Raj Singhaniya
0112cs221101@gmail.com

Assignment - Level B

1. Create a procedure InsertOrderDetails that takes OrderID, ProductID, UnitPrice, Quantity, Discount as input parameters and inserts that order information in the Order Details table. After each order is 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 your procedure should have these functionalities.Make the UnitPrice and Discount parameters 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. 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.

Ans:
```
CREATE PROCEDURE InsertOrderDetails
   @OrderID INT,
   @ProductID INT,
   @Quantity SMALLINT,
   @UnitPrice MONEY = NULL,
   @Discount MONEY = 0.0
AS
BEGIN
   SET NOCOUNT ON;

     IF @UnitPrice IS NULL
         SET @UnitPrice = (SELECT UnitPrice FROM Products WHERE
ProductID = @ProductID);
   IF @Discount IS NULL
      SET @Discount = 0;

   DECLARE @CurrentStock INT;
```

```
    SELECT @CurrentStock = UnitsInStock FROM Products WHERE
ProductID = @ProductID;

   IF @CurrentStock < @Quantity
   BEGIN
      PRINT 'Insufficient stock. Order aborted.';
      RETURN;
   END

   BEGIN TRANSACTION;

   BEGIN TRY

          INSERT INTO [Order Details] (OrderID, ProductID, UnitPrice,
Quantity, Discount)
            VALUES (@OrderID, @ProductID, @UnitPrice, @Quantity,
@Discount);

          IF @@ROWCOUNT = 0
      BEGIN
         PRINT 'Failed to place the order. Please try again.';
         ROLLBACK TRANSACTION;
         RETURN;
      END


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

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

        SELECT @CurrentStock = UnitsInStock FROM Products WHERE
ProductID = @ProductID;
```

```
        IF @CurrentStock < @ReorderLevel
        BEGIN
            PRINT 'Warning: Product stock is below reorder level.';
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Failed to place the order due to an error. Please try again.';
    END CATCH
END;
```

2. 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 s/he 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 the products table accordingly.

Ans:
```
CREATE PROCEDURE UpdateOrderDetails
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(10,2) = NULL,
    @Quantity INT = NULL,
    @Discount DECIMAL(5,2) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    IF @UnitPrice IS NOT NULL
    BEGIN
        UPDATE OrderDetails
        SET UnitPrice = @UnitPrice
        WHERE OrderID = @OrderID AND ProductID = @ProductID;
```

```
    END

    IF @Quantity IS NOT NULL
    BEGIN
        UPDATE OrderDetails
        SET Quantity = @Quantity
        WHERE OrderID = @OrderID AND ProductID = @ProductID;

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

    IF @Discount IS NOT NULL
    BEGIN
        UPDATE OrderDetails
   SET Discount = @Discount
    WHERE OrderID = @OrderID AND ProductID = @ProductID;
    END
END;
```

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

```
Ans: CREATE PROCEDURE GetOrderDetails
    @OrderID INT
AS
BEGIN
    SET NOCOUNT ON;

   IF EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID = @OrderID)
    BEGIN
        SELECT * FROM OrderDetails WHERE OrderID = @OrderID;
    END
    ELSE
    BEGIN
    PRINT 'The OrderID ' + CAST(@OrderID AS VARCHAR(10)) + ' does not
      exist.';
    RETURN 1;
```

```
END
END;
```

4. **Create a procedure DeleteOrderDetails that takes OrderID and ProductID and deletes that from the 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 given product ID appears in that order.**

```
Ans: CREATE PROCEDURE DeleteOrderDetails
   @OrderID INT,
   @ProductID INT
AS
BEGIN
   SET NOCOUNT ON;


      IF NOT EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID =
      @OrderID AND ProductID = @ProductID)
   BEGIN
         PRINT 'Error: The specified ProductID does not exist for the given
      OrderID.';
       RETURN -1;
   END

   DELETE FROM OrderDetails
   WHERE OrderID = @OrderID AND ProductID = @ProductID;

   PRINT 'Record deleted successfully.';
END;
```

5. **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**

```
Ans: CREATE FUNCTION FormatDate (@InputDate DATETIME)
RETURNS VARCHAR(10)
AS
BEGIN
   RETURN CONVERT(VARCHAR(10), @InputDate, 101)
END;
```

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

Ans:
```
CREATE FUNCTION FormatDateYYYYMMDD (@InputDate DATETIME)
RETURNS VARCHAR(8)
AS
BEGIN
RETURN CONVERT(VARCHAR(8), @InputDate, 112)
END;
```

Views

7. Create a view vwCustomerOrders which returns CompanyName,OrderID,OrderDate,ProductID,ProductName,Quantity,UnitPrice,Quantity * od.UnitPrice

Ans:
```
CREATE VIEW vwCustomerOrders AS
SELECT
   c.CompanyName,
   o.OrderID,
   o.OrderDate,
   od.ProductID,
   p.ProductName,
   od.Quantity,
   od.UnitPrice,
   (od.Quantity * od.UnitPrice) AS TotalPrice
FROM
   Orders o
JOIN
   Customers c ON o.CustomerID = c.CustomerID
JOIN
   [Order Details] od ON o.OrderID = od.OrderID
JOIN
   Products p ON od.ProductID = p.ProductID;
```

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

Ans:
```
CREATE VIEW vwCustomerOrdersYesterday AS
SELECT
   c.CompanyName,
   o.OrderID,
   o.OrderDate,
```

```
    od.ProductID,
    p.ProductName,
    od.Quantity,
    od.UnitPrice,
    od.Quantity * od.UnitPrice AS ExtendedPrice
FROM
    Orders o
JOIN
    Customers c ON o.CustomerID = c.CustomerID
JOIN
    [Order Details] od ON o.OrderID = od.OrderID
JOIN
    Products p ON od.ProductID = p.ProductID
WHERE
    o.OrderDate = DATEADD(day, -1, CAST(GETDATE() AS DATE));
```

Triggers

9. If someone cancels an order in the 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 the Orders table so that if someone tries to delete an Order that trigger gets fired and that trigger should first delete everything in the order details table and then delete that order from the Orders table.

Ans:
```
CREATE TRIGGER trgInsteadOfDeleteOrder
ON Orders
INSTEAD OF DELETE
AS
BEGIN
    DELETE od
    FROM [Order Details] od
    INNER JOIN deleted d ON od.OrderID = d.OrderID;
    DELETE o
    FROM Orders o
    INNER JOIN deleted d ON o.OrderID = d.OrderID;
END;
```

10. 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 (i.e. do not insert it) and notify the user that the order could not be filled because of insufficient stock.

Ans:
```
CREATE TRIGGER trgCheckStockBeforeInsert
ON [Order Details]
INSTEAD OF INSERT
AS
BEGIN
   DECLARE @ProductID INT, @Quantity INT, @UnitsInStock INT;
   DECLARE order_cursor CURSOR FOR
      SELECT ProductID, Quantity
      FROM inserted;

   OPEN order_cursor;
   FETCH NEXT FROM order_cursor INTO @ProductID, @Quantity;

   WHILE @@FETCH_STATUS = 0
   BEGIN
      SELECT @UnitsInStock = UnitsInStock
      FROM Products
      WHERE ProductID = @ProductID;

      IF @UnitsInStock >= @Quantity
      BEGIN
      INSERT INTO [Order Details] (OrderID, ProductID, Quantity, UnitPrice,
     Discount)
         SELECT OrderID, ProductID, Quantity, UnitPrice, Discount
         FROM inserted
         WHERE ProductID = @ProductID;

         UPDATE Products
         SET UnitsInStock = UnitsInStock - @Quantity
         WHERE ProductID = @ProductID;
      END
      ELSE
      BEGIN
      RAISERROR('Insufficient stock for ProductID %d. Order cannot be
     placed.', 16, 1, @ProductID);
```

```
        END

        FETCH NEXT FROM order_cursor INTO @ProductID, @Quantity;
    END

    CLOSE order_cursor;
    DEALLOCATE order_cursor;
END;
```