# Module 6: Database Objects

# Lab: Using SQL Server

## Scenario

You work for an organization called Adventure Works, which is a retailer of bicycles and associated products. You are implementing a database called **OrdersDatabase**, which is an OLTP database that tracks customers, the orders that they place, and the products that are included in the orders. You need to perform the following actions in the new database:

- Implement a table, that you have already designed, to record the line items in each order. You will create the table in the **Sales** schema, and also add a DEFAULT constraint to the **Sales.Order** table to insert the current date into the **OrderDate** column.
- Create and test a view to provide a standardized way of accessing customer data, including the details of the orders that they place.
- Create and test a stored procedure to insert new orders into the **Sales.Order** table.

## Objectives

After completing this lab, you will have:

- Created a table and a DEFAULT constraint.
- Created a view.
- Created a stored procedure.

## Lab Setup

Estimated Time: 60 minutes

Before you start this lab:

1. Ensure that the 10985C-MIA-DC and 10985C-MIA-SQL virtual machines are both running.
2. Log on to 10985C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa$$w0rd**.
3. Browse to the **D:\Labfiles\Lab06\Starter** folder, and then run **Setup.cmd** as Administrator.

## Exercise 1: Creating a Table and Constraints

### Scenario

The **OrdersDatabase** needs to store the details of each order, including the line items, quantity, and unit price. In this exercise, you will create a table called **Sales.LineItems** to store this information, and implement the required constraints to ensure referential integrity. You will also create a DEFAULT constraint on the **OrderDate** column in the **Sales.Orders** table to add the current date as a default value.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Create the Sales.LineItems Table

3. Create and Test a DEFAULT Constraint

Task 1: 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. Open **SQL Server Management Studio** and connect to the MIA-SQL Database Engine instance by using Windows Authentication.

2. In a new query window, type and execute a Transact-SQL statement to create a table called **LineItems** in the **Sales** schema in the **OrdersDatabase** database. Include the following columns and constraints. None of the columns should allow NULLs:

    - **OrderID** with the data type **int**.
    - **ProductID** with the data type **int**.
    - **UnitPrice** with the data type **money**.
    - **Quantity** with the data type **smallint**.
    - A primary key constraint called **PK_LineItems** on the **OrderID** and **ProductID** columns.
    - A foreign key constraint called **FK_LineItems_Orders** between the **OrderID** column in the **Sales.LineItems** table and the **OrderID** column in the **Sales.Orders** table.

3. Browse to the **D:\Labfiles\Lab06\Starter** folder, and then run **Populate LineItems.cmd**, as Administrator. This script adds rows to the newly created **Sales.LineItems** table.

4. Query the contents of the **Sales.LineItems** table. It should now contain 8 rows.

Task 2: Create and Test a DEFAULT Constraint

1. Type and execute a Transact-SQL statement to create a DEFAULT constraint called **DF_OrderDate** that adds the current date as the default value for the **Orderdate** column of the **Sales.Orders** table. Use the **GETDATE()** function to retrieve the current date.

2. Test the DEFAULT constraint by executing an INSERT statement to add a row to the **Sales.Orders** table using the following values:

    - **OrderID** =**110**
    - **CustomerID** = **3**
    - **OrderDate** = **DEFAULT**

3. Check that the INSERT statement worked by executing a SELECT statement against the **Sales.Orders** table to return the row with the **OrderID** value of **110**. Note that the date was added by the DEFAULT constraint.

4. Close the query window, but leave SQL Server Management Studio open for the next exercise.

> **Results**: After completing this exercise, you will have a new table named **Sales.LineItems** and a new DEFAULT constraint on the **Sales.Orders** table.

## Exercise 2: Creating a View

### Scenario

An application that customers use needs to display all of the orders that a customer has placed and the total value of each order. Instead of hard coding the Transact-SQL query into the application, you have decided to create a view that the application can use to access the required data.

The main tasks for this exercise are as follows:

1. Write a Transact-SQL Query to Retrieve the Required Data
2. Create the VW_CustomerOrders 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, in a new query window, in the **OrdersDatabase** database, write a Transact-SQL query that returns the following columns. Include the appropriate JOIN statements in the FROM clause, and use the alias **O** for the **Sales.Orders** table, the alias **C** for the **Person.Customer** table, and the alias **L** for the **Sales.LineItems** table:
    - **FirstName** (in the **Person.Customers** table).
    - **Lastname** (in the **Person.Customers** table).
    - **OrderID** (in the **Sales.Orders** table).
    - **OrderDate** (in the **Sales.Orders** table).
    - **UnitPrice** (in the **Sales.LineItems** table).
    - **Quantity** (in the **Sales.LineItems** table).
2. Execute the query to ensure that it returns the required columns and rows.
3. In the SELECT list, remove the **UnitPrice** and **Quantity** columns, and replace them with the following Transact-SQL code:

    ```
    SUM (UnitPrice * Quantity) AS [Order Total]
    ```

4. In the SELECT list, remove the **FirstName** and **LastName** columns, and replace them with the following Transact-SQL code:

    ```
    (FirstName + ' ' + LastName) AS [Customer Name]
    ```

5. After the FROM clause and the JOIN statements, type the following Transact-SQL code:

    ```
    GROUP BY O.OrderID, Firstname, Lastname, OrderDate;
    GO
    ```

6. Execute the query, which should return four rows, including a column that shows the total cost for each order.

Task 2: Create the VW_CustomerOrders View

1. Create a view called **VW_CustomerOrders** that includes the Transact-SQL statement that you created in the previous task.
2. Test the view by writing a Transact-SQL SELECT statement that selects all columns from the view.
3. Close the query window, but leave SQL Server Management Studio open for the next exercise.

> **Results**: After completing this exercise, you will have a new view in the database.

## Exercise 3: Creating a Stored Procedure

### Scenario

To ensure that new orders are entered consistently into the **Sales.Orders** table, you will create a stored procedure. The procedure will use input parameters to accept values for the **OrderID** and **CustomerID** columns, but the value for the **OrderDate** column will be supplied by the DEFAULT constraint on that column.

The main tasks for this exercise are as follows:

1. Create the USP_InsertOrders Stored Procedure
2. Test the 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.

1. In SQL Server Management Studio, in a new query window, ensure that the database context is the **OrdersDatabase** database. Write a Transact-SQL INSERT statement that inserts a row into the **Sales.Orders** table. In the INSERT clause, in the column list, specify the following columns:
    - **OrderID**
    - **CustomerID**

> **Note:** You do not need to specify the **OrderDate** column because this column has a DEFAULT constraint that supplies its value.

2. In the VALUES list, use the following values:
    - **120**
    - **2**
3. Use the INSERT statement that you just typed to write a CREATE STORED PROCEDURE statement. Use the following specifications:
    - Name: **USP_InsertOrders**.
    - Input parameters:
        - **@OrderID** (data type **int**).

        - **@CustomerID** (data type **int**).
    - Replace the literal values (**120** and **2**) in the VALUES list with the input parameter values you added above.
    - Execute the statement to create the procedure.

Task 2: Test the Stored Procedure

1. Test the procedure by executing it using the following values:
    - **@OrderID** = **150**
    - **@CustomerID** = **1**
2. Write a Transact-SQL SELECT statement to view the newly inserted rows in the **Sales.Orders** table.
3. Close SQL Server Management Studio without saving changes.

> **Results**: After completing this exercise, you will have a new stored procedure in the database.

## 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?

**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?