Module 6: Database Objects

Lab: Using SQL Server

Exercise 1: Creating a Table and Constraints

Task 2: Create the Sales.LineItems Table

> **Note:** You can find the completed Transact-SQL statements for this exercise in the **Create Line Items Table.sql** file in the **D:\Labfiles\Lab06\Solution** folder.

1. On the taskbar, click **Microsoft SQL Server Management Studio**, in the **Connect to Server** dialog box, in the **Server type** field, select **Database Engine**, in the **Server name** field, type **MIA-SQL**, in the **Authentication** field, select **Windows Authentication**, and then click **Connect**.
2. Click **New Query**, and then in the new query window, type the following Transact-SQL statements, and then click **Execute**:

```
USE OrdersDatabase;
GO

CREATE TABLE Sales.LineItems
(OrderID INT NOT NULL
,ProductID INT NOT NULL
,UnitPrice MONEY NOT NULL
,Quantity SMALLINT NOT NULL
CONSTRAINT PK_LineItems PRIMARY KEY (OrderID,ProductID)
CONSTRAINT FK_LineItems_Orders FOREIGN KEY (OrderID)
REFERENCES Sales.Orders (OrderID));
GO
```

3. In Object Explorer, expand **Databases**, expand **OrdersDatabase**, expand **Tables**, and verify that the **Sales.LineItems** table appears in the list of tables.
4. On the taskbar, click **File Explorer**, browse to the D:\Labfiles\Lab06\Starter folder, right-click **Populate LineItems.cmd**, and then click **Run as administrator**.
5. In the **User Account Control** dialog box, click **Yes**, and then wait for setup to complete. This script adds rows to the newly created **Sales.LineItems** table.
6. In the query window, type the following Transact-SQL statement, select it, and then click **Execute**:

```
SELECT *
FROM Sales.LineItems
```

   This query should return 8 rows.

Task 3: Create and Test a DEFAULT Constraint

1. In SQL Server Management Studio, in the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
ALTER TABLE Sales.Orders
ADD CONSTRAINT DF_OrderDate DEFAULT GETDATE() FOR OrderDate;
GO
```

2. Under the existing Transact-SQL statements, type the following Transact-SQL statements, select them, and then click **Execute**:

```
INSERT INTO Sales.Orders
VALUES (110, 3, DEFAULT);
GO

SELECT * FROM Sales.Orders
WHERE OrderID = 110;
GO
```

3. Review the results, and note that the value in the **OrderDate** column was added by the DEFAULT constraint.

4. Close the **SQLQuery1.sql** window without saving changes.

5. Leave SQL Server Management Studio open for the next exercise.

Exercise 2: Creating a View

Task 1: Write a Transact-SQL Query to Retrieve the Required Data

> **Note:** The file **Create View.sql** in the **D:\Labfiles\Lab06\Solution** folder contains the Transact-SQL statements used in this exercise.

1. In SQL Server Management Studio, click **New Query**. In the query window, type the following Transact-SQL statements, and then click **Execute**:

```
USE OrdersDatabase;
GO

SELECT (FirstName + ' ' + LastName) AS [Customer Name]
,O.OrderID, OrderDate
,SUM(UnitPrice*Quantity) AS [Order Total]
FROM Person.Customers as C
JOIN Sales.Orders as O
ON C.CustomerID = O.CustomerID
JOIN Sales.LineItems as L
ON O.OrderID = L.OrderID
GROUP BY O.OrderID, Firstname, Lastname, OrderDate;
GO
```

2. Review the results, noting that the query returns one row for each of the orders in the **Sales.Orders** table, along with a single total cost for each order, which is calculated in the query by multiplying the **UnitPrice** column by the **Quantity** column in the **Sales.LineItems**

table. Note also that the **Customer Name** column is created by concatenating the **FirstName** and **LastName** columns in the **Person.Customers** table.

Task 2: Create the VW_CustomerOrders View

1. In SQL Server Management Studio, in the query window, edit the Transact-SQL statement that you typed in the previous task by typing the following Transact-SQL code directly above the SELECT line:

```
CREATE VIEW VW_CustomerOrders
AS
```

2. Click **Execute** to create the view.
3. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statement, select it, and then click **Execute**:

```
SELECT *
FROM VW_CustomerOrders;
GO
```

4. Review the results, noting that they are the same as the results that the SELECT query returned in the previous exercise.
5. Close the **SQLQuery2.sql** window without saving changes.
6. Leave SQL Server Management Studio open for the next exercise.

Exercise 3: Creating a Stored Procedure

Task 1: Create the USP_InsertOrders Stored Procedure

> **Note:** You can find the completed Transact-SQL statements for this exercise in the **Create Stored Procedure.sql** file in the **D:\Labfiles\Lab06\Solution** folder.

- In SQL Server Management Studio, click **New Query**, type the following Transact-SQL statements, and then click **Execute**:

```
USE OrdersDatabase;
GO

CREATE PROCEDURE USP_InsertOrders
@OrderID int, @CustomerID int
AS
INSERT INTO Sales.Orders (OrderID, CustomerID)
VALUES
(@OrderID, @CustomerID);
GO
```

Task 2: Test the Stored Procedure

1. In the query window, under the existing Transact-SQL statements, type the following Transact-SQL statements, select it, and then click **Execute**:

```
EXEC USP_InsertOrders 150,2;
GO

SELECT * FROM Sales.Orders
WHERE OrderID = 150
```

2. Review the results set, noting that it shows the row that the stored procedure added, and that the **OrderDate** value was added by the DEFAULT constraint.
3. Close SQL Server Management Studio without saving changes.

Lab Review

Question: The PRIMARY KEY column in the Sales.Orders table does not have the IDENTITY property set. If the IDENTITY property were set for this column, how would you have to modify the USP_InsertOrders stored procedure?

Answer: You would need to modify the procedure by removing the @OrderID input parameter and the reference to the OrderID column in the INSERT statement.

Question: You created a stored procedure to insert new rows into the Sales.Orders table. What else would you need to do to ensure that new orders are recorded in full, including details of the products in each order, prices, and so on?

Answer: You would need to update the Sales.LineItems table with the appropriate details for each order. You could extend the existing stored procedure to do this.