# ADO.NET Disconnected Layer

CSIS 3540

Client Server Systems

Class 08

# Topics

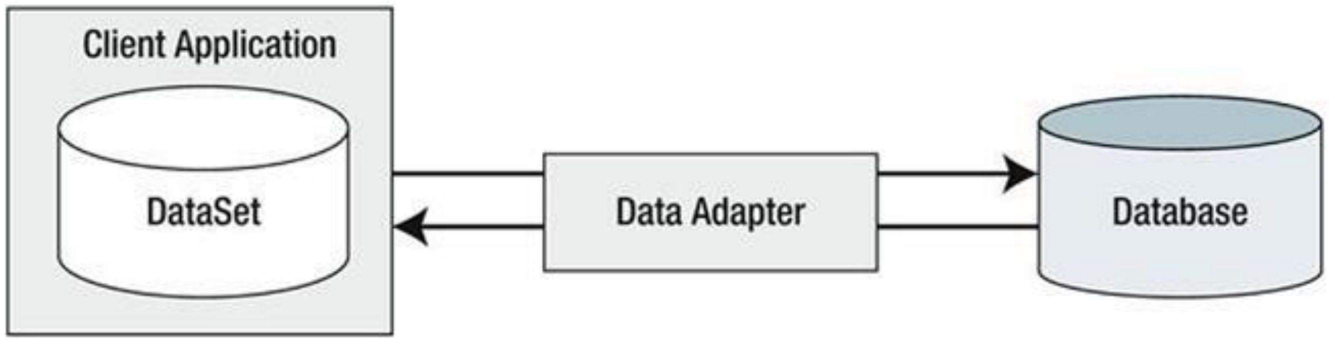
- Overview
- DataSet
- DataTables, DataColumns, DataRows
- Binding DataTable objects to DataGridView
- Data Adapters
- Multiple tables and relationships
- Using Windows Forms DB designer tools
- LINQ and DataSet

# Overview of Disconnected Layer

- Build an in-memory model of the database
- Manipulate the database
- Sync to actual stored database
- DB can be different forms
  - XML file
  - SQL Server
  - …

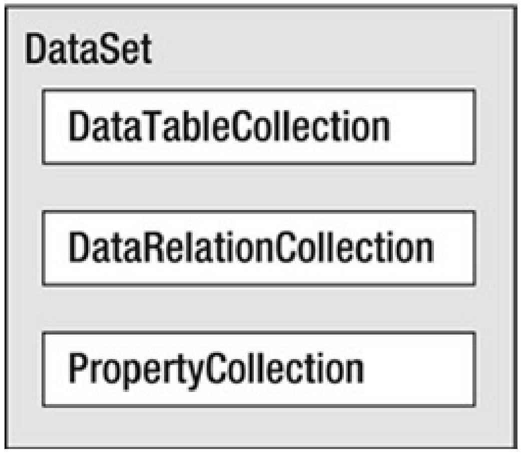
## DataAdapter and DataSet

- DataAdapter object uses DataSet objects
- Think of DataSet as an in memory database
- DataAdapter generates SQL code to get records and update database.
- The data adapter object of your data provider handles the database connection automatically.
- In an effort to increase scalability, data adapters keep the connection open for the shortest amount of time possible. After the caller receives the DataSet object, the calling tier is completely disconnected from thedatabase and left with a local copy of the remote data.
- The caller is free to insert, delete, or update rows from a given DataTable, **but the physical database is not updated until the caller explicitly passes a DataTable in the DataSet to the data adapter for updating.**



# DataSet

- Basically, an in-memory database schema consisting of the following:
  - DataTableCollection
    - A set of database tables
  - DataRelationCollection
    - A set of relations, equivalent to foreign key constraints
  - PropertyCollection
    - metadata



# DataSet key properties

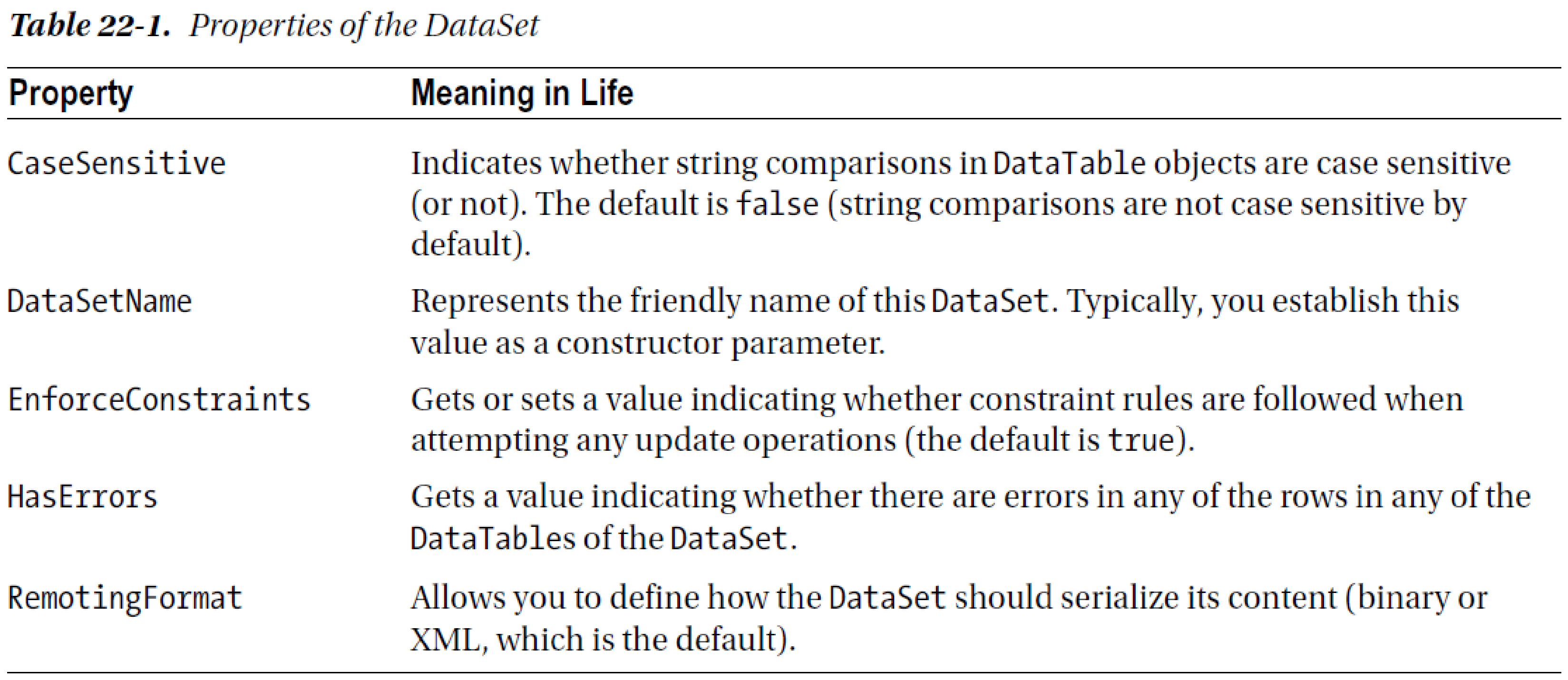


*Table 22-1. Properties of the DataSet*

| Property | Meaning in Life |
|---|---|
| `CaseSensitive` | Indicates whether string comparisons in `DataTable` objects are case sensitive (or not). The default is `false` (string comparisons are not case sensitive by default). |
| `DataSetName` | Represents the friendly name of this `DataSet`. Typically, you establish this value as a constructor parameter. |
| `EnforceConstraints` | Gets or sets a value indicating whether constraint rules are followed when attempting any update operations (the default is `true`). |
| `HasErrors` | Gets a value indicating whether there are errors in any of the rows in any of the `DataTables` of the `DataSet`. |
| `RemotingFormat` | Allows you to define how the `DataSet` should serialize its content (binary or XML, which is the default). |

# DataSet key methods

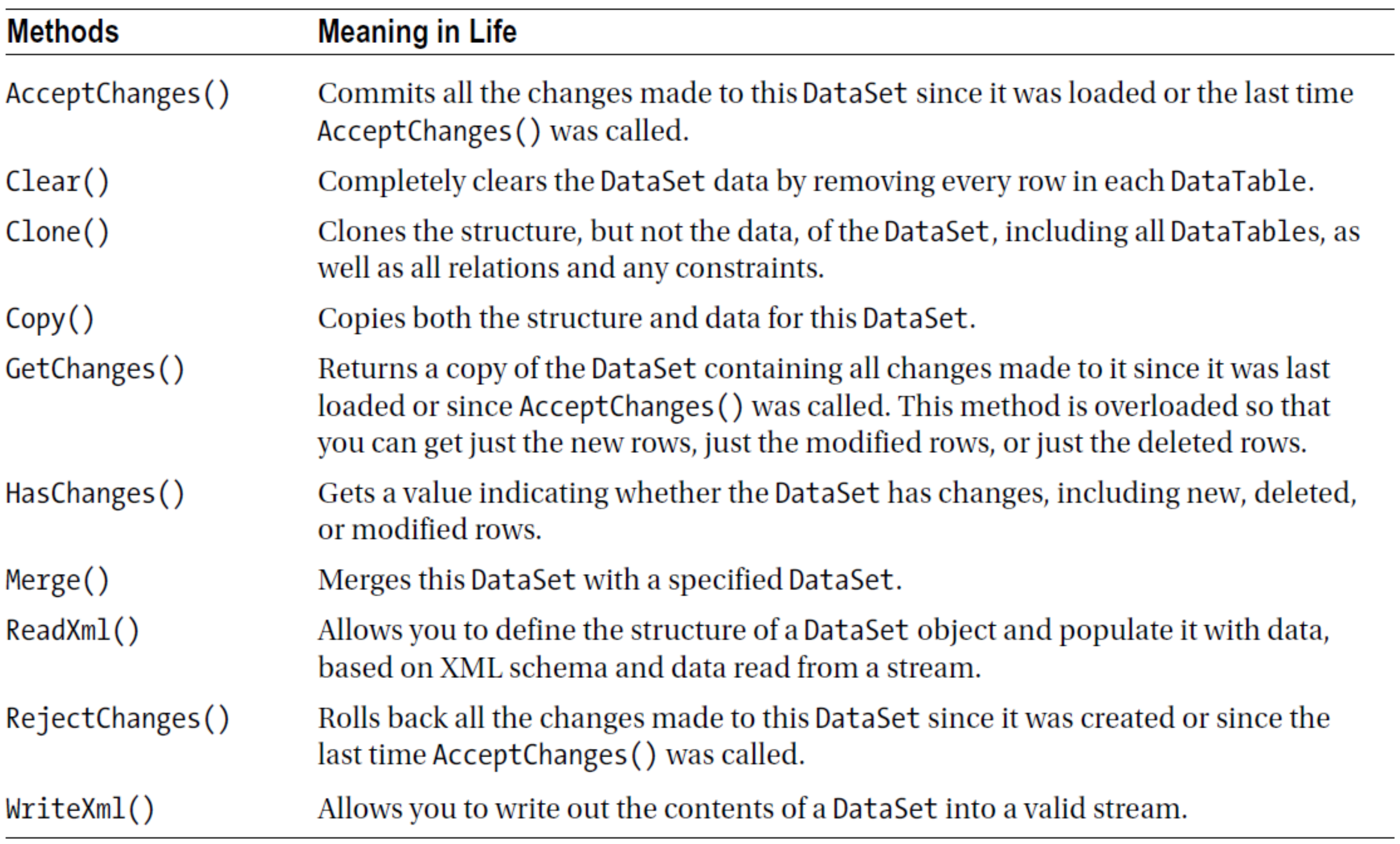


***Table 22-2.*** *Select Methods of the DataSet*

| Methods | Meaning in Life |
|---|---|
| `AcceptChanges()` | Commits all the changes made to this `DataSet` since it was loaded or the last time `AcceptChanges()` was called. |
| `Clear()` | Completely clears the `DataSet` data by removing every row in each `DataTable`. |
| `Clone()` | Clones the structure, but not the data, of the `DataSet`, including all `DataTables`, as well as all relations and any constraints. |
| `Copy()` | Copies both the structure and data for this `DataSet`. |
| `GetChanges()` | Returns a copy of the `DataSet` containing all changes made to it since it was last loaded or since `AcceptChanges()` was called. This method is overloaded so that you can get just the new rows, just the modified rows, or just the deleted rows. |
| `HasChanges()` | Gets a value indicating whether the `DataSet` has changes, including new, deleted, or modified rows. |
| `Merge()` | Merges this `DataSet` with a specified `DataSet`. |
| `ReadXml()` | Allows you to define the structure of a `DataSet` object and populate it with data, based on XML schema and data read from a stream. |
| `RejectChanges()` | Rolls back all the changes made to this `DataSet` since it was created or since the last time `AcceptChanges()` was called. |
| `WriteXml()` | Allows you to write out the contents of a `DataSet` into a valid stream. |

## Creating an in-memory database with ADO

- Create a DataSet
- Create DataColumns for each DataTable
- Create DataRelations
- Add data
- Done!
- For all examples, see **SimpleDataSet**

## Create a DataSet

- Create it with a friendly name
- Set metadata if need be
  - Note that this could be used for Windows Forms labels etc later
- DataSet has DataTables, which consist of DataColumns and DataRows

```
// Create the DataSet object and add a few properties.
var carsInventoryDS = new DataSet("Car Inventory");

carsInventoryDS.ExtendedProperties["TimeStamp"] = DateTime.Now;
carsInventoryDS.ExtendedProperties["DataSetID"] = Guid.NewGuid();
carsInventoryDS.ExtendedProperties["Company"] = "Mikko’s Hot Tub Super Store";
```

# DataColumns for a DataTable

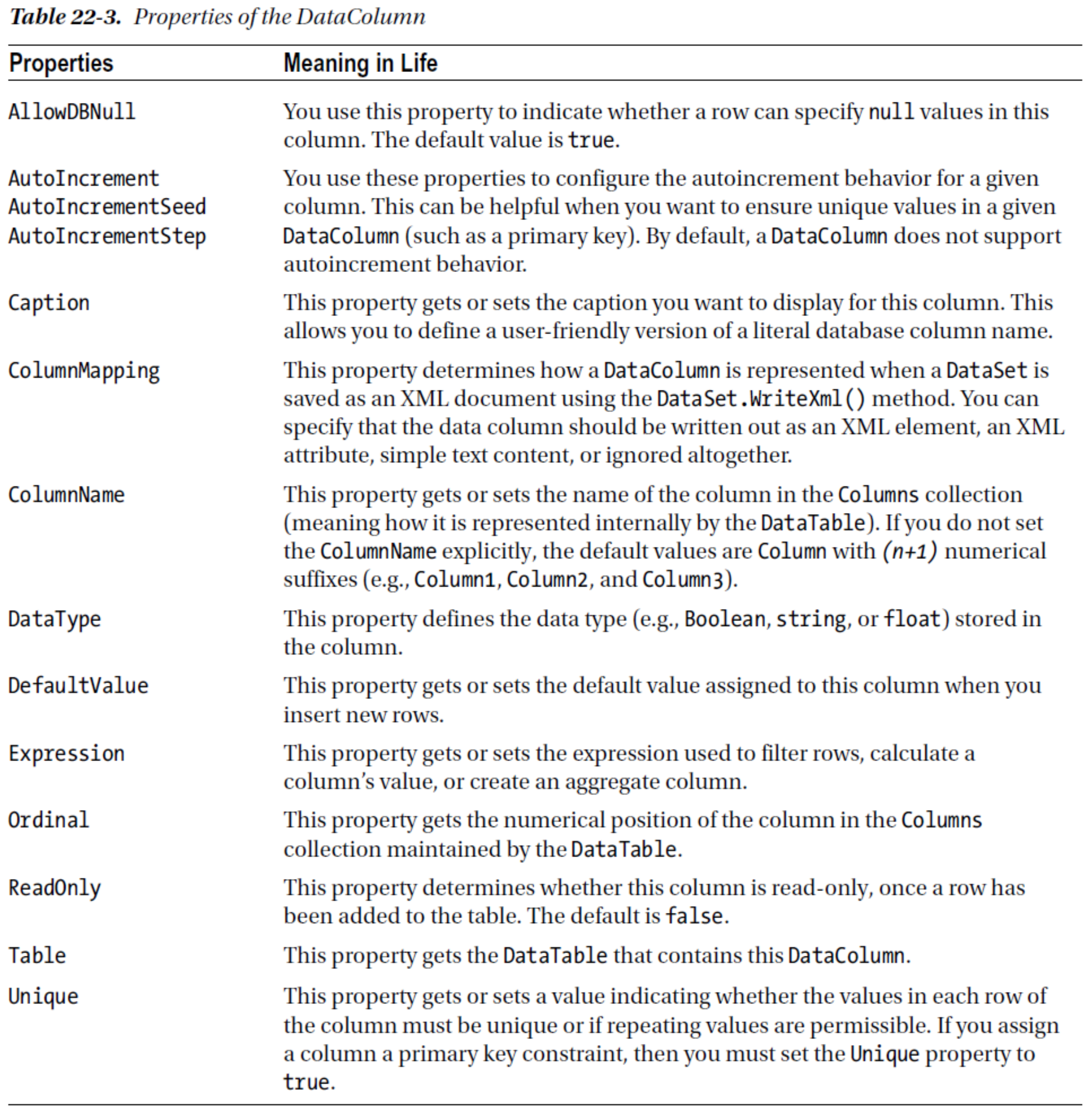


***Table 22-3.*** *Properties of the DataColumn*

| Properties | Meaning in Life |
|---|---|
| `AllowDBNull` | You use this property to indicate whether a row can specify `null` values in this column. The default value is `true`. |
| `AutoIncrement`<br>`AutoIncrementSeed`<br>`AutoIncrementStep` | You use these properties to configure the autoincrement behavior for a given column. This can be helpful when you want to ensure unique values in a given `DataColumn` (such as a primary key). By default, a `DataColumn` does not support autoincrement behavior. |
| `Caption` | This property gets or sets the caption you want to display for this column. This allows you to define a user-friendly version of a literal database column name. |
| `ColumnMapping` | This property determines how a `DataColumn` is represented when a `DataSet` is saved as an XML document using the `DataSet.WriteXml()` method. You can specify that the data column should be written out as an XML element, an XML attribute, simple text content, or ignored altogether. |
| `ColumnName` | This property gets or sets the name of the column in the `Columns` collection (meaning how it is represented internally by the `DataTable`). If you do not set the `ColumnName` explicitly, the default values are `Column` with *(n+1)* numerical suffixes (e.g., `Column1`, `Column2`, and `Column3`). |
| `DataType` | This property defines the data type (e.g., `Boolean`, `string`, or `float`) stored in the column. |
| `DefaultValue` | This property gets or sets the default value assigned to this column when you insert new rows. |
| `Expression` | This property gets or sets the expression used to filter rows, calculate a column's value, or create an aggregate column. |
| `Ordinal` | This property gets the numerical position of the column in the `Columns` collection maintained by the `DataTable`. |
| `ReadOnly` | This property determines whether this column is read-only, once a row has been added to the table. The default is `false`. |
| `Table` | This property gets the `DataTable` that contains this `DataColumn`. |
| `Unique` | This property gets or sets a value indicating whether the values in each row of the column must be unique or if repeating values are permissible. If you assign a column a primary key constraint, then you must set the `Unique` property to `true`. |

## Create DataColumn

- Note that primary key does not have nulls, is unique, and has autoincrement
    - Set with properties
- Every column is constructed with a column name and data type
    - Note use of typeof(), indicating basic SQL types

```
var carIDColumn = new DataColumn("CarID", typeof(int))
{
    Caption = "Car ID",
    ReadOnly = true,
    AllowDBNull = false,
    Unique = true,
    AutoIncrement = true,
    AutoIncrementSeed = 1,
    AutoIncrementStep = 1
};

var carMakeColumn = new DataColumn("Make", typeof(string));
var carColorColumn = new DataColumn("Color", typeof(string));

// this shows how to use the Caption property for a column

var carNameColumn = new DataColumn("Name", typeof(string))
{
    Caption = "Car Name"
};
```

## Create DataTable and add columns

- Create a DataTable with a name identifier
- Add columns
  - Note use of AddRange, but could do this in order using Add() as well.

```
// Now add DataColumns to a DataTable.
var inventoryTable = new DataTable("Inventory");
inventoryTable.Columns.AddRange(new[]
{carIDColumn, carMakeColumn, carColorColumn, carPetNameColumn});
```

# DataRow

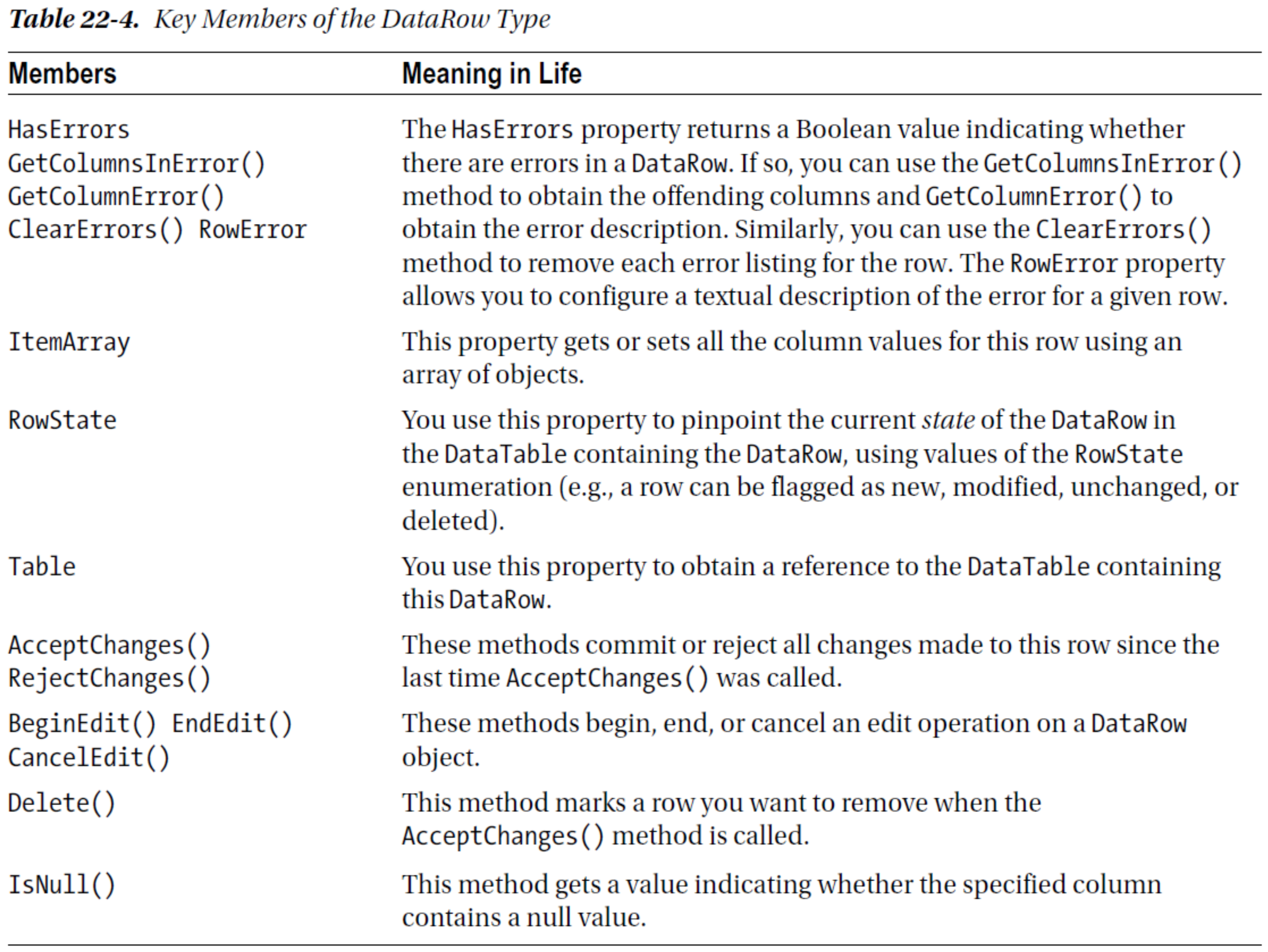


***Table 22-4.*** *Key Members of the DataRow Type*

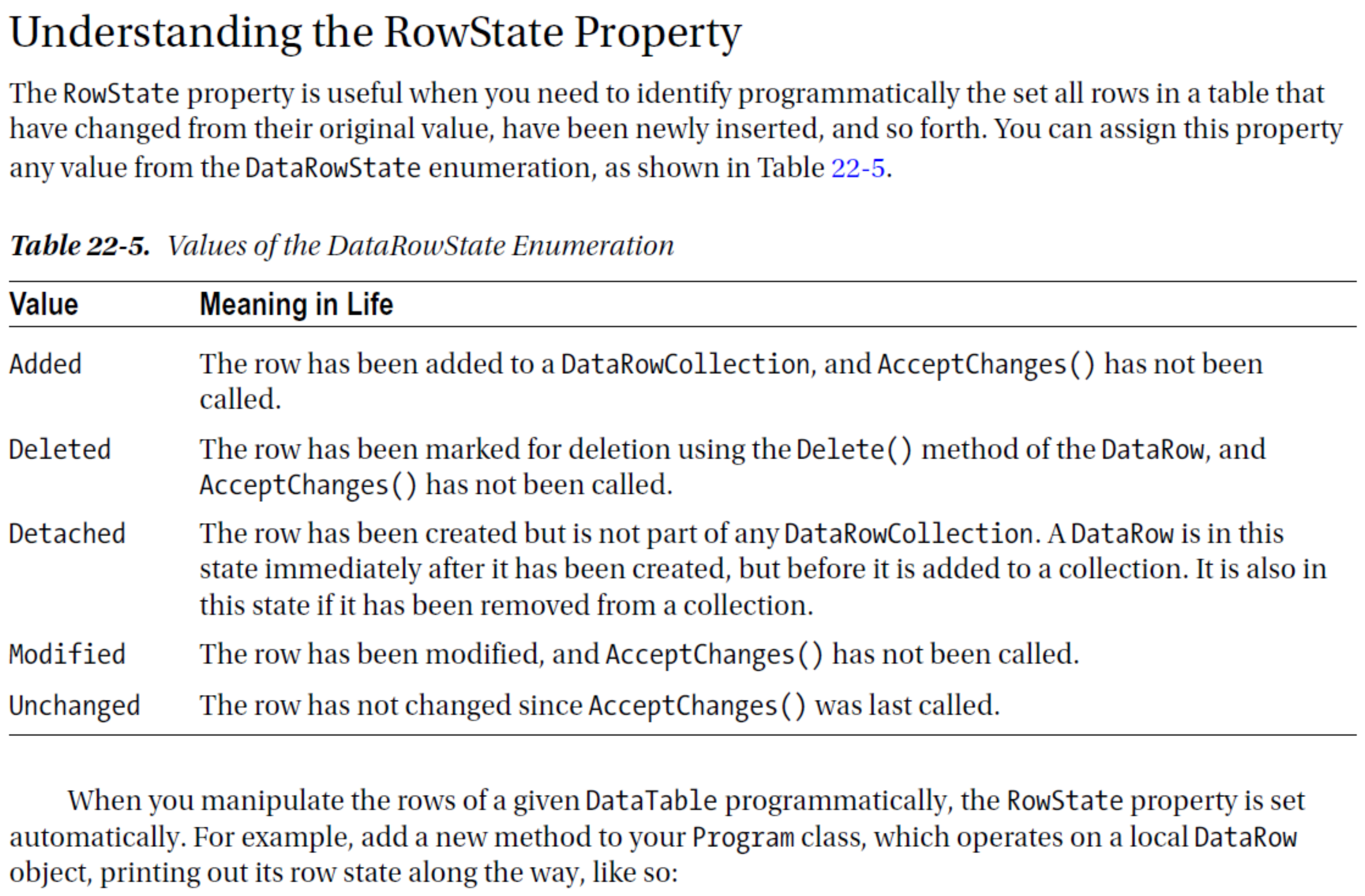
| Members | Meaning in Life |
|---|---|
| `HasErrors` `GetColumnsInError()` `GetColumnError()` `ClearErrors()` `RowError` | The `HasErrors` property returns a Boolean value indicating whether there are errors in a `DataRow`. If so, you can use the `GetColumnsInError()` method to obtain the offending columns and `GetColumnError()` to obtain the error description. Similarly, you can use the `ClearErrors()` method to remove each error listing for the row. The `RowError` property allows you to configure a textual description of the error for a given row. |
| `ItemArray` | This property gets or sets all the column values for this row using an array of objects. |
| `RowState` | You use this property to pinpoint the current *state* of the `DataRow` in the `DataTable` containing the `DataRow`, using values of the `RowState` enumeration (e.g., a row can be flagged as new, modified, unchanged, or deleted). |
| `Table` | You use this property to obtain a reference to the `DataTable` containing this `DataRow`. |
| `AcceptChanges()` `RejectChanges()` | These methods commit or reject all changes made to this row since the last time `AcceptChanges()` was called. |
| `BeginEdit()` `EndEdit()` `CancelEdit()` | These methods begin, end, or cancel an edit operation on a `DataRow` object. |
| `Delete()` | This method marks a row you want to remove when the `AcceptChanges()` method is called. |
| `IsNull()` | This method gets a value indicating whether the specified column contains a null value. |

# Adding data with NewRow()

- Rows do not have a constructor!!
- Instead, a DataTable method NewRow()
- Then add the row using Rows.Add()

```
// Now add a few more rows to the Inventory Table.
           DataRow carRow = inventoryTable.NewRow();
           carRow["Make"] = "BMW";
           carRow["Color"] = "Black";
           carRow["Name"] = "X3";
           inventoryTable.Rows.Add(carRow);

           carRow = inventoryTable.NewRow();
           // Column 0 is the autoincremented ID field,
           // so start at 1.
           carRow[1] = "Saab";
           carRow[2] = "Red";
           carRow[3] = "9000";
           inventoryTable.Rows.Add(carRow);
```

# RowState property



## Understanding the RowState Property

The RowState property is useful when you need to identify programmatically the set all rows in a table that have changed from their original value, have been newly inserted, and so forth. You can assign this property any value from the DataRowState enumeration, as shown in Table 22-5.

***Table 22-5.*** *Values of the DataRowState Enumeration*

| Value | Meaning in Life |
|---|---|
| Added | The row has been added to a DataRowCollection, and AcceptChanges() has not been called. |
| Deleted | The row has been marked for deletion using the Delete() method of the DataRow, and AcceptChanges() has not been called. |
| Detached | The row has been created but is not part of any DataRowCollection. A DataRow is in this state immediately after it has been created, but before it is added to a collection. It is also in this state if it has been removed from a collection. |
| Modified | The row has been modified, and AcceptChanges() has not been called. |
| Unchanged | The row has not changed since AcceptChanges() was last called. |

When you manipulate the rows of a given DataTable programmatically, the RowState property is set automatically. For example, add a new method to your Program class, which operates on a local DataRow object, printing out its row state along the way, like so:

# Editing Rows

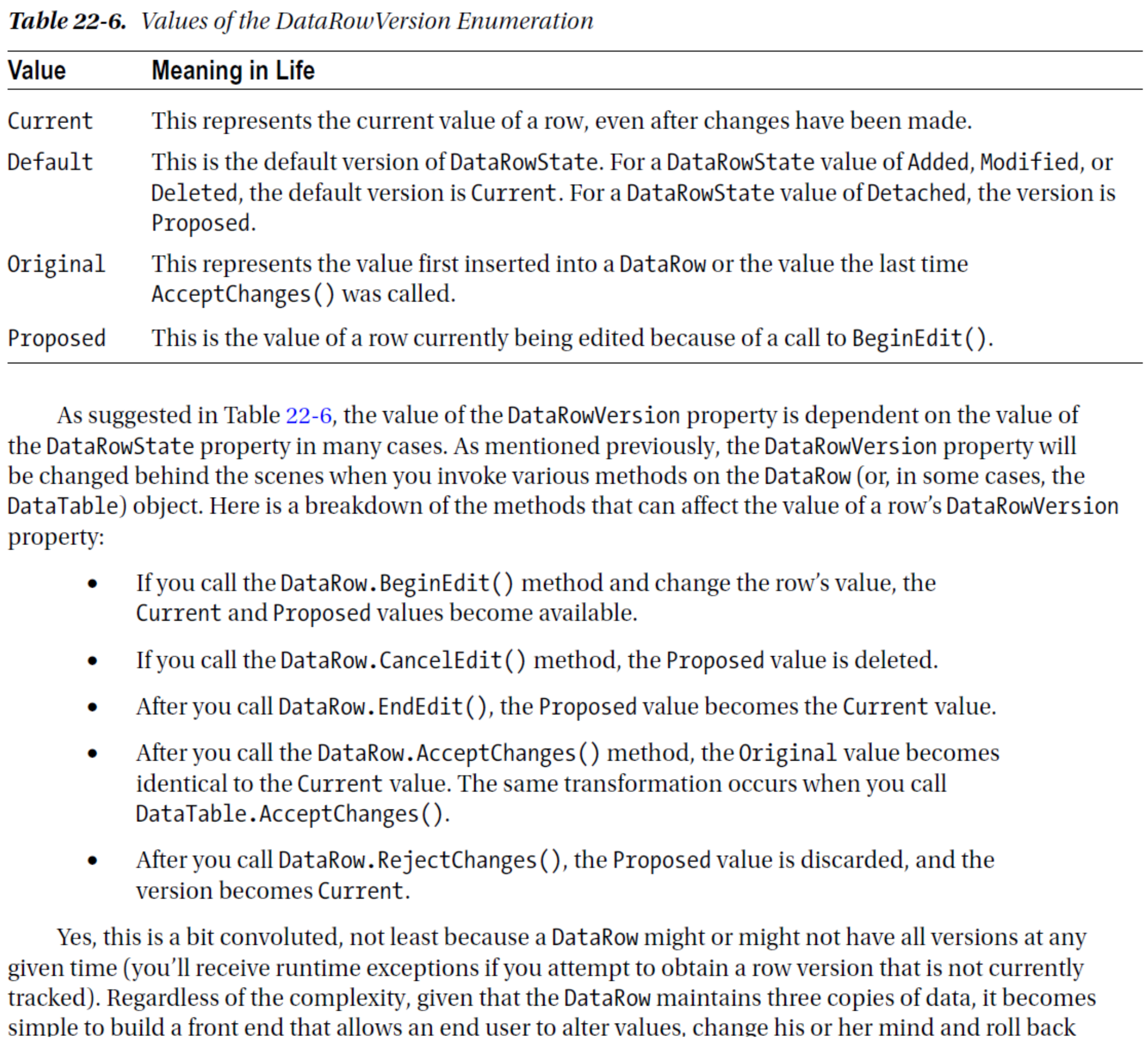


***Table 22-6.*** *Values of the DataRowVersion Enumeration*

| Value | Meaning in Life |
|---|---|
| `Current` | This represents the current value of a row, even after changes have been made. |
| `Default` | This is the default version of `DataRowState`. For a `DataRowState` value of `Added`, `Modified`, or `Deleted`, the default version is `Current`. For a `DataRowState` value of `Detached`, the version is `Proposed`. |
| `Original` | This represents the value first inserted into a `DataRow` or the value the last time `AcceptChanges()` was called. |
| `Proposed` | This is the value of a row currently being edited because of a call to `BeginEdit()`. |

As suggested in Table 22-6, the value of the `DataRowVersion` property is dependent on the value of the `DataRowState` property in many cases. As mentioned previously, the `DataRowVersion` property will be changed behind the scenes when you invoke various methods on the `DataRow` (or, in some cases, the `DataTable`) object. Here is a breakdown of the methods that can affect the value of a row's `DataRowVersion` property:

- If you call the `DataRow.BeginEdit()` method and change the row's value, the `Current` and `Proposed` values become available.
- If you call the `DataRow.CancelEdit()` method, the `Proposed` value is deleted.
- After you call `DataRow.EndEdit()`, the `Proposed` value becomes the `Current` value.
- After you call the `DataRow.AcceptChanges()` method, the `Original` value becomes identical to the `Current` value. The same transformation occurs when you call `DataTable.AcceptChanges()`.
- After you call `DataRow.RejectChanges()`, the `Proposed` value is discarded, and the version becomes `Current`.

Yes, this is a bit convoluted, not least because a `DataRow` might or might not have all versions at any given time (you'll receive runtime exceptions if you attempt to obtain a row version that is not currently tracked). Regardless of the complexity, given that the `DataRow` maintains three copies of data, it becomes simple to build a front end that allows an end user to alter values, change his or her mind and roll back

## Editing Rows

- Once an AcceptChanges() has been invoked, one needs to edit rows
- Get the Row from the table, BeginEdit(), change data, EndEdit(), then AcceptChanges()
- This is basically the same as a Transaction with Commit and Rollback

```
DataRow carRow = ds.Tables["Inventory"].Rows[1];
carRow.BeginEdit();
WriteLine($"BeginEdit row, state: {carRow.RowState}");

carRow["Make"] = "Lincoln";
WriteLine($"Changed value of Make to {carRow["Make"]}, state: {carRow.RowState}");
carRow.EndEdit();
WriteLine($"EndEdit row, state: {carRow.RowState}");
carRow.AcceptChanges();
WriteLine($"AcceptChanges row, state: {carRow.RowState}");
```

# DataTables (setting PrimaryKey)

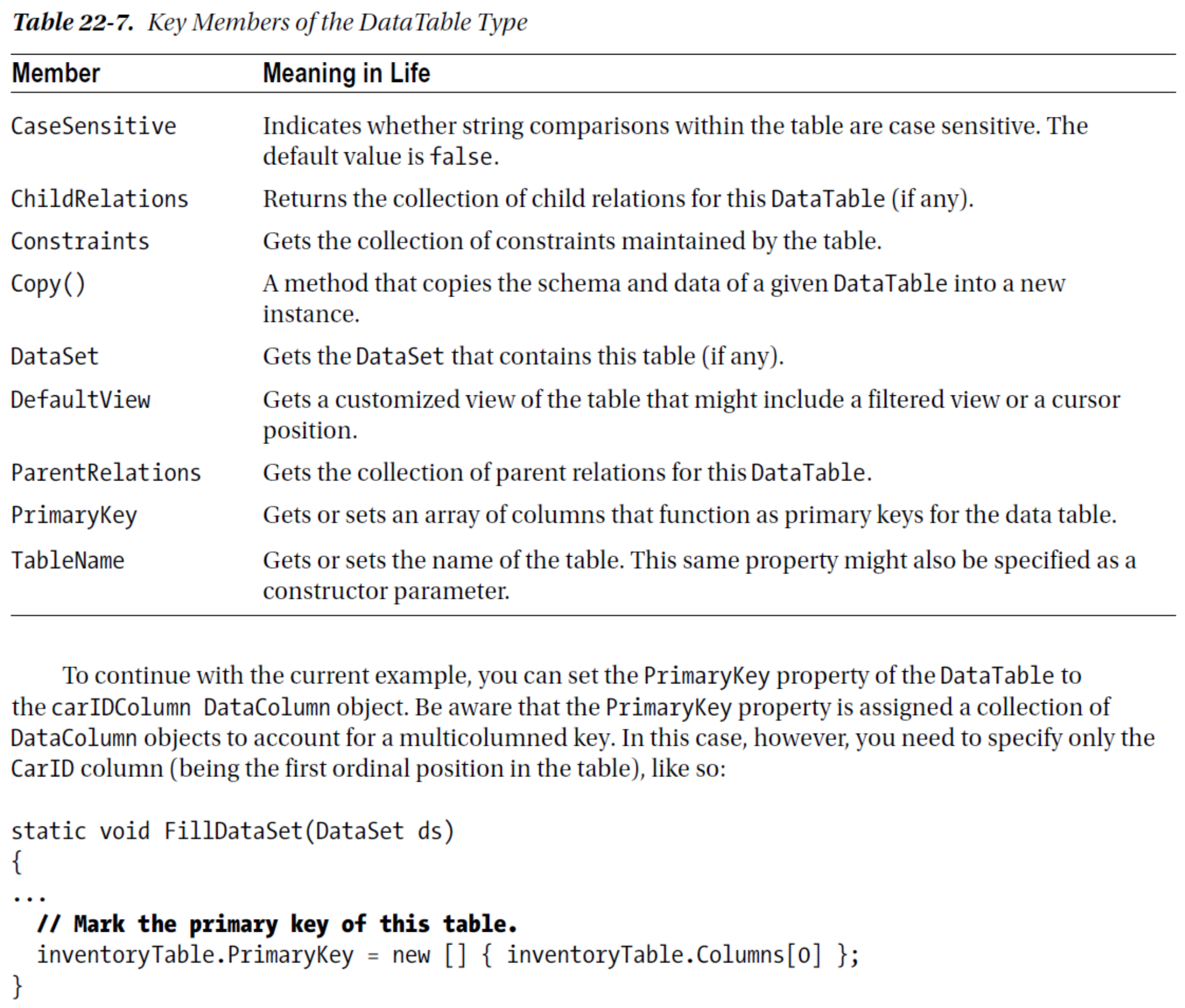


***Table 22-7.*** *Key Members of the DataTable Type*

| Member | Meaning in Life |
|---|---|
| `CaseSensitive` | Indicates whether string comparisons within the table are case sensitive. The default value is `false`. |
| `ChildRelations` | Returns the collection of child relations for this `DataTable` (if any). |
| `Constraints` | Gets the collection of constraints maintained by the table. |
| `Copy()` | A method that copies the schema and data of a given `DataTable` into a new instance. |
| `DataSet` | Gets the `DataSet` that contains this table (if any). |
| `DefaultView` | Gets a customized view of the table that might include a filtered view or a cursor position. |
| `ParentRelations` | Gets the collection of parent relations for this `DataTable`. |
| `PrimaryKey` | Gets or sets an array of columns that function as primary keys for the data table. |
| `TableName` | Gets or sets the name of the table. This same property might also be specified as a constructor parameter. |

To continue with the current example, you can set the `PrimaryKey` property of the `DataTable` to the `carIDColumn DataColumn` object. Be aware that the `PrimaryKey` property is assigned a collection of `DataColumn` objects to account for a multicolumned key. In this case, however, you need to specify only the `CarID` column (being the first ordinal position in the table), like so:

```
static void FillDataSet(DataSet ds)
{
...
  // Mark the primary key of this table.
  inventoryTable.PrimaryKey = new [] { inventoryTable.Columns[0] };
}
```

## Add the table to the DataSet

- Note the AcceptChanges() finishes table creation in memory
- Then add to DataSet

```
// Mark the primary key of this table.
inventoryTable.PrimaryKey = new[] { inventoryTable.Columns[0] };
inventoryTable.AcceptChanges();

// Finally, add our table to the DataSet.
ds.Tables.Add(inventoryTable);
```

## Reading Data (Rows and Columns)

- Columns[i].ColumnName to get the name of the column
- Notice use of Rows[i][j] as a way to iterate through the table

```
        static void DisplayDataSetUsingRowsAndColumns(DataSet dataSet)
        {
            const int columnWidth = 10;

            // Print out each table using rows and columns
            foreach (DataTable dataTable in dataSet.Tables)
            {
                WriteLine($"Rows and Columns => {dataTable.TableName} Table:");

                // Print out the column names.
                for (var column = 0; column < dataTable.Columns.Count; column++)
                {
                    Write($"{dataTable.Columns[column].ColumnName, columnWidth}");
                }
                WriteLine("\n----------------------------------");

                // Print the DataTable.
                for (var row = 0; row < dataTable.Rows.Count; row++)
                {
                    for (var column = 0; column < dataTable.Columns.Count; column++)
                    {
                        Write($"{dataTable.Rows[row][column], columnWidth}");
                    }
                    WriteLine();
                }
            }
        }
```

# Reading Data (DataReader)

- Works just like connected layer DataReader!

```
/// <summary>
/// Display an individual table using DataReader
/// </summary>
/// <param name="dataTable">DataTable</param>
static void DisplayTableUsingDataReader(DataTable dataTable)
{
    const int columnWidth = 10;

    WriteLine($"Data Reader => {dataTable.TableName} Table:");

    // Display the column names.
    for (int column = 0; column < dataTable.Columns.Count; column++)
    {
        Write($"{dataTable.Columns[column].ColumnName.Trim(),columnWidth}");
    }
    WriteLine("\n----------------------------------");

    // Get the DataTableReader type.
    DataTableReader dataTableReader = dataTable.CreateDataReader();

    // The DataTableReader works just like the DataReader.
    while (dataTableReader.Read())
    {
        for (var i = 0; i < dataTableReader.FieldCount; i++)
        {
            Write($"{dataTableReader.GetValue(i).ToString().Trim(),columnWidth}");
        }
        WriteLine();
    }
    dataTableReader.Close();
}
```

## Manipulating RowState

```
var temp = new DataTable("Temp");
temp.Columns.Add(new DataColumn("TempColumn",
typeof(int)));

// RowState = Detached.
var row = temp.NewRow();
WriteLine($"After calling NewRow(): {row.RowState}");

// RowState = Added.
temp.Rows.Add(row);
WriteLine($"After calling Rows.Add(): {row.RowState}");

// RowState = Added.
row["TempColumn"] = 10;
WriteLine($"After first assignment: {row.RowState}");

// RowState = Unchanged.
temp.AcceptChanges();
WriteLine($"After calling AcceptChanges:
{row.RowState}");

// RowState = Modified.
row["TempColumn"] = 11;
WriteLine($"After first assignment: {row.RowState}");

// RowState = Deleted.
temp.Rows[0].Delete();
WriteLine($"After calling Delete: {row.RowState}");
```

```
***** Fun with DataSets *****
After calling NewRow(): Detached
After calling Rows.Add(): Added
After first assignment: Added
After calling AcceptChanges: Unchanged
After first assignment: Modified
After calling Delete: Deleted
```

# Save and Load as XML

- Notice no need for File, etc.
- Very simple
- Resulting files on next slides

```
// Save this DataSet as XML.
carsInventoryDS.WriteXml("carsDataSet.xml");
carsInventoryDS.WriteXmlSchema("carsDataSet.xsd");

// Clear out DataSet.
carsInventoryDS.Clear();

// Load DataSet from XML file.
carsInventoryDS.ReadXml("carsDataSet.xml");
```

# XML Schema

```xml
<?xml version="1.0" standalone="yes"?>
<xs:schema id="Car_x0020_Inventory" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:msprop="urn:schemas-microsoft-com:xml-
msprop">
  <xs:element name="Car_x0020_Inventory" msdata:IsDataSet="true" msdata:UseCurrentLocale="true"
msprop:TimeStamp="02/19/2017 21:19:13" msprop:DataSetID="0472acbc-6b48-40bc-a1db-5e75e3a12e13"
msprop:Company="Mikko’s Hot Tub Super Store">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Inventory">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CarID" msdata:ReadOnly="true" msdata:AutoIncrement="true"
msdata:AutoIncrementSeed="1" msdata:Caption="Car ID" type="xs:int" />
              <xs:element name="Make" type="xs:string" minOccurs="0" />
              <xs:element name="Color" type="xs:string" minOccurs="0" />
              <xs:element name="PetName" msdata:Caption="Pet Name" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    <xs:unique name="Constraint1" msdata:PrimaryKey="true">
      <xs:selector xpath=".//Inventory" />
      <xs:field xpath="CarID" />
    </xs:unique>
  </xs:element>
</xs:schema>
```

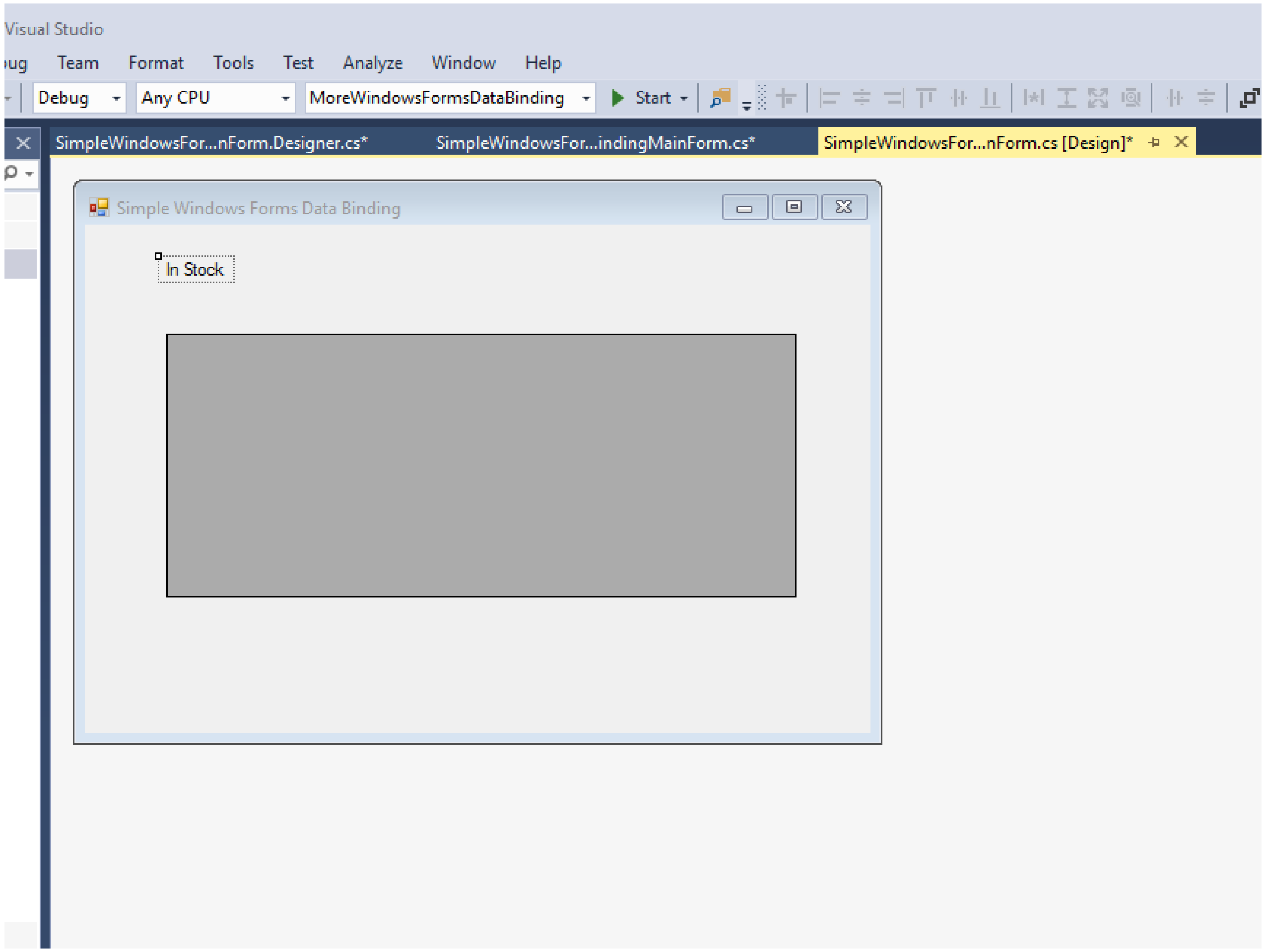
## XML Data

```xml
<?xml version="1.0" standalone="yes"?>
<Car_x0020_Inventory>
  <Inventory>
    <CarID>1</CarID>
    <Make>BMW</Make>
    <Color>Black</Color>
    <PetName>Hamlet</PetName>
  </Inventory>
  <Inventory>
    <CarID>2</CarID>
    <Make>Lincoln</Make>
    <Color>Red</Color>
    <PetName>Sea Breeze</PetName>
  </Inventory>
</Car_x0020_Inventory>
```

## Windows Forms and ADO

- See SimpleWindowsFormsDataBinding
    - And other variations with filters
- Can bind a DataTable to a DataGridView
- As the DataGridView is edited, the Update() and Fill() methods will automatically sync the DB with the control

# Create DataGridView control


Visual Studio
ug Team Format Tools Test Analyze Window Help
Debug
Any CPU
MoreWindowsFormsDataBinding
Start
SimpleWindowsFor...nForm.Designer.cs*
SimpleWindowsFor...indingMainForm.cs*
SimpleWindowsFor...nForm.cs [Design]*
Simple Windows Forms Data Binding
In Stock

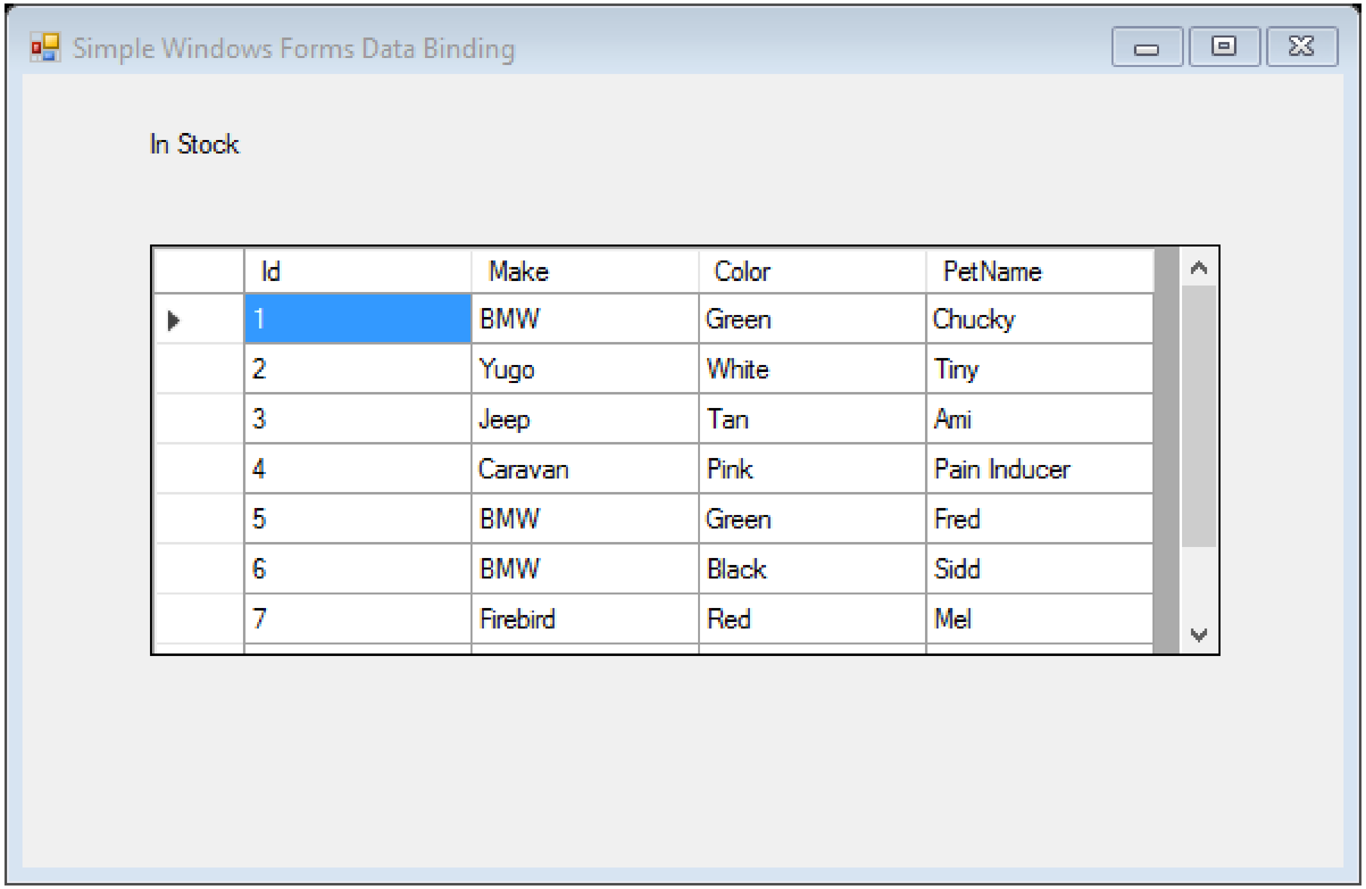
# Add DataTable, and data

```
listCars = new List<Car>()
{
        new Car { CarId = 1, Name = "Chucky", Make = "BMW", Color = "Green" },
        new Car { CarId = 2, Name = "Tiny", Make = "Yugo", Color = "White" },
        new Car { CarId = 3, Name = "Ami", Make = "Jeep", Color = "Tan" },
        new Car { CarId = 4, Name = "Pain Inducer", Make = "Caravan", Color = "Pink"
},
        new Car { CarId = 5, Name = "Fred", Make = "BMW", Color = "Green" },
        new Car { CarId = 6, Name = "Sidd", Make = "BMW", Color = "Black" },
        new Car { CarId = 7, Name = "Mel", Make = "Firebird", Color = "Red" },
        new Car { CarId = 8, Name = "Sarah", Make = "Colt", Color = "Black" },
};
CreateDataTable();
```

## CreateDataTable

```
        void CreateDataTable()
        {
            // Create table schema.
            var carIDColumn = new DataColumn("CarId", typeof(int))
                { Caption = "Car ID" };
            var carMakeColumn = new DataColumn("Make", typeof(string));
            var carColorColumn = new DataColumn("Color", typeof(string));
            var carNameColumn = new DataColumn("Name", typeof(string))
                { Caption = "Car Name" };

            inventoryTable.Columns.AddRange(
                new[] { carIDColumn, carMakeColumn, carColorColumn, carNameColumn });

            // Iterate over the array list to make rows.
            foreach (var c in listCars)
            {
                var newRow = inventoryTable.NewRow();
                newRow["CarId"] = c.CarId;
                newRow["Make"] = c.Make;
                newRow["Color"] = c.Color;
                newRow["Name"] = c.Name;
                inventoryTable.Rows.Add(newRow);
            }
            // Bind the DataTable to the carInventoryGridView.
            dataGridViewInventory.DataSource = inventoryTable;
            dataGridViewInventory.AllowUserToAddRows = false;
            dataGridViewInventory.AllowUserToDeleteRows = false;
        }
```

# Output

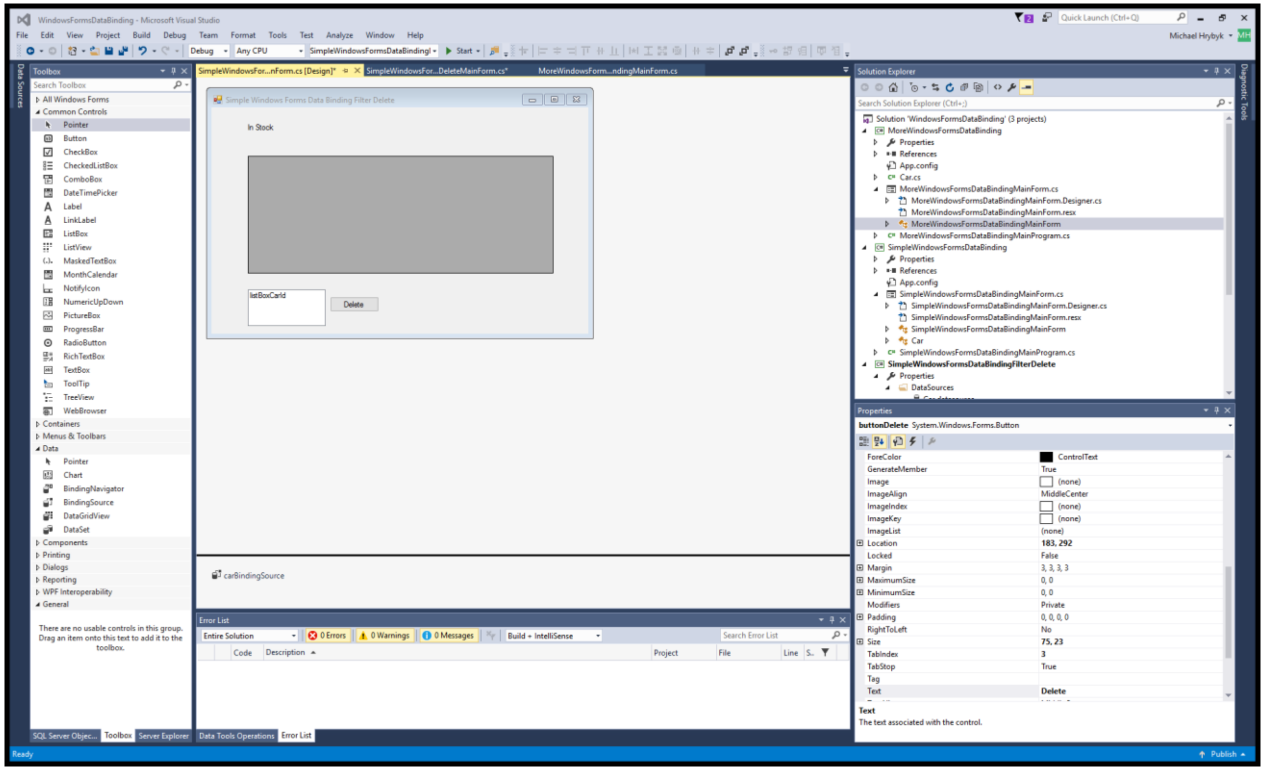

Simple Windows Forms Data Binding

In Stock

| | Id | Make | Color | PetName |
|---|---|---|---|---|
| ▶ | 1 | BMW | Green | Chucky |
| | 2 | Yugo | White | Tiny |
| | 3 | Jeep | Tan | Ami |
| | 4 | Caravan | Pink | Pain Inducer |
| | 5 | BMW | Green | Fred |
| | 6 | BMW | Black | Sidd |
| | 7 | Firebird | Red | Mel |

## Delete an item

- Let's add a ListBox and Button
- See SimpleWindowsFormsDataBindingDelete


WindowsFormsDataBinding - Microsoft Visual Studio
Simple Windows Forms Data Binding Filter Delete
In Stock
listBoxCarId
Delete
carBindingSource
Solution Explorer
Properties
Error List

buttonDelete System.Windows.Forms.Button

| | |
|---|---|
| ForeColor | ControlText |
| GenerateMember | True |
| Image | (none) |
| ImageAlign | MiddleCenter |
| ImageIndex | (none) |
| ImageKey | (none) |
| ImageList | (none) |
| Location | 183, 292 |
| Locked | False |
| Margin | 3, 3, 3, 3 |
| MaximumSize | 0, 0 |
| MinimumSize | 0, 0 |
| Modifiers | Private |
| Padding | 0, 0, 0, 0 |
| RightToLeft | No |
| Size | 75, 23 |
| TabIndex | 3 |
| TabStop | True |
| Tag | |
| Text | Delete |

**Text**
The text associated with the control.

# Fill in the listBox

- Simply add a line where we are adding to the table, add CarId to the listBox
- or use LINQ

```
        foreach (var c in listCars)
        {
            var newRow = inventoryTable.NewRow();
            newRow["CarId"] = c.CarId;
            newRow["Make"] = c.Make;
            newRow["Color"] = c.Color;
            newRow["Name"] = c.Name;
            inventoryTable.Rows.Add(newRow);
            //listBoxCarId.Items.Add(c.Id.ToString()); // add Id to listBox while we are at it
        }

        listBoxCarId.DataSource = GetCarIds(inventoryTable);
-------------------------------------------------------------------------------------------

    List<int> GetCarIds(DataTable dataTable)
    {
        // Here is a way to use LINQ to get a list from a datatable
        return dataTable.AsEnumerable()
            .Select(x => x.Field<int>("CarId"))
            .OrderBy(x => x).ToList();
    }
```

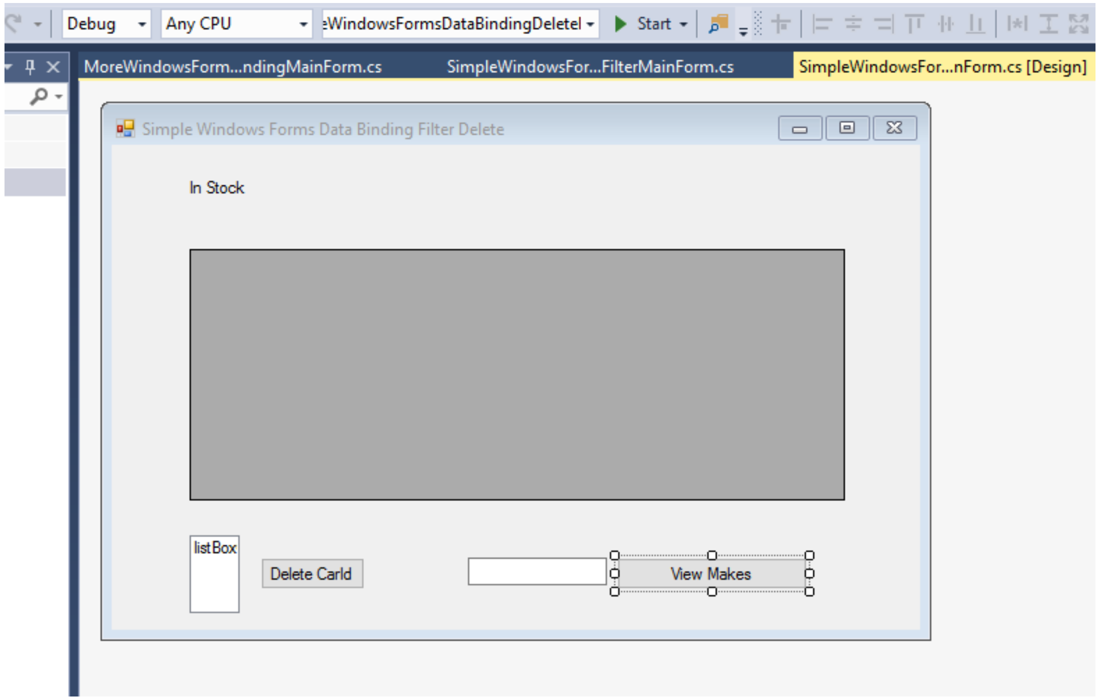
## Button action

- Get the selected CarId from the listBox
- Lookup the Id using Select, which will return a list of rows
- Delete the row and AcceptChanges
- Remove the item from the listBox

```
        private void ButtonDelete_Click(object sender, EventArgs e)
        {
            // get the ID to delete from the list box

            if (listBoxCarId.Items.Count == 0)
                return;

            // Find the correct row to delete.
            DataRow[] rowToDelete = inventoryTable
                .Select($"CarId={listBoxCarId.SelectedItem.ToString()}");

            // Delete it!
            rowToDelete[0].Delete();
            inventoryTable.AcceptChanges();

            listBoxCarId.DataSource = GetCarIds(inventoryTable);
        }
```

# Add a filter

- Add a button and textbox to filter on car names
- See SimpleWindowsFormsDataBindingDeleteFilter


Debug
Any CPU
:WindowsFormsDataBindingDeletel
Start
MoreWindowsForm...ndingMainForm.cs
SimpleWindowsFor...FilterMainForm.cs
SimpleWindowsFor...nForm.cs [Design]
Simple Windows Forms Data Binding Filter Delete
In Stock
listBox
Delete Carld
View Makes

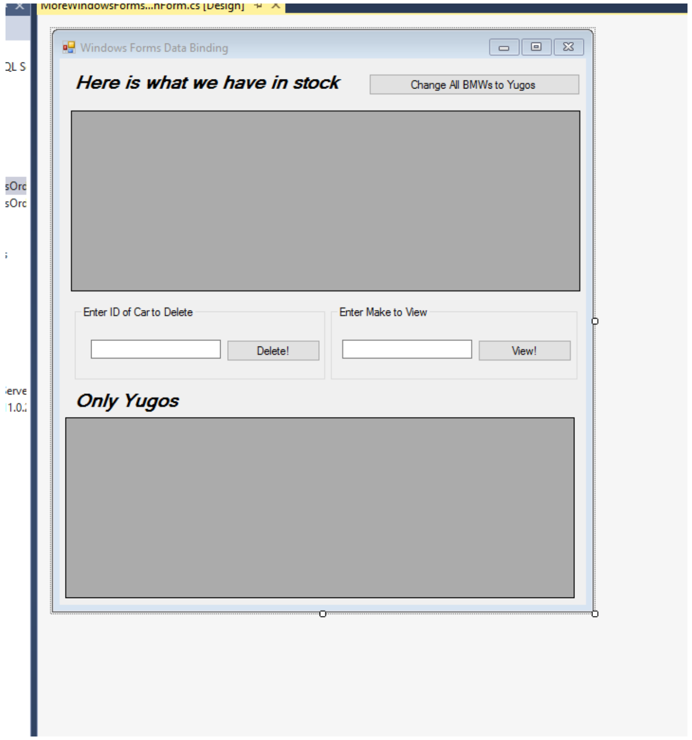
# View Makes button action

- Select method takes two arguments:
  - Filter string: criteria, like a sql where clause
  - sort: column and direction
- But easier to do with LINQ!!

```
private void ButtonViewMakes_Click(object sender, EventArgs e)
       {
           // Build a filter based on user input.
           string makesFilter = $"Make='{textBoxInputMake.Text}'";

           // Find all rows matching the filter.
           DataRow[] makes = inventoryTable.Select(makesFilter, "Name DESC");

          // use of standard linq

           var query =
               from DataRow row in inventoryTable.AsEnumerable()
               where row.Field<string>("Make") == textBoxInputMake.Text
               orderby row["Name"] descending
               select row;

           MessageBox.Show($"LINQ result count {query.Count()}", "Query Result", MessageBoxButtons.OK);

           string makesOutput = "";

           foreach(DataRow row in query)
           {
               makesOutput += "Name: " + row["Name"] + ", Color: " + row["Color"] + "\n";
           }

           MessageBox.Show(makesOutput, $"LINQ We have {query.Count()} {textBoxInputMake.Text}'s named:");

           // just for fun, change datasource

           //dataGridViewInventory.DataSource = query.CopyToDataTable();
           //dataGridViewInventory.Refresh();
       }
```

# Add another DataGridView bound to DataView

- Create a DataView and bind that to the control
  - Similar to what was done with DataTable
- See MoreWindowsFormsDataBinding

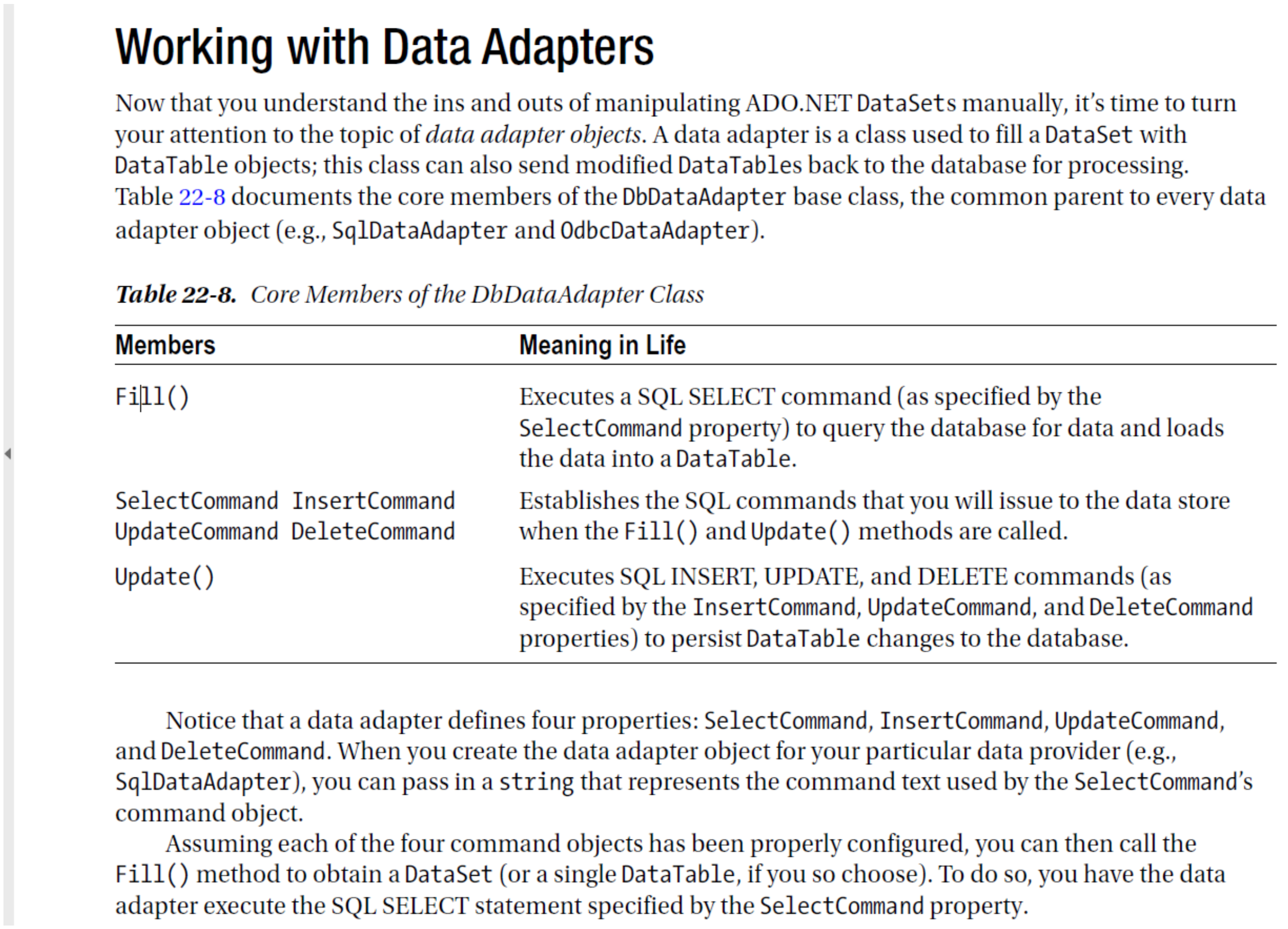

MoreWindowsForms...nForm.cs [Design]
Windows Forms Data Binding
Here is what we have in stock
Change All BMWs to Yugos
Enter ID of Car to Delete
Delete!
Enter Make to View
View!
Only Yugos

## CreateDataView

- called after CreateDataTable
- hydrates the lower DataGridViewControl
- sets RowFilter to an expression

```
private void CreateDataView()
{
      // Set the table that is used to construct this view.
      yugosOnlyView = new DataView(inventoryTable);

      // Now configure the views using a filter.
      yugosOnlyView.RowFilter = "Make = 'Yugo'";

      // Bind to the new grid.
      dataGridYugosView.DataSource = yugosOnlyView;
}
```

## Data Adapters



### Working with Data Adapters

Now that you understand the ins and outs of manipulating ADO.NET `DataSets` manually, it's time to turn your attention to the topic of *data adapter objects*. A data adapter is a class used to fill a `DataSet` with `DataTable` objects; this class can also send modified `DataTables` back to the database for processing. Table 22-8 documents the core members of the `DbDataAdapter` base class, the common parent to every data adapter object (e.g., `SqlDataAdapter` and `OdbcDataAdapter`).

***Table 22-8.*** *Core Members of the DbDataAdapter Class*

| Members | Meaning in Life |
|---|---|
| `Fill()` | Executes a SQL SELECT command (as specified by the `SelectCommand` property) to query the database for data and loads the data into a `DataTable`. |
| `SelectCommand` `InsertCommand` `UpdateCommand` `DeleteCommand` | Establishes the SQL commands that you will issue to the data store when the `Fill()` and `Update()` methods are called. |
| `Update()` | Executes SQL INSERT, UPDATE, and DELETE commands (as specified by the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` properties) to persist `DataTable` changes to the database. |

Notice that a data adapter defines four properties: `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand`. When you create the data adapter object for your particular data provider (e.g., `SqlDataAdapter`), you can pass in a `string` that represents the command text used by the `SelectCommand`'s command object.

Assuming each of the four command objects has been properly configured, you can then call the `Fill()` method to obtain a `DataSet` (or a single `DataTable`, if you so choose). To do so, you have the data adapter execute the SQL SELECT statement specified by the `SelectCommand` property.

## Data Adapters

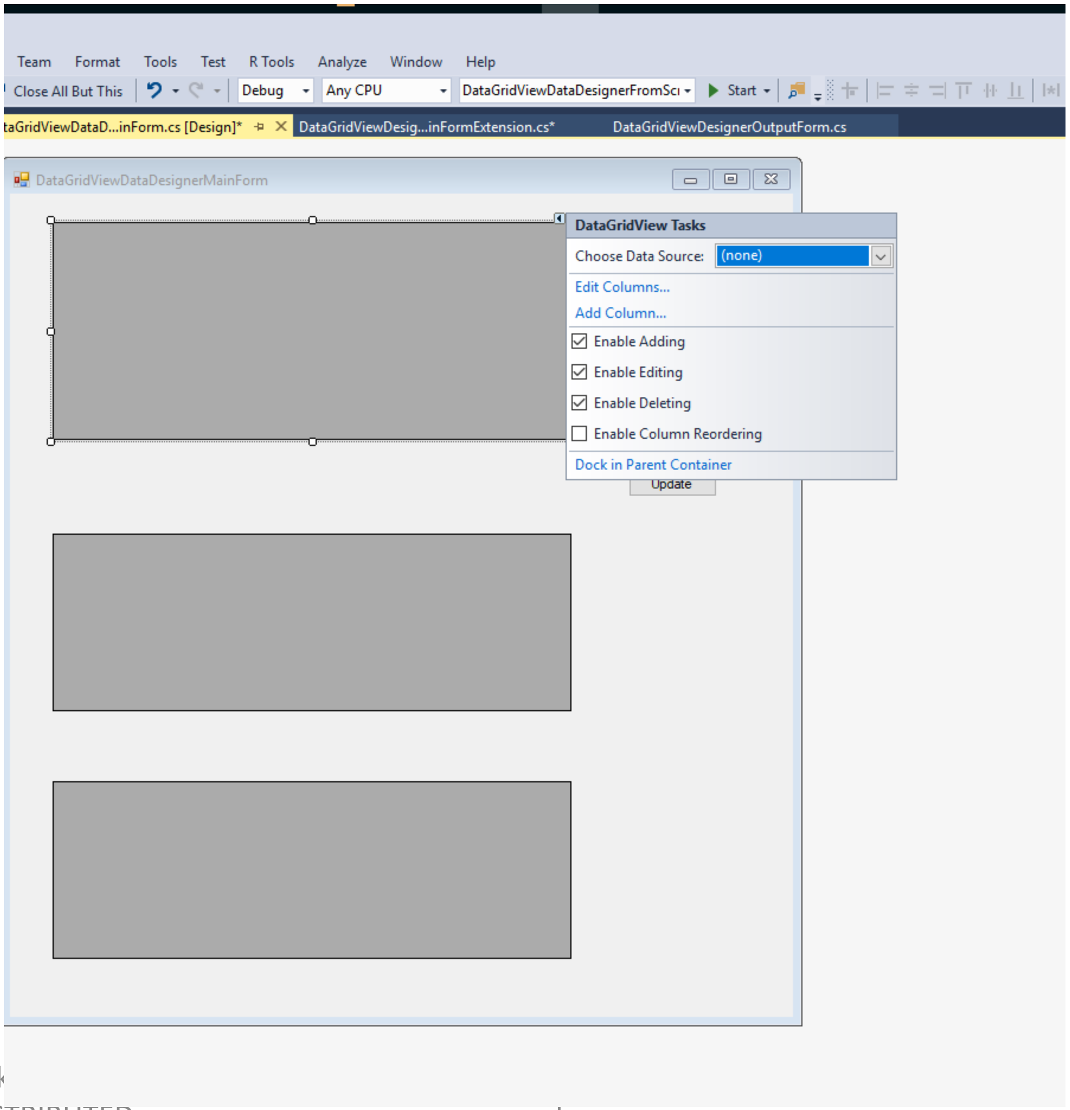
- See FillDataSetUsingSqlDataAdapter
- Note that Fill() creates CommandText for Insert, Delete, Select commands automatically from database!
- Once table is “Fill()”d, changes can be made, and Update() called
  - DataGridView.DataSource can be set to the table
  - When TableAdapter.Update() is called, DataGridView is updated!
- See InventoryDALDIscconnectedGUI

```
            new SqlCommandBuilder(inventoryAdapter);  // needed to set up all SQL commands in adapter

            inventoryAdapter.Fill(autoLotDataSet, "Inventory"); // note use of Fill
            DisplayDataSet(autoLotDataSet);

            // change command to only select id and name
            // This will create a new table in the dataset

            inventoryAdapter.SelectCommand.CommandText = "Select CarId,Name from Inventory";

            // create new column mappings using friendly names

            DataTableMapping namesColumnMappings = new DataTableMapping("Names", "Only Car Names");
            namesColumnMappings.ColumnMappings.Add("CarId", "Car Id");
            namesColumnMappings.ColumnMappings.Add("Name", "Car Name");
            inventoryAdapter.TableMappings.Add(namesColumnMappings);

            inventoryAdapter.Fill(autoLotDataSet, "Names"); // this actually creates a new table in
memory, but not in DB

            DisplayDataSet(autoLotDataSet);
```

## Using Windows Forms Data Designer

- First, (re)create the AutoLot database
  - Note the use of Seed sql script to seed the database
- Take a look at the database design view
  - Note constraints!
- Adapters – already done!
- Just fire it up and tables are loaded!
- See
  - DataGridViewDesigner – just inventory table
  - MultiTableDataGridViewDesigner – add other tables
  - AutoLotDataDesigner – include all tables and views

# Create three DGVs, and start wizard


Team Format Tools Test R Tools Analyze Window Help
Close All But This
Debug
Any CPU
DataGridViewDataDesignerFromSc
Start
taGridViewDataD...inForm.cs [Design]*
DataGridViewDesig...inFormExtension.cs*
DataGridViewDesignerOutputForm.cs
DataGridViewDataDesignerMainForm
Update

DataGridView Tasks

Choose Data Source: (none)

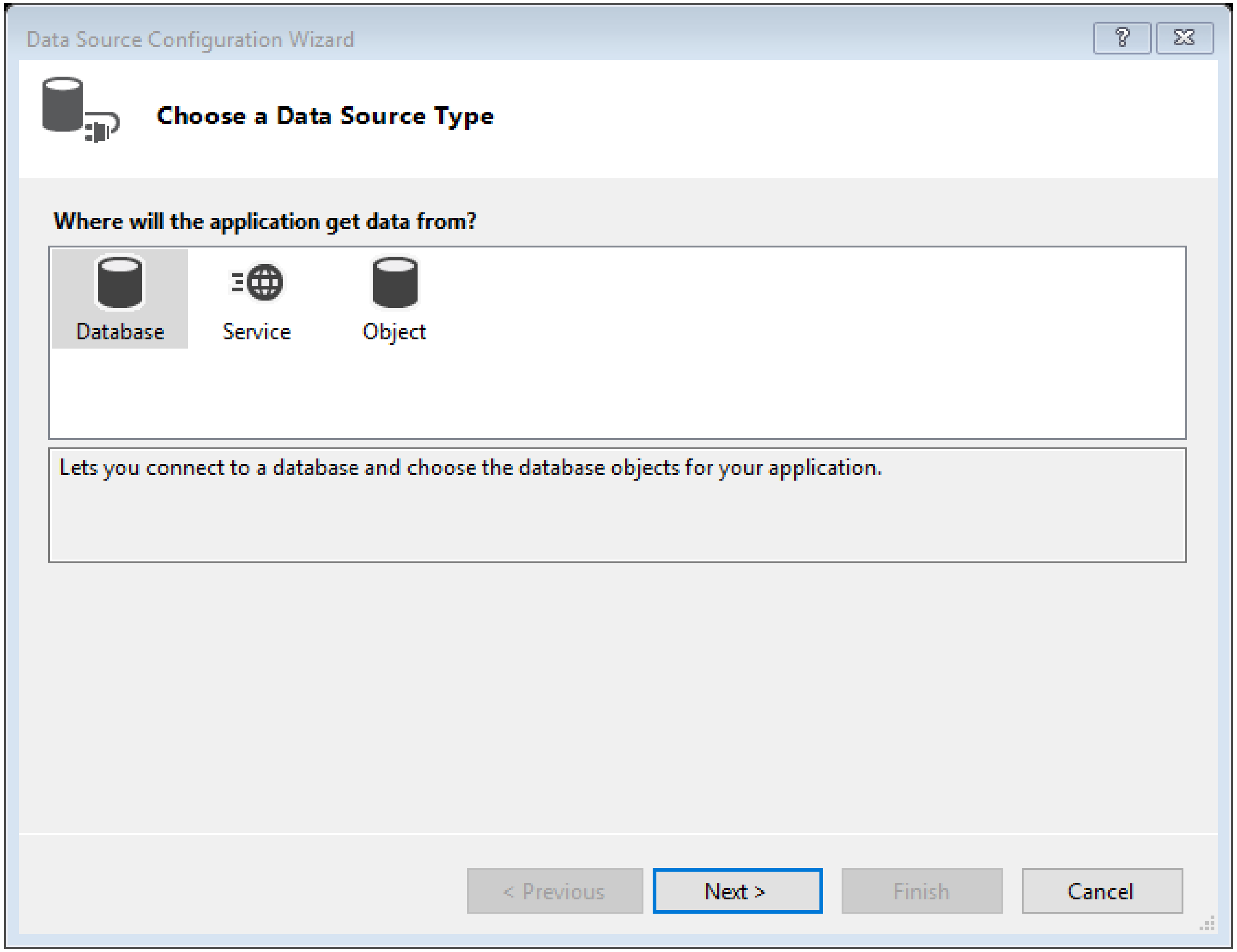
Edit Columns...

Add Column...

- [x] Enable Adding
- [x] Enable Editing
- [x] Enable Deleting
- [ ] Enable Column Reordering

Dock in Parent Container

# Add Project Data Source

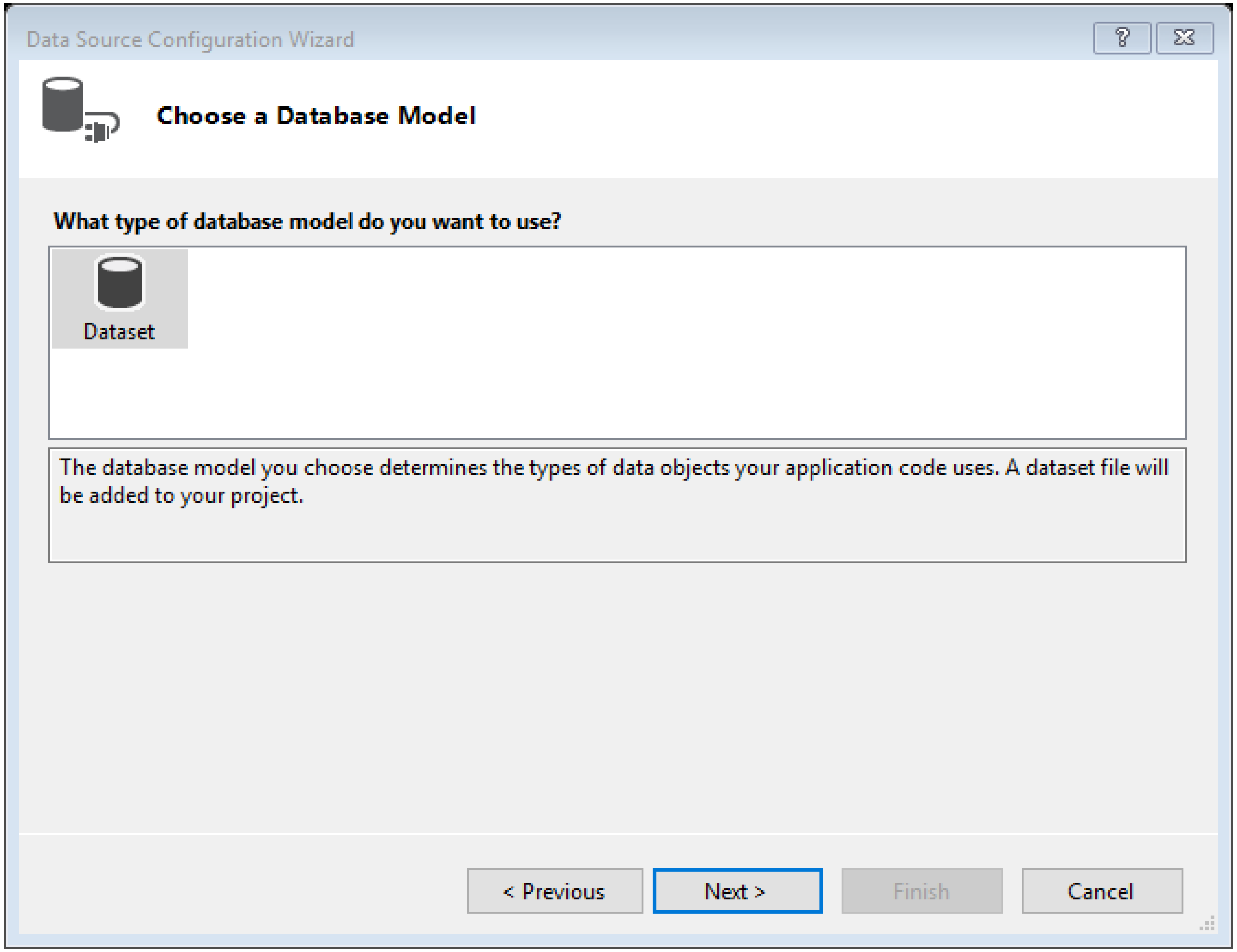


**Choose a Data Source Type**

**Where will the application get data from?**

Lets you connect to a database and choose the database objects for your application.

## Select Dataset as database model

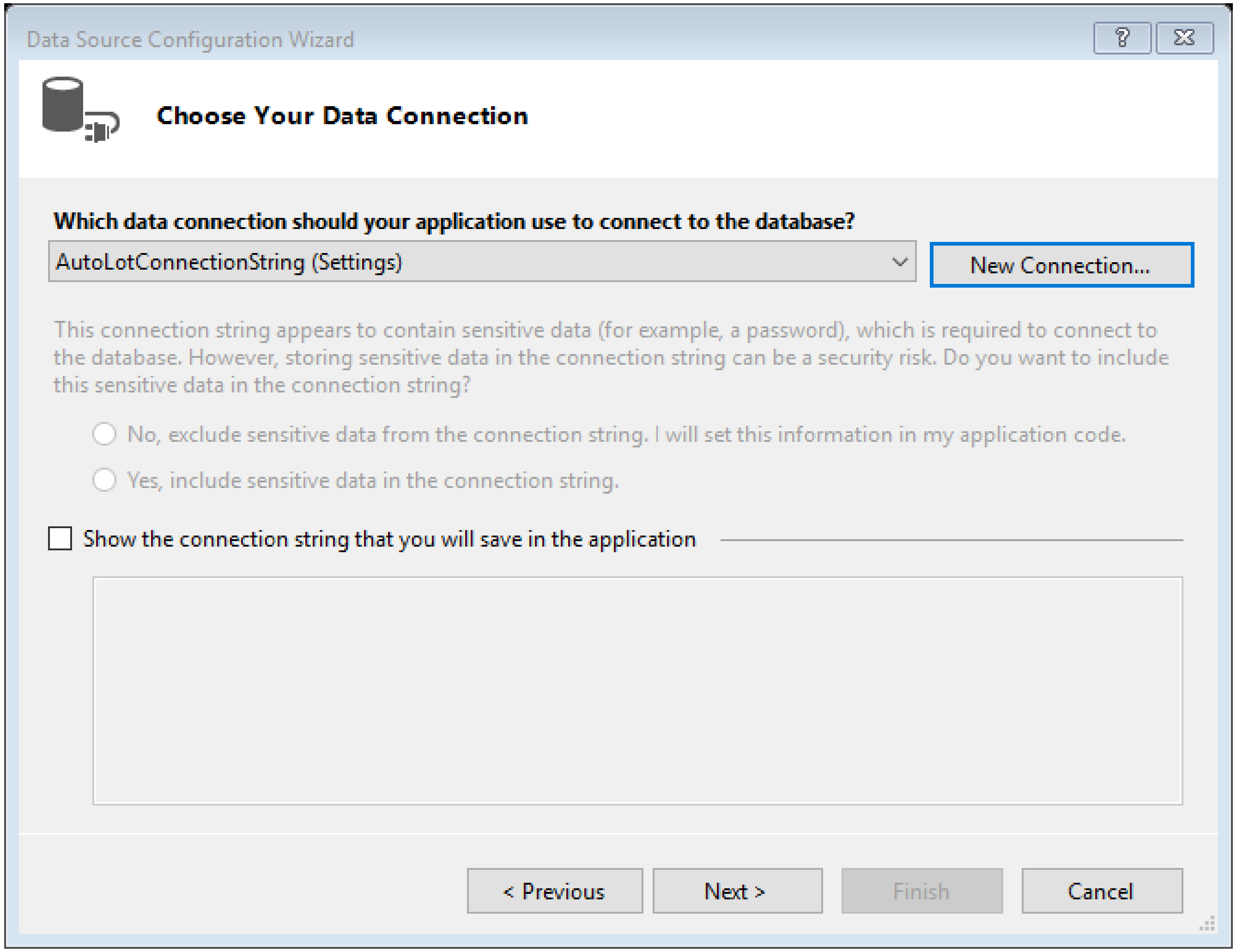

Data Source Configuration Wizard

**Choose a Database Model**

**What type of database model do you want to use?**

The database model you choose determines the types of data objects your application code uses. A dataset file will be added to your project.

# Create a connection to the database


Data Source Configuration Wizard

**Choose Your Data Connection**

**Which data connection should your application use to connect to the database?**

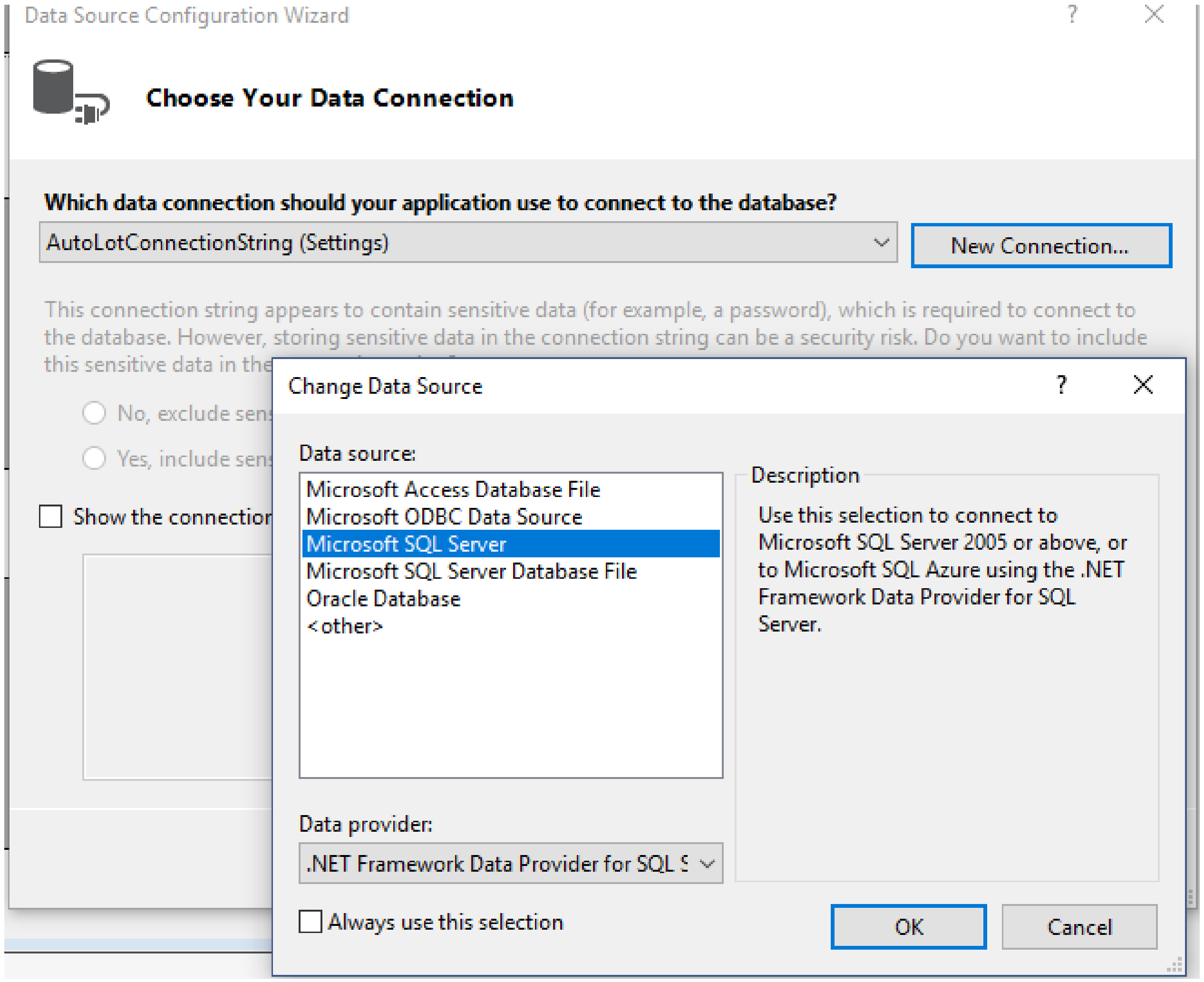
AutoLotConnectionString (Settings)

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

- ○ No, exclude sensitive data from the connection string. I will set this information in my application code.
- ○ Yes, include sensitive data in the connection string.

☐ Show the connection string that you will save in the application

# Specify SQL Server



Data Source Configuration Wizard

**Choose Your Data Connection**

**Which data connection should your application use to connect to the database?**

AutoLotConnectionString (Settings)

New Connection...

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the

◯ No, exclude sens

◯ Yes, include sens

☐ Show the connection

Change Data Source

Data source:

- Microsoft Access Database File
- Microsoft ODBC Data Source
- Microsoft SQL Server
- Microsoft SQL Server Database File
- Oracle Database
- <other>

Description

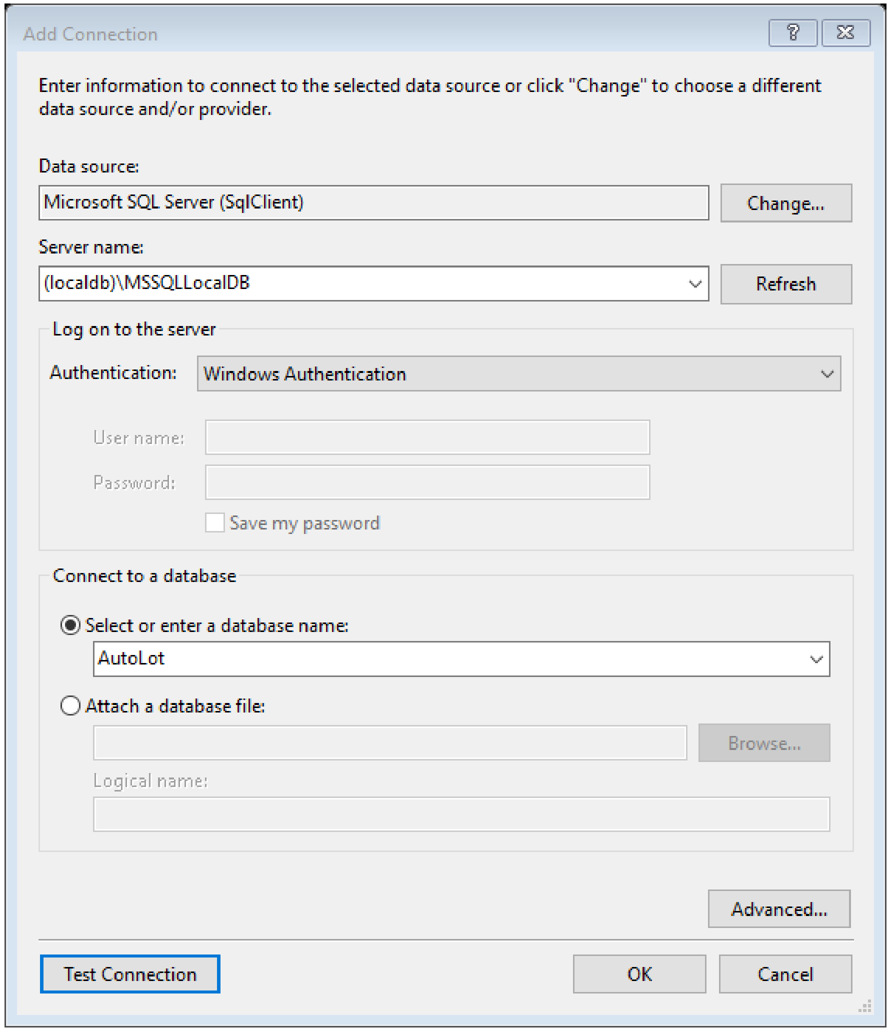
Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.

Data provider: .NET Framework Data Provider for SQL S

☐ Always use this selection

OK Cancel

# Set up the database connection



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient)

Server name:
(localdb)\MSSQLLocalDB

Log on to the server

Authentication: Windows Authentication

User name:

Password:

[ ] Save my password

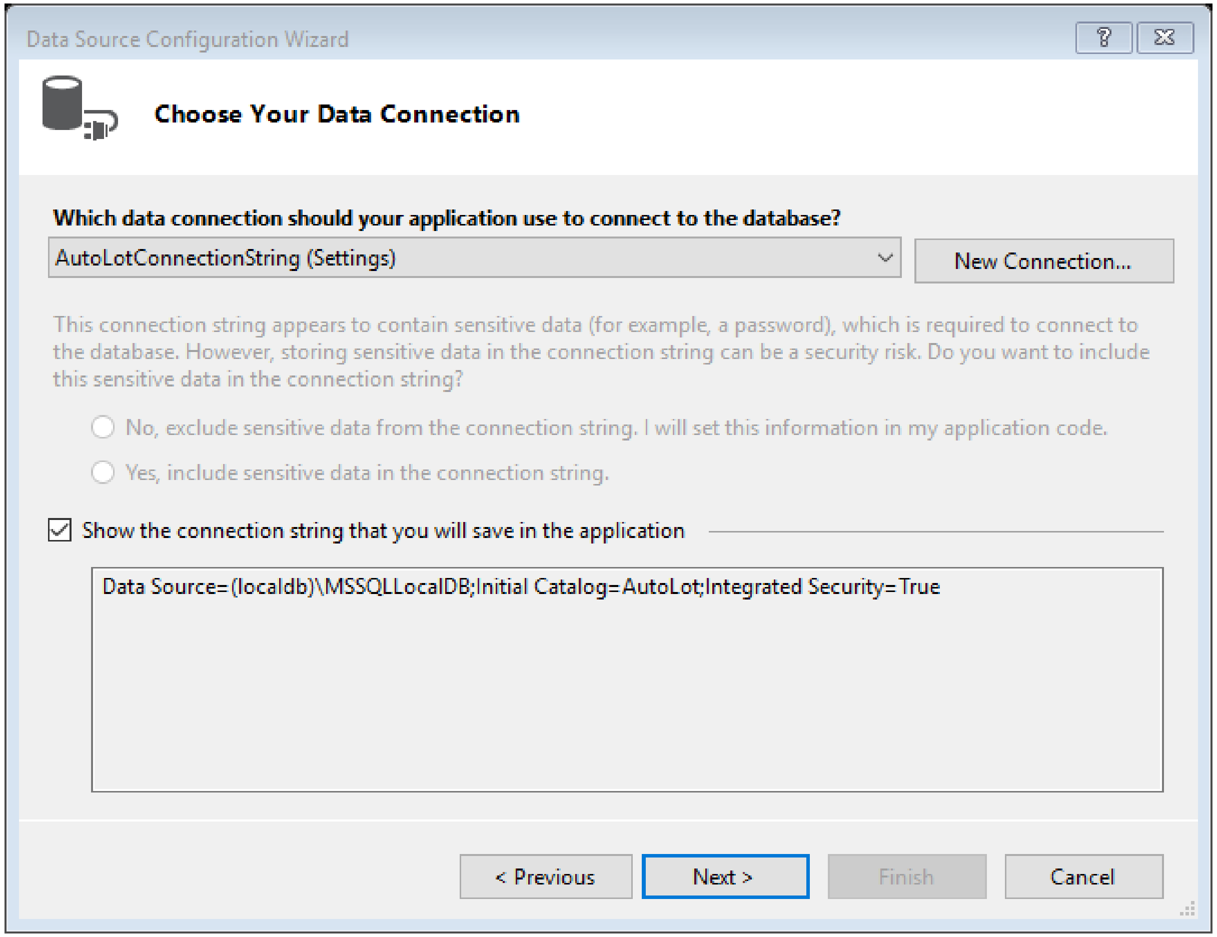
Connect to a database

[● Select or enter a database name:]
AutoLot

○ Attach a database file:

Logical name:

## Check that everything is ok (App.Config)



Which data connection should your application use to connect to the database?

AutoLotConnectionString (Settings)

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

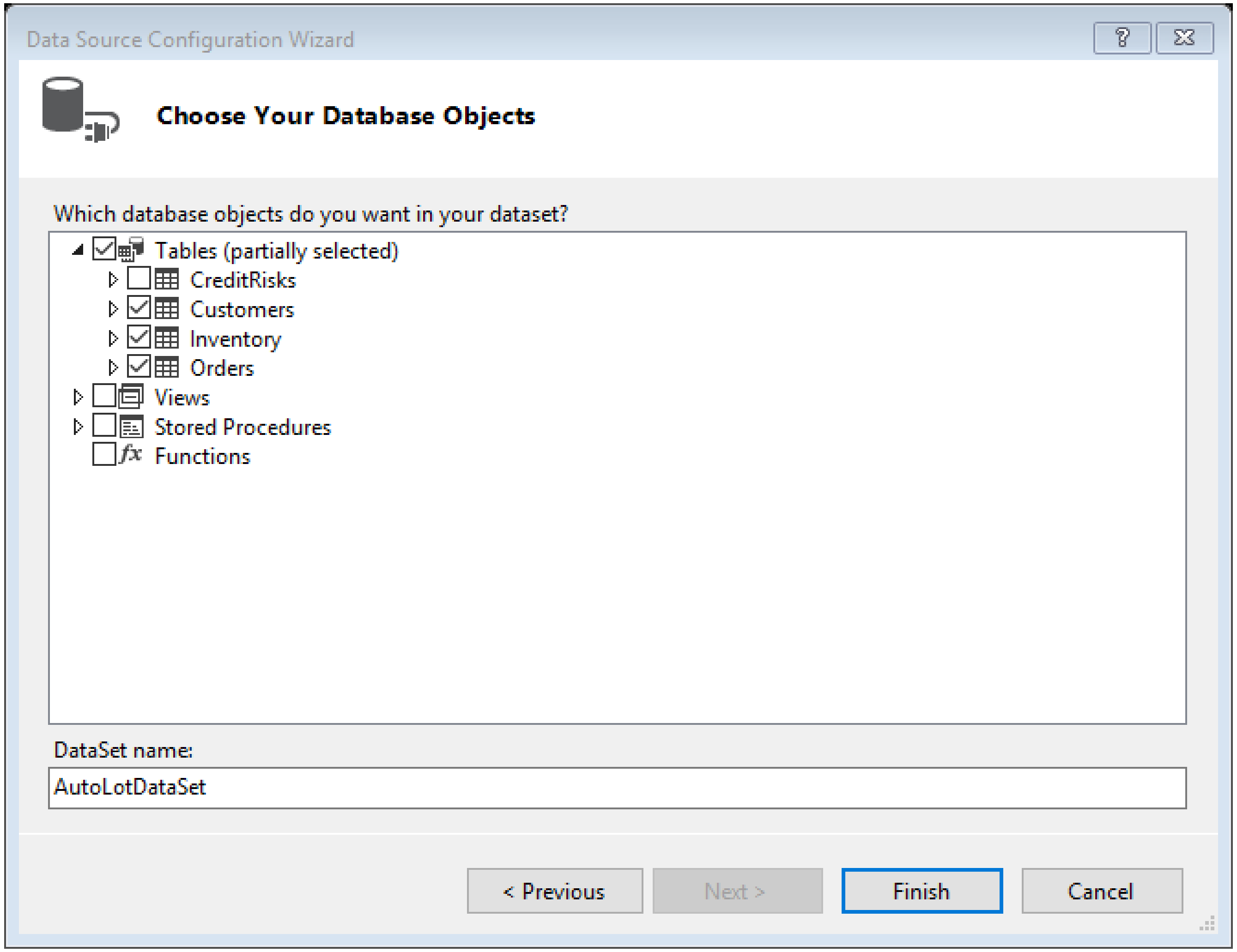
○ No, exclude sensitive data from the connection string. I will set this information in my application code.

○ Yes, include sensitive data in the connection string.

☑ Show the connection string that you will save in the application

```
Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=AutoLot;Integrated Security=True
```

# Select the database objects



Data Source Configuration Wizard

**Choose Your Database Objects**

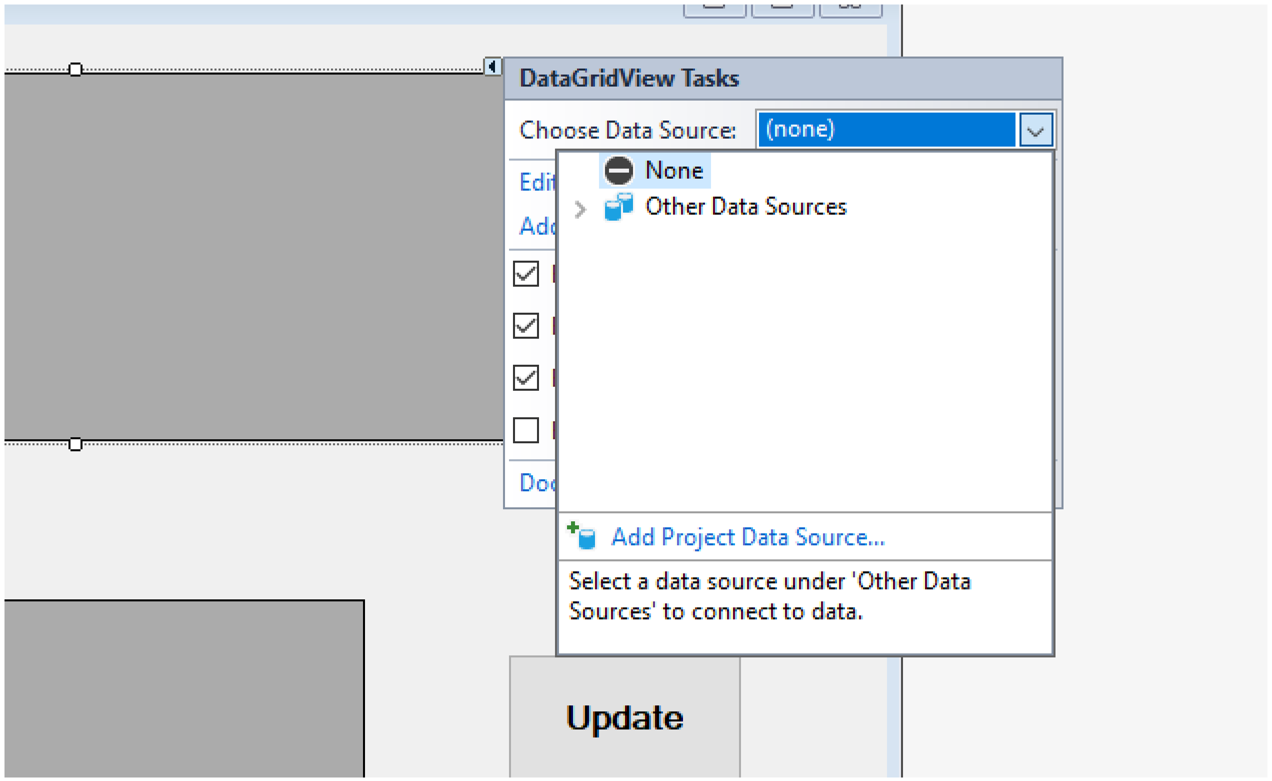
Which database objects do you want in your dataset?

- [✓] Tables (partially selected)
  - [ ] CreditRisks
  - [✓] Customers
  - [✓] Inventory
  - [✓] Orders
- [ ] Views
- [ ] Stored Procedures
- [ ] Functions

DataSet name:

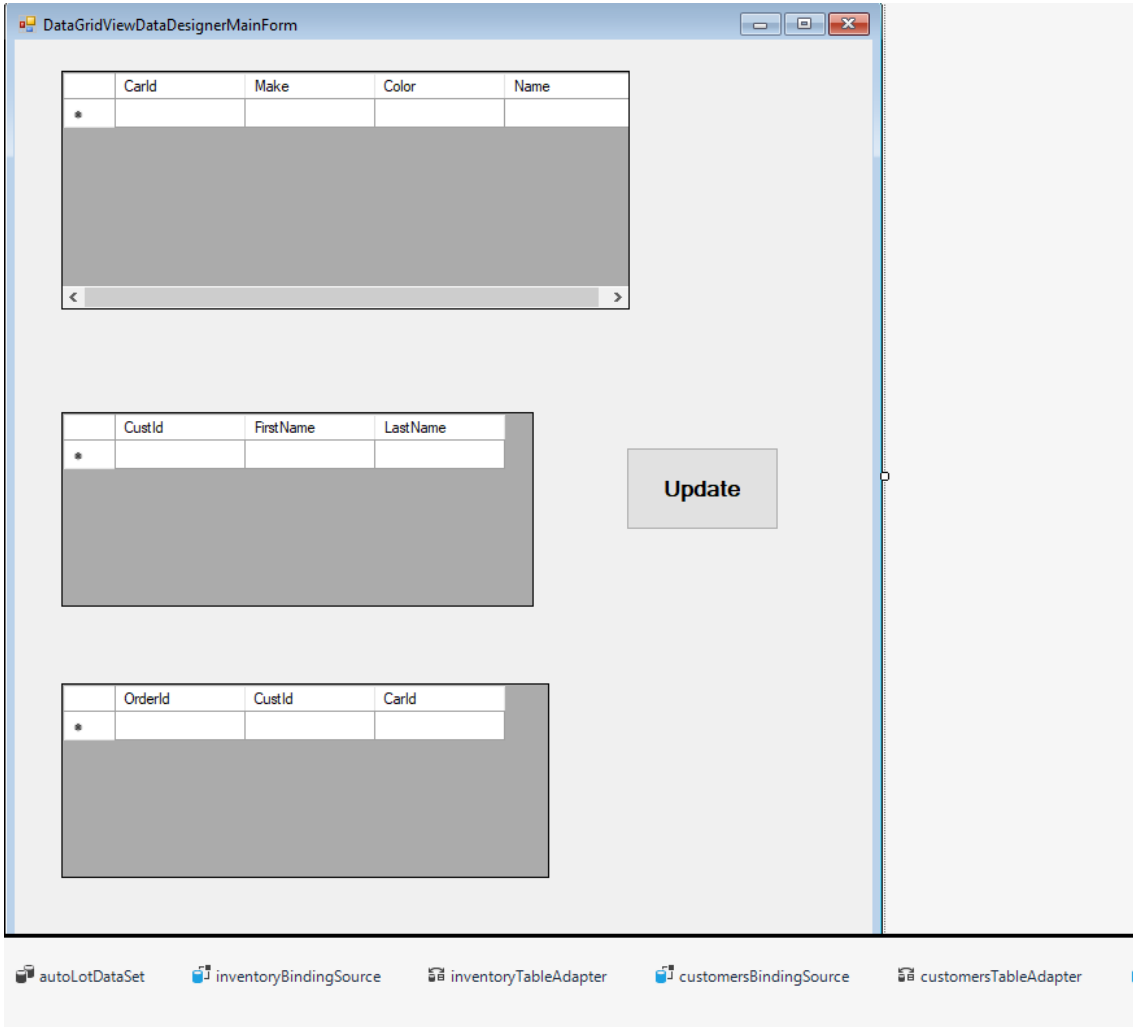
AutoLotDataSet

# Now for each DGV choose a table



Select a data source under 'Other Data Sources' to connect to data.

## Finished controls



|  | CarId | Make | Color | Name |
|---|---|---|---|---|
| * |  |  |  |  |

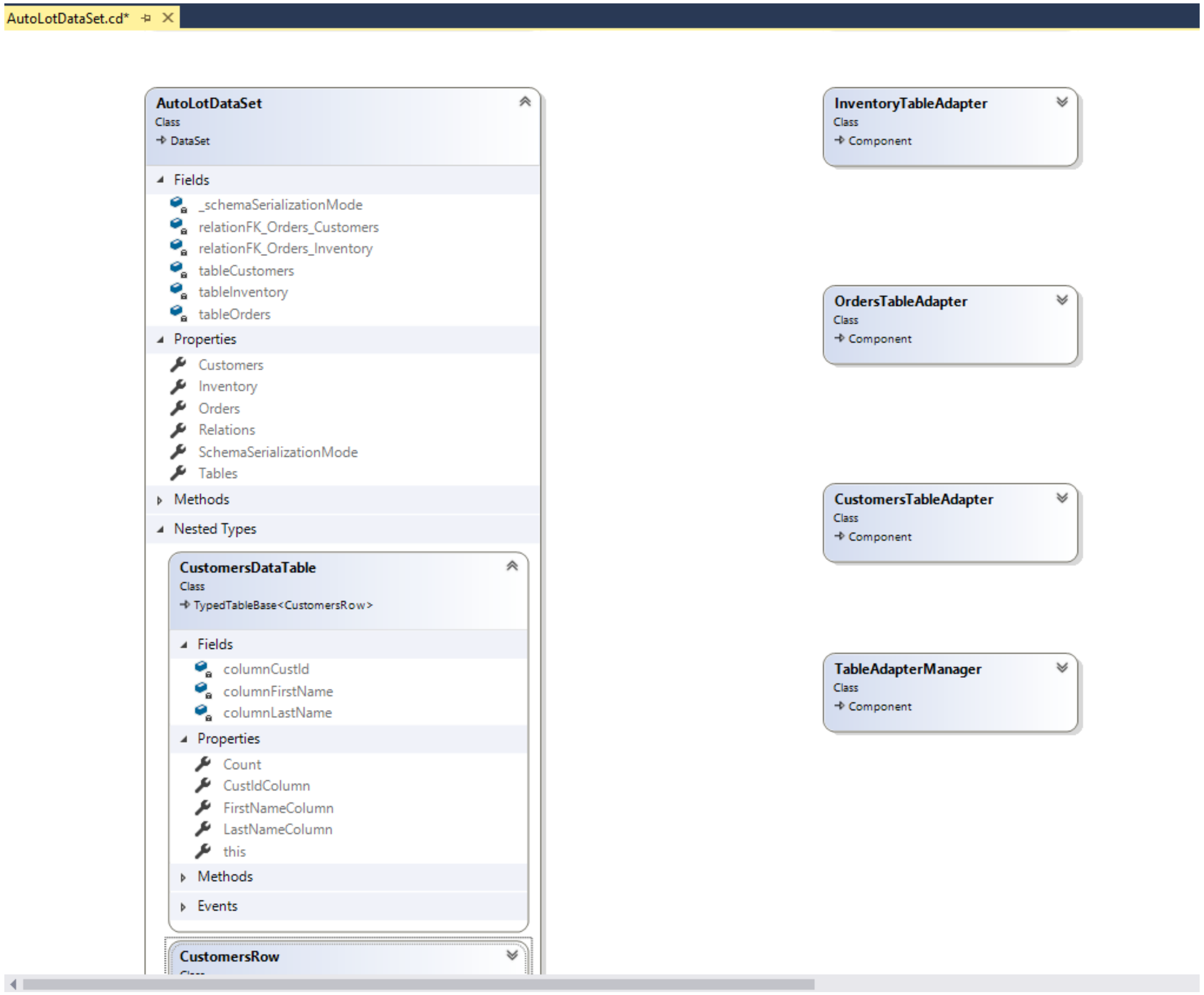
|  | CustId | FirstName | LastName |
|---|---|---|---|
| * |  |  |  |

|  | OrderId | CustId | CarId |
|---|---|---|---|
| * |  |  |  |

# Windows Form Data Designer

- Use custom methods
- Strong types created for *each* table
    - Customers, Inventory, Orders
    - No need for Tables[“Customers”] anymore
- Now add a button to update the database
    - Fill() with each table

# Fields are now typed (see class diagram)



AutoLotDataSet
Class
DataSet

Fields
- _schemaSerializationMode
- relationFK_Orders_Customers
- relationFK_Orders_Inventory
- tableCustomers
- tableInventory
- tableOrders

Properties
- Customers
- Inventory
- Orders
- Relations
- SchemaSerializationMode
- Tables

Methods

Nested Types

CustomersDataTable
Class
TypedTableBase<CustomersRow>

Fields
- columnCustId
- columnFirstName
- columnLastName

Properties
- Count
- CustIdColumn
- FirstNameColumn
- LastNameColumn
- this

Methods

Events

CustomersRow

InventoryTableAdapter
Class
Component

OrdersTableAdapter
Class
Component

CustomersTableAdapter
Class
Component

TableAdapterManager
Class
Component

# App.config automatically set up

- Everything is set up!

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="MultiTableDataGridViewDataDesigner.Properties.Settings.AutoLotConnectionString"
            connectionString="Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=AutoLot;Integrated
Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
    </startup>
</configuration>
```

## Code to run is preset

- Notice all of the Fill() methods are generated.

```
            // TODO: This line of code loads data into the 'autoLotDataSet.Orders' table. 
You can move, or remove it, as needed.
            this.ordersTableAdapter.Fill(this.autoLotDataSet.Orders);
            // TODO: This line of code loads data into the 'autoLotDataSet.Customers' 
table. You can move, or remove it, as needed.
            this.customersTableAdapter.Fill(this.autoLotDataSet.Customers);
            // TODO: This line of code loads data into the 'autoLotDataSet.Inventory' 
table. You can move, or remove it, as needed.
            this.inventoryTableAdapter.Fill(this.autoLotDataSet.Inventory);
```

# Add an update button

- Button event: Update() and Fill() all tables
  - Note lack of exception processing here, which is NEEDED

```
        public MainForm()
        /// <summary>
        /// Update the database with user input from datagridview controls, then
        /// redisplay the reports in the output form.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButtonUpdate_Click(object sender, EventArgs e)
        {
            ordersTableAdapter.Update(autoLotDataSet.Orders);
            customersTableAdapter.Update(autoLotDataSet.Customers);
            inventoryTableAdapter.Update(autoLotDataSet.Inventory);

            ordersTableAdapter.Fill(autoLotDataSet.Orders);
            customersTableAdapter.Fill(autoLotDataSet.Customers);
            inventoryTableAdapter.Fill(autoLotDataSet.Inventory);

            outputForm.Close();
            outputForm = new MultiTableDataGridViewDesignerOutputForm();
            outputForm.Show();

        }
```

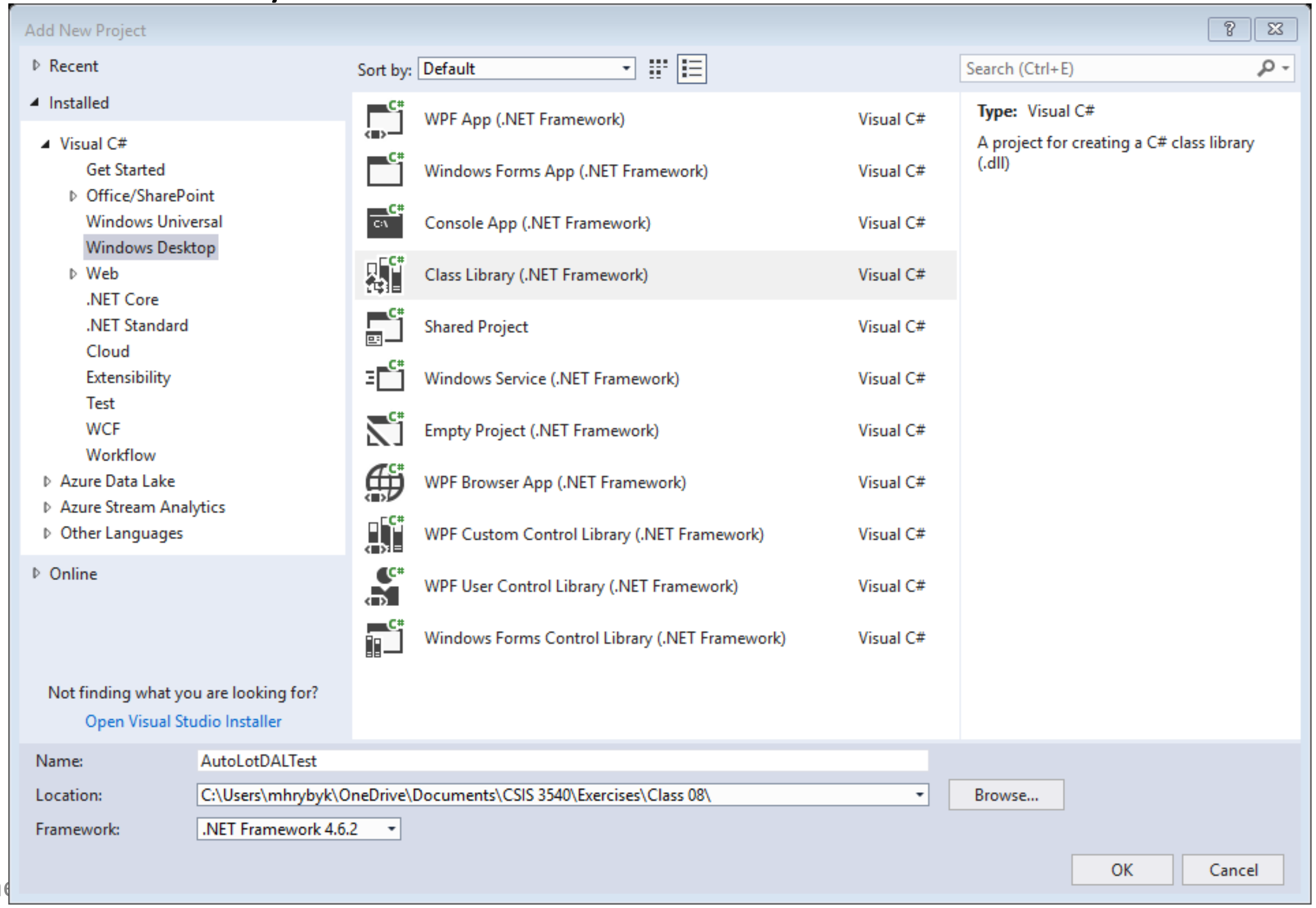
## Can also auto generate and use

- For standalone, add Data item,
  - add xsd schema and drag tables from database onto it
- Code will be generated to add strongly typed dataset
- Note that no column or row indices are now needed!
  - part of MultiTableDataGridViewDesigner, has additional output form

```
// Create new strongly typed datatables, and associated table adapters.

            var inventoryTable = new AutoLotDataSet.InventoryDataTable();
            var ordersTable = new AutoLotDataSet.OrdersDataTable();
            var customersTable = new AutoLotDataSet.CustomersDataTable();

            InventoryTableAdapter inventoryAdapter = new InventoryTableAdapter();
            CustomersTableAdapter customersAdapter = new CustomersTableAdapter();
            OrdersTableAdapter ordersAdapter = new OrdersTableAdapter();

            // now fill the tables with data

            inventoryAdapter.Fill(inventoryTable);
            customersAdapter.Fill(customersTable);
            ordersAdapter.Fill(ordersTable);


            textBoxOutput.AppendText($"Inventory Count = {inventoryTable.Count()}{Environment.NewLine}");

            foreach (var car in inventoryTable)
                textBoxOutput.AppendText($"{car.CarId} {car.Make} {car.Color} {car.Name}{Environment.NewLine}");
```

# Create custom DAL

- Create our own AutoLotDAL from the database
- Will include table adapters and strongly typed classes corresponding to the database tables
- Add a class library named AutoLotDALTest



| | |
|---|---|
| WPF App (.NET Framework) | Visual C# |
| Windows Forms App (.NET Framework) | Visual C# |
| Console App (.NET Framework) | Visual C# |
| Class Library (.NET Framework) | Visual C# |
| Shared Project | Visual C# |
| Windows Service (.NET Framework) | Visual C# |
| Empty Project (.NET Framework) | Visual C# |
| WPF Browser App (.NET Framework) | Visual C# |
| WPF Custom Control Library (.NET Framework) | Visual C# |
| WPF User Control Library (.NET Framework) | Visual C# |
| Windows Forms Control Library (.NET Framework) | Visual C# |

**Type:** Visual C#

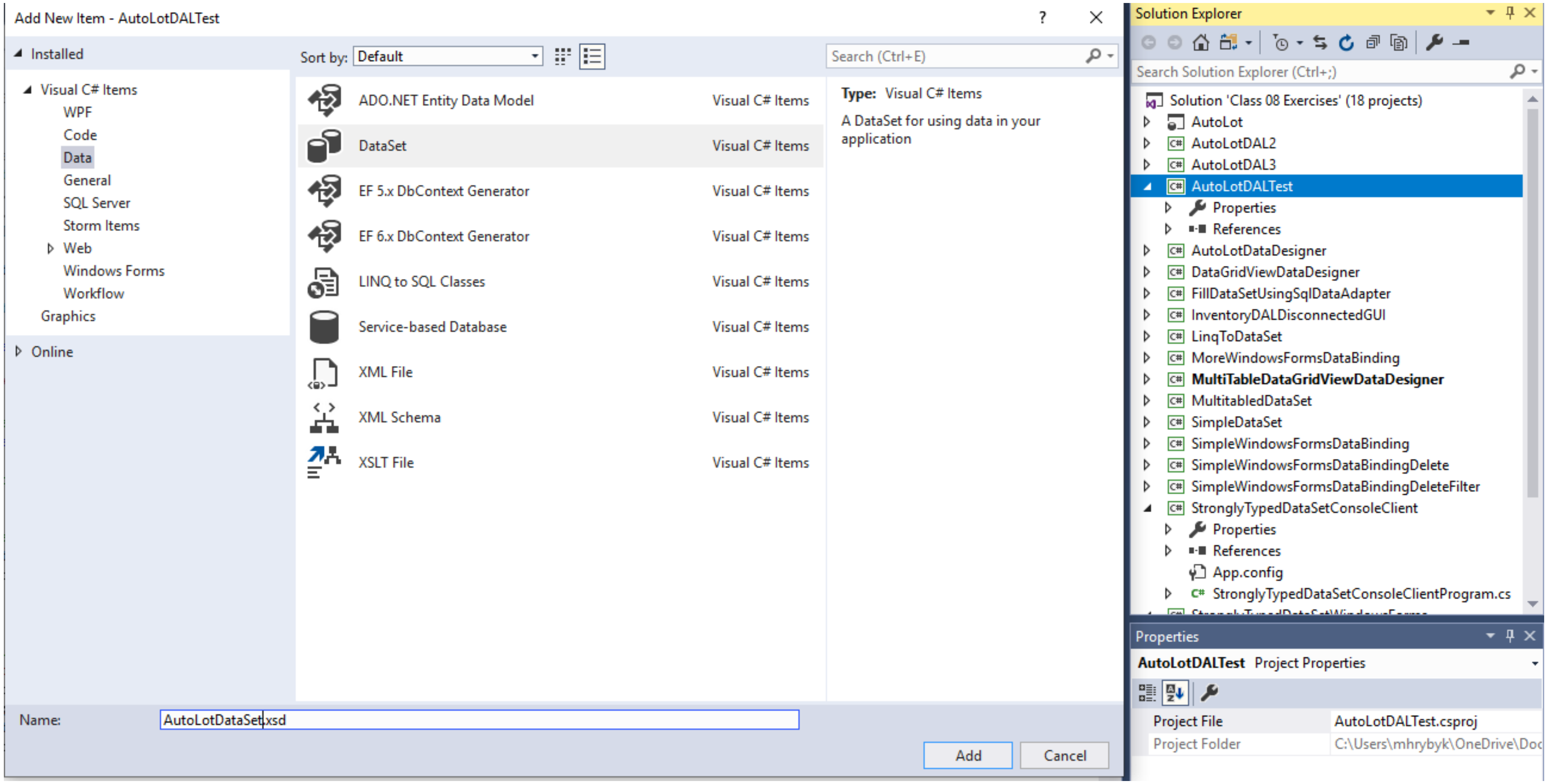
A project for creating a C# class library (.dll)

Name: AutoLotDALTest

Location: C:\Users\mhrybyk\OneDrive\Documents\CSIS 3540\Exercises\Class 08\

Framework: .NET Framework 4.6.2

# Add a dataset to the project



Add New Item - AutoLotDALTest

Installed

- Visual C# Items
  - WPF
  - Code
  - Data
  - General
  - SQL Server
  - Storm Items
  - Web
  - Windows Forms
  - Workflow
- Graphics

Online

Sort by: Default

Search (Ctrl+E)

| | |
|---|---|
| ADO.NET Entity Data Model | Visual C# Items |
| DataSet | Visual C# Items |
| EF 5.x DbContext Generator | Visual C# Items |
| EF 6.x DbContext Generator | Visual C# Items |
| LINQ to SQL Classes | Visual C# Items |
| Service-based Database | Visual C# Items |
| XML File | Visual C# Items |
| XML Schema | Visual C# Items |
| XSLT File | Visual C# Items |

**Type:** Visual C# Items

A DataSet for using data in your application

Name: AutoLotDataSet.xsd

Solution Explorer

Search Solution Explorer (Ctrl+;)

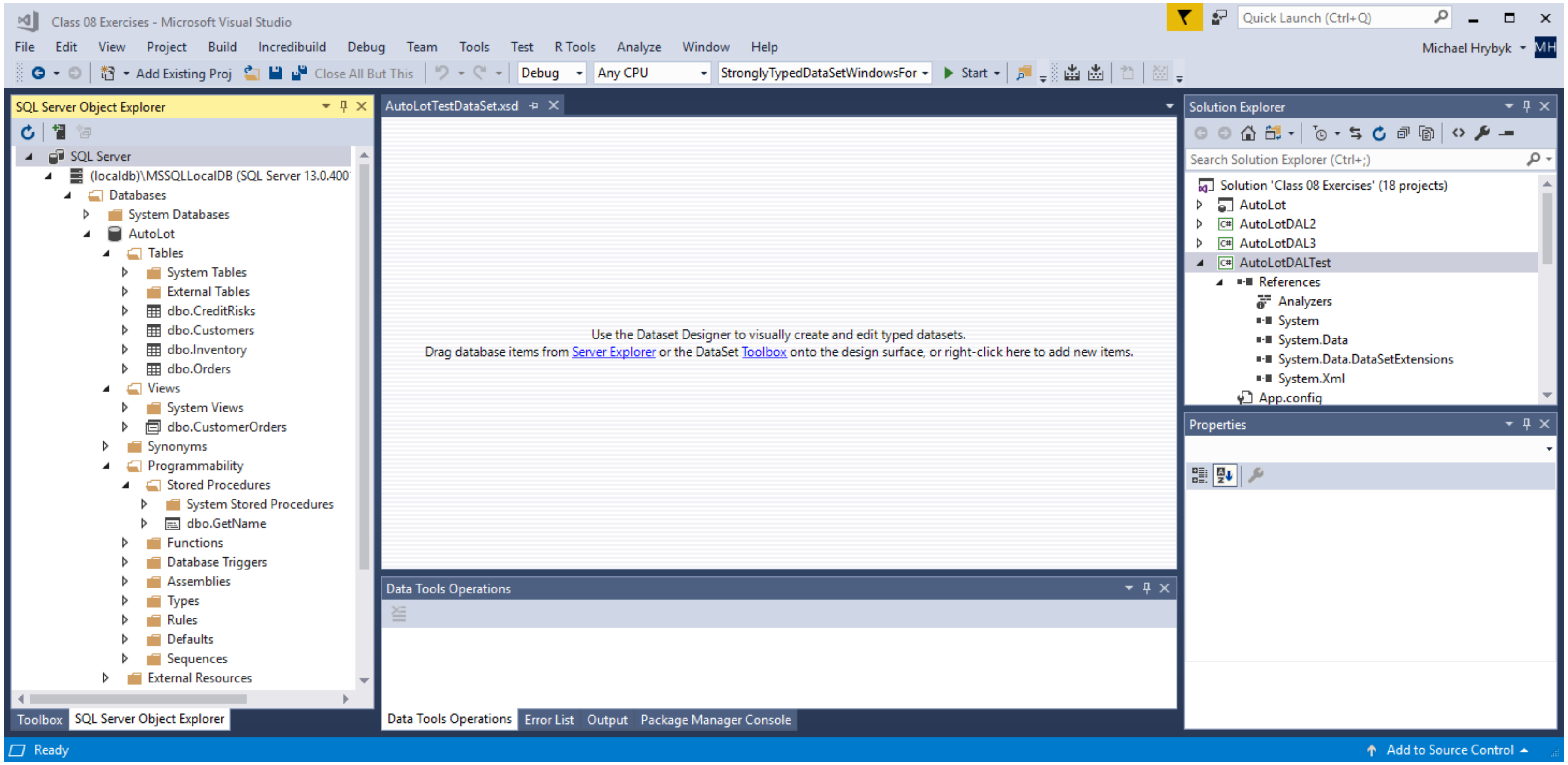
Solution 'Class 08 Exercises' (18 projects)
- AutoLot
- AutoLotDAL2
- AutoLotDAL3
- AutoLotDALTest
  - Properties
  - References
- AutoLotDataDesigner
- DataGridViewDataDesigner
- FillDataSetUsingSqlDataAdapter
- InventoryDALDisconnectedGUI
- LinqToDataSet
- MoreWindowsFormsDataBinding
- **MultiTableDataGridViewDataDesigner**
- MultitabledDataSet
- SimpleDataSet
- SimpleWindowsFormsDataBinding
- SimpleWindowsFormsDataBindingDelete
- SimpleWindowsFormsDataBindingDeleteFilter
- StronglyTypedDataSetConsoleClient
  - Properties
  - References
  - App.config
  - StronglyTypedDataSetConsoleClientProgram.cs

Properties

**AutoLotDALTest** Project Properties

| | |
|---|---|
| Project File | AutoLotDALTest.csproj |
| Project Folder | C:\Users\mhrybyk\OneDrive\Doc |

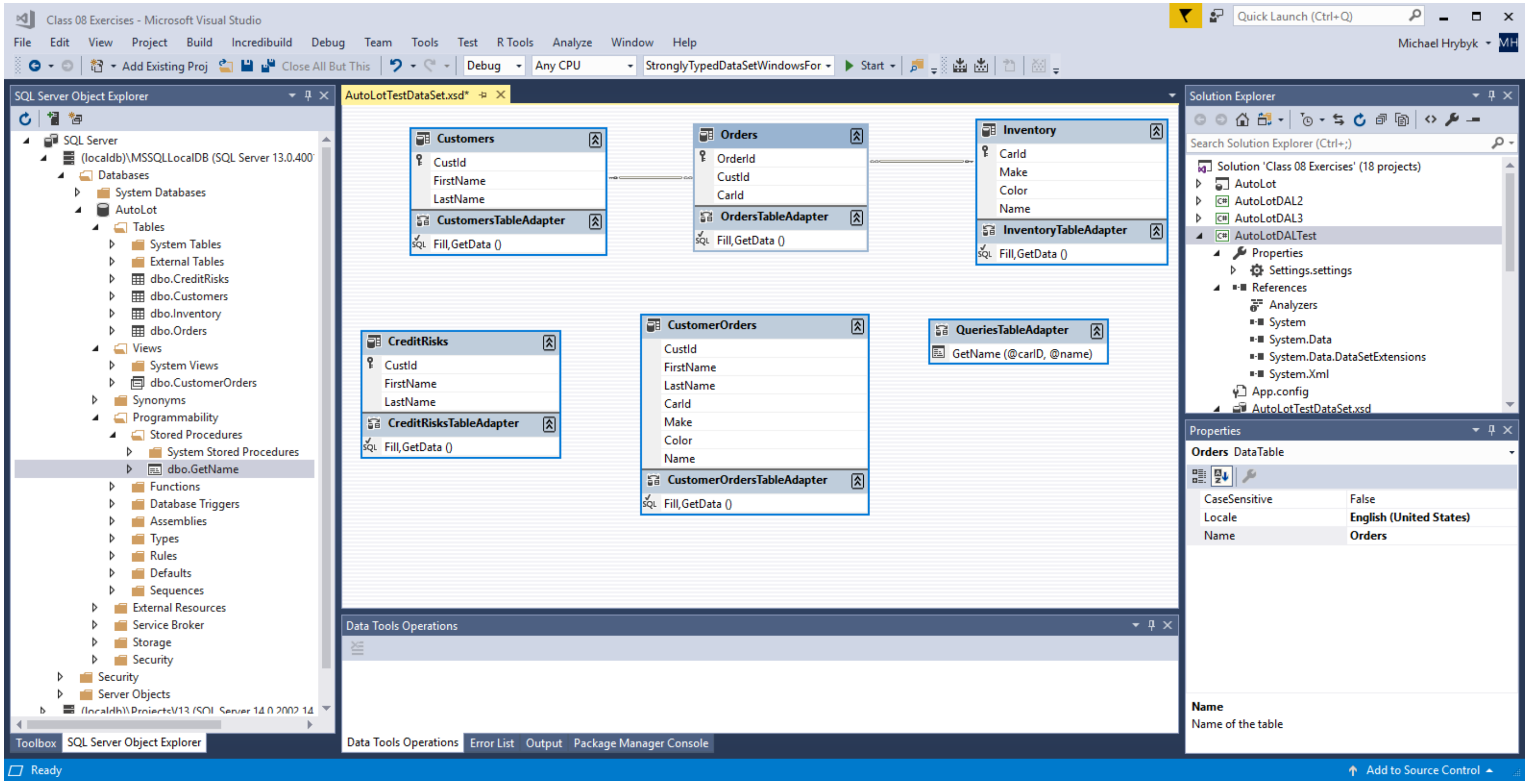
# Expand AutoLot DB in Explorer to show all tables



Use the Dataset Designer to visually create and edit typed datasets.
Drag database items from Server Explorer or the DataSet Toolbox onto the design surface, or right-click here to add new items.

# Create DAL library

- Drag tables, views, stored procedures to middle window
  - Rearrange as needed
- Save project
- Build
- Now ready to use!!



Customers: CustId, FirstName, LastName
CustomersTableAdapter: Fill,GetData ()

Orders: OrderId, CustId, CarId
OrdersTableAdapter: Fill,GetData ()

Inventory: CarId, Make, Color, Name
InventoryTableAdapter: Fill,GetData ()

CreditRisks: CustId, FirstName, LastName
CreditRisksTableAdapter: Fill,GetData ()

CustomerOrders: CustId, FirstName, LastName, CarId, Make, Color, Name
CustomerOrdersTableAdapter: Fill,GetData ()

QueriesTableAdapter: GetName (@carID, @name)

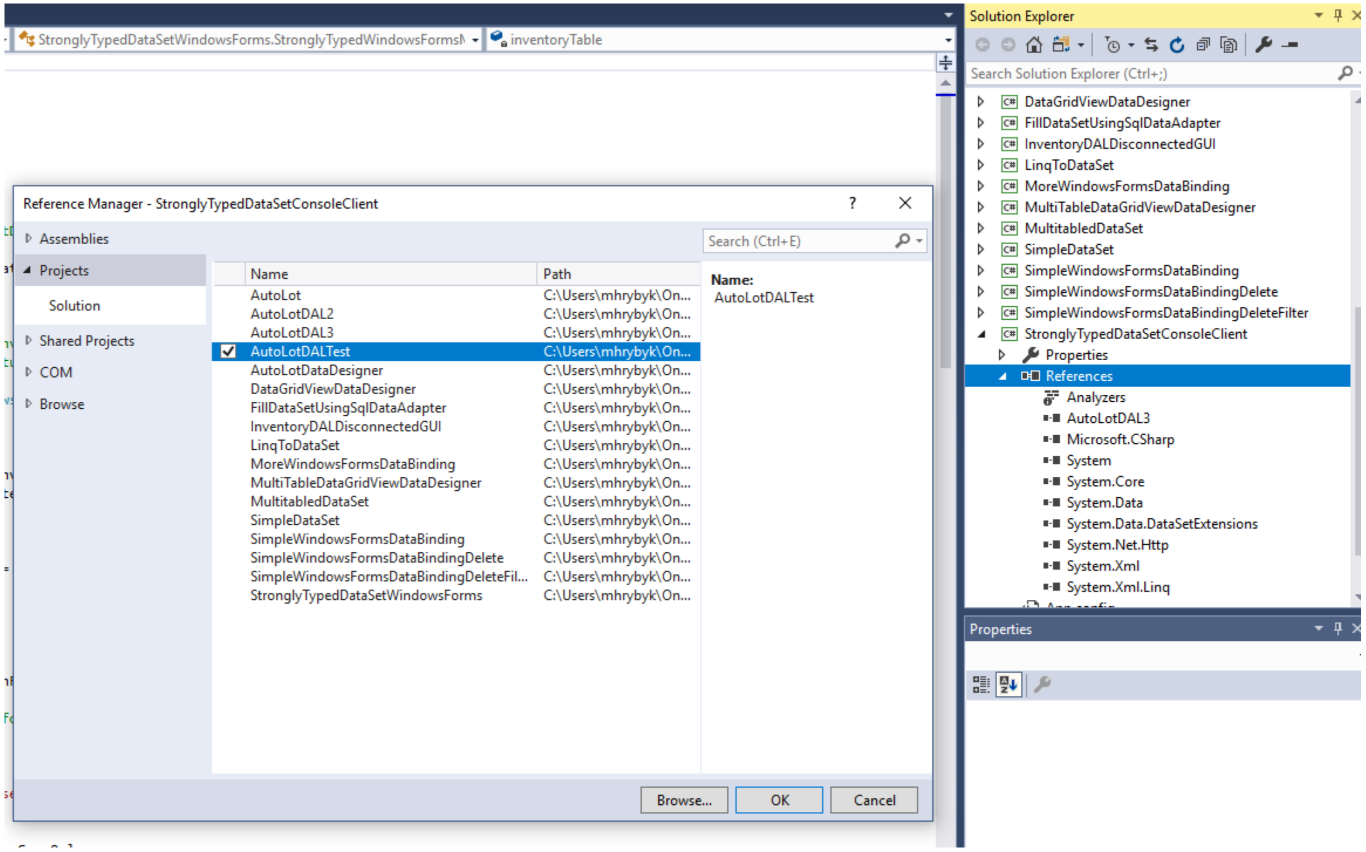
Orders DataTable

| | |
|---|---|
| CaseSensitive | False |
| Locale | English (United States) |
| Name | Orders |

**Name**
Name of the table

# Use in a program

- In StronglyTypedDataSetConsoleClient, add a reference to AutoLotDALTest



StronglyTypedDataSetWindowsForms.StronglyTypedWindowsFormsN | inventoryTable

Reference Manager - StronglyTypedDataSetConsoleClient

- Assemblies
- Projects
  - Solution
- Shared Projects
- COM
- Browse

Search (Ctrl+E)

| | Name | Path |
|---|---|---|
| | AutoLot | C:\Users\mhrybyk\On... |
| | AutoLotDAL2 | C:\Users\mhrybyk\On... |
| | AutoLotDAL3 | C:\Users\mhrybyk\On... |
| ✓ | AutoLotDALTest | C:\Users\mhrybyk\On... |
| | AutoLotDataDesigner | C:\Users\mhrybyk\On... |
| | DataGridViewDataDesigner | C:\Users\mhrybyk\On... |
| | FillDataSetUsingSqlDataAdapter | C:\Users\mhrybyk\On... |
| | InventoryDALDisconnectedGUI | C:\Users\mhrybyk\On... |
| | LinqToDataSet | C:\Users\mhrybyk\On... |
| | MoreWindowsFormsDataBinding | C:\Users\mhrybyk\On... |
| | MultiTableDataGridViewDataDesigner | C:\Users\mhrybyk\On... |
| | MultitabledDataSet | C:\Users\mhrybyk\On... |
| | SimpleDataSet | C:\Users\mhrybyk\On... |
| | SimpleWindowsFormsDataBinding | C:\Users\mhrybyk\On... |
| | SimpleWindowsFormsDataBindingDelete | C:\Users\mhrybyk\On... |
| | SimpleWindowsFormsDataBindingDeleteFil... | C:\Users\mhrybyk\On... |
| | StronglyTypedDataSetWindowsForms | C:\Users\mhrybyk\On... |

**Name:**
AutoLotDALTest

Browse... | OK | Cancel

Solution Explorer

Search Solution Explorer (Ctrl+;)

- DataGridViewDataDesigner
- FillDataSetUsingSqlDataAdapter
- InventoryDALDisconnectedGUI
- LinqToDataSet
- MoreWindowsFormsDataBinding
- MultiTableDataGridViewDataDesigner
- MultitabledDataSet
- SimpleDataSet
- SimpleWindowsFormsDataBinding
- SimpleWindowsFormsDataBindingDelete
- SimpleWindowsFormsDataBindingDeleteFilter
- StronglyTypedDataSetConsoleClient
  - Properties
  - References
    - Analyzers
    - AutoLotDAL3
    - Microsoft.CSharp
    - System
    - System.Core
    - System.Data
    - System.Data.DataSetExtensions
    - System.Net.Http
    - System.Xml
    - System.Xml.Linq

Properties

## In the program edit to make use of DAL

- Add using statements
- Change AutoLotDataSet to AutoLotTestDataSet throughout the program
- The class library can now be used in any program!

```
using AutoLotDALTest;
using AutoLotDALTest.AutoLotTestDataSetTableAdapters;
```

# Use of DataSet Designer

- You can use DataSet Designer to create a dataset within a project and use it with forms controls **without** using the wizard.
    - This makes your code more robust when changing the database.
- Create your form (view)
- Create the dataset using designer (model)
- Create DataTables and TableAdapters.
- Set controls Datasource property to a DataTable.
- Done!
- Examples:
    - StronglyTypedDataSet projects: Console, WindowsForms, WindowsFormsDesigner

# Use of LINQ

- Examples
  - MultiTableDataGridViewDesigner
  - WindowsFormsDataBinding
  - LinqToDataSetApp
- DataSet table AsEnumerable(), then most LINQ methods available
- Use of  Field<type>("ColumnName") to get strong typing and avoid casts
- CopyToDataTable() also useful when hydrating/binding LINQ result to DataSource

```
var ordersQuery =
from order in ordersTable
from car in inventoryTable
from customer in customersTable
where order.CustId == customer.CustId
where order.CarId == car.CarId
select new
{
    car.CarId,
    car.Name,
    car.Make,
    CustomerId = customer.CustId,
    CustomerName = customer.FirstName + " " + customer.LastName,
};
```