1.

CREATE PROCEDURE InsertOrderDetails

@OrderID INT,

@ProductID INT,

@UnitPrice DECIMAL = NULL,

@Quantity INT,

@Discount DECIMAL = 0

AS

BEGIN

SET NOCOUNT ON;

DECLARE @OriginalUnitsInStock INT, @ReorderLevel INT;

-- Check if UnitPrice is NULL, then fetch from Products table

IF @UnitPrice IS NULL

SELECT @UnitPrice = UnitPrice

FROM Production.Product

WHERE ProductID = @ProductID;

-- Adjust UnitsInStock and check for availability

SELECT @OriginalUnitsInStock = UnitsInStock, @ReorderLevel = ReorderLevel

FROM Production.Product

WHERE ProductID = @ProductID;

IF @OriginalUnitsInStock < @Quantity

BEGIN

PRINT 'Insufficient stock. Order cannot be placed.';

RETURN;

END

-- Insert into Order Details table

INSERT INTO Sales.OrderDetail (OrderID, ProductID, UnitPrice, Quantity, Discount)

VALUES (@OrderID, @ProductID, @UnitPrice, @Quantity, @Discount);

-- Check @@ROWCOUNT

IF @@ROWCOUNT = 0

BEGIN

PRINT 'Failed to place the order. Please try again.';

RETURN;

END

-- Adjust UnitsInStock

UPDATE Production.Product

SET UnitsInStock = @OriginalUnitsInStock - @Quantity

WHERE ProductID = @ProductID;

-- Check if UnitsInStock drops below ReorderLevel

IF (@OriginalUnitsInStock - @Quantity) < @ReorderLevel

BEGIN

PRINT 'Quantity in stock has dropped below Reorder Level.';

END

END

2.

CREATE PROCEDURE UpdateOrderDetails

@OrderID INT,

@ProductID INT,

@UnitPrice DECIMAL = NULL,

@Quantity INT = NULL,

@Discount DECIMAL = NULL

AS

BEGIN

SET NOCOUNT ON;

DECLARE @CurrentQuantity INT, @CurrentUnitPrice DECIMAL, @OriginalUnitsInStock INT;

-- Retrieve current values from OrderDetail

SELECT @CurrentQuantity = Quantity, @CurrentUnitPrice = UnitPrice

FROM Sales.OrderDetail

WHERE OrderID = @OrderID AND ProductID = @ProductID;

-- Check for NULL parameters and retain original values

SET @Quantity = ISNULL(@Quantity, @CurrentQuantity);

SET @UnitPrice = ISNULL(@UnitPrice, @CurrentUnitPrice);

SET @Discount = ISNULL(@Discount, 0);

-- Update OrderDetail

UPDATE Sales.OrderDetail

SET UnitPrice = @UnitPrice,

Quantity = @Quantity,

Discount = @Discount

WHERE OrderID = @OrderID AND ProductID = @ProductID;

-- Adjust UnitsInStock in Products table

SELECT @OriginalUnitsInStock = UnitsInStock

FROM Production.Product

WHERE ProductID = @ProductID;

UPDATE Production.Product

SET UnitsInStock = @OriginalUnitsInStock + @CurrentQuantity - @Quantity

WHERE ProductID = @ProductID;

END

3.

CREATE PROCEDURE GetOrderDetails

@OrderID INT

AS

BEGIN

SET NOCOUNT ON;

IF NOT EXISTS (SELECT \* FROM Sales.OrderDetail WHERE OrderID = @OrderID)

BEGIN

PRINT 'The OrderID ' + CAST(@OrderID AS VARCHAR) + ' does not exist.';

RETURN 1;

END

-- Return all records for the OrderID

SELECT \*

FROM Sales.OrderDetail

WHERE OrderID = @OrderID;

END

4.

CREATE PROCEDURE DeleteOrderDetails

@OrderID INT,

@ProductID INT

AS

BEGIN

SET NOCOUNT ON;

-- Validate parameters

IF NOT EXISTS (SELECT \* FROM Sales.OrderDetail WHERE OrderID = @OrderID AND ProductID = @ProductID)

BEGIN

PRINT 'Invalid parameters. OrderID or ProductID does not exist.';

RETURN -1;

END

-- Delete from OrderDetail table

DELETE FROM Sales.OrderDetail WHERE OrderID = @OrderID AND ProductID = @ProductID;

-- Adjust UnitsInStock in Products table if needed

-- (Assuming there's no requirement to adjust stock here, since it's handled during Insert/Update)

END

5.

CREATE FUNCTION FormatDateMMDDYYYY (@InputDate DATETIME)

RETURNS VARCHAR(10)

AS

BEGIN

RETURN CONVERT(VARCHAR(10), @InputDate, 101);

END

6.

CREATE FUNCTION FormatDateYYYYMMDD (@InputDate DATETIME)

RETURNS VARCHAR(8)

AS

BEGIN

RETURN CONVERT(VARCHAR(8), @InputDate, 112);

END

7.

CREATE VIEW vwCustomerorders

AS

SELECT

c.CompanyName,

od.OrderID,

o.OrderDate,

od.ProductID,

p.Name AS ProductName,

od.Quantity,

od.UnitPrice,

od.Quantity \* od.UnitPrice AS TotalPrice

FROM

Sales.Customer c

JOIN Sales.OrderHeader o ON c.CustomerID = o.CustomerID

JOIN Sales.OrderDetail od ON o.OrderID = od.OrderID

JOIN Production.Product p ON od.ProductID = p.ProductID;

8.

CREATE VIEW vwCustomerordersYesterday

AS

SELECT

c.CompanyName,

od.OrderID,

o.OrderDate,

od.ProductID,

p.Name AS ProductName,

od.Quantity,

od.UnitPrice,

od.Quantity \* od.UnitPrice AS TotalPrice

FROM

Sales.Customer c

JOIN Sales.OrderHeader o ON c.CustomerID = o.CustomerID

JOIN Sales.OrderDetail od ON o.OrderID = od.OrderID

JOIN Production.Product p ON od.ProductID = p.ProductID

WHERE

CONVERT(DATE, o.OrderDate) = DATEADD(DAY, -1, CONVERT(DATE, GETDATE()));

9.

CREATE VIEW MyProducts

AS

SELECT

p.ProductID,

p.Name AS ProductName,

p.QuantityPerUnit,

p.UnitPrice,

s.CompanyName,

c.Name AS CategoryName

FROM

Production.Product p

JOIN Production.Supplier s ON p.SupplierID = s.SupplierID

JOIN Production.Category c ON p.CategoryID = c.CategoryID

WHERE

p.Discontinued = 0;

10.

CREATE TRIGGER trgInsteadOfDeleteOrders

ON Sales.OrderHeader

INSTEAD OF DELETE

AS

BEGIN

SET NOCOUNT ON;

-- Delete from OrderDetail first

DELETE FROM Sales.OrderDetail

WHERE OrderID IN (SELECT OrderID FROM DELETED);

-- Delete from Orders table

DELETE FROM Sales.OrderHeader

WHERE OrderID IN (SELECT OrderID FROM DELETED);

END

11.

CREATE TRIGGER trgCheckStockOnOrderDetails

ON Sales.OrderDetail

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @ProductID INT, @Quantity INT, @UnitsInStock INT;

SELECT @ProductID = ProductID, @Quantity = Quantity

FROM inserted;

SELECT @UnitsInStock = UnitsInStock

FROM Production.Product

WHERE ProductID = @ProductID;

IF @UnitsInStock < @Quantity

BEGIN

RAISERROR ('Insufficient stock to fill the order.', 16, 1);

ROLLBACK TRANSACTION;

END

ELSE

BEGIN

UPDATE Production.Product

SET UnitsInStock = @UnitsInStock - @Quantity

WHERE ProductID = @ProductID;

END

END