ADO.NET
Disconnected Models

Disconnected Model

Steps used in the disconnected model:

1. Create the `SqlConnection` object.
   - You'll need the connection string to create it.
2. Create the `SqlDataAdapter` object.
   - You need the SQL SELECT query in string format.
   - And `SqlConnection` object created in the previous step.
3. Create `DataSet` object.
   - It'll be used to store the local copy of the database.
4. Fill the `DataSet` object using `SqlDataAdapter` object.
   - `adapterObj.Fill(dataSetObj);`
5. Use the `DataSet` object to display the result.

SqlDataAdapter Class

- The SqlDataAdapter manages connections with the data source and gives us the disconnected behavior.
- The SqlDataAdapter opens a connection only when required and closes it as soon as it has performed its task.
- For example: the SqlDataAdapter performs the following tasks when filling a DataSet with data:
  1. Open connection.
  2. Store data into DataSet.
  3. Close connection.

- And it performs the following actions when updating the data source with DataSet changes:
  - Open connection.
  - Save changes from DataSet to the data source.
  - Close connection.

`SqlDataAdapter` Class

- The `SqlDataAdapter` has two very useful methods:

- `Fill`:
  - This method opens the connection, stores data from data source into `DataSet` and then closes the connection.

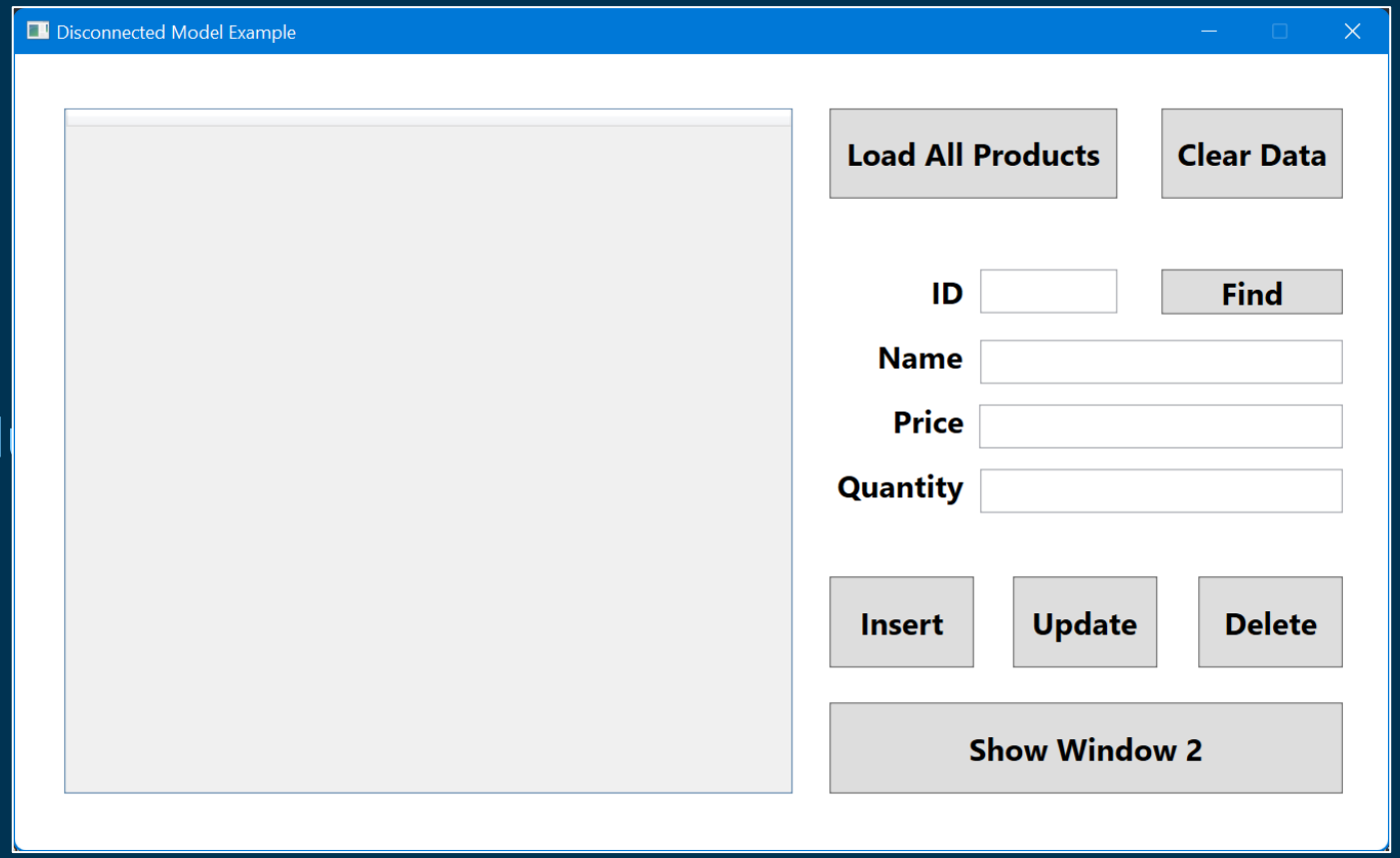
- `Update`:
  - This method opens the connection, saves data from `DataSet` to the data source and then closes the connection.

DataSet Class

- In the disconnected scenario, the data retrieved from the database is stored in a local buffer called DataSet.
- This class is defined in the System.Data namespace.
  - Notice that DataSet is not prefixed with Sql like in SqlDataAdapter and SqlConnection.
  - This is because DataSet is independent of the data source.
  - You can populate DataSet with data from various data sources such as SQL Server, Oracle, Excel files, XML files, text files etc.
- A DataSet object is an in-memory representation of the data and can hold multiple tables.
- It is specially designed to manage data in memory and to support disconnected operations on data.
- A DataSet is a collection of DataTables and DataRelations.
- Each DataTable is a collection of DataColumns, DataRows and Constraints.

Create WPF App

- Create a new WPF app.
- Design a window that looks something like this.
- Give meaningful names to all the controls:
  - DataGrid: `grdProducts`
  - Load All Products: `btnLoadAllProd`
  - Clear Data button: `btnClearData`
  - ID textbox: `txtId`
  - Find button: `btnFind`
  - Name: `txtName`
  - Price: `txtPrice`
  - Quantity: `txtQuantity`
  - Insert button: `btnInsert`
  - Update button: `btnUpdate`
  - Delete button: `btnDelete`
  - Show Window 2 button: `btnShowWindow2`



Create Data Class

- Add a class and name it something like Data.
- Add a private static field which stores the connection string.
- Add a public static property that returns the connection string.

```
public class Data
{
   private static string connStr =  @"Data Source=(LocalDB)\MSSQLLocalDB;
                                    Initial Catalog=Northwind;
                                    Integrated Security=True";

   public static string ConnectionString { get => connStr; }
}
```

Install NuGet Package

- We need to use the NuGet Package Manager to install a package.
- Right-click on project name in the Solution Explorer and select Manage NuGet Packages….

- Click on the Browse tab and search for the following package:
  - Microsoft.Data.SqlClient

- Install this package.
  - Accept any license agreement that might pop up.

Getting All The Products

- Go into `Data` class.
- Include the namespaces at the top:
  - `using Microsoft.Data.SqlClient;`
  - `using Microsoft.Data;`

- To fetch all the products, create a method `GetAllProducts()`.
- Set its return type to `DataTable`.

```
public DataTable GetAllProducts()
{

}
```

SqlConnection Object

- Within the `GetAllProducts` method, create the `SqlConnection` object.
  - Requires the connection string.

```
public DataTable GetAllProducts()
{
  SqlConnection conn = new SqlConnection(ConnectionString);
}
```

SqlDataAdapter Object

- Create the `SqlDataAdapter` object.
  - Need the SQL SELECT query in string format.
    - SELECT query is required because first it needs to fetch data and store it within `DataSet`.
  - Need the `SqlConnection` object created in the previous step.

```
public DataTable GetAllProducts()
{
        SqlConnection conn = new SqlConnection(ConnectionString);

        string query = "Select ProductID, ProductName, UnitPrice,
                         UnitsInStock from Products";
        SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
}
```

DataSet Object

- Create the DataSet object.

```
public DataTable GetAllProducts()
{
        SqlConnection conn = new SqlConnection(ConnectionString);

        string query = "Select ProductID, ProductName, UnitPrice,
                         UnitsInStock from Products";
        SqlDataAdapter adapter = new SqlDataAdapter(query, conn);

        DataSet ds = new DataSet();
}
```

Fill the DataSet

- Call the SqlDataAdapter's Fill method to populate the DataSet.
- The Fill method takes two parameters: a DataSet and a table name.
  - The second parameter is the name of the table that will be created in the DataSet.
  - You can name the table anything you want. Typically, I'll give it the same name as the database table.

```
public DataTable GetAllProducts()
{
        SqlConnection conn = new SqlConnection(ConnectionString);

        string query = "Select ProductID, ProductName, UnitPrice,
                        UnitsInStock from Products";
        SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
        DataSet ds = new DataSet();

        adapter.Fill(ds, "Products");
}
```

Return the DataTable

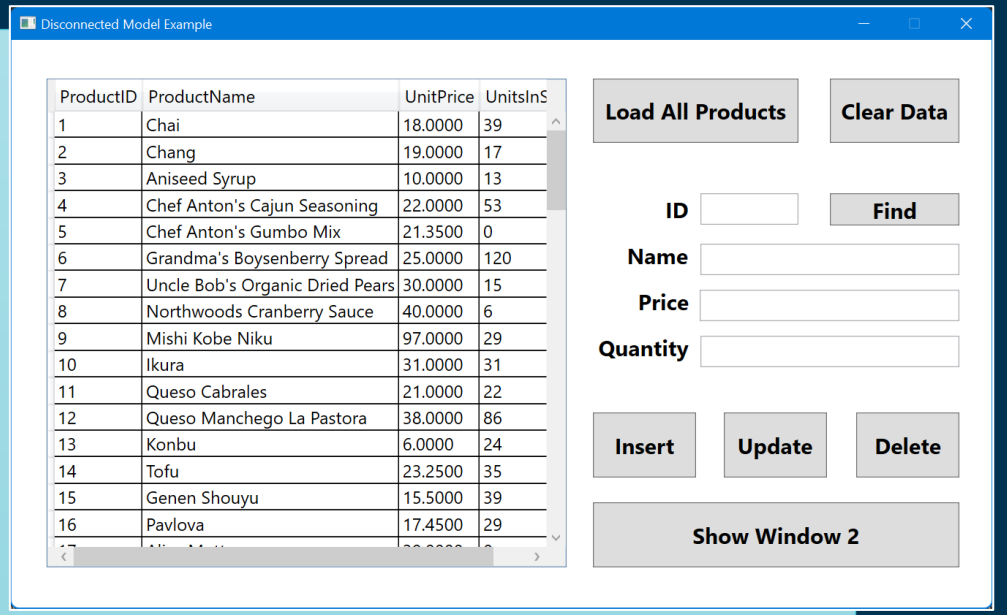
- Then, read and assign the table from the DataSet to DataTable.
- And return the DataTable.

```
public DataTable GetAllProducts()
{
        SqlConnection conn = new SqlConnection(ConnectionString);

        string query = "Select ProductID, ProductName, UnitPrice,
                         UnitsInStock from Products";
        SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
        DataSet ds = new DataSet();

        adapter.Fill(ds, "Products");

        DataTable tblProducts = ds.Tables["Products"];
        return tblProducts;
}
```

Implement the Load All Products Click Event

- Now, add the click event handler to the Load All Products button.
- In the Code View, instantiate the `Data` class.
- And call the `GetAllProducts` method in the Load All Products click event.

```
public partial class MainWindow : Window
{
    private Data data = new Data();

    public MainWindow()
    {
        InitializeComponent();
    }

    private void btnLoadAllProducts_Click(object sender, RoutedEventArgs e)
    {
        grdProducts.ItemsSource = data.GetAllProducts().DefaultView;
    }
}
```



`DataSet` having Multiple Tables

- `DataSet` is the in-memory representation of database.
- Just like how there can be multiple tables in a database, `DataSet` can also have multiple tables.
- When you fill `DataSet` with multiple tables, the first table is named `Table`, the second table is `Table1`, then `Table2`, and so on.
- `DataSet` does not maintain table names.
  - You have to manually name them.
- Also, the data types of columns are not maintained.
- Each column is of type `object`.
- This type of `DataSet` is called Untyped DataSet.
- Like real databases, tables in a `DataSet` can have relations as well.

CRUD Operations in DataSet

- Lets look at an example of CRUD operations in DataSet.
- With DataSet, we only need the SELECT SQL query to fetch the data for the first time.
- There is no need to write INSERT, UPDATE or DELETE queries.
- These queries are auto-generated with the help of SqlCommandBuilder.
- SqlCommandBuilder needs the SqlDataAdapter object to be passed to its constructor.
- That SqlDataAdapter object already contains the SELECT query.
- So, based on that SELECT query information, SqlCommandBuilder generates the INSERT, UPDATE and DELETE queries.

CRUD Operations in DataSet

- For this example, we'll work on the table Products and the user can perform the following operations:

1. Get all Products
2. Get Product by ID
3. Insert Product
4. Update Product
5. Delete Product

- This will be the SELECT query:
  - `SELECT ProductID, ProductName, UnitPrice, UnitsInStock FROM Products;`

CRUD Operations in DataSet

- The following classes will be used in this example:

- SqlConnection:
  - Used to create the connection.

- SqlDataAdapter:
  - Used to fill and update the DataSet.

- SqlCommandBuilder:
  - Used to auto-generate the INSERT, UPDATE, DELETE queries.

- DataSet:
  - Used for in-memory storage and representation of data.

- DataTable:
  - Used to read the table from DataSet.

CRUD Operations in `DataSet`

- First, create a class that'll perform these actions.
- Ideally, this class should be named `DataAccess` or something similar.
- But for now, I'll name it `CrudOperationsInDataSet`.
- Declare the previous mentioned class objects as fields.
- And use the constructor to initialize these objects.

CRUD Operations in DataSet

- Make a method `FillDataSet` that'll fetch the fresh copy of the data from the database.
- Since, this `DataSet` is Untyped, I'll also define the primary key column.
- Call this method in the constructor of the `CrudOperationsInDataSet` class.

```
private void FillDataSet()
{
    // reset the dataset
    ds = new DataSet();

    adapter.Fill(ds);
    tblProducts = ds.Tables[0];

    // define primary key
    DataColumn[] pk = new DataColumn[1];
    pk[0] = tblProducts.Columns["ProductID"];
    pk[0].AutoIncrement = true;
    tblProducts.PrimaryKey = pk;
}
```

CRUD Operations in DataSet

- A method called `GetAllProducts()` to display all the products fetched from the database.
- This method first calls the `FillDataSet` method.
- Then simply returns the `DataTable`.

```
public DataTable GetAllProducts()
{
    FillDataSet();
    return tblProducts;
}
```

CRUD Operations in DataSet

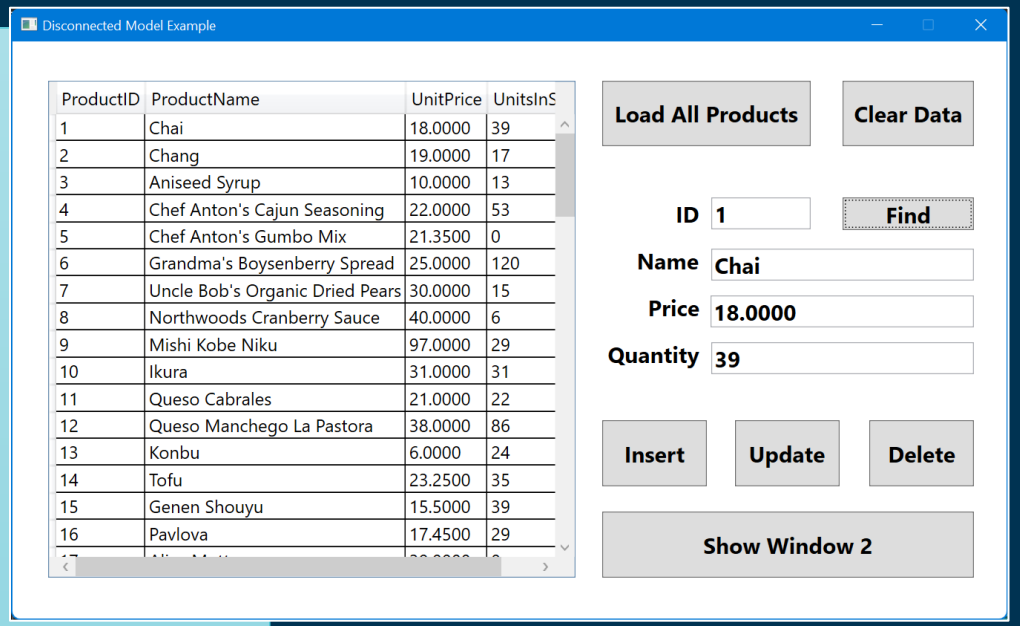
- A method called `GetProductById(int id)` to get a single product found by its primary key.
- `Find` method can be used to get a single row based on primary key of the `DataTable`.
- Then return the fetched row.

```
public DataRow GetProductById(int id)
{
    // find a row based on its primary key
    DataRow row = tblProducts.Rows.Find(id);

    return row;
}
```

CRUD Operations in DataSet

- Implement the Find button's click event.
- Read the ID from the textbox and pass it to the `GetProductById`.
- If a row is returned, display the data in the respective textboxes.

```
private void btnFind_Click(object sender,
                           RoutedEventArgs e)
{
    int id = int.Parse(txtId.Text);
    DataRow row = crud.GetProductById(id);

    if (row != null)
    {
        txtName.Text = row["ProductName"].ToString();
        txtPrice.Text = row["UnitPrice"].ToString();
        txtQuantity.Text = row["UnitsInStock"].ToString();
    }
    else
        MessageBox.Show("Invalid Product ID. Please try again.");
}
```



CRUD Operations in DataSet

- A method called `InsertProduct(string name, decimal price, short quantity)` to insert a new product.
- First, create a new row in the `DataTable`.
- Assign the values to this new row.
- Notice that ID isn't required because it is set as Identity (Auto-increment).
- Then, add this new row to the `Rows` collection of the `DataTable`.

- Now, read the INSERT query from the `SqlCommandBuilder` object and assign it to the `InsertCommand` property of the `SqlDataAdapter` object.

```
adapter.InsertCommand = cmdBuilder.GetInsertCommand();
```

- Call the `SqlDataAdapter`'s `Update` method which can take in the `DataTable` object to commit the changes to the database.
- Code on next slide.

CRUD Operations in DataSet

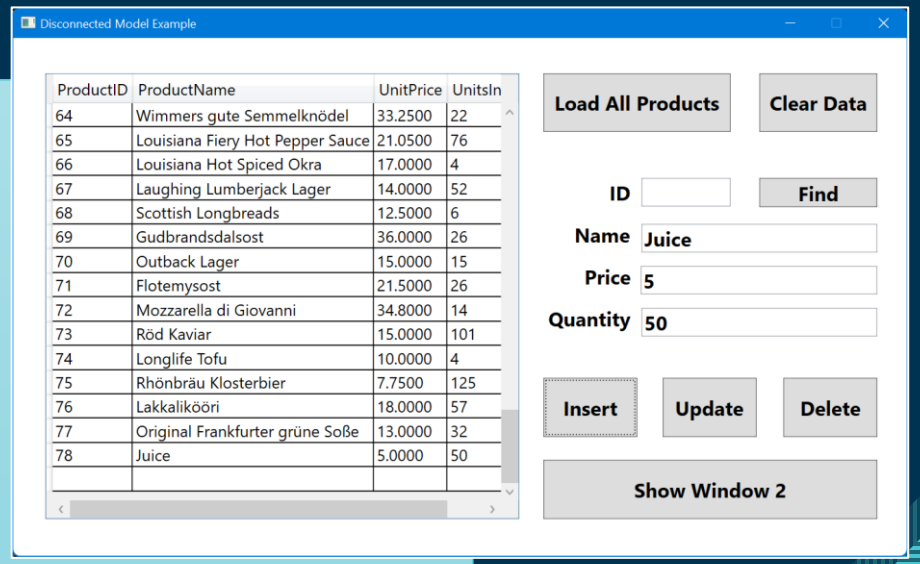
- A method called `InsertProduct(string name, decimal price, short quantity)` to insert a new product.

```
public void InsertProduct(string name, decimal price, short quantity)
{
    DataRow newRow = tblProducts.NewRow(); // create a new row

    // assign the values from variables to the new row
    newRow["ProductName"] = name;
    newRow["UnitPrice"] = price;
    newRow["UnitsInStock"] = quantity;

    tblProducts.Rows.Add(newRow); // add the new row to the Rows collection

    adapter.InsertCommand = cmdBuilder.GetInsertCommand();
    adapter.Update(tblProducts);
}
```

CRUD Operations in DataSet

- Implement the Insert button's click event.
- Read the values from the textboxes and pass them to the `InsertProduct` method.
- Then refresh (or reload) the `DataGrid` using the `GetAllProducts` method.

```
private void btnInsert_Click(object sender,
                             RoutedEventArgs e)
{
        string name = txtName.Text;
        decimal price = decimal.Parse(txtPrice.Text);
        short quantity = short.Parse(txtQuantity.Text);

        crud.InsertProduct(name, price, quantity);
        grdProducts.ItemsSource = crud.GetAllProducts().DefaultView;
}
```



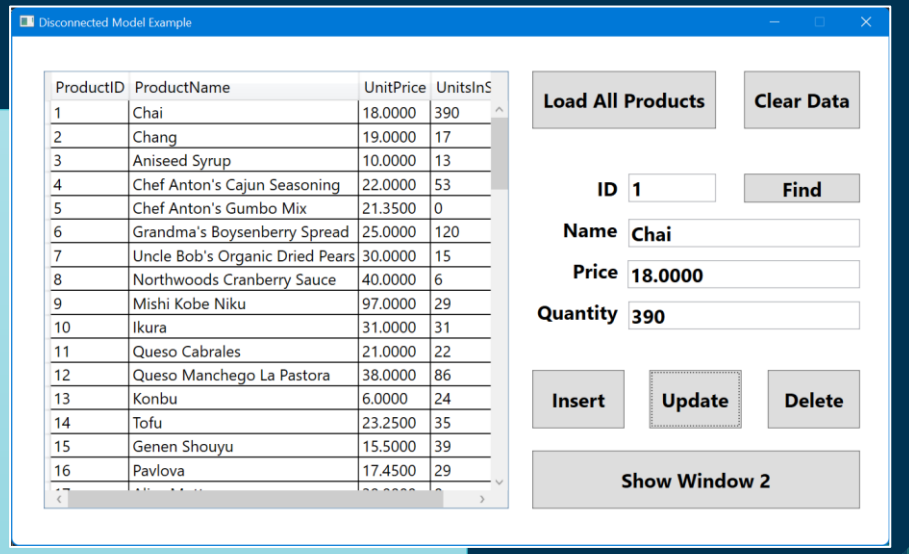
CRUD Operations in DataSet

- A method called `UpdateProduct(int id, string name, decimal price, short quantity)` to update a product.
- While updating, you need the ID of the product so that only that product is updated.
- Remaining code is almost similar to `InsertProduct`.

```
public void UpdateProduct(int id, string name, decimal price, short quantity)
{
    DataRow row = tblProducts.Rows.Find(id);

    row["ProductName"] = name;
    row["UnitPrice"] = price;
    row["UnitsInStock"] = quantity;

    adapter.UpdateCommand = cmdBuilder.GetUpdateCommand();
    adapter.Update(tblProducts);
}
```

CRUD Operations in DataSet

- Implement the Update button's click event.
- Read the values from the textboxes and pass them to the `UpdateProduct` method.
- Then refresh (or reload) the `DataGrid` using the `GetAllProducts` method.




```
private void btnUpdate_Click(object sender,
                             RoutedEventArgs e)
{
        int id = int.Parse(txtId.Text);
        string name = txtName.Text;
        decimal price = decimal.Parse(txtPrice.Text);
        short quantity = short.Parse(txtQuantity.Text);

        crud.UpdateProduct(id, name, price, quantity);
        grdProducts.ItemsSource = crud.GetAllProducts().DefaultView;
}
```

CRUD Operations in DataSet

- A method called `DeleteProduct(int id)` to delete a product.
- While deleting, you only need the ID of the product.
- **Note:**
  - With Northwind database, you can only delete the products you've inserted yourself.
  - The built-in products will throw an exception because of the foreign key constraint with other tables.

```
public void DeleteProduct(int id)
{
    DataRow row = tblProducts.Rows.Find(id);

    row.Delete();

    adapter.DeleteCommand = cmdBuilder.GetDeleteCommand();
    adapter.Update(tblProducts);
}
```

CRUD Operations in DataSet

- Implement the Delete button's click event.
- Read the ID from the textbox and pass them to the DeleteProduct method.
- Then refresh (or reload) the DataGrid using the GetAllProducts method.

```
private void btnDelete_Click(object sender, RoutedEventArgs e)
{
        int id = int.Parse(txtId.Text);

        crud.DeleteProduct(id);
        grdProducts.ItemsSource = crud.GetAllProducts().DefaultView;
}
```

Do It Yourself!

- Create and design a WPF app where the user can perform the following actions:
  1. Get all Employees
  2. Search Employee by first name
  3. Search Employee by last name

- Use Employee table in Northwind database.
- Allow partial matches when searching by first name or last name.
  - For example: If searching by first name user enters **an**, it should display all employees that contain **an** in their name.
  - Hint: Use SQL's LIKE operator.

- Use disconnected model.

Do It Yourself!

- In the Northwind database, CategoryID of Categories table is a foreign key in Products table.
- Refer to the following diagram:
  - https://docs.yugabyte.com/images/sample-data/northwind/northwind-er-diagram.png
- Create and design a WPF app where the user can get all products filtered by a category.
- If the user enters CategoryID, fetch all the products based on that CategoryID.
- Use either connected or disconnected model.

Do It Yourself!

- Use the Categories and Products tables in the Northwind database.
- Fetch all records from the Products table and display the following columns:
  - ProductID ProductName CategoryName
- Note that CategoryName doesn't belong to Products table, but CategoryID does.
- Use SQL's INNER JOIN to replace CategoryID with CategoryName.
- Refer to the following diagram:
  - https://docs.yugabyte.com/images/sample-data/northwind/northwind-er-diagram.png
- Use either connected or disconnected model.

Thank You

Copyright

The materials provided in class and in SLATE are protected by copyright. They are intended for the personal, educational uses of students in this course and should not be shared externally or on websites such as Chegg, Course Hero or OneClass. Unauthorized distribution may result in copyright infringement and violation of Sheridan policies.

References

Material has been taken from:

- Basics of ADO.NET:
  - https://www.c-sharpcorner.com/UploadFile/18fc30/understanding-the-basics-of-ado-net/
- Professional C# 7 and .NET Core 2.0:
  - https://learning.oreilly.com/library/view/professional-c-7/9781119449270/c25.xhtml
- Working with Disconnected Data – The DataSet and SqlDataAdapter:
  - https://csharp-station.com/Tutorial/AdoDotNet/Lesson05