

## **Assignment - Week II**

**Domain: SQL**

**Name: Harsh Bhasin**

**Student ID: CT\_CSI\_SQ\_3533**

**Contact No :9098835618**

**Email ID :Harshbha30@gmail.com**

## Part 1: Stored Procedures

### 1. InsertOrderDetails Procedure:

```
Query: CREATE PROCEDURE InsertOrderDetails
(
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(10,2) = NULL,
    @Quantity INT,
    @Discount DECIMAL(5,2) = 0
)
AS
BEGIN
    IF @UnitPrice IS NULL
    BEGIN
        SELECT @UnitPrice = UnitPrice FROM Products WHERE ProductID = @ProductID
    END

    IF @Discount IS NULL
        SET @Discount = 0

    DECLARE @StockQuantity INT
    SELECT @StockQuantity = UnitsInStock FROM Products WHERE ProductID = @ProductID

    IF @StockQuantity >= @Quantity
    BEGIN
        INSERT INTO [Order Details] (OrderID, ProductID, UnitPrice, Quantity, Discount)
        VALUES (@OrderID, @ProductID, @UnitPrice, @Quantity, @Discount)

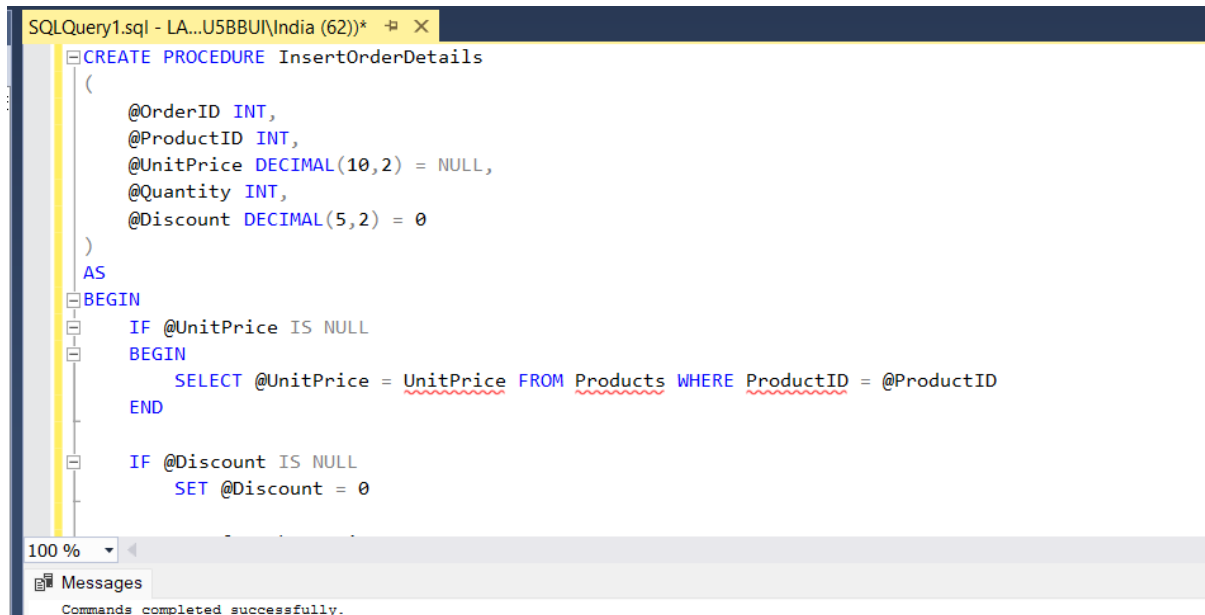
        UPDATE Products
        SET UnitsInStock = UnitsInStock - @Quantity
        WHERE ProductID = @ProductID

        DECLARE @ReorderLevel INT
        SELECT @ReorderLevel = ReorderLevel FROM Products WHERE ProductID = @ProductID

        IF (@StockQuantity - @Quantity) < @ReorderLevel
        BEGIN
            PRINT 'Warning: Product stock is below reorder level!'
        END

        PRINT 'Order placed successfully!'
    END
    ELSE
    BEGIN
        PRINT 'Failed to place order. Not enough stock available.'
    END
END
```

Result:



The screenshot shows a SQL Server Enterprise Manager window with a query editor. The query is a T-SQL statement to create a stored procedure named 'InsertOrderDetails'. The procedure has five parameters: @OrderID (INT), @ProductID (INT), @UnitPrice (DECIMAL(10,2) = NULL), @Quantity (INT), and @Discount (DECIMAL(5,2) = 0). The procedure body starts with a BEGIN statement, followed by an IF statement that checks if @UnitPrice is NULL. If it is, a SELECT statement is used to retrieve the UnitPrice from the Products table where ProductID matches @ProductID. This is followed by an END statement, then another IF statement checking if @Discount is NULL, and finally a SET statement to set @Discount to 0. The procedure ends with an AS statement. The bottom of the window shows a Messages pane with the text 'Commands completed successfully.'

```
SQLQuery1.sql - LA...U5BBUI\India (62))* X
CREATE PROCEDURE InsertOrderDetails
(
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(10,2) = NULL,
    @Quantity INT,
    @Discount DECIMAL(5,2) = 0
)
AS
BEGIN
    IF @UnitPrice IS NULL
    BEGIN
        SELECT @UnitPrice = UnitPrice FROM Products WHERE ProductID = @ProductID
    END

    IF @Discount IS NULL
        SET @Discount = 0
END
```

100 %  
Messages  
Commands completed successfully.

## 2. UpdateOrderDetails Procedure:

### Query:

CREATE PROCEDURE UpdateOrderDetails

```
(
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(10,2) = NULL,
    @Quantity INT = NULL,
    @Discount DECIMAL(5,2) = NULL
)
AS
BEGIN
    DECLARE @CurrentUnitPrice DECIMAL(10,2)
    DECLARE @CurrentQuantity INT
    DECLARE @CurrentDiscount DECIMAL(5,2)

    SELECT @CurrentUnitPrice = UnitPrice, @CurrentQuantity = Quantity,
    @CurrentDiscount = Discount
    FROM [Order Details]
    WHERE OrderID = @OrderID AND ProductID = @ProductID

    IF @UnitPrice IS NULL SET @UnitPrice = @CurrentUnitPrice
    IF @Quantity IS NULL SET @Quantity = @CurrentQuantity
```

```
IF @Discount IS NULL SET @Discount = @CurrentDiscount
```

```
UPDATE [Order Details]
```

```
SET UnitPrice = @UnitPrice, Quantity = @Quantity, Discount = @Discount
```

```
WHERE OrderID = @OrderID AND ProductID = @ProductID
```

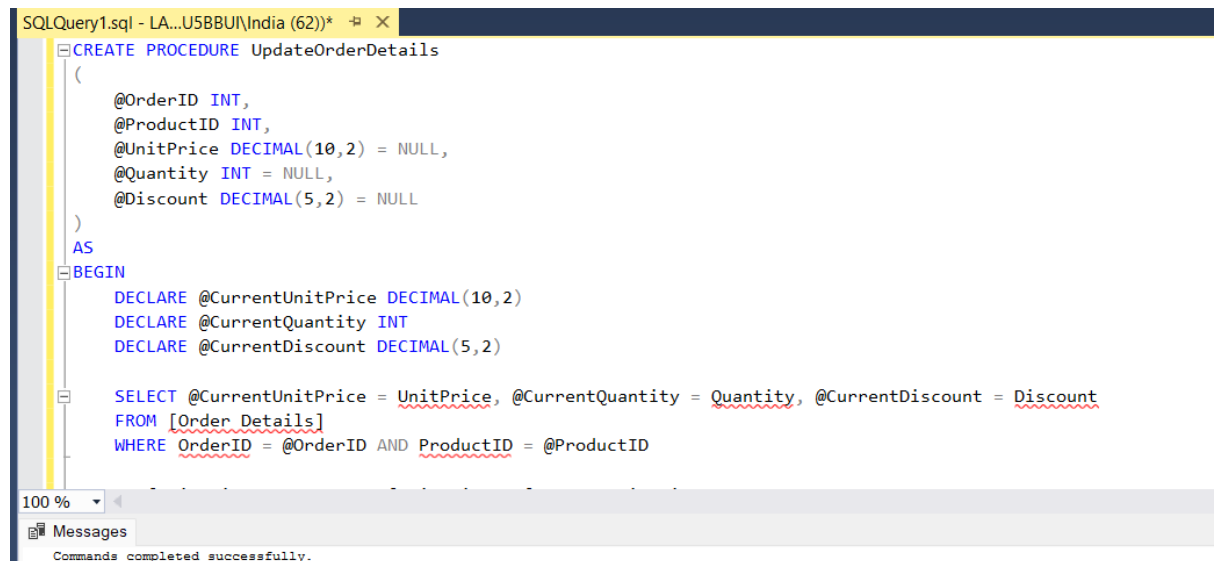
```
UPDATE Products
```

```
SET UnitsInStock = UnitsInStock + @CurrentQuantity - @Quantity
```

```
WHERE ProductID = @ProductID
```

```
END
```

Result:



The screenshot shows a SQL Server Enterprise Manager window titled 'SQLQuery1.sql - LA...USBBUI\India (62)\*'. The main pane displays the following T-SQL code:

```
CREATE PROCEDURE UpdateOrderDetails
(
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(10,2) = NULL,
    @Quantity INT = NULL,
    @Discount DECIMAL(5,2) = NULL
)
AS
BEGIN
    DECLARE @CurrentUnitPrice DECIMAL(10,2)
    DECLARE @CurrentQuantity INT
    DECLARE @CurrentDiscount DECIMAL(5,2)

    SELECT @CurrentUnitPrice = UnitPrice, @CurrentQuantity = Quantity, @CurrentDiscount = Discount
    FROM [Order Details]
    WHERE OrderID = @OrderID AND ProductID = @ProductID
```

The bottom pane shows a 'Messages' tab with the text: 'Commands completed successfully.'

### 3. GetOrderDetails Procedure

Query:

```
CREATE PROCEDURE GetOrderDetails
```

```
(
```

```
    @OrderID INT
```

```
)
```

```
AS
```

```
BEGIN
```

```
    SELECT * FROM [Order Details] WHERE OrderID = @OrderID
```

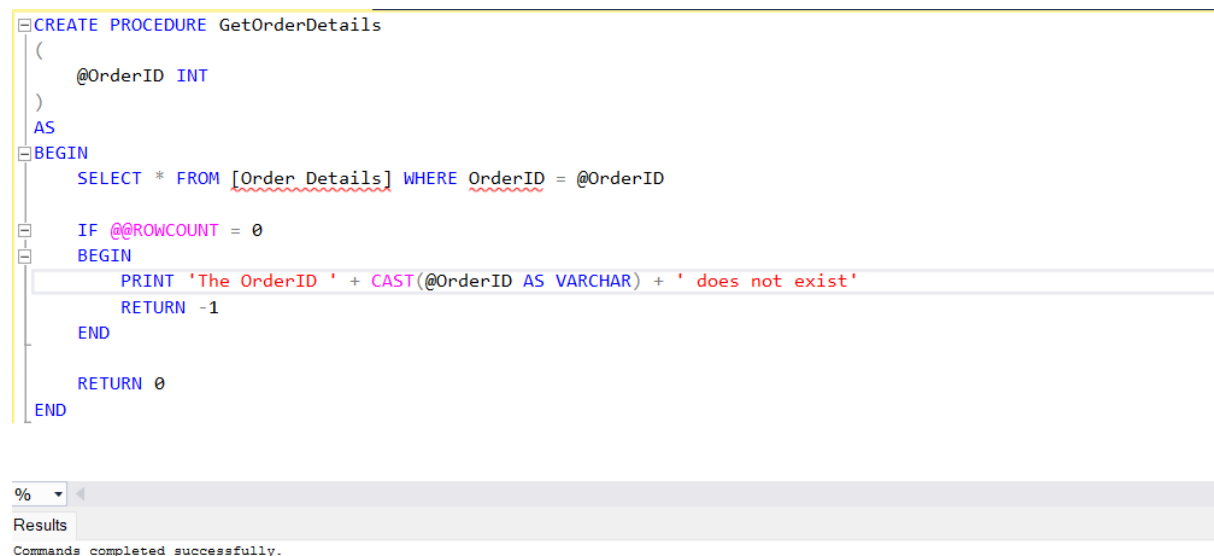
```

IF @@ROWCOUNT = 0
BEGIN
    PRINT 'The OrderID ' + CAST(@OrderID AS VARCHAR) + ' does not exist'
    RETURN -1
END

RETURN 0
END

```

## Result:



```

CREATE PROCEDURE GetOrderDetails
(
    @OrderID INT
)
AS
BEGIN
    SELECT * FROM [Order_Details] WHERE OrderID = @OrderID

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'The OrderID ' + CAST(@OrderID AS VARCHAR) + ' does not exist'
        RETURN -1
    END

    RETURN 0
END

```

Results

Commands completed successfully.

## 4. DeleteOrderDetails Procedure:

### Query:

```

CREATE PROCEDURE DeleteOrderDetails
(
    @OrderID INT,
    @ProductID INT
)
AS

```

```
BEGIN

    IF NOT EXISTS (SELECT 1 FROM [Order Details] WHERE OrderID = @OrderID AND ProductID =
@ProductID)

        BEGIN

            PRINT 'Order details not found'

            RETURN -1

        END

    DECLARE @Quantity INT

    SELECT @Quantity = Quantity FROM [Order Details]

    WHERE OrderID = @OrderID AND ProductID = @ProductID

    DELETE FROM [Order Details] WHERE OrderID = @OrderID AND ProductID = @ProductID

    UPDATE Products SET UnitsInStock = UnitsInStock + @Quantity WHERE ProductID =
@ProductID

    PRINT 'Order details deleted successfully'

    RETURN 0

END
```

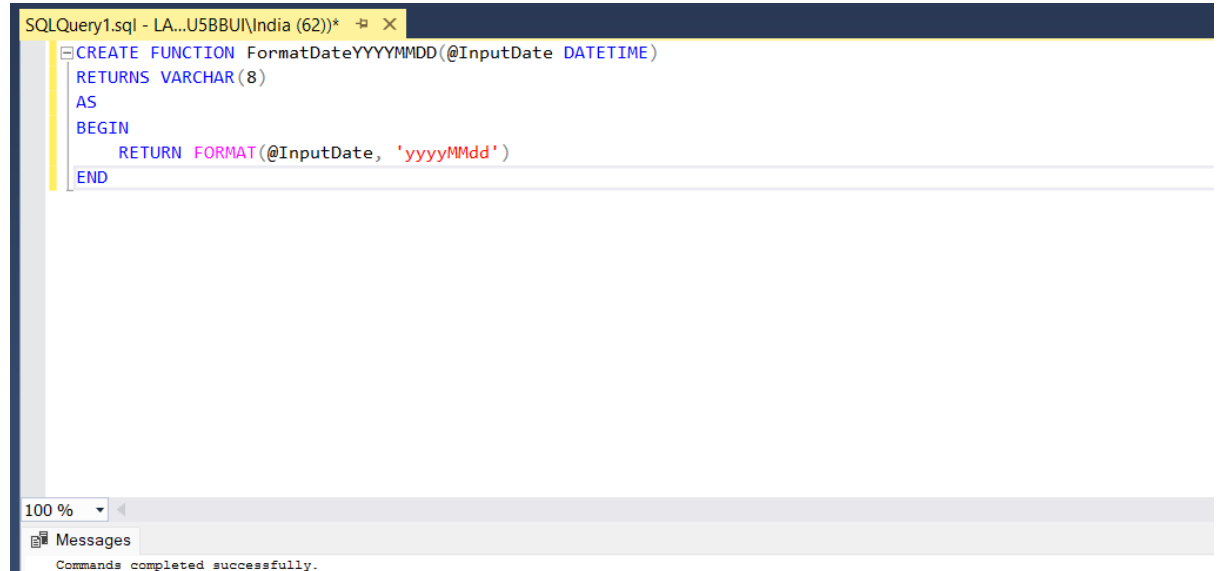
## **Functions:**

### **1. Date Function:**

#### **Query:**

```
CREATE FUNCTION FormatDateYYYYMMDD(@InputDate DATETIME)
RETURNS VARCHAR(8)
AS
BEGIN
    RETURN FORMAT(@InputDate, 'yyyyMMdd')
END
```

## Output:



The screenshot shows a SQL Server Enterprise Manager window titled 'SQLQuery1.sql - LA...U5BBUI\India (62))\*'. The main pane displays the following T-SQL code:

```
CREATE FUNCTION FormatDateYYYYMMDD(@InputDate DATETIME)
RETURNS VARCHAR(8)
AS
BEGIN
    RETURN FORMAT(@InputDate, 'yyyyMMdd')
END
```

The bottom pane shows a 'Messages' tab with the text: 'Commands completed successfully.'

## Views:

### 1. Customer Orders View

#### Query:

```
CREATE VIEW vwCustomerOrders
AS
SELECT
    c.CustomerID,
    soh.SalesOrderID,
    soh.OrderDate,
    sod.ProductID,
    p.Name AS ProductName,
    sod.OrderQty,
    sod.UnitPrice,
    sod.OrderQty * sod.UnitPrice AS TotalPrice
FROM Sales.Customer c
JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN Production.Product p ON sod.ProductID = p.ProductID
```

Output:

```
SQLQuery1.sql - LA...U58BUI\India (62))* X
CREATE VIEW vwCustomerOrders
AS
SELECT
    c.CustomerID,
    soh.SalesOrderID,
    soh.OrderDate,
    sod.ProductID,
    p.Name AS ProductName,
    sod.OrderQty,
    sod.UnitPrice,
    sod.OrderQty * sod.UnitPrice AS TotalPrice
FROM Sales.Customer c
JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN Production.Product p ON sod.ProductID = p.ProductID

100 %
Messages
Commands completed successfully.
```

## 2. Yesterday Orders View:

### Query:

```
CREATE VIEW vwYesterdayOrders
```

```
AS
```

```
SELECT
```

```
    c.CustomerID,
```

```
    soh.SalesOrderID,
```

```
    soh.OrderDate,
```

```
    sod.ProductID,
```

```
    p.Name AS ProductName,
```

```
    sod.OrderQty,
```

```
    sod.UnitPrice,
```

```
    sod.OrderQty * sod.UnitPrice AS TotalPrice
```

```
FROM Sales.Customer c
```

```
JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
```

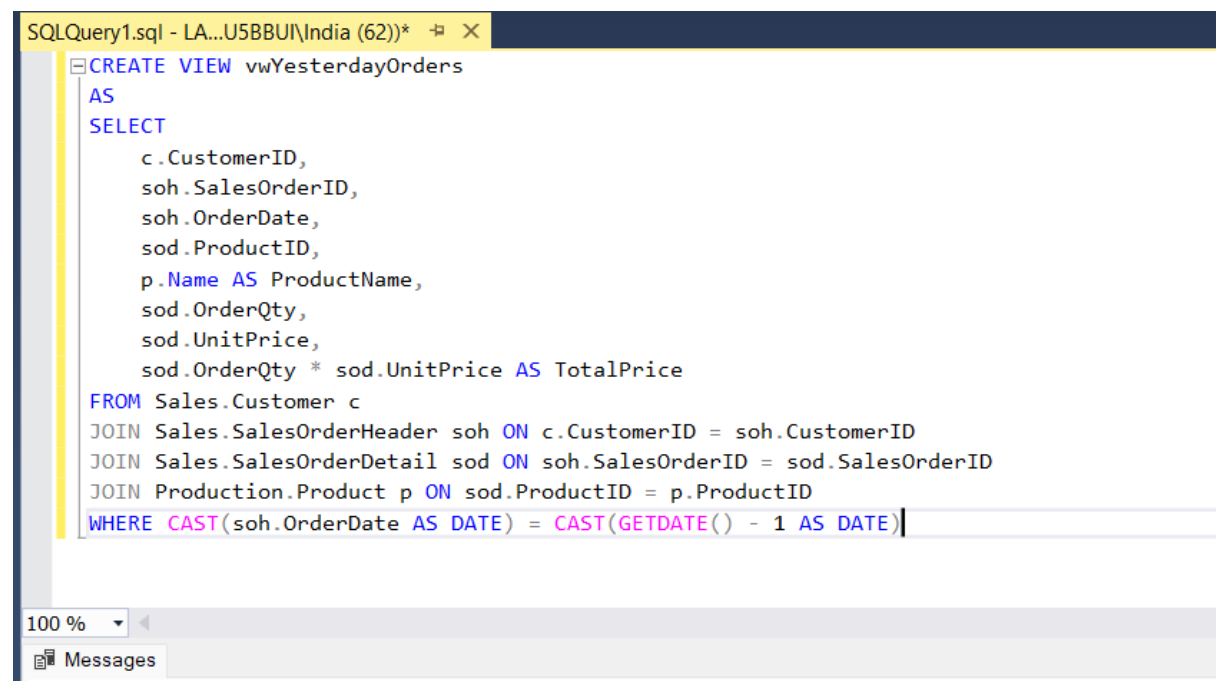
```
JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
```

```
JOIN Production.Product p ON sod.ProductID = p.ProductID
```

```
WHERE CAST(soh.OrderDate AS DATE) = CAST(GETDATE() - 1 AS DATE)
```



Output:



The screenshot shows a SQL Server Enterprise Manager window with a query editor. The title bar reads 'SQLQuery1.sql - LA...U5BBUI\India (62))\*'. The query text is as follows:

```
CREATE VIEW vwYesterdayOrders
AS
SELECT
    c.CustomerID,
    soh.SalesOrderID,
    soh.OrderDate,
    sod.ProductID,
    p.Name AS ProductName,
    sod.OrderQty,
    sod.UnitPrice,
    sod.OrderQty * sod.UnitPrice AS TotalPrice
FROM Sales.Customer c
JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN Production.Product p ON sod.ProductID = p.ProductID
WHERE CAST(soh.OrderDate AS DATE) = CAST(GETDATE() - 1 AS DATE)
```

At the bottom of the window, there is a 'Messages' tab and a zoom level set to '100 %'.

### 3. MyProducts View:

Query:

```
CREATE VIEW MyProducts
```

```
AS
```

```
SELECT
```

```
    p.ProductID,
```

```
    p.Name AS ProductName,
```

```
    p.ProductNumber,
```

```
    p.ListPrice,
```

```
    pc.Name AS CategoryName
```

```
FROM Production.Product p
```

```
JOIN Production.ProductSubcategory ps ON p.ProductSubcategoryID =
ps.ProductSubcategoryID
```

```
JOIN Production.ProductCategory pc ON ps.ProductCategoryID = pc.ProductCategoryID
```

```
WHERE p.DiscontinuedDate IS NULL
```

Output:

A screenshot of the SQL Server Enterprise Manager interface. The top pane shows a SQL query window titled 'SQLQuery1.sql - LA...U58BUI\India (62))'. The query is a CREATE VIEW statement for 'MyProducts'. The bottom pane shows a 'Messages' window with the text 'Commands completed successfully.'

```
CREATE VIEW MyProducts
AS
SELECT
    p.ProductID,
    p.Name AS ProductName,
    p.ProductNumber,
    p.ListPrice,
    pc.Name AS CategoryName
FROM Production.Product p
JOIN Production.ProductSubcategory ps ON p.ProductSubcategoryID = ps.ProductSubcategoryID
JOIN Production.ProductCategory pc ON ps.ProductCategoryID = pc.ProductCategoryID
WHERE p.DiscontinuedDate IS NULL
```

100 %

Messages

Commands completed successfully.

## Triggers

### 1. Delete Trigger:

**Query:** CREATE TRIGGER tr\_DeleteOrder

ON Sales.SalesOrderHeader

INSTEAD OF DELETE

AS

BEGIN

DELETE FROM Sales.SalesOrderDetail

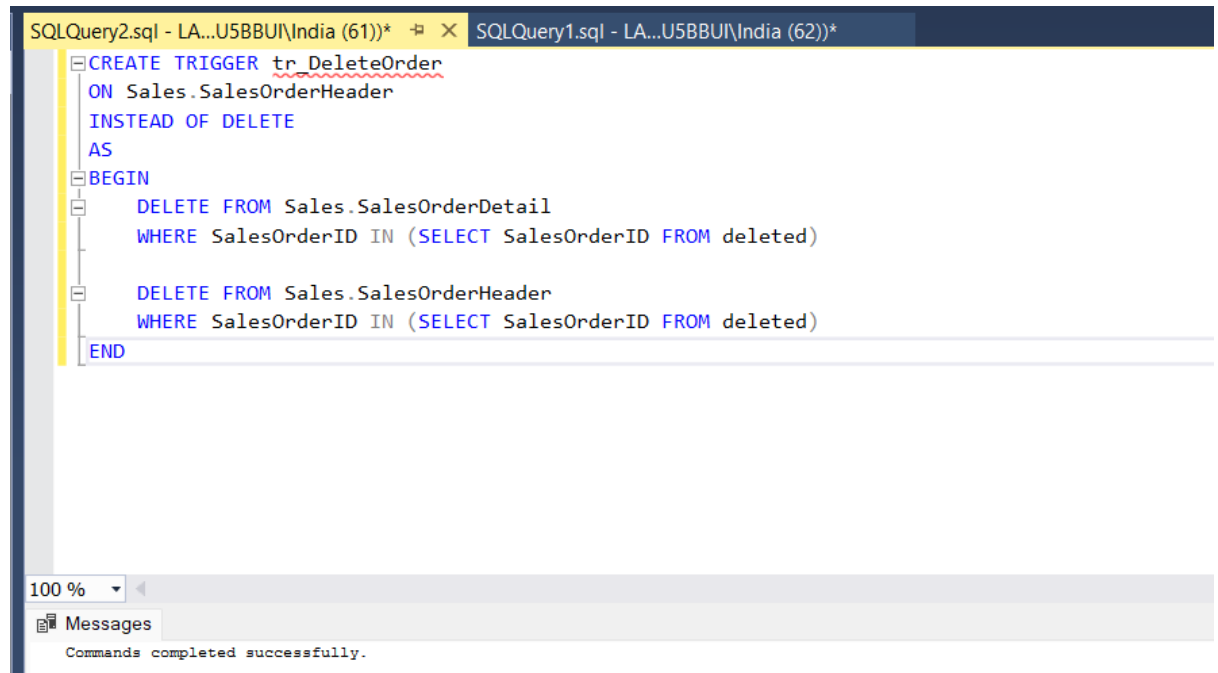
WHERE SalesOrderID IN (SELECT SalesOrderID FROM deleted)

DELETE FROM Sales.SalesOrderHeader

WHERE SalesOrderID IN (SELECT SalesOrderID FROM deleted)

END

## Output:



The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery2.sql - LA...U5BBUI\India (61))\*' and 'SQLQuery1.sql - LA...U5BBUI\India (62))\*'. The active tab displays the following T-SQL code:

```
CREATE TRIGGER tr_DeleteOrder
ON Sales.SalesOrderHeader
INSTEAD OF DELETE
AS
BEGIN
    DELETE FROM Sales.SalesOrderDetail
    WHERE SalesOrderID IN (SELECT SalesOrderID FROM deleted)

    DELETE FROM Sales.SalesOrderHeader
    WHERE SalesOrderID IN (SELECT SalesOrderID FROM deleted)
END
```

Below the code editor, the 'Messages' pane shows the status: 'Commands completed successfully.'

## 2. Stock Check Trigger

### Query:

```
CREATE TRIGGER tr_CheckStock
ON Sales.SalesOrderDetail
FOR INSERT
AS
BEGIN
    DECLARE @ProductID INT, @Quantity INT, @StockQuantity INT

    SELECT @ProductID = ProductID, @Quantity = OrderQty FROM inserted

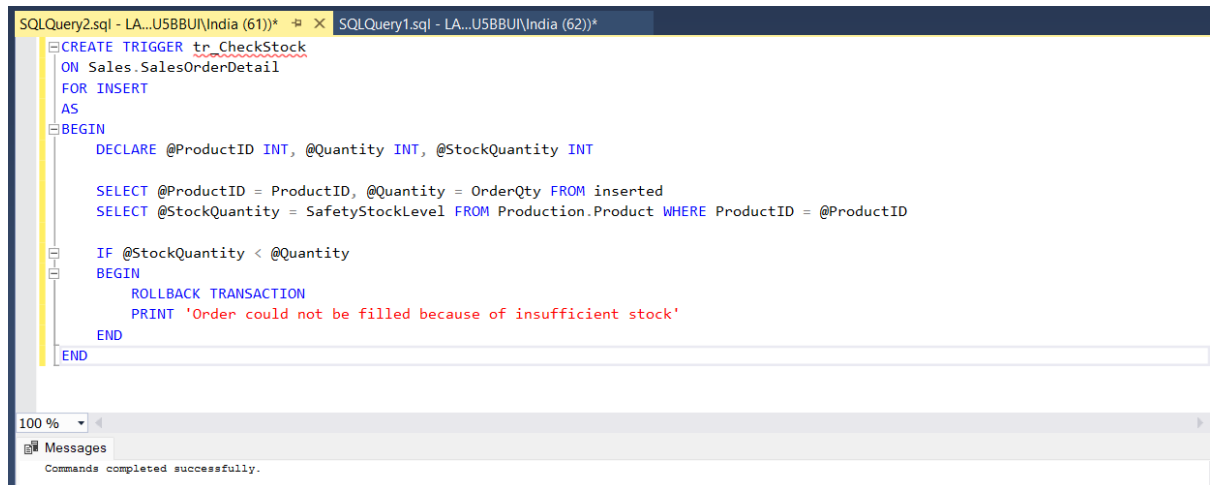
    SELECT @StockQuantity = SafetyStockLevel FROM Production.Product WHERE ProductID =
@ProductID

    IF @StockQuantity < @Quantity
    BEGIN
        ROLLBACK TRANSACTION

        PRINT 'Order could not be filled because of insufficient stock'
    END
END
```

END

Output:



```
SQLQuery2.sql - LA...U5BBUI\India (61))* x SQLQuery1.sql - LA...U5BBUI\India (62))*  
CREATE TRIGGER tr_CheckStock  
ON Sales.SalesOrderDetail  
FOR INSERT  
AS  
BEGIN  
    DECLARE @ProductID INT, @Quantity INT, @StockQuantity INT  
  
    SELECT @ProductID = ProductID, @Quantity = OrderQty FROM inserted  
    SELECT @StockQuantity = SafetyStockLevel FROM Production.Product WHERE ProductID = @ProductID  
  
    IF @StockQuantity < @Quantity  
    BEGIN  
        ROLLBACK TRANSACTION  
        PRINT 'Order could not be filled because of insufficient stock'  
    END  
END
```

100 %

Messages  
Commands completed successfully.