Lab 01 - Extend.
Building an Product Management Application using Dabatase, LINQ and WPF

1. Introduction

This lab explores creating an application using WPF with .NET Core, and C# extended Lab 01. An "in-memory database" using collection of products is called List will be replaced by a Database instead.

The application has to support adding, viewing, modifying, and removing products—a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD) using Database connection and SQL commands.

2. Lab Objectives

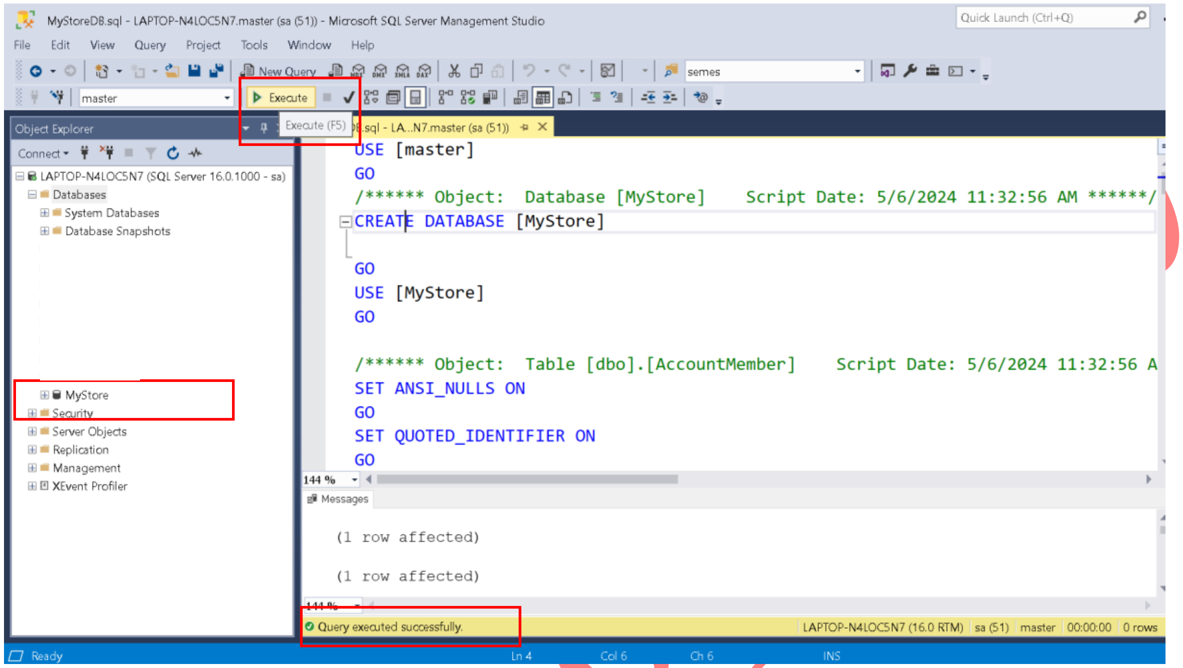
In this lab, you will:

- Use the Visual Studio.NET to create WPF application and Class Library (.dll) project.
- Use the Visual Studio.NET to create Windows Forms and Class Library (.dll) project.
- Create a Database in order to persist products, and use LINQ to Object to find items.
- Apply passing data in WPF application
- Apply Repository pattern in a project.
- Add CRUD action methods to WPF application.
- Run the project and test the WPF application actions.

Activity 01: Create a Database

Create a database from exsiting DB script

Step 01. Open a **MyStoreDB.sql** using SQL Server Management Studio, click button Execute to run script, check if Query executed successfully and database **MyStore** appears in list Databases (Object Explorer)

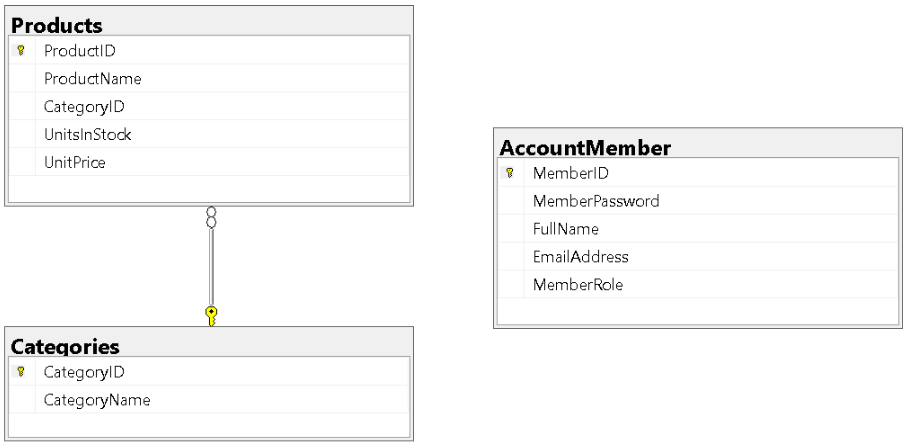


```
USE [master]
GO
/****** Object:  Database [MyStore]    Script Date: 5/6/2024 11:32:56 AM ******/
CREATE DATABASE [MyStore]

GO
USE [MyStore]
GO

/****** Object:  Table [dbo].[AccountMember]    Script Date: 5/6/2024 11:32:56 A
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

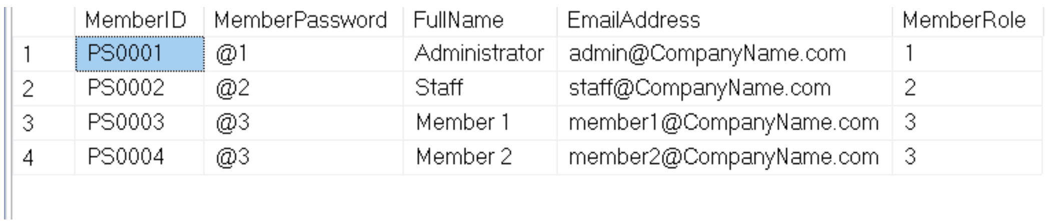
```
(1 row affected)

(1 row affected)
```

Query executed successfully. LAPTOP-N4LOC5N7 (16.0 RTM) sa (51) master 00:00:00 0 rows

+ Check diagram and data in MyStore database

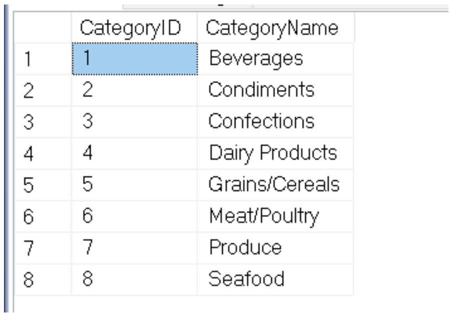


AccountMember



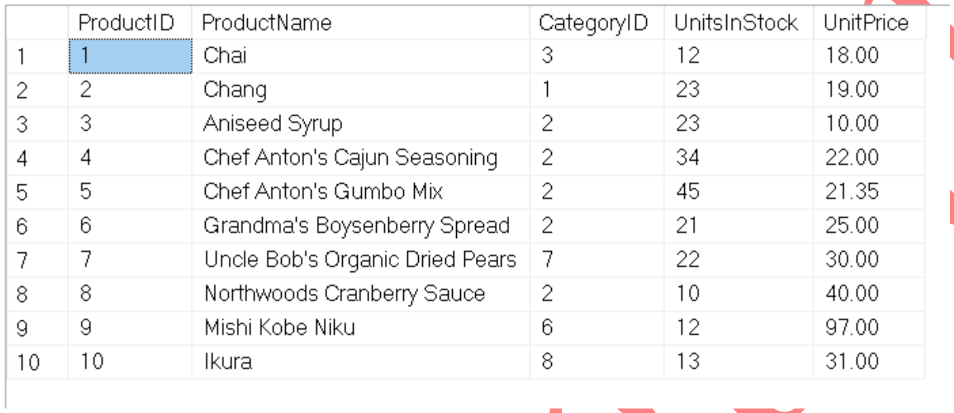
	MemberID	MemberPassword	FullName	EmailAddress	MemberRole
1	PS0001	@1	Administrator	admin@CompanyName.com	1
2	PS0002	@2	Staff	staff@CompanyName.com	2
3	PS0003	@3	Member 1	member1@CompanyName.com	3
4	PS0004	@3	Member 2	member2@CompanyName.com	3

Categories



	CategoryID	CategoryName
1	1	Beverages
2	2	Condiments
3	3	Confections
4	4	Dairy Products
5	5	Grains/Cereals
6	6	Meat/Poultry
7	7	Produce
8	8	Seafood

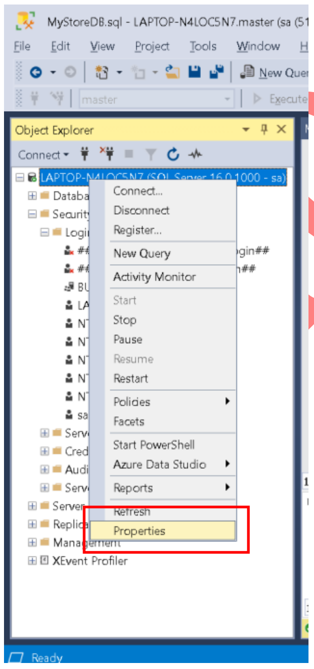
Products

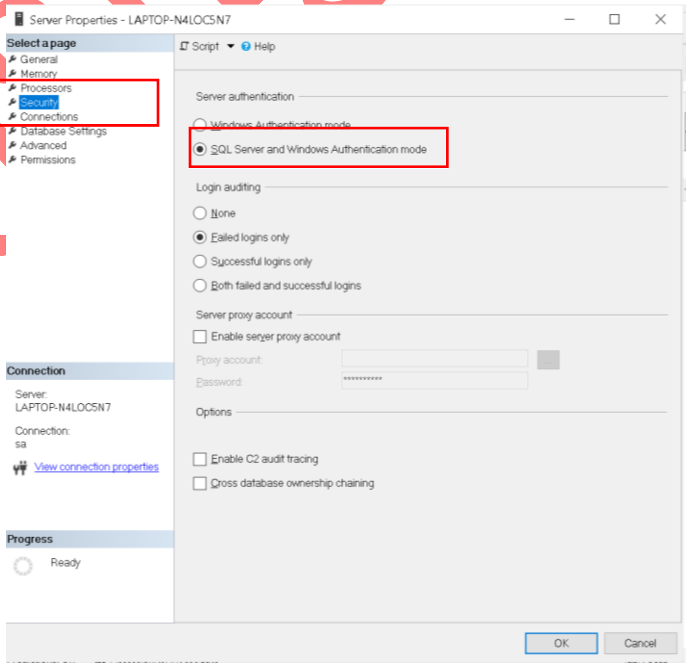


	ProductID	ProductName	CategoryID	UnitsInStock	UnitPrice
1	1	Chai	3	12	18.00
2	2	Chang	1	23	19.00
3	3	Aniseed Syrup	2	23	10.00
4	4	Chef Anton's Cajun Seasoning	2	34	22.00
5	5	Chef Anton's Gumbo Mix	2	45	21.35
6	6	Grandma's Boysenberry Spread	2	21	25.00
7	7	Uncle Bob's Organic Dried Pears	7	22	30.00
8	8	Northwoods Cranberry Sauce	2	10	40.00
9	9	Mishi Kobe Niku	6	12	97.00
10	10	Ikura	8	13	31.00

<u>Step 02</u>. Config **system administrator user** to Database.

+ Check if sa user is enable: right-click connection, choose Properties, choose Security, check SQL Server and Windows Authentication mode under Server authentication


MyStoreDB.sql - LAPTOP-N4LOC5N7.master (sa (51


Server Properties - LAPTOP-N4LOC5N7

Connection
Server: LAPTOP-N4LOC5N7
Connection: sa
View connection properties

Progress
Ready

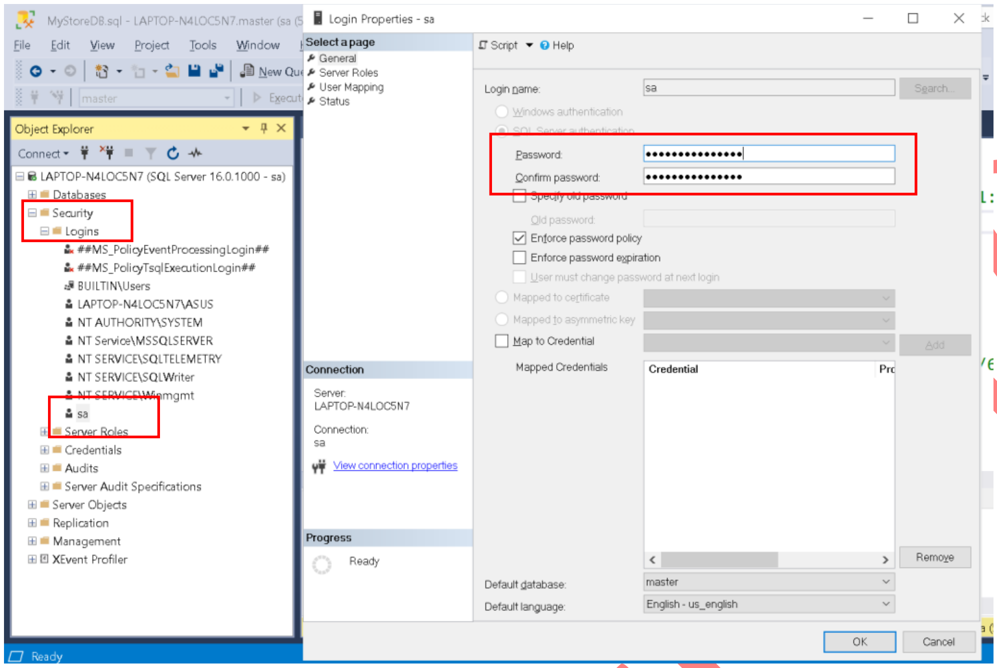
Server authentication
- ○ Windows Authentication mode
- ● SQL Server and Windows Authentication mode

Login auditing
- ○ None
- ● Failed logins only
- ○ Successful logins only
- ○ Both failed and successful logins

Server proxy account
- ☐ Enable server proxy account
- Proxy account:
- Password: **********

Options
- ☐ Enable C2 audit tracing
- ☐ Cross database ownership chaining

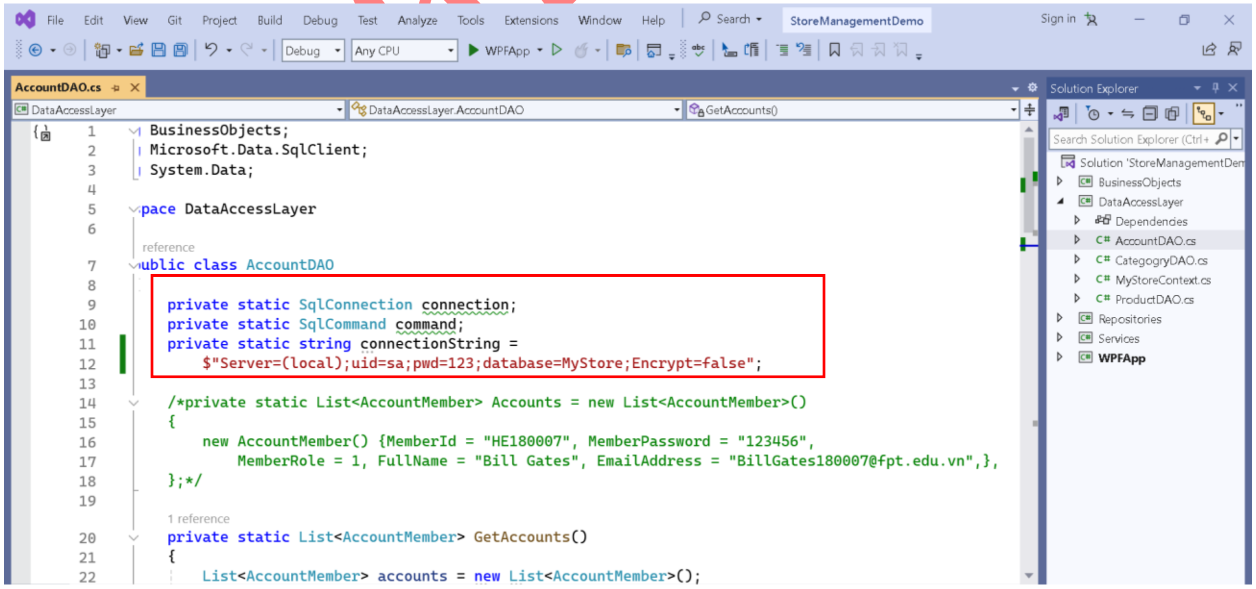
+ Config sa user password: In Object Explorer, choose Security => Logins => sa, right-click and choose Properties, set Password and Confirm Password to **123**, restarts SQL Server and relogin using sa user.



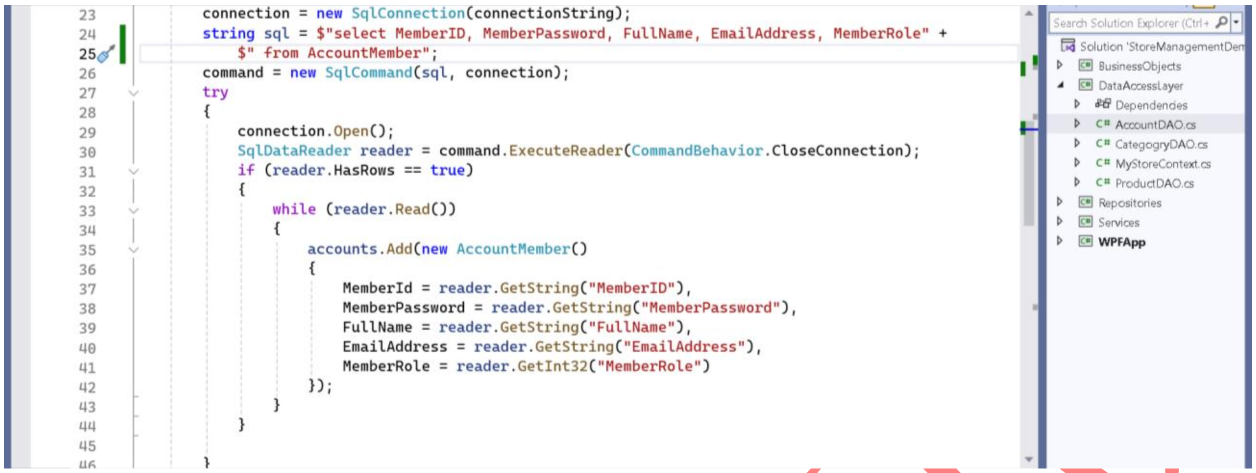
Activity 02: Rewrite codes for the AccountDAO.cs

<u>Step 01</u>. Open project created in Lab-01, On the **DataAccess** project, open a class named **AccountDAO.cs** and rewrite codes as follows:

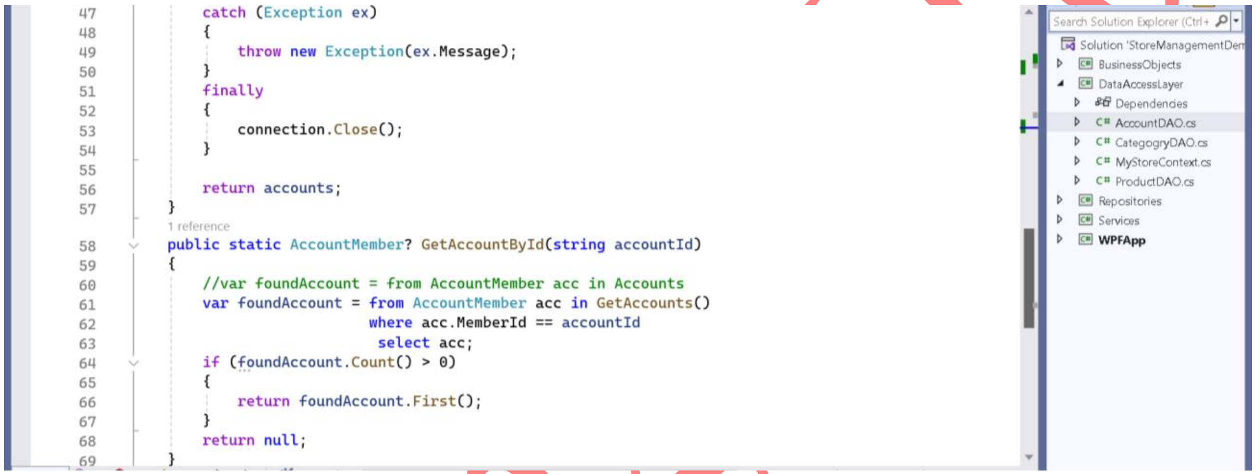
Note: consider about connection, command, connectionString



```
BusinessObjects;
Microsoft.Data.SqlClient;
System.Data;

pace DataAccessLayer

reference
ublic class AccountDAO

    private static SqlConnection connection;
    private static SqlCommand command;
    private static string connectionString =
        $"Server=(local);uid=sa;pwd=123;database=MyStore;Encrypt=false";

    /*private static List<AccountMember> Accounts = new List<AccountMember>()
    {
        new AccountMember() {MemberId = "HE180007", MemberPassword = "123456",
            MemberRole = 1, FullName = "Bill Gates", EmailAddress = "BillGates180007@fpt.edu.vn",},
    };*/

    1 reference
    private static List<AccountMember> GetAccounts()
    {
        List<AccountMember> accounts = new List<AccountMember>();
```

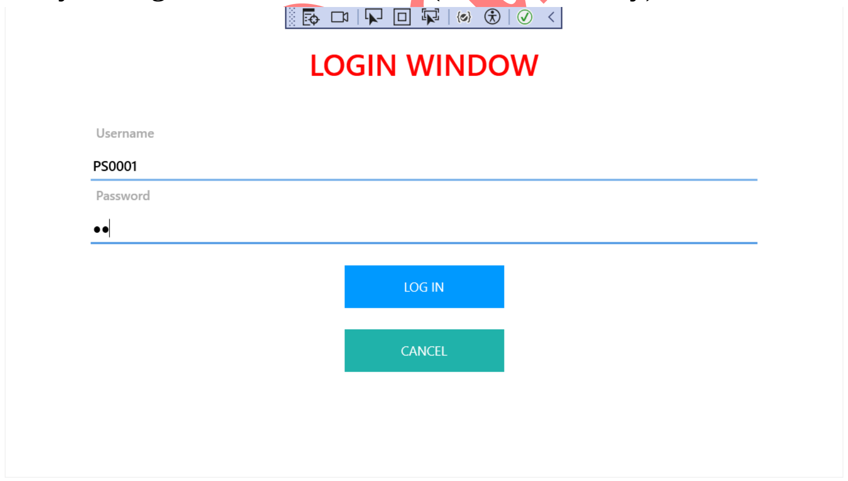


```
        connection = new SqlConnection(connectionString);
        string sql = $"select MemberID, MemberPassword, FullName, EmailAddress, MemberRole" +
            $" from AccountMember";
        command = new SqlCommand(sql, connection);
        try
        {
            connection.Open();
            SqlDataReader reader = command.ExecuteReader(CommandBehavior.CloseConnection);
            if (reader.HasRows == true)
            {
                while (reader.Read())
                {
                    accounts.Add(new AccountMember()
                    {
                        MemberId = reader.GetString("MemberID"),
                        MemberPassword = reader.GetString("MemberPassword"),
                        FullName = reader.GetString("FullName"),
                        EmailAddress = reader.GetString("EmailAddress"),
                        MemberRole = reader.GetInt32("MemberRole")
                    });
                }
            }

        }
```



```
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            connection.Close();
        }

        return accounts;
    }
    1 reference
    public static AccountMember? GetAccountById(string accountId)
    {
        //var foundAccount = from AccountMember acc in Accounts
        var foundAccount = from AccountMember acc in GetAccounts()
                           where acc.MemberId == accountId
                            select acc;
        if (foundAccount.Count() > 0)
        {
            return foundAccount.First();
        }
        return null;
    }
```

<u>Step 02</u>. Build and run application.
+ Try to Login with user PS0001 (successfully)
+ Try to Login with another user (not successfully)



LOGIN WINDOW

Username

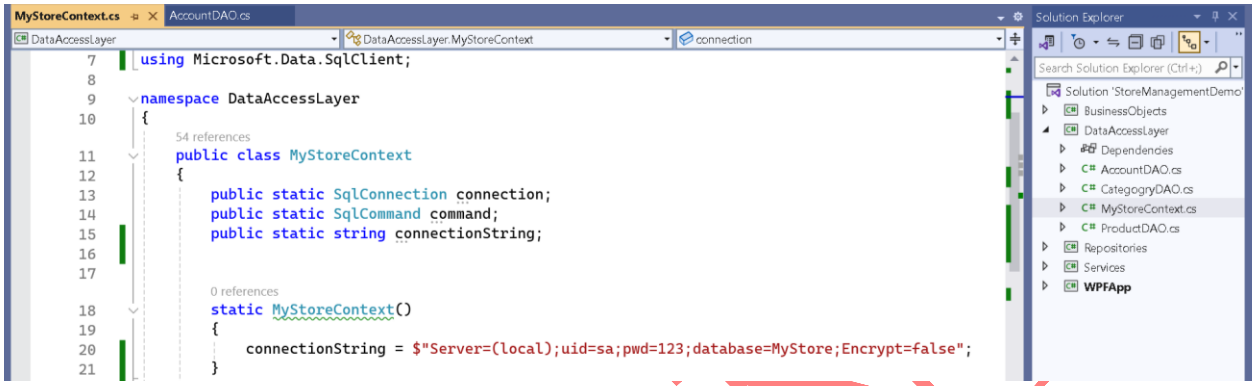
PS0001

Password

••

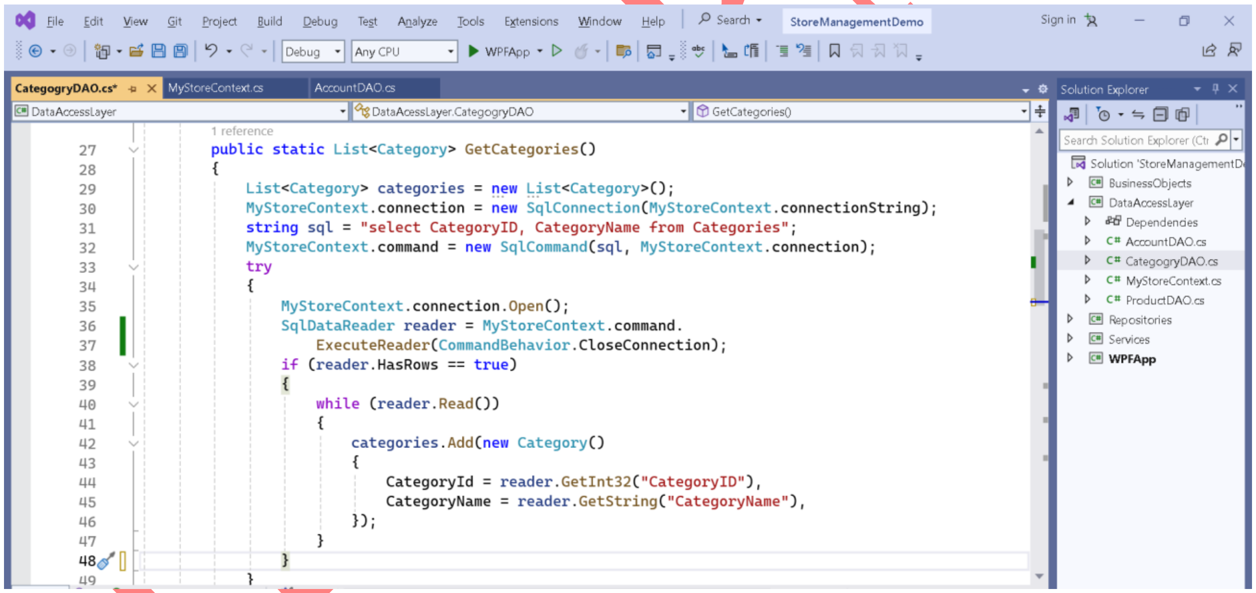
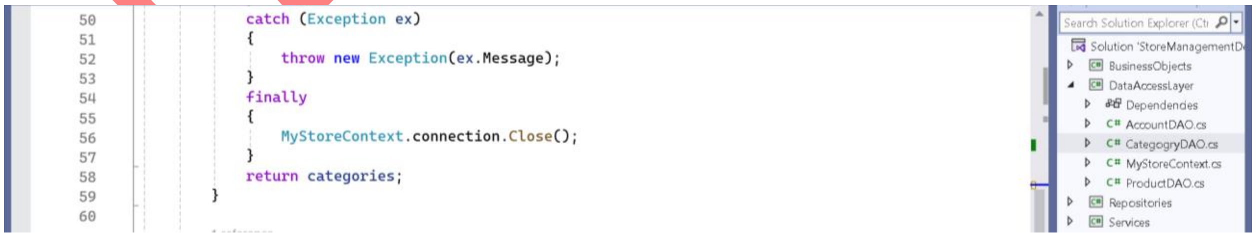
Activity 03: Write codes for the CategoryDAO.cs

Step 01. On the **DataAccessLayer** project, add a class named **MyStoreContext.cs** and write codes as follows:



```
using Microsoft.Data.SqlClient;

namespace DataAccessLayer
{
    54 references
    public class MyStoreContext
    {
        public static SqlConnection connection;
        public static SqlCommand command;
        public static string connectionString;


        0 references
        static MyStoreContext()
        {
            connectionString = $"Server=(local);uid=sa;pwd=123;database=MyStore;Encrypt=false";
        }
```

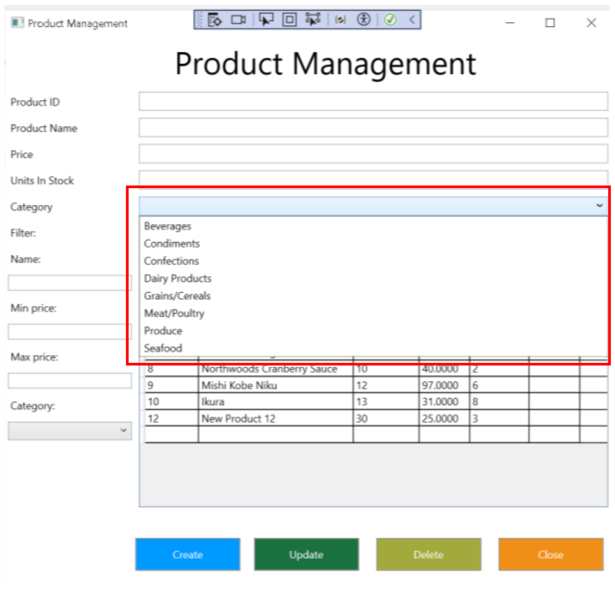
<u>Step 02</u>. On the **DataAccessLayer** project, open class named **CategoryDAO.cs** and rewrite codes for method **GetCategories()** as follows:

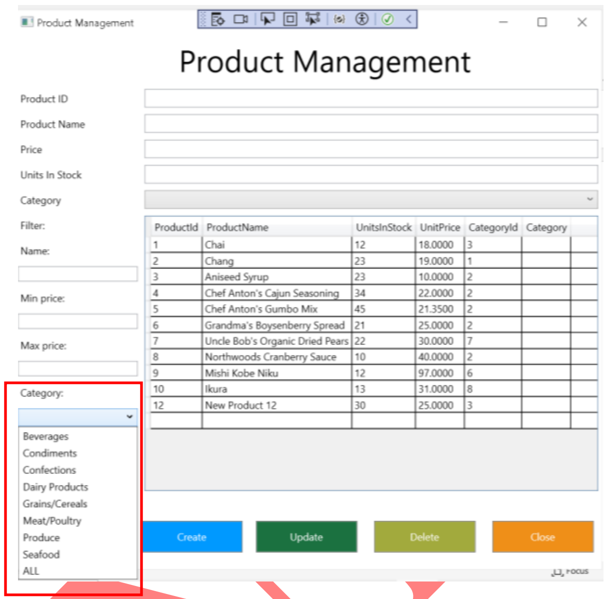
```
        1 reference
        public static List<Category> GetCategories()
        {
            List<Category> categories = new List<Category>();
            MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
            string sql = "select CategoryID, CategoryName from Categories";
            MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
            try
            {
                MyStoreContext.connection.Open();
                SqlDataReader reader = MyStoreContext.command.
                    ExecuteReader(CommandBehavior.CloseConnection);
                if (reader.HasRows == true)
                {
                    while (reader.Read())
                    {
                        categories.Add(new Category()
                        {
                            CategoryId = reader.GetInt32("CategoryID"),
                            CategoryName = reader.GetString("CategoryName"),
                        });
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
            finally
            {
                MyStoreContext.connection.Close();
            }
            return categories;
        }
```

<u>Step 03</u>. Build and run application.

+ Check if list of categories appears Category and FilterCategory (combobox)



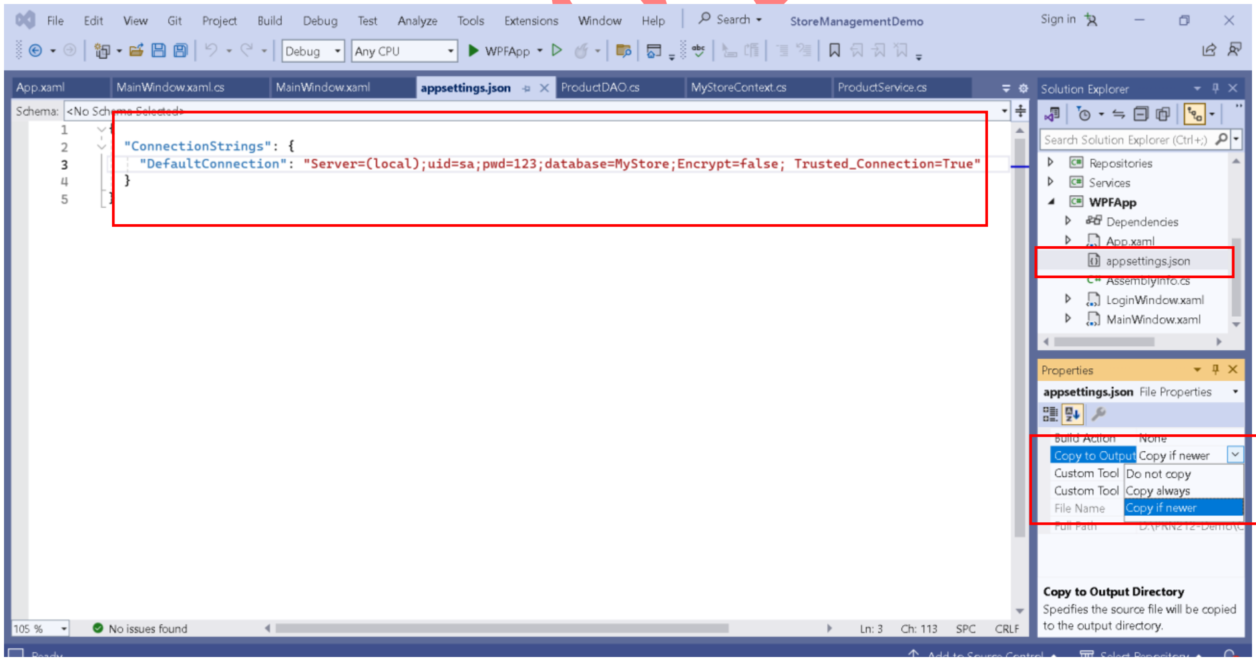
8	Northwoods Cranberry Sauce	10	40.0000	2	
9	Mishi Kobe Niku	12	97.0000	6	
10	Ikura	13	31.0000	8	
12	New Product 12	30	25.0000	3	



ProductId	ProductName	UnitsInStock	UnitPrice	CategoryId	Category
1	Chai	12	18.0000	3	
2	Chang	23	19.0000	1	
3	Aniseed Syrup	23	10.0000	2	
4	Chef Anton's Cajun Seasoning	34	22.0000	2	
5	Chef Anton's Gumbo Mix	45	21.3500	2	
6	Grandma's Boysenberry Spread	21	25.0000	2	
7	Uncle Bob's Organic Dried Pears	22	30.0000	7	
8	Northwoods Cranberry Sauce	10	40.0000	2	
9	Mishi Kobe Niku	12	97.0000	6	
10	Ikura	13	31.0000	8	
12	New Product 12	30	25.0000	3	

Activity 04: Write codes for the ProductDAO.cs

Step 01. On the **WPF** project, add an json file named **appsettings.josn** and write codes for setting **connection string** as follows. Then right click to appsettings.json, choose properties, choose **Copy if newer** in Copy to Output Directory build action

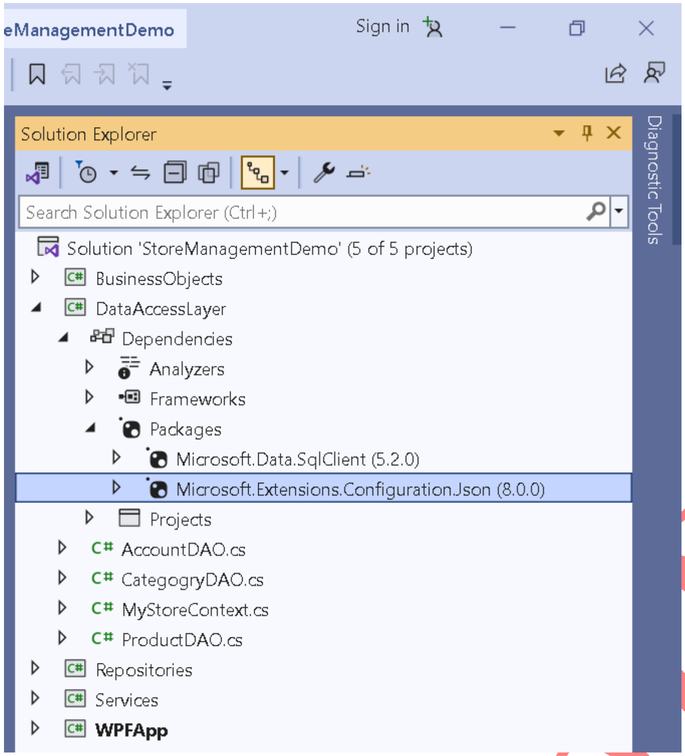


```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(local);uid=sa;pwd=123;database=MyStore;Encrypt=false; Trusted_Connection=True"
  }
}
```

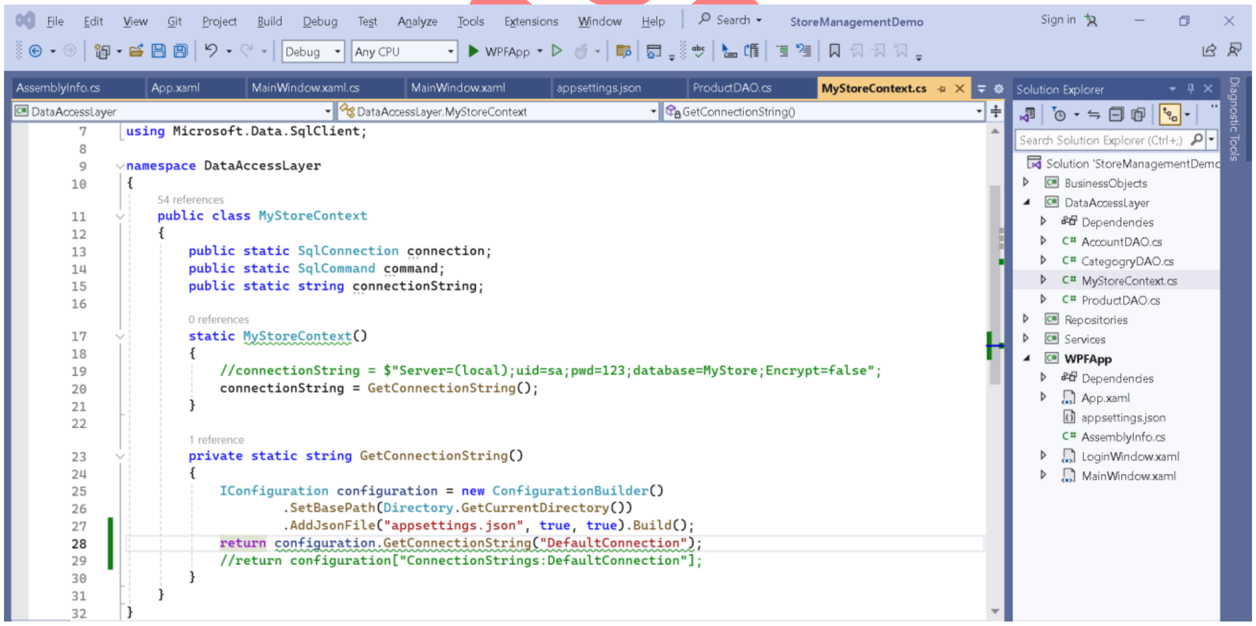
Build Action: None
Copy to Output Directory: Copy if newer (Do not copy / Copy always / Copy if newer)

Copy to Output Directory
Specifies the source file will be copied to the output directory.

Step 02. On the **DataAccessLayer** project, add dependency named **Microsoft.Extension.Configuration.Json (8.0.0)** using Manage NuGet Packages …

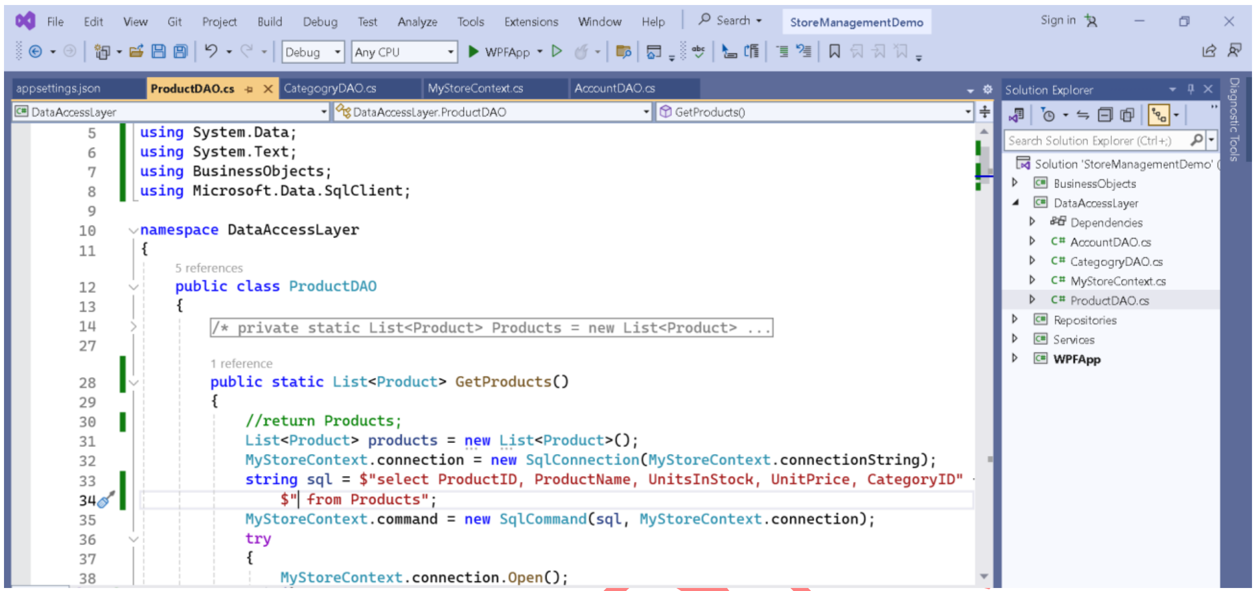


+ Modify code in MyStoreContext.cs as follows:

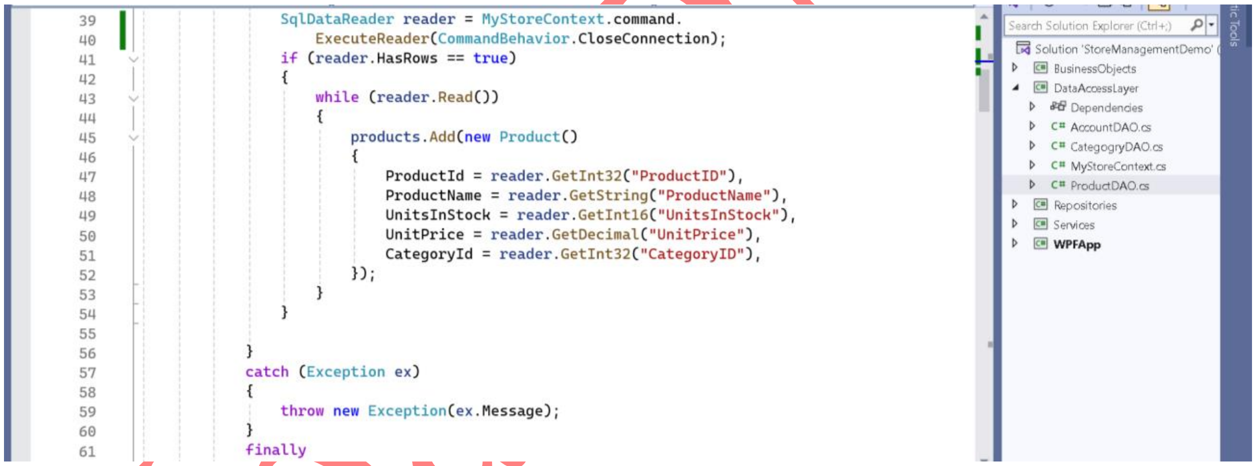


```
using Microsoft.Data.SqlClient;

namespace DataAccessLayer
{
    54 references
    public class MyStoreContext
    {
        public static SqlConnection connection;
        public static SqlCommand command;
        public static string connectionString;

        0 references
        static MyStoreContext()
        {
            //connectionString = $"Server=(local);uid=sa;pwd=123;database=MyStore;Encrypt=false";
            connectionString = GetConnectionString();
        }

        1 reference
        private static string GetConnectionString()
        {
            IConfiguration configuration = new ConfigurationBuilder()
                    .SetBasePath(Directory.GetCurrentDirectory())
                    .AddJsonFile("appsettings.json", true, true).Build();
            return configuration.GetConnectionString("DefaultConnection");
            //return configuration["ConnectionStrings:DefaultConnection"];
        }
    }
}
```

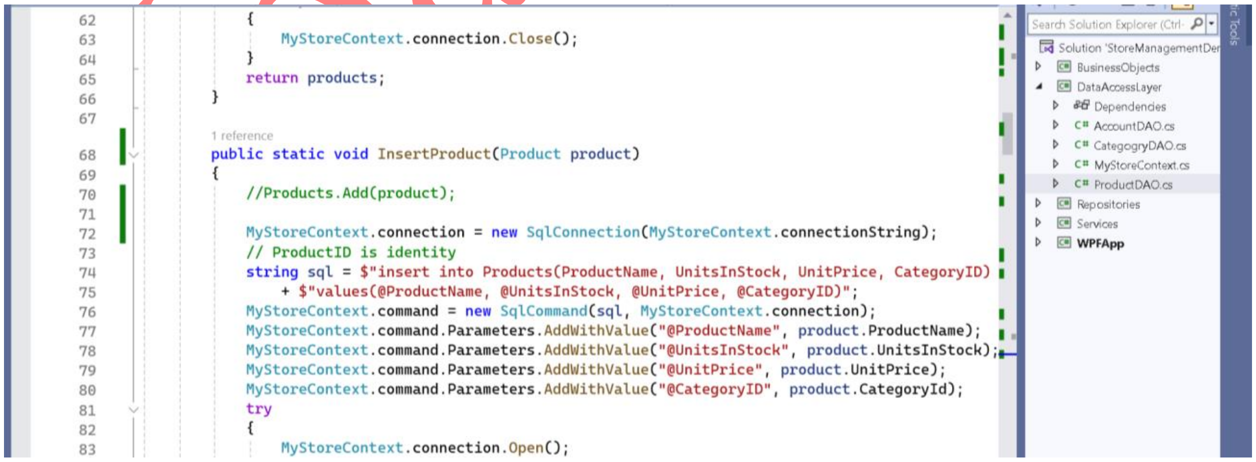
Step 03. On the **DataAccessLayer** project, open a class named **ProductDAO.cs** and rewrite codes for CRUD methods of product as follows:



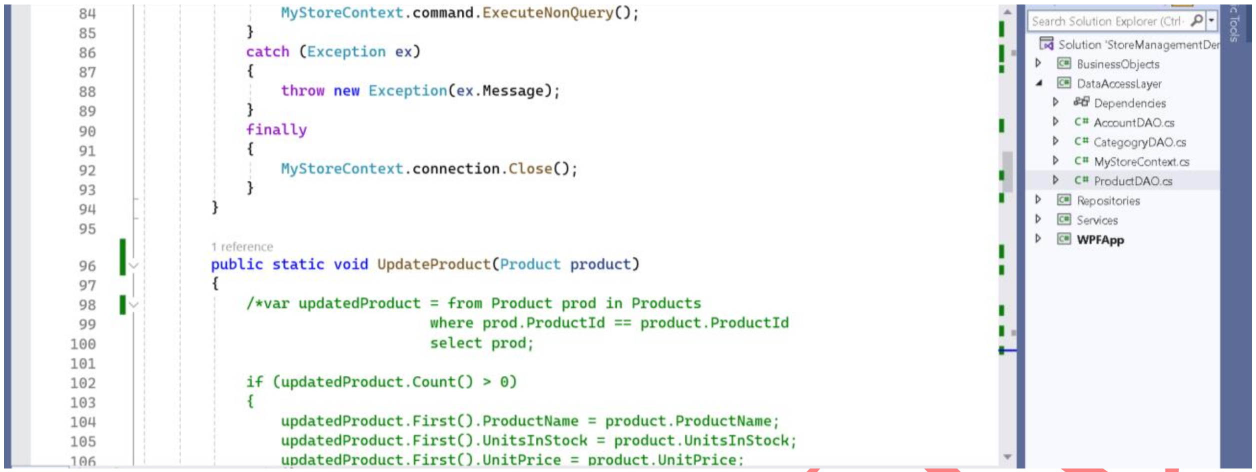
```
using System.Data;
using System.Text;
using BusinessObjects;
using Microsoft.Data.SqlClient;

namespace DataAccessLayer
{
    5 references
    public class ProductDAO
    {
        /* private static List<Product> Products = new List<Product> ...

        1 reference
        public static List<Product> GetProducts()
        {
            //return Products;
            List<Product> products = new List<Product>();
            MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
            string sql = $"select ProductID, ProductName, UnitsInStock, UnitPrice, CategoryID"
                $" from Products";
            MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
            try
            {
                MyStoreContext.connection.Open();
```



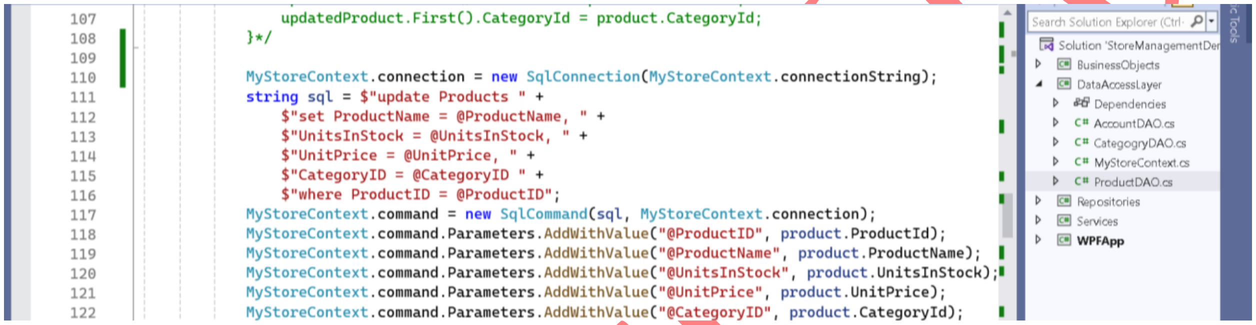
```
                SqlDataReader reader = MyStoreContext.command.
                    ExecuteReader(CommandBehavior.CloseConnection);
                if (reader.HasRows == true)
                {
                    while (reader.Read())
                    {
                        products.Add(new Product()
                        {
                            ProductId = reader.GetInt32("ProductID"),
                            ProductName = reader.GetString("ProductName"),
                            UnitsInStock = reader.GetInt16("UnitsInStock"),
                            UnitPrice = reader.GetDecimal("UnitPrice"),
                            CategoryId = reader.GetInt32("CategoryID"),
                        });
                    }
                }

            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
            finally
```



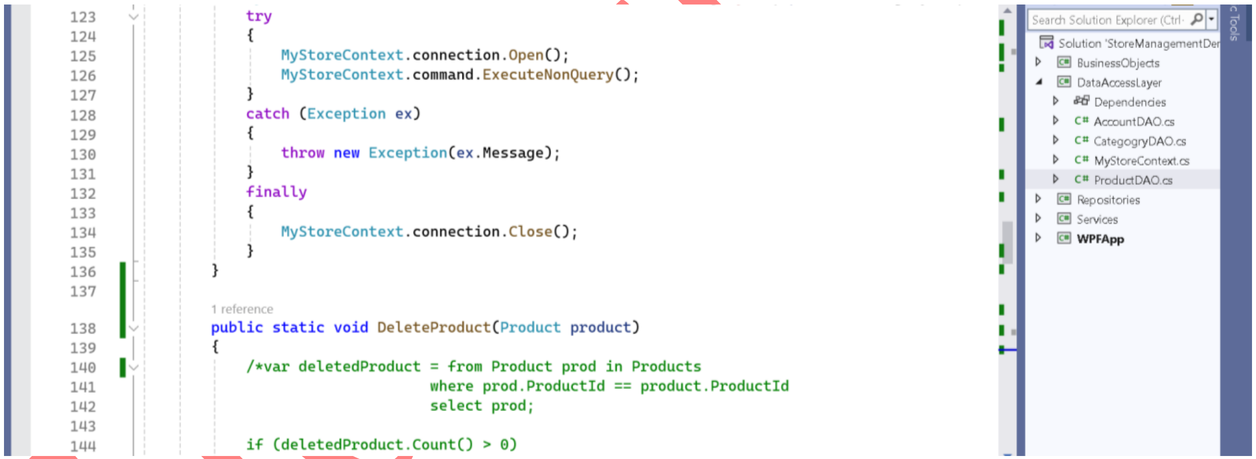
```
            {
                MyStoreContext.connection.Close();
            }
            return products;
        }

        1 reference
        public static void InsertProduct(Product product)
        {
            //Products.Add(product);

            MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
            // ProductID is identity
            string sql = $"insert into Products(ProductName, UnitsInStock, UnitPrice, CategoryID)
                + $"values(@ProductName, @UnitsInStock, @UnitPrice, @CategoryID)";
            MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
            MyStoreContext.command.Parameters.AddWithValue("@ProductName", product.ProductName);
            MyStoreContext.command.Parameters.AddWithValue("@UnitsInStock", product.UnitsInStock);
            MyStoreContext.command.Parameters.AddWithValue("@UnitPrice", product.UnitPrice);
            MyStoreContext.command.Parameters.AddWithValue("@CategoryID", product.CategoryId);
            try
            {
                MyStoreContext.connection.Open();
```



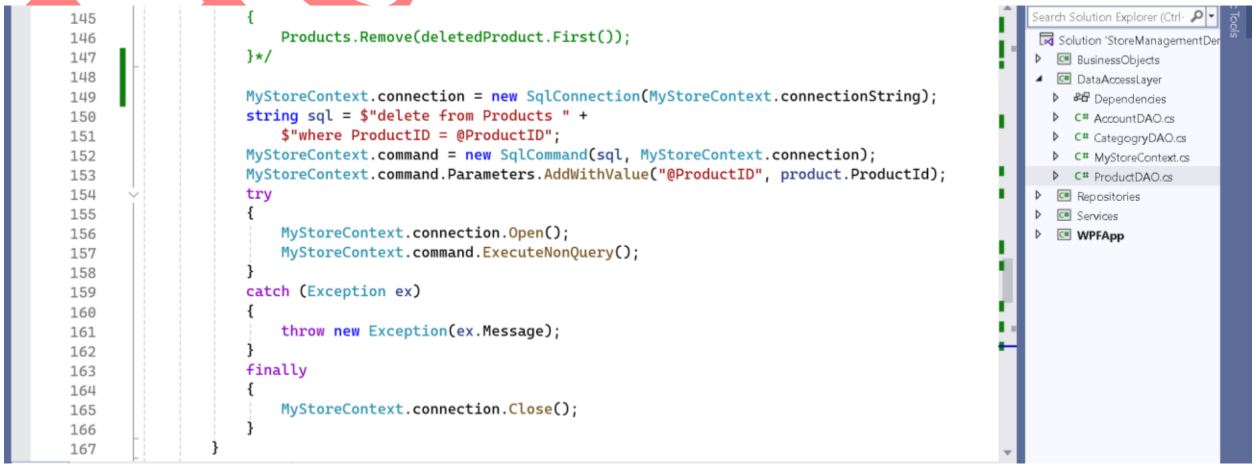
```
            MyStoreContext.command.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            MyStoreContext.connection.Close();
        }
    }

    1 reference
    public static void UpdateProduct(Product product)
    {
        /*var updatedProduct = from Product prod in Products
                             where prod.ProductId == product.ProductId
                             select prod;

        if (updatedProduct.Count() > 0)
        {
            updatedProduct.First().ProductName = product.ProductName;
            updatedProduct.First().UnitsInStock = product.UnitsInStock;
            updatedProduct.First().UnitPrice = product.UnitPrice;
```



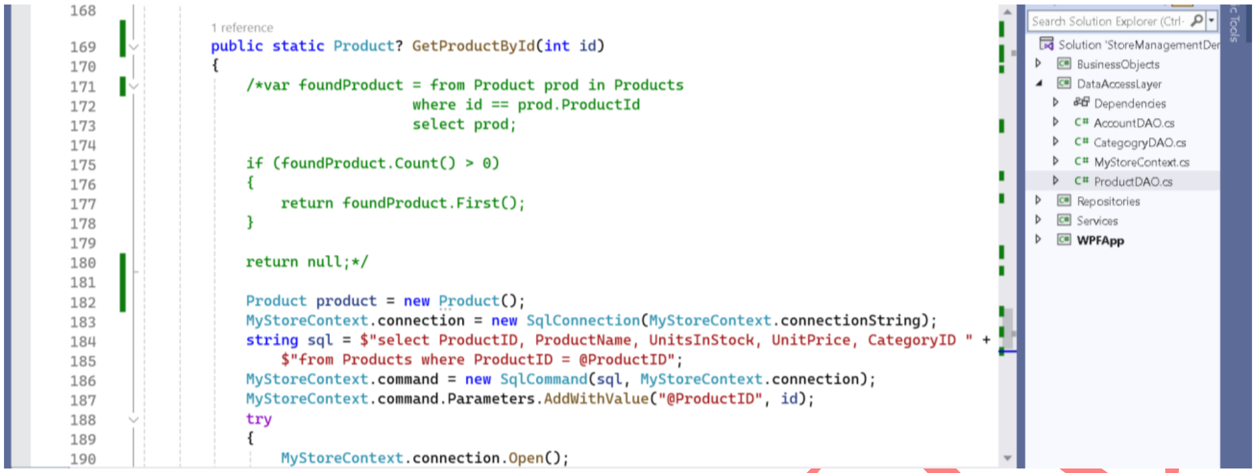
```
            updatedProduct.First().CategoryId = product.CategoryId;
        }*/

        MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
        string sql = $"update Products " +
            $"set ProductName = @ProductName, " +
            $"UnitsInStock = @UnitsInStock, " +
            $"UnitPrice = @UnitPrice, " +
            $"CategoryID = @CategoryID " +
            $"where ProductID = @ProductID";
        MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
        MyStoreContext.command.Parameters.AddWithValue("@ProductID", product.ProductId);
        MyStoreContext.command.Parameters.AddWithValue("@ProductName", product.ProductName);
        MyStoreContext.command.Parameters.AddWithValue("@UnitsInStock", product.UnitsInStock);
        MyStoreContext.command.Parameters.AddWithValue("@UnitPrice", product.UnitPrice);
        MyStoreContext.command.Parameters.AddWithValue("@CategoryID", product.CategoryId);
```



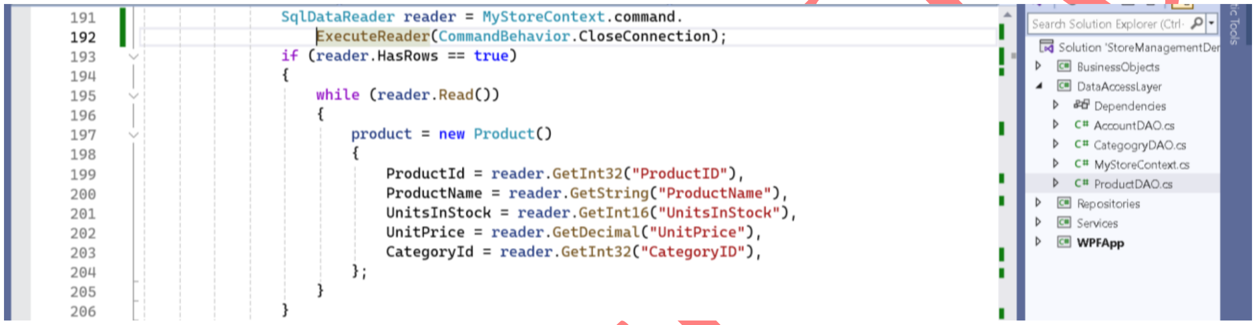
```
        try
        {
            MyStoreContext.connection.Open();
            MyStoreContext.command.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            MyStoreContext.connection.Close();
        }
    }

    1 reference
    public static void DeleteProduct(Product product)
    {
        /*var deletedProduct = from Product prod in Products
                             where prod.ProductId == product.ProductId
                             select prod;

        if (deletedProduct.Count() > 0)
```



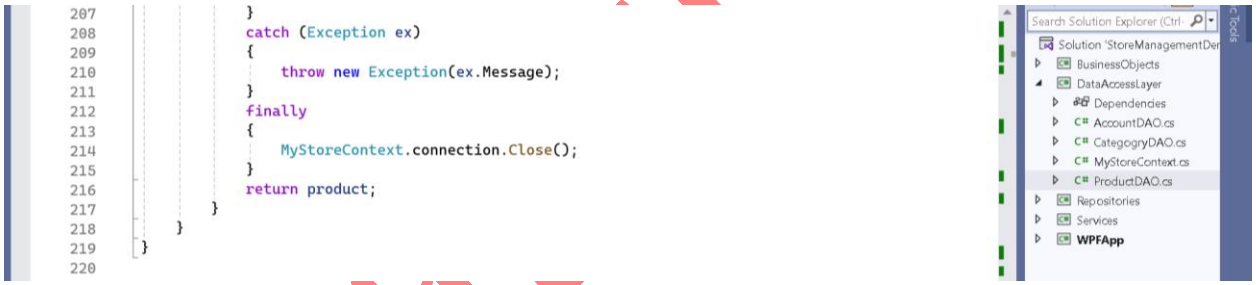
```
        {
            Products.Remove(deletedProduct.First());
        }*/

        MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
        string sql = $"delete from Products " +
            $"where ProductID = @ProductID";
        MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
        MyStoreContext.command.Parameters.AddWithValue("@ProductID", product.ProductId);
        try
        {
            MyStoreContext.connection.Open();
            MyStoreContext.command.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            MyStoreContext.connection.Close();
        }
    }
```



```
        1 reference
        public static Product? GetProductById(int id)
        {
            /*var foundProduct = from Product prod in Products
                               where id == prod.ProductId
                               select prod;

            if (foundProduct.Count() > 0)
            {
                return foundProduct.First();
            }

            return null;*/

            Product product = new Product();
            MyStoreContext.connection = new SqlConnection(MyStoreContext.connectionString);
            string sql = $"select ProductID, ProductName, UnitsInStock, UnitPrice, CategoryID " +
                $"from Products where ProductID = @ProductID";
            MyStoreContext.command = new SqlCommand(sql, MyStoreContext.connection);
            MyStoreContext.command.Parameters.AddWithValue("@ProductID", id);
            try
            {
                MyStoreContext.connection.Open();
```



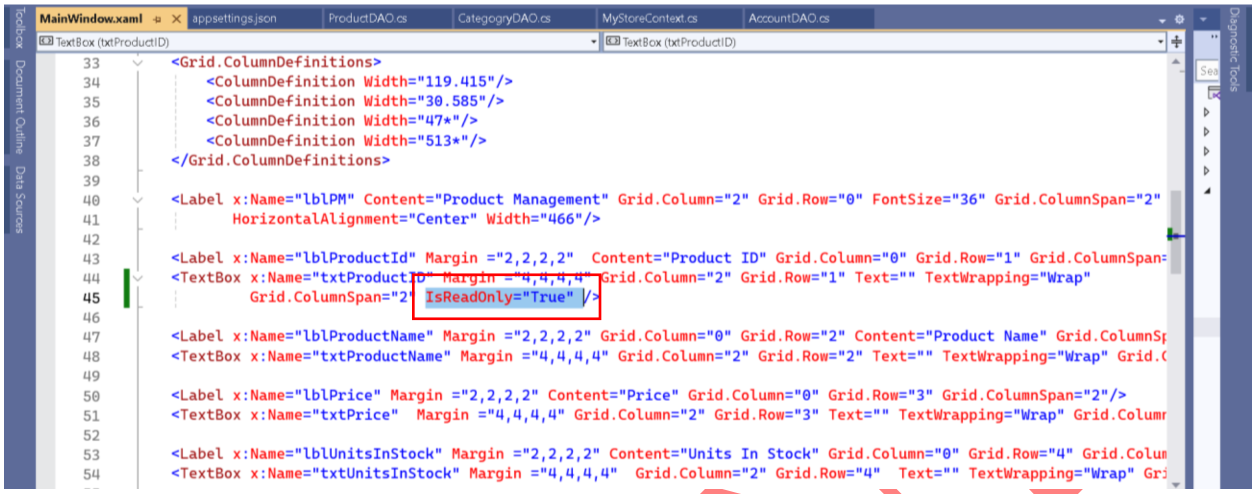
```
                SqlDataReader reader = MyStoreContext.command.
                    ExecuteReader(CommandBehavior.CloseConnection);
                if (reader.HasRows == true)
                {
                    while (reader.Read())
                    {
                        product = new Product()
                        {
                            ProductId = reader.GetInt32("ProductID"),
                            ProductName = reader.GetString("ProductName"),
                            UnitsInStock = reader.GetInt16("UnitsInStock"),
                            UnitPrice = reader.GetDecimal("UnitPrice"),
                            CategoryId = reader.GetInt32("CategoryID"),
                        };
                    }
                }
```



```
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
            finally
            {
                MyStoreContext.connection.Close();
            }
            return product;
        }
    }
}
```

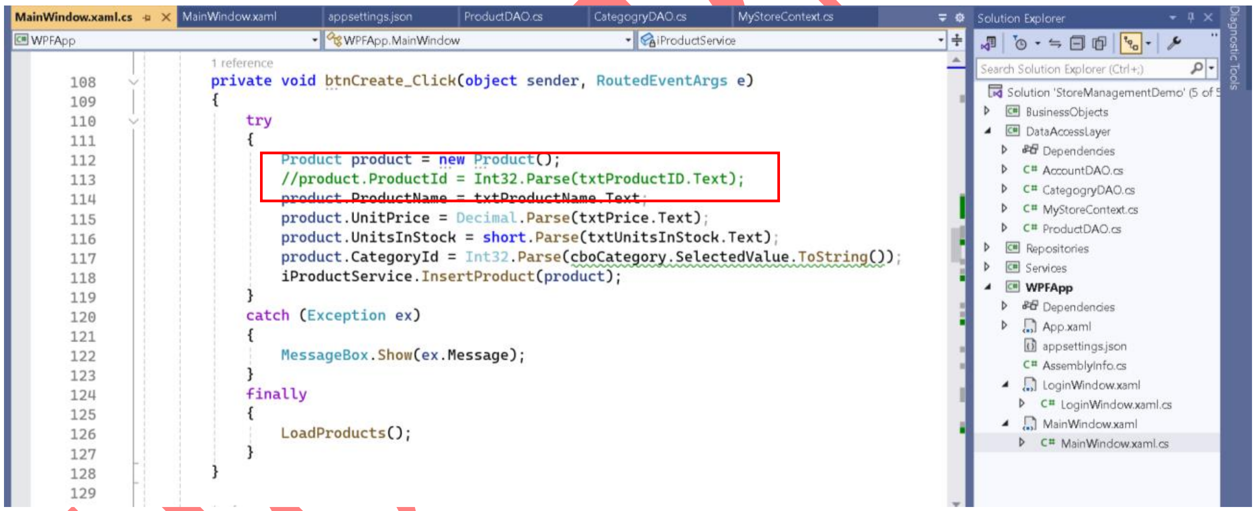
Activity 05: Write codes for the WPF project

Step 01. On the **WFP** project, open **MainWindow.xaml** and add codes to make txtProductID read only as follows:



```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="119.415"/>
    <ColumnDefinition Width="30.585"/>
    <ColumnDefinition Width="47*"/>
    <ColumnDefinition Width="513*"/>
</Grid.ColumnDefinitions>

<Label x:Name="lblPM" Content="Product Management" Grid.Column="2" Grid.Row="0" FontSize="36" Grid.ColumnSpan="2"
       HorizontalAlignment="Center" Width="466"/>

<Label x:Name="lblProductId" Margin ="2,2,2,2"  Content="Product ID" Grid.Column="0" Grid.Row="1" Grid.ColumnSpan=
<TextBox x:Name="txtProductID" Margin ="4,4,4,4" Grid.Column="2" Grid.Row="1" Text="" TextWrapping="Wrap"
         Grid.ColumnSpan="2" IsReadOnly="True" />

<Label x:Name="lblProductName" Margin ="2,2,2,2" Grid.Column="0" Grid.Row="2" Content="Product Name" Grid.ColumnSp
<TextBox x:Name="txtProductName" Margin ="4,4,4,4" Grid.Column="2" Grid.Row="2" Text="" TextWrapping="Wrap" Grid.C

<Label x:Name="lblPrice" Margin ="2,2,2,2" Content="Price" Grid.Column="0" Grid.Row="3" Grid.ColumnSpan="2"/>
<TextBox x:Name="txtPrice"  Margin ="4,4,4,4" Grid.Column="2" Grid.Row="3" Text="" TextWrapping="Wrap" Grid.Colum

<Label x:Name="lblUnitsInStock" Margin ="2,2,2,2" Content="Units In Stock" Grid.Column="0" Grid.Row="4" Grid.Colun
<TextBox x:Name="txtUnitsInStock" Margin ="4,4,4,4"  Grid.Column="2" Grid.Row="4"  Text="" TextWrapping="Wrap" Gri
```

Step 02. On the **WPF** project, open behind code class named **MainWindow.xaml.cs** and rewrite codes for method btnCreate_Click() as follows:



```
private void btnCreate_Click(object sender, RoutedEventArgs e)
{
    try
    {
        Product product = new Product();
        //product.ProductId = Int32.Parse(txtProductID.Text);
        product.ProductName = txtProductName.Text;
        product.UnitPrice = Decimal.Parse(txtPrice.Text);
        product.UnitsInStock = short.Parse(txtUnitsInStock.Text);
        product.CategoryId = Int32.Parse(cboCategory.SelectedValue.ToString());
        iProductService.InsertProduct(product);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        LoadProducts();
    }
}
```

Step 03. Build and run application. Test CRUD methods for product.