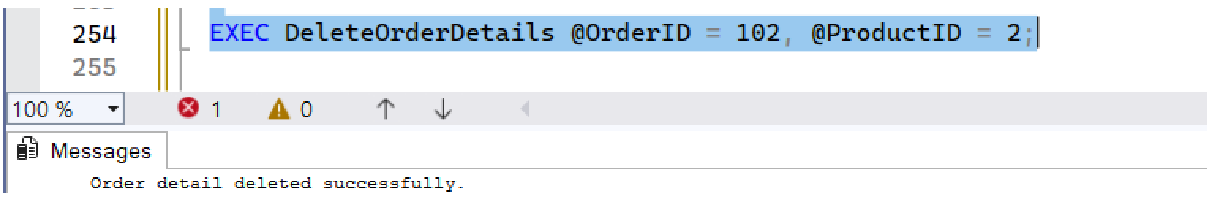
TASK 4 :-

Create a procedure DeleteOrderDetails that takes OrderID and ProductID and deletes that from Order Details table. Your procedure should validate parameters. It should return an error code (-1) and print a message if the parameters are invalid. Parameters are valid if the given order ID appears in the table and if the given product ID appears in that order.



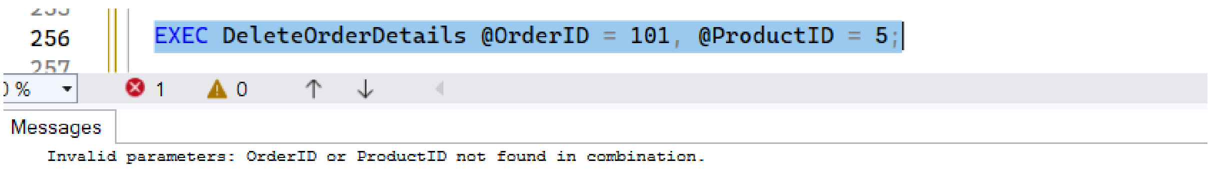
```sql
--TASK 4 --
CREATE PROCEDURE DeleteOrderDetails
    @OrderID INT,
    @ProductID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Step 1: Validate OrderID + ProductID combination
    IF NOT EXISTS (
        SELECT 1
        FROM OrderDetails
        WHERE OrderID = @OrderID AND ProductID = @ProductID
    )
    BEGIN
        PRINT 'Invalid parameters: OrderID or ProductID not found in combination.';
        RETURN -1;
    END

    -- Step 2: Delete the order detail
    DELETE FROM OrderDetails
    WHERE OrderID = @OrderID AND ProductID = @ProductID;

    PRINT 'Order detail deleted successfully.';
END;
```

```
Commands completed successfully.
```



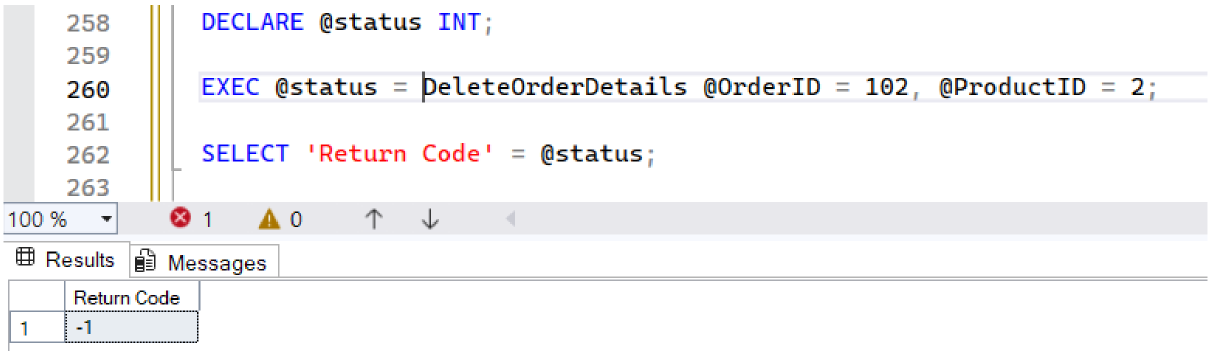
```sql
EXEC DeleteOrderDetails @OrderID = 102, @ProductID = 2;
```

```
Order detail deleted successfully.
```



```sql
EXEC DeleteOrderDetails @OrderID = 101, @ProductID = 5;
```

```
Invalid parameters: OrderID or ProductID not found in combination.
```



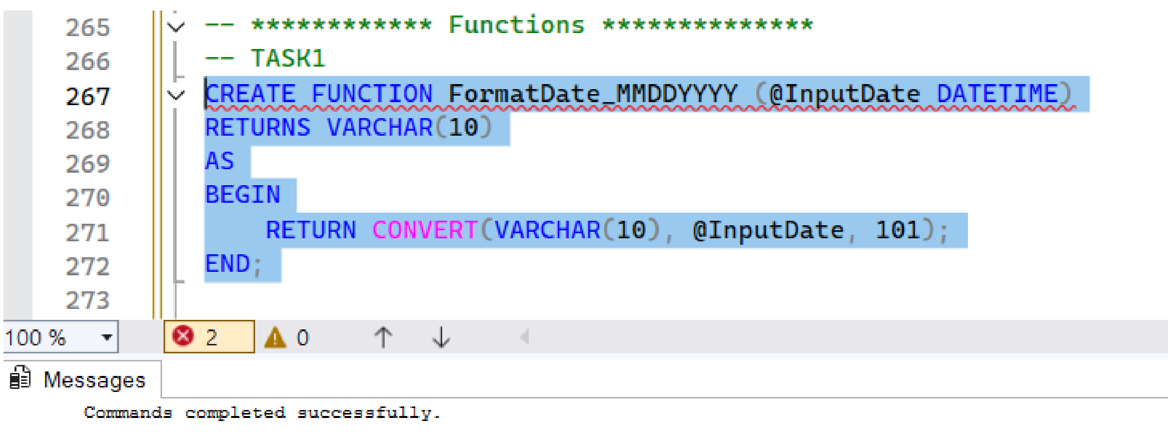
```sql
DECLARE @status INT;

EXEC @status = DeleteOrderDetails @OrderID = 102, @ProductID = 2;

SELECT 'Return Code' = @status;
```

	Return Code
1	-1

****

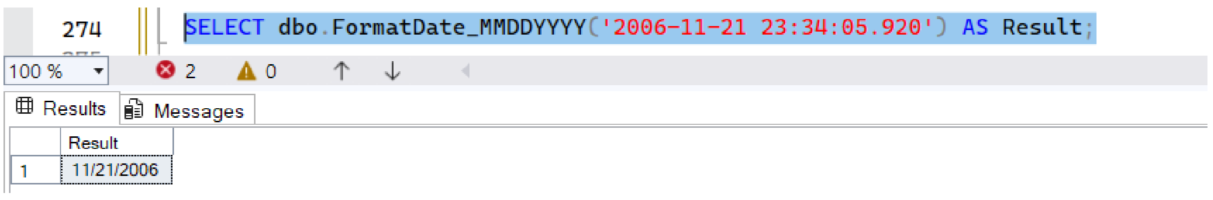
Functions :-

TASK 1 : Create a function that takes an input parameter type datetime and returns the date in the format MM/DD/YYYY. For example if I pass in '2006-11-21 23:34:05.920', the output of the functions should be 11/21/2006



```sql
-- ************* Functions **************
-- TASK1
CREATE FUNCTION FormatDate_MMDDYYYY (@InputDate DATETIME)
RETURNS VARCHAR(10)
AS
BEGIN
    RETURN CONVERT(VARCHAR(10), @InputDate, 101);
END;
```

```
Commands completed successfully.
```



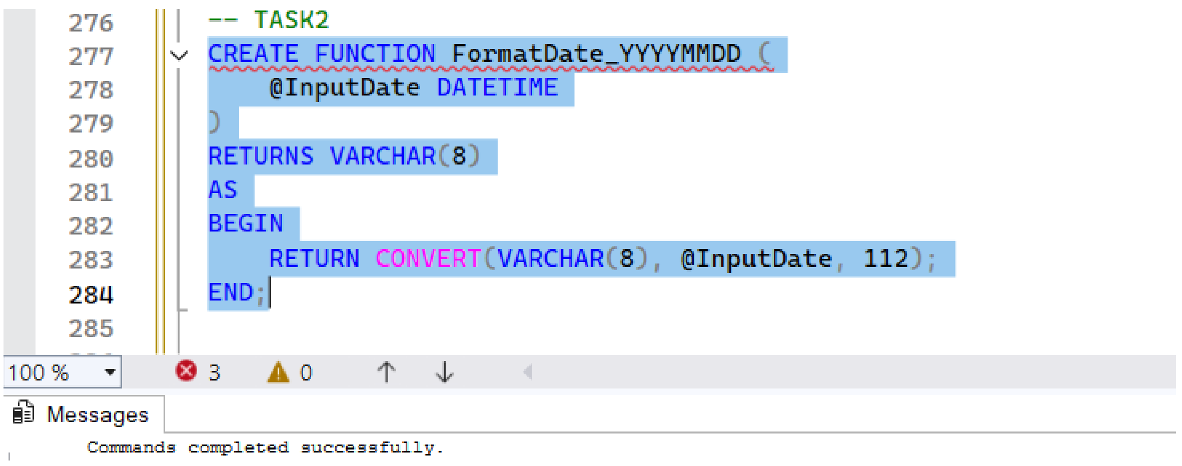
```sql
SELECT dbo.FormatDate_MMDDYYYY('2006-11-21 23:34:05.920') AS Result;
```

	Result
1	11/21/2006

****

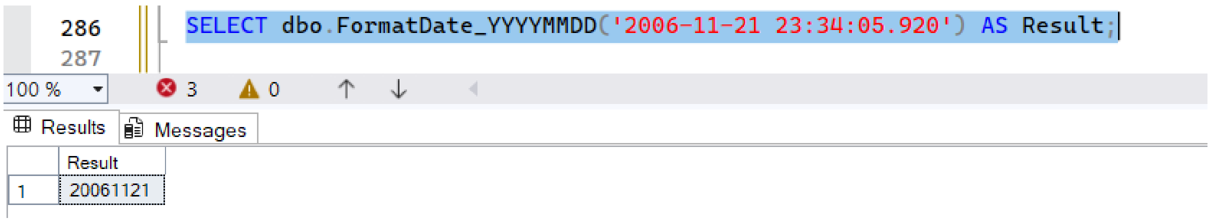
TASK 2:-

Create a function that takes an input parameter type datetime and returns the date in the format YYYYMMDD



```sql
-- TASK2
CREATE FUNCTION FormatDate_YYYYMMDD (
    @InputDate DATETIME
)
RETURNS VARCHAR(8)
AS
BEGIN
    RETURN CONVERT(VARCHAR(8), @InputDate, 112);
END;
```

```
Commands completed successfully.
```



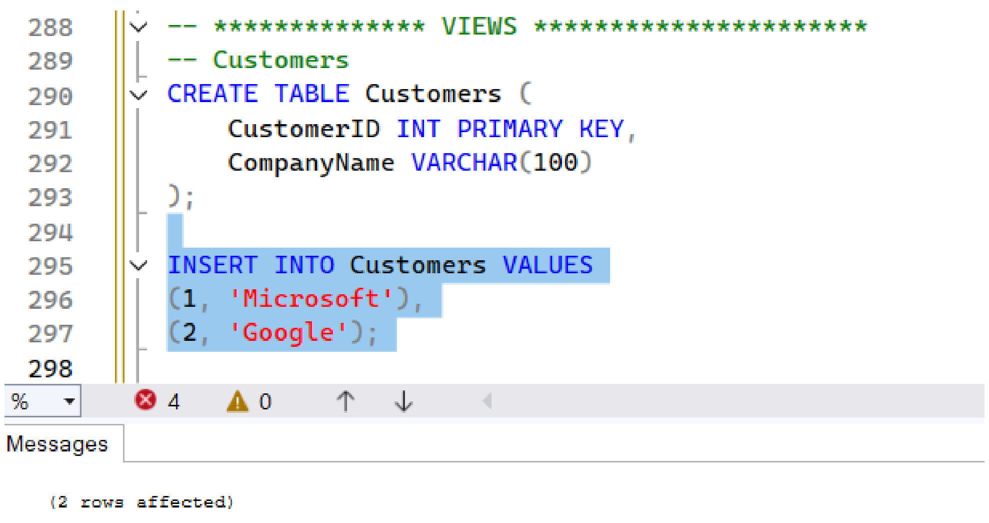
```sql
SELECT dbo.FormatDate_YYYYMMDD('2006-11-21 23:34:05.920') AS Result;
```

	Result
1	20061121

****

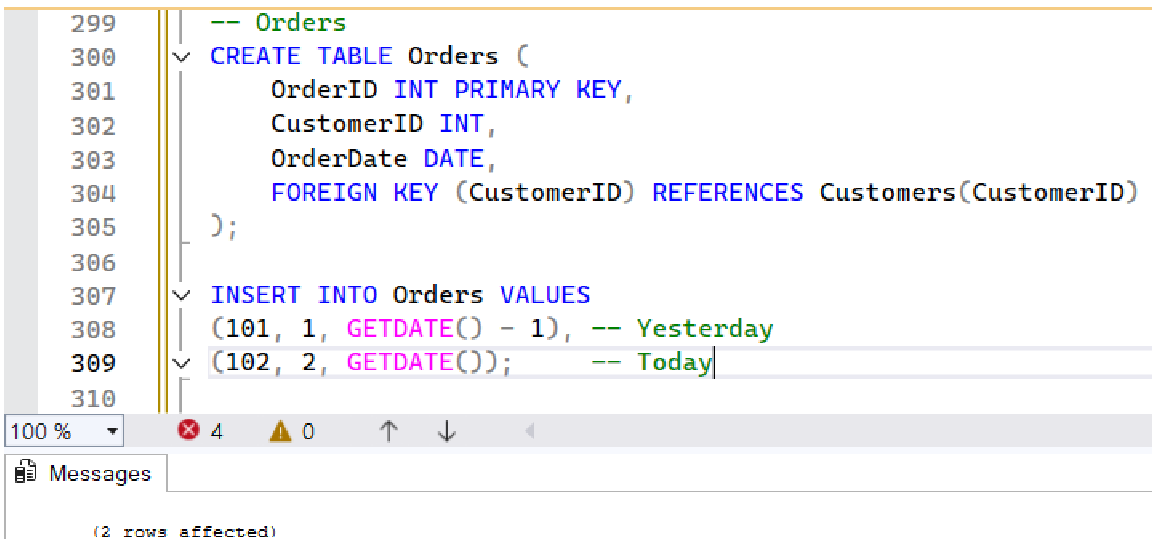
Views :

TASK 1 :- Create a view vwCustomerOrders which returns CompanyName, OrderID, OrderDate ProductID, Product Name, Quantity, Unit Price, Quantity od UnitPrice



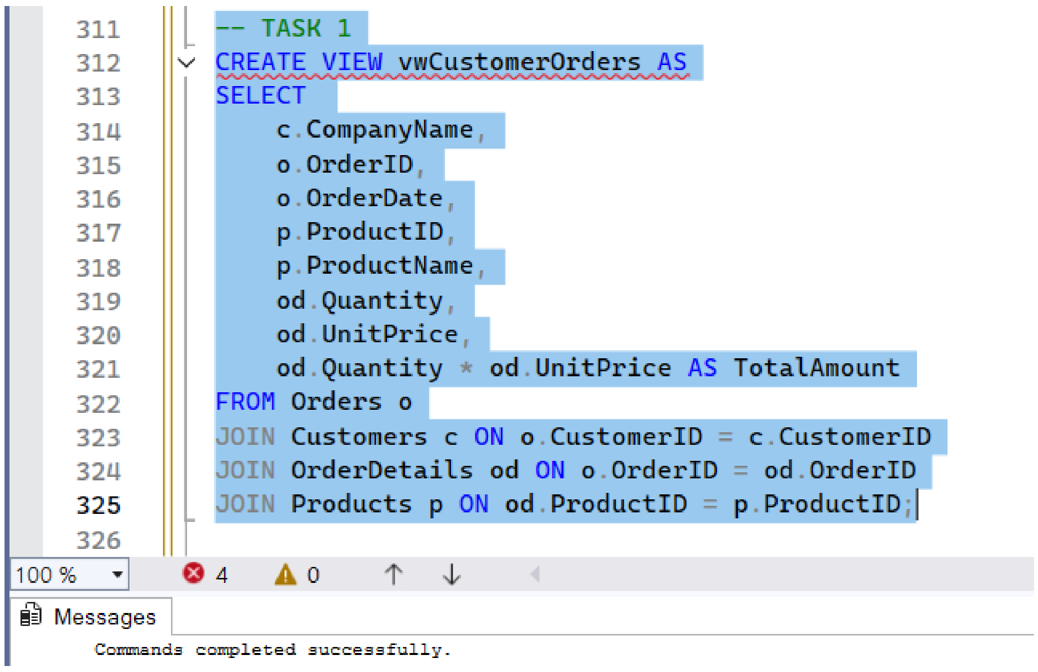
```sql
-- ************** VIEWS **********************
-- Customers
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CompanyName VARCHAR(100)
);

INSERT INTO Customers VALUES
(1, 'Microsoft'),
(2, 'Google');
```

```
(2 rows affected)
```



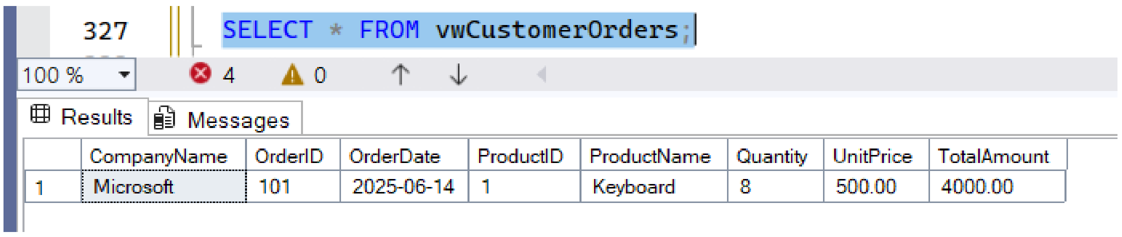
```sql
-- Orders
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

INSERT INTO Orders VALUES
(101, 1, GETDATE() - 1), -- Yesterday
(102, 2, GETDATE());     -- Today
```

```
(2 rows affected)
```



```sql
-- TASK 1
CREATE VIEW vwCustomerOrders AS
SELECT
    c.CompanyName,
    o.OrderID,
    o.OrderDate,
    p.ProductID,
    p.ProductName,
    od.Quantity,
    od.UnitPrice,
    od.Quantity * od.UnitPrice AS TotalAmount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products p ON od.ProductID = p.ProductID;
```

```
Commands completed successfully.
```



```
SELECT * FROM vwCustomerOrders;
```

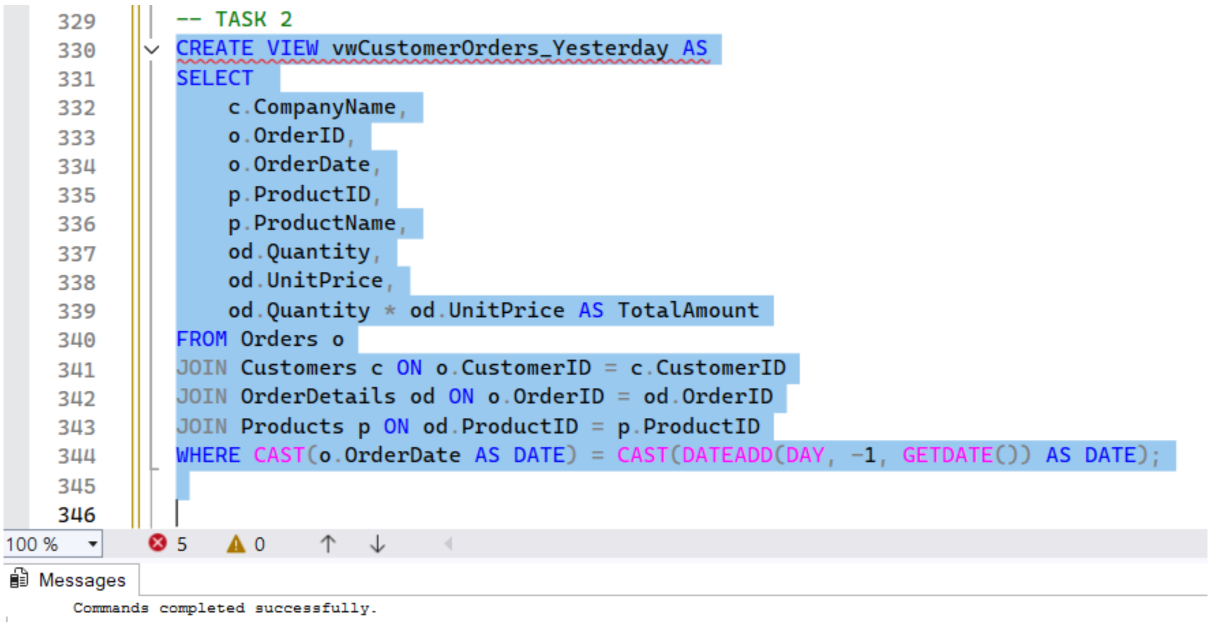
```
100 %    ⊗ 4   ⚠ 0
Results   Messages
```

	CompanyName	OrderID	OrderDate	ProductID	ProductName	Quantity	UnitPrice	TotalAmount
1	Microsoft	101	2025-06-14	1	Keyboard	8	500.00	4000.00

****

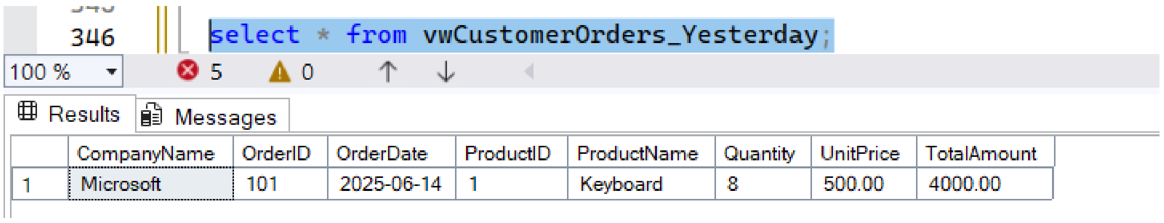
TASK 2 :-

Create a copy of the above view and modify it so that it only returns the above information for orders that were placed yesterday



```
-- TASK 2
CREATE VIEW vwCustomerOrders_Yesterday AS
SELECT
    c.CompanyName,
    o.OrderID,
    o.OrderDate,
    p.ProductID,
    p.ProductName,
    od.Quantity,
    od.UnitPrice,
    od.Quantity * od.UnitPrice AS TotalAmount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
JOIN OrderDetails od ON o.OrderID = od.OrderID
JOIN Products p ON od.ProductID = p.ProductID
WHERE CAST(o.OrderDate AS DATE) = CAST(DATEADD(DAY, -1, GETDATE()) AS DATE);
```

```
100 %    ⊗ 5   ⚠ 0
Messages
   Commands completed successfully.
```



```
select * from vwCustomerOrders_Yesterday;
```

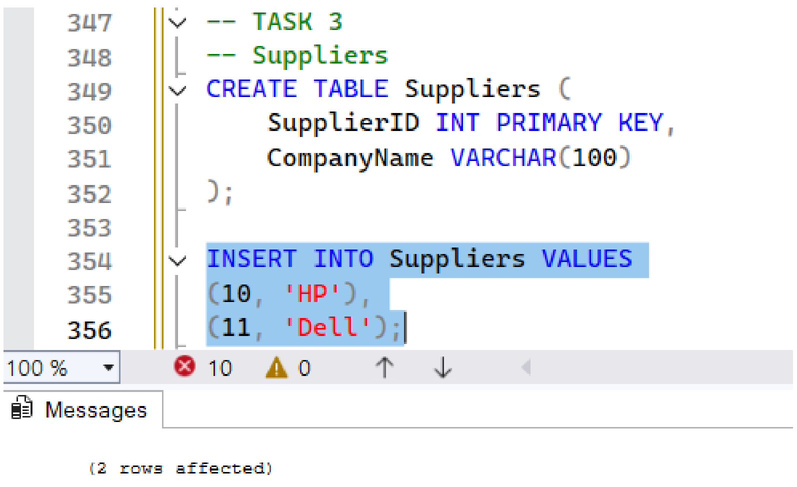
```
100 %    ⊗ 5   ⚠ 0
Results   Messages
```

	CompanyName	OrderID	OrderDate	ProductID	ProductName	Quantity	UnitPrice	TotalAmount
1	Microsoft	101	2025-06-14	1	Keyboard	8	500.00	4000.00

****

TASK 3 :-

Use a CREATE VIEW statement to create a view called MyProducts. Your view should contain the ProductID, ProductName, QuantityPerUnit and UnitPrice columns from the Products table. It should also contain the Company Name column from the Suppliers table and the CategoryName column from the Categories table. Your view should only contain products that are not discontinued.

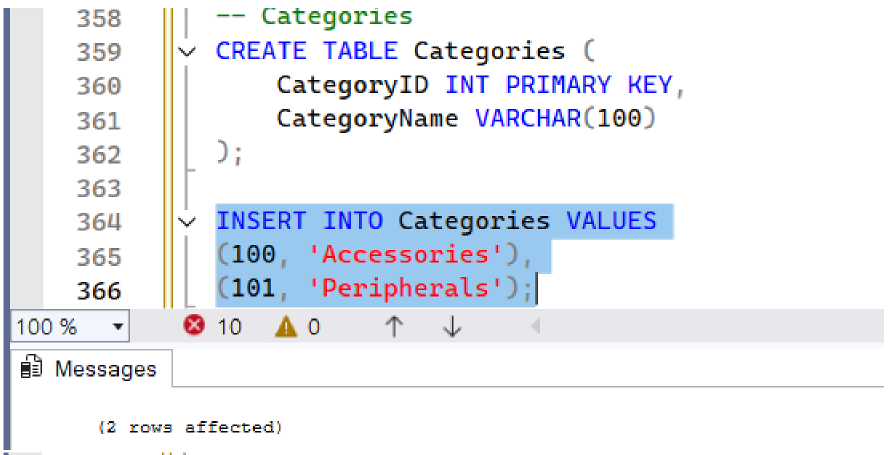


```sql
-- TASK 3
-- Suppliers
CREATE TABLE Suppliers (
    SupplierID INT PRIMARY KEY,
    CompanyName VARCHAR(100)
);

INSERT INTO Suppliers VALUES
(10, 'HP'),
(11, 'Dell');
```

100 % · 10 · 0

Messages

```
(2 rows affected)
```

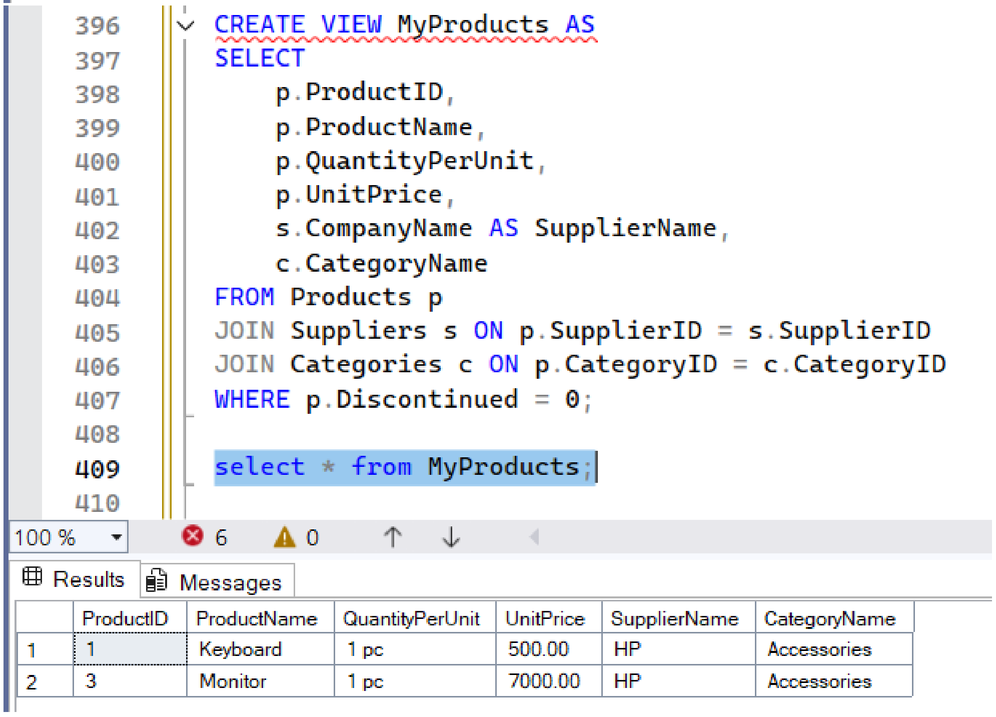


```sql
-- Categories
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(100)
);

INSERT INTO Categories VALUES
(100, 'Accessories'),
(101, 'Peripherals');
```

100 % · 10 · 0

Messages

```
(2 rows affected)
```



```sql
CREATE VIEW MyProducts AS
SELECT
    p.ProductID,
    p.ProductName,
    p.QuantityPerUnit,
    p.UnitPrice,
    s.CompanyName AS SupplierName,
    c.CategoryName
FROM Products p
JOIN Suppliers s ON p.SupplierID = s.SupplierID
JOIN Categories c ON p.CategoryID = c.CategoryID
WHERE p.Discontinued = 0;

select * from MyProducts;
```

100 % · 6 · 0

Results · Messages

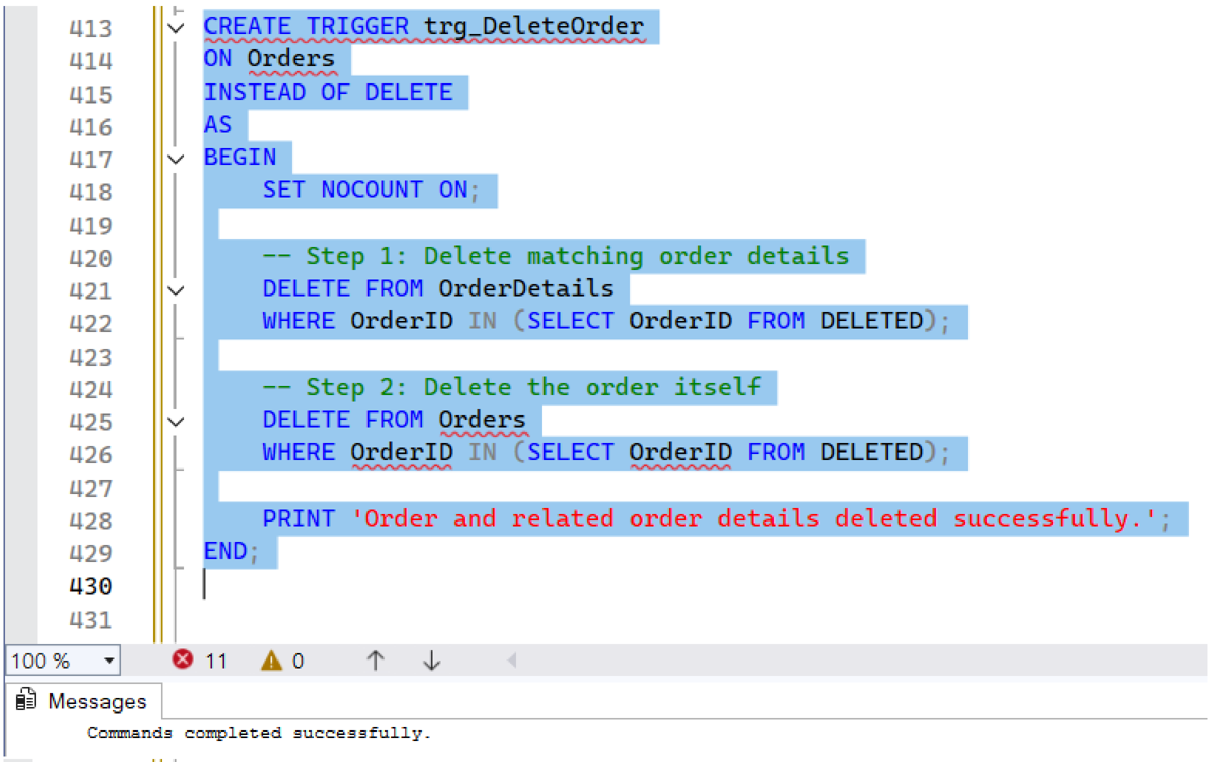
	ProductID	ProductName	QuantityPerUnit	UnitPrice	SupplierName	CategoryName
1	1	Keyboard	1 pc	500.00	HP	Accessories
2	3	Monitor	1 pc	7000.00	HP	Accessories

****

<u>**Triggers :-**</u>

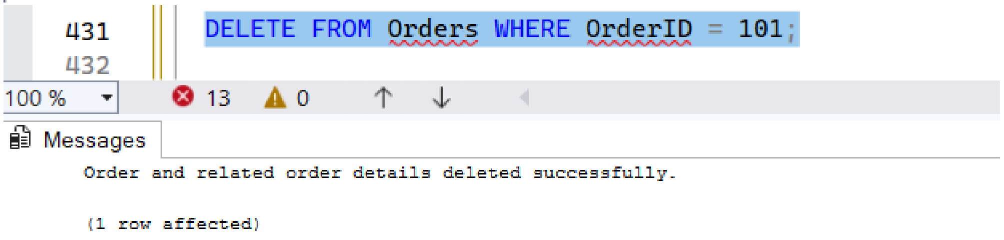
TASK 1 :

If someone cancels an order in northwind database, then you want to delete that order from the Orders table. But you will not be able to delete that Order before deleting the records from Order Details table for that particular order due to referential integrity constraints. Create an Instead of Delete trigger on Orders table so that if some one tries to delete an Order that trigger gets fired and that trigger should first delete everything in order details table and then delete that order from the Orders table



```
CREATE TRIGGER trg_DeleteOrder
ON Orders
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- Step 1: Delete matching order details
    DELETE FROM OrderDetails
    WHERE OrderID IN (SELECT OrderID FROM DELETED);

    -- Step 2: Delete the order itself
    DELETE FROM Orders
    WHERE OrderID IN (SELECT OrderID FROM DELETED);

    PRINT 'Order and related order details deleted successfully.';
END;
```

```
Commands completed successfully.
```



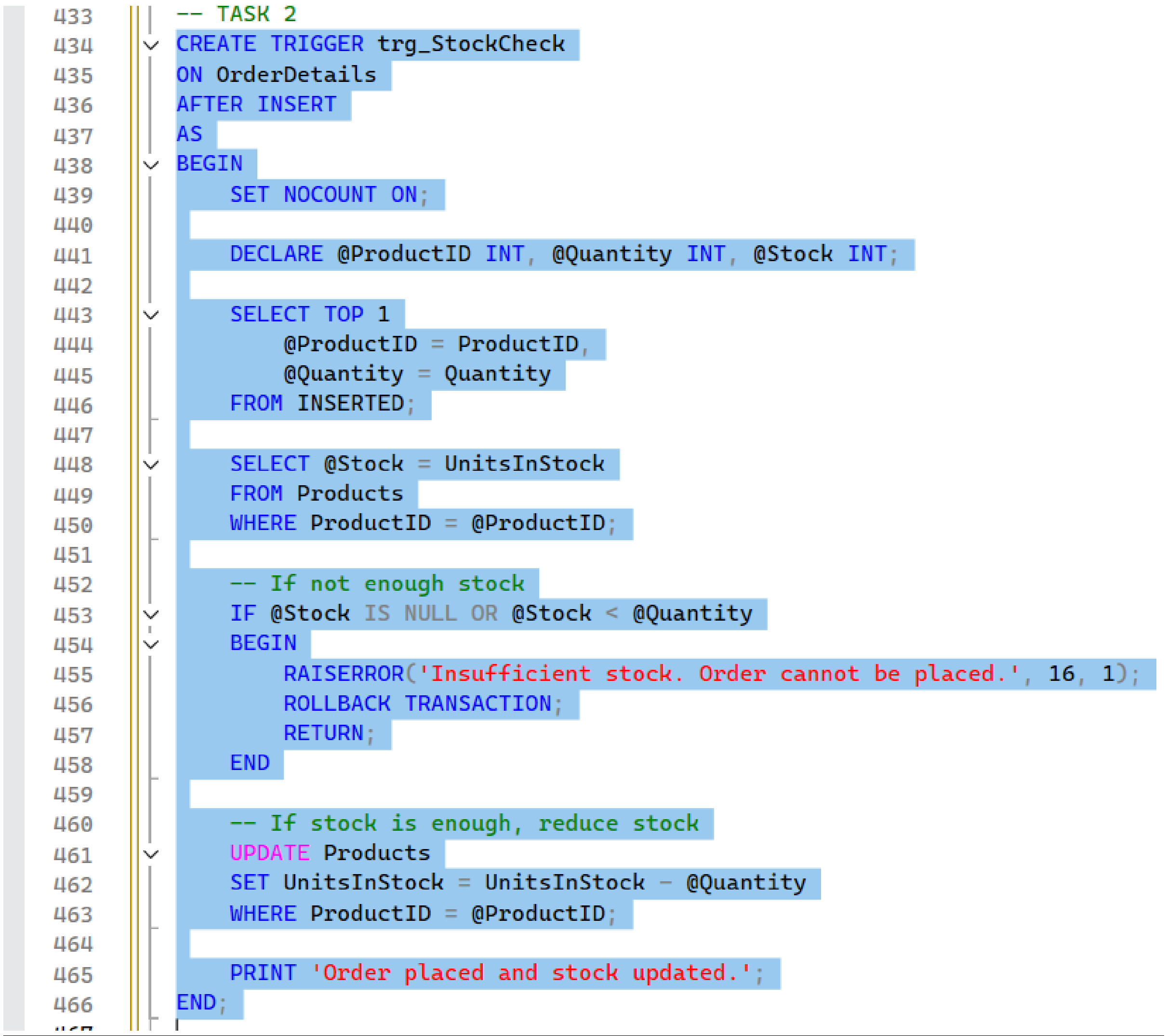
```
DELETE FROM Orders WHERE OrderID = 101;
```

```
Order and related order details deleted successfully.

(1 row affected)
```

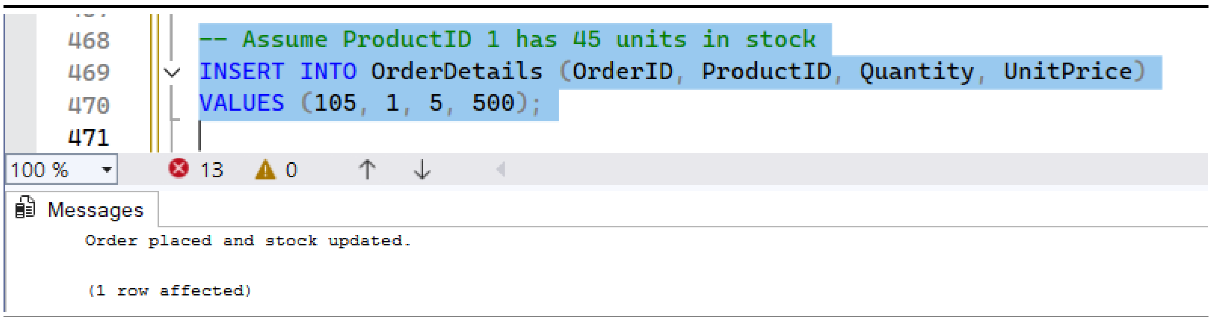
****

TASK 2 :

When an order is placed for X units of product Y, we must first check the Products table to ensure that there is sufficient stock to fill the order. This trigger will operate on the Order Details table. If sufficient stock exists, then fill the order and decrement X units from the UnitsInStock column in Products. If insufficient stock exists, then refuse the order (ie do not insert it) and notify the user that the order could not be filled because of insufficient stock.

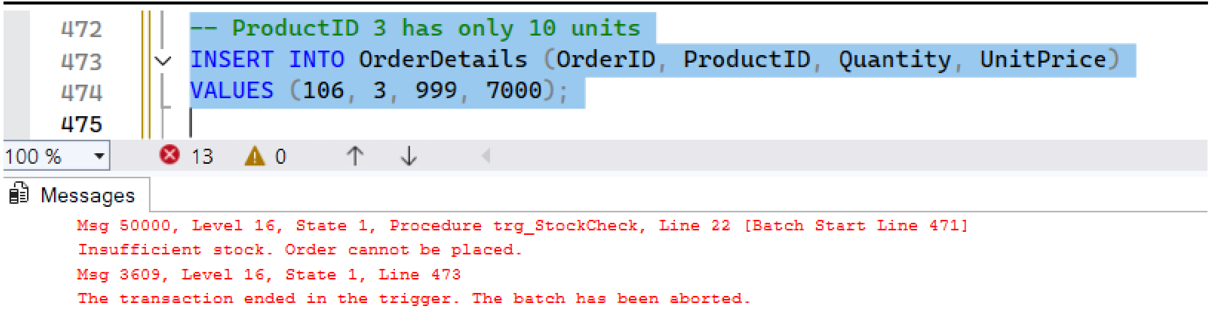


```sql
-- TASK 2
CREATE TRIGGER trg_StockCheck
ON OrderDetails
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ProductID INT, @Quantity INT, @Stock INT;

    SELECT TOP 1
        @ProductID = ProductID,
        @Quantity = Quantity
    FROM INSERTED;

    SELECT @Stock = UnitsInStock
    FROM Products
    WHERE ProductID = @ProductID;

    -- If not enough stock
    IF @Stock IS NULL OR @Stock < @Quantity
    BEGIN
        RAISERROR('Insufficient stock. Order cannot be placed.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- If stock is enough, reduce stock
    UPDATE Products
    SET UnitsInStock = UnitsInStock - @Quantity
    WHERE ProductID = @ProductID;

    PRINT 'Order placed and stock updated.';
END;
```



```sql
-- Assume ProductID 1 has 45 units in stock
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, UnitPrice)
VALUES (105, 1, 5, 500);
```

```
Order placed and stock updated.

(1 row affected)
```



```sql
-- ProductID 3 has only 10 units
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, UnitPrice)
VALUES (106, 3, 999, 7000);
```

```
Msg 50000, Level 16, State 1, Procedure trg_StockCheck, Line 22 [Batch Start Line 471]
Insufficient stock. Order cannot be placed.
Msg 3609, Level 16, State 1, Line 473
The transaction ended in the trigger. The batch has been aborted.
```