LEVEL B(Answers)

-- Procedure: InsertOrderDetails

-- Takes OrderID, ProductID, UnitPrice, Quantity, Discount as input parameters and inserts that order information in the Order Details table.

-- After each order inserted, checks the @@rowcount value to make sure that order was inserted properly.

-- If for any reason the order was not inserted, prints the message: Failed to place the order. Please try again.

-- UnitPrice and Discount parameters are optional.

-- If no UnitPrice is given, then use the UnitPrice value from the product table.

-- If no Discount is given, then use a discount of 0.

-- Adjusts the quantity in stock (UnitsInStock) for the product by subtracting the quantity sold from inventory.

-- If there is not enough of a product in stock, then aborts the stored procedure without making any changes to the database.

-- Prints a message if the quantity in stock of a product drops below its Reorder Level as a result of the update.

CREATE PROCEDURE InsertOrderDetails

    @OrderID INT,

    @ProductID INT,

    @UnitPrice DECIMAL(18, 2) = NULL,

    @Quantity INT,

    @Discount DECIMAL(4, 2) = 0

AS

BEGIN

    DECLARE @CurrentStock INT

    DECLARE @ReorderLevel INT

    DECLARE @DefaultUnitPrice DECIMAL(18, 2)

    -- Get current stock and reorder level from the product table

    SELECT @CurrentStock = UnitsInStock, @ReorderLevel = ReorderLevel, @DefaultUnitPrice = ListPrice

    FROM Production.Product

    WHERE ProductID = @ProductID

    -- Use default unit price if not provided

    SET @UnitPrice = ISNULL(@UnitPrice, @DefaultUnitPrice)

    -- Check if there is enough stock

    IF @CurrentStock < @Quantity

    BEGIN

        PRINT 'Not enough stock available. Order cannot be placed.'

        RETURN

    END

    -- Insert order details

    INSERT INTO Sales.OrderDetail (SalesOrderID, ProductID, UnitPrice, OrderQty, UnitPriceDiscount)

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

    -- Check if the order was inserted

    IF @@ROWCOUNT = 0

    BEGIN

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

        RETURN

    END

    -- Update the stock

    UPDATE Production.Product

    SET UnitsInStock = UnitsInStock - @Quantity

    WHERE ProductID = @ProductID

    -- Check if stock drops below reorder level

    IF @CurrentStock - @Quantity < @ReorderLevel

    BEGIN

        PRINT 'The quantity in stock of the product has dropped below its reorder level.'

    END

END

GO

-- Procedure: UpdateOrderDetails

-- 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.

-- If any of the values are being passed in as NULL, then retains the original value instead of overwriting it with NULL.

-- Adjusts the UnitsInStock value in the products table accordingly.

CREATE PROCEDURE UpdateOrderDetails

    @OrderID INT,

    @ProductID INT,

    @UnitPrice DECIMAL(18, 2) = NULL,

    @Quantity INT = NULL,

    @Discount DECIMAL(4, 2) = NULL

AS

BEGIN

    DECLARE @OriginalQuantity INT

    DECLARE @CurrentStock INT

    -- Get original quantity from order details

    SELECT @OriginalQuantity = OrderQty

    FROM Sales.OrderDetail

    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID

    -- Update order details

    UPDATE Sales.OrderDetail

    SET UnitPrice = ISNULL(@UnitPrice, UnitPrice),

        OrderQty = ISNULL(@Quantity, OrderQty),

        UnitPriceDiscount = ISNULL(@Discount, UnitPriceDiscount)

    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID

    -- Check if the order was updated

    IF @@ROWCOUNT = 0

    BEGIN

        PRINT 'Failed to update the order. Please try again.'

        RETURN

    END

    -- Adjust stock based on new quantity

    SET @CurrentStock = (SELECT UnitsInStock FROM Production.Product WHERE ProductID = @ProductID)

    IF @Quantity IS NOT NULL

    BEGIN

        UPDATE Production.Product

        SET UnitsInStock = UnitsInStock + @OriginalQuantity - @Quantity

        WHERE ProductID = @ProductID

    END

END

GO

-- Procedure: GetOrderDetails

-- Takes OrderID as input parameter and returns all the records for that OrderID.

-- If no records are found in Order Details table, prints the line: "The OrderID XXXX does not exist", where XXXX is the OrderID entered by the user.

-- Returns the value 1 if no records are found.

CREATE PROCEDURE GetOrderDetails

    @OrderID INT

AS

BEGIN

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

    BEGIN

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

        RETURN 1

    END

    SELECT \* FROM Sales.OrderDetail

    WHERE SalesOrderID = @OrderID

END

GO

-- Procedure: DeleteOrderDetails

-- Takes OrderID and ProductID and deletes that from Order Details table.

-- Validates parameters. Returns an error code (-1) and prints a message if the parameters are invalid.

-- Parameters are valid if the given OrderID appears in the table and if the given ProductID appears in that order.

CREATE PROCEDURE DeleteOrderDetails

    @OrderID INT,

    @ProductID INT

AS

BEGIN

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

    BEGIN

        PRINT 'Invalid OrderID or ProductID.'

        RETURN -1

    END

    DELETE FROM Sales.OrderDetail

    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID

    IF @@ROWCOUNT = 0

    BEGIN

        PRINT 'Failed to delete the order detail. Please try again.'

        RETURN -1

    END

END

GO

-- Function: FormatDate\_MMDDYYYY

-- Takes an input parameter type datetime and returns the date in the format MM/DD/YYYY.

-- Example: if '2006-11-21 23:34:05.920' is passed in, the output should be '11/21/2006'.

CREATE FUNCTION FormatDate\_MMDDYYYY (@InputDate DATETIME)

RETURNS NVARCHAR(10)

AS

BEGIN

    RETURN CONVERT(NVARCHAR(10), @InputDate, 101)

END

GO

-- Function: FormatDate\_YYYYMMDD

-- Takes an input parameter type datetime and returns the date in the format YYYYMMDD.

CREATE FUNCTION FormatDate\_YYYYMMDD (@InputDate DATETIME)

RETURNS NVARCHAR(8)

AS

BEGIN

    RETURN CONVERT(NVARCHAR(8), @InputDate, 112)

END

GO

-- Views --

-- Create a view vwCustomerOrders which returns Company Name, OrderID, OrderDate, ProductID, ProductName, Quantity, UnitPrice

CREATE VIEW vwCustomerOrders AS

SELECT

    c.CompanyName,

    o.OrderID,

    o.OrderDate,

    od.ProductID,

    p.ProductName,

    od.Quantity,

    od.UnitPrice

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;

-- Create a copy of vwCustomerOrders to return orders placed yesterday

CREATE VIEW vwCustomerOrdersYesterday AS

SELECT \*

FROM vwCustomerOrders

WHERE OrderDate = DATEADD(DAY, -1, GETDATE());

-- Create MyProducts view with ProductID, ProductName, QuantityPerUnit, UnitPrice, CompanyName, CategoryName

-- Only include products that are not discontinued

CREATE VIEW MyProducts AS

SELECT

    p.ProductID,

    p.ProductName,

    p.QuantityPerUnit,

    p.UnitPrice,

    s.CompanyName,

    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;

-- Triggers --

-- Instead of Delete trigger on Orders table to handle cascading delete from OrderDetails first

CREATE TRIGGER tr\_DeleteOrderInsteadOf

ON Orders

INSTEAD OF DELETE

AS

BEGIN

    DELETE FROM OrderDetails

    WHERE OrderID IN (SELECT OrderID FROM deleted);

    DELETE FROM Orders

    WHERE OrderID IN (SELECT OrderID FROM deleted);

END;

-- Trigger to check stock availability before inserting into OrderDetails

CREATE TRIGGER tr\_CheckStock

ON OrderDetails

INSTEAD OF INSERT

AS

BEGIN

    DECLARE @ProductID INT, @Quantity INT;

    SELECT @ProductID = ProductID, @Quantity = Quantity

    FROM inserted;

    IF EXISTS (

        SELECT 1

        FROM Products

        WHERE ProductID = @ProductID AND UnitsInStock >= @Quantity

    )

    BEGIN

        -- Sufficient stock, proceed with insertion

        INSERT INTO OrderDetails (OrderID, ProductID, UnitPrice, Quantity)

        SELECT OrderID, ProductID, UnitPrice, Quantity

        FROM inserted;

        -- Update UnitsInStock

        UPDATE Products

        SET UnitsInStock = UnitsInStock - @Quantity

        WHERE ProductID = @ProductID;

    END

    ELSE

    BEGIN

        -- Insufficient stock, notify user (do not insert)

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

    END

END;