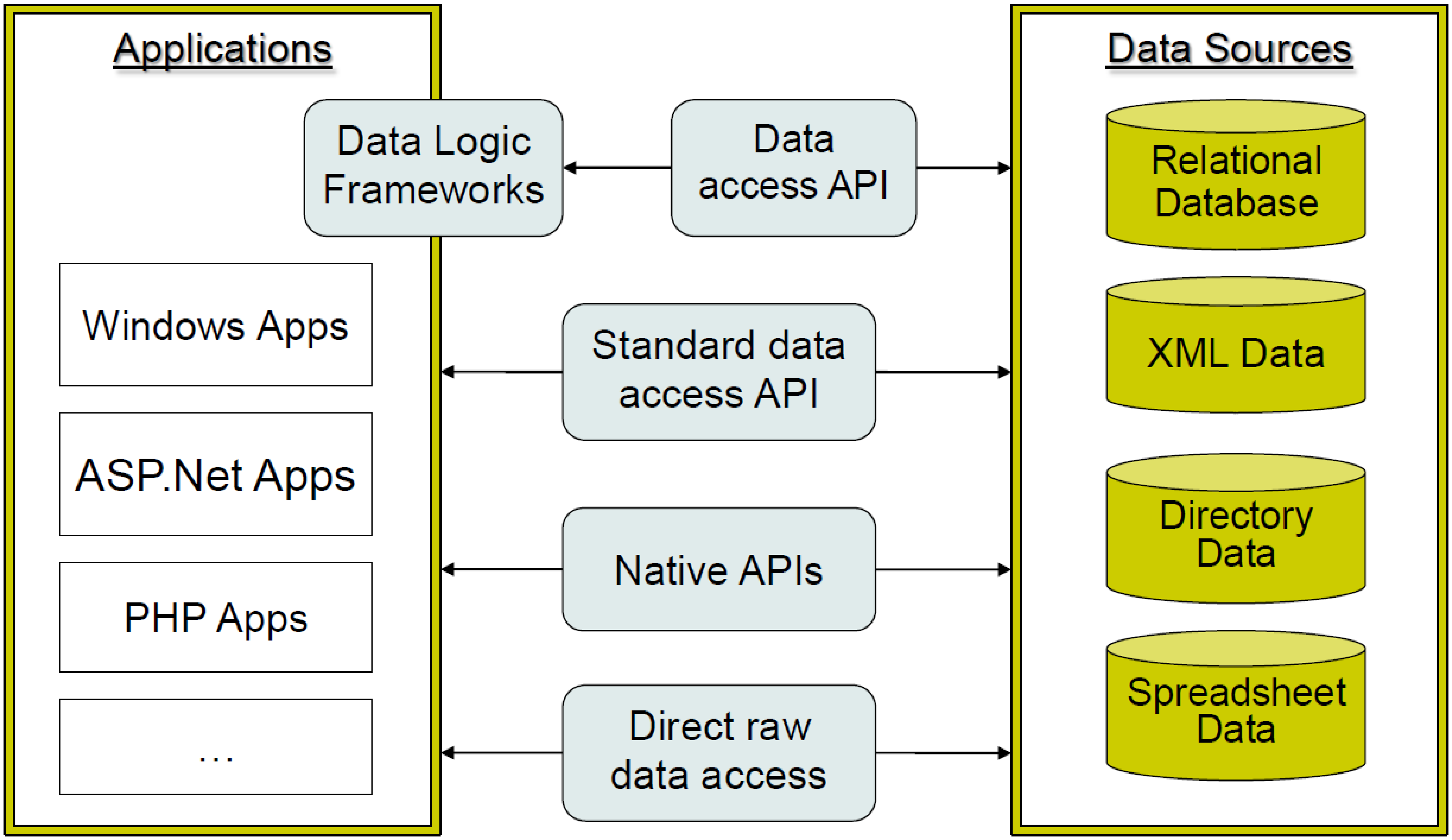
# CS108: Advanced Database - ASP.NET Programming

Lecture 03:

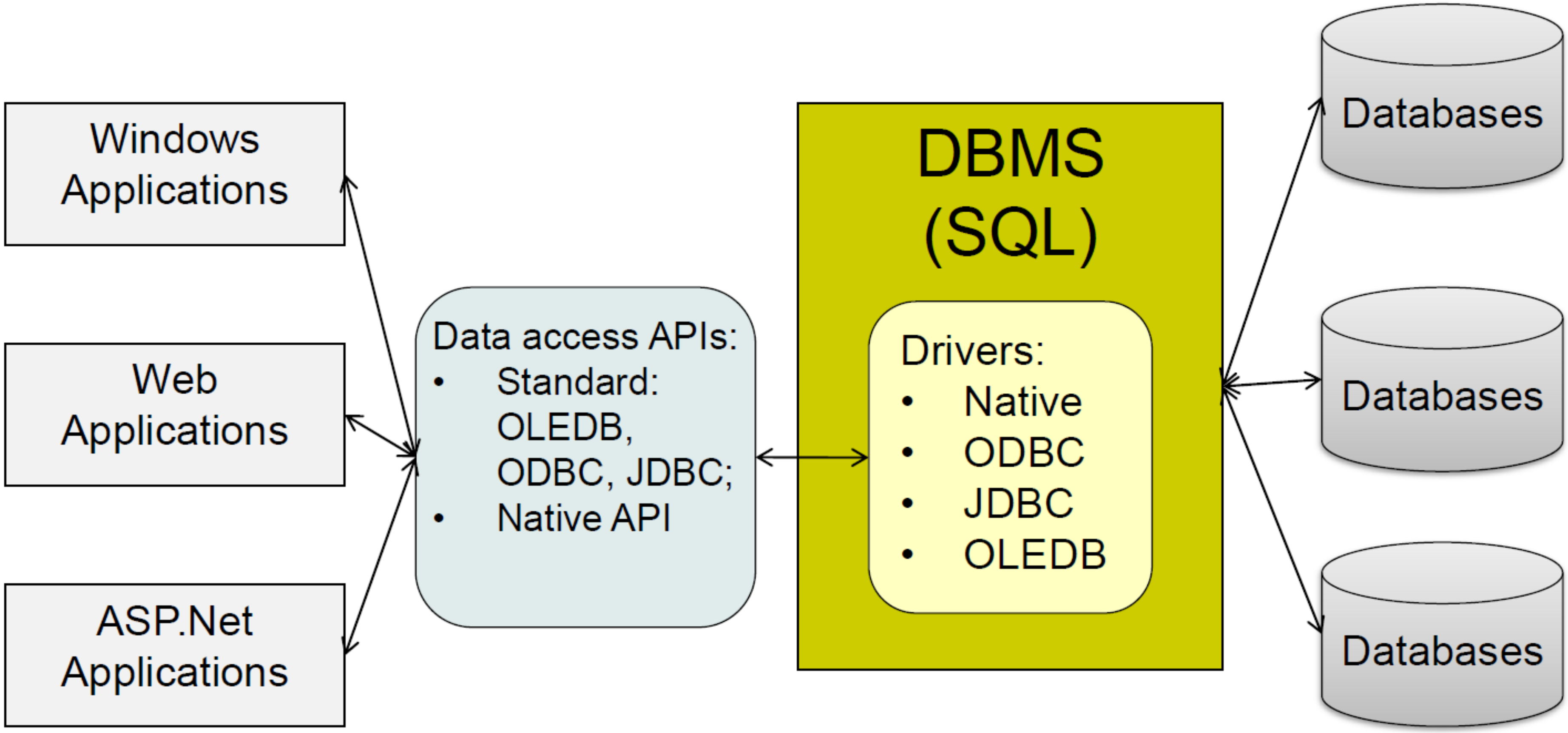
ADO.NET

# ADO.NET, ASP.NET and SQL Server

# Data Access: the Big Picture



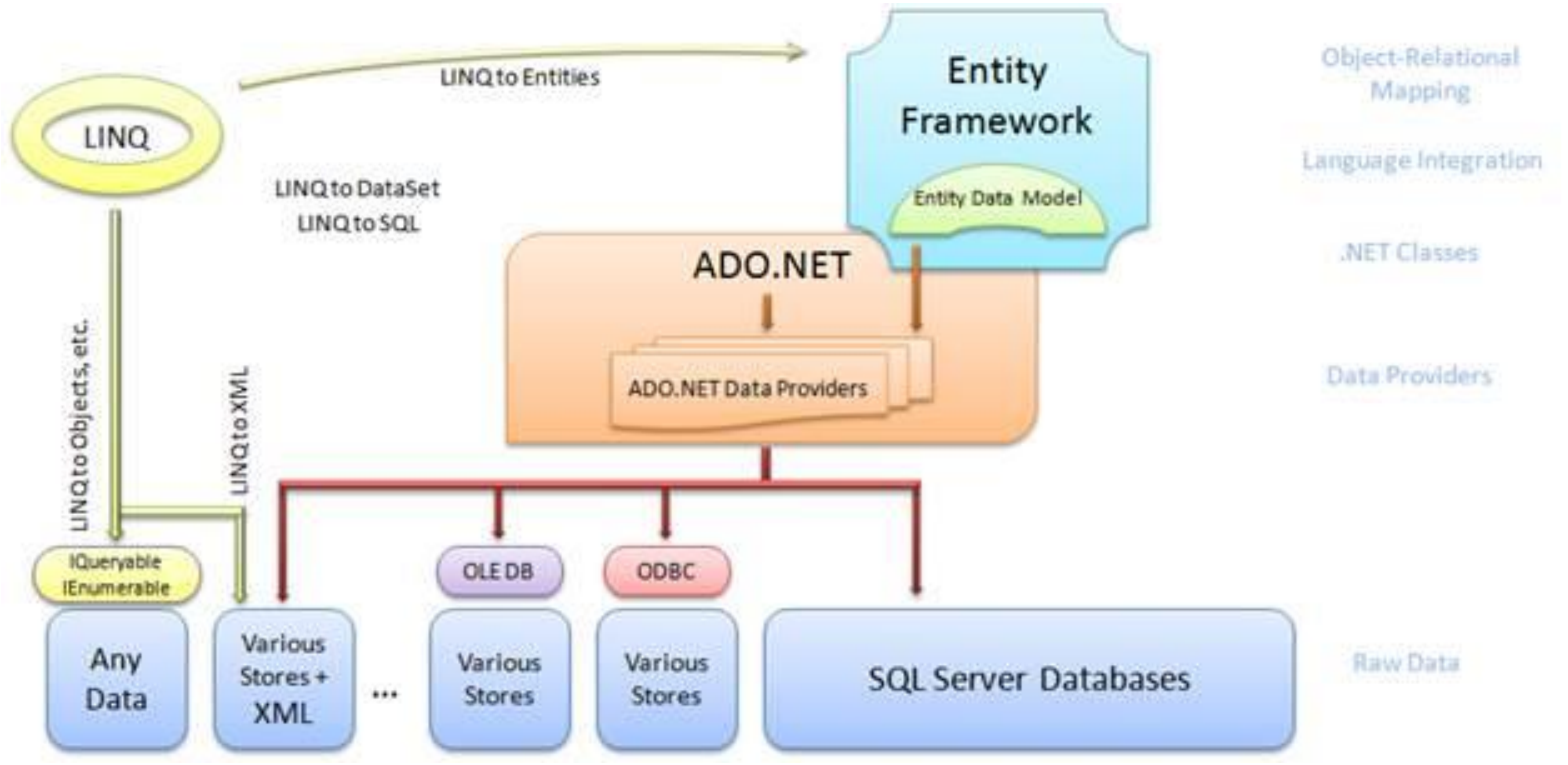
# Accessing Relational Database



## Standard Data Access APIs

- ODBC: Open Database Connectivity
  - ODBC is an uniform interface that allows applications to access data from a variety of relational Database Management Systems (DBMS).
- OLEDB: Object Linking and Embedding for Databases
  - OLE DB is a comprehensive set of COM interfaces for accessing a diverse range of data in a variety of data stores.
  - Designed as a higher-level replacement for ODBC.
- JDBC: Java Database Connectivity

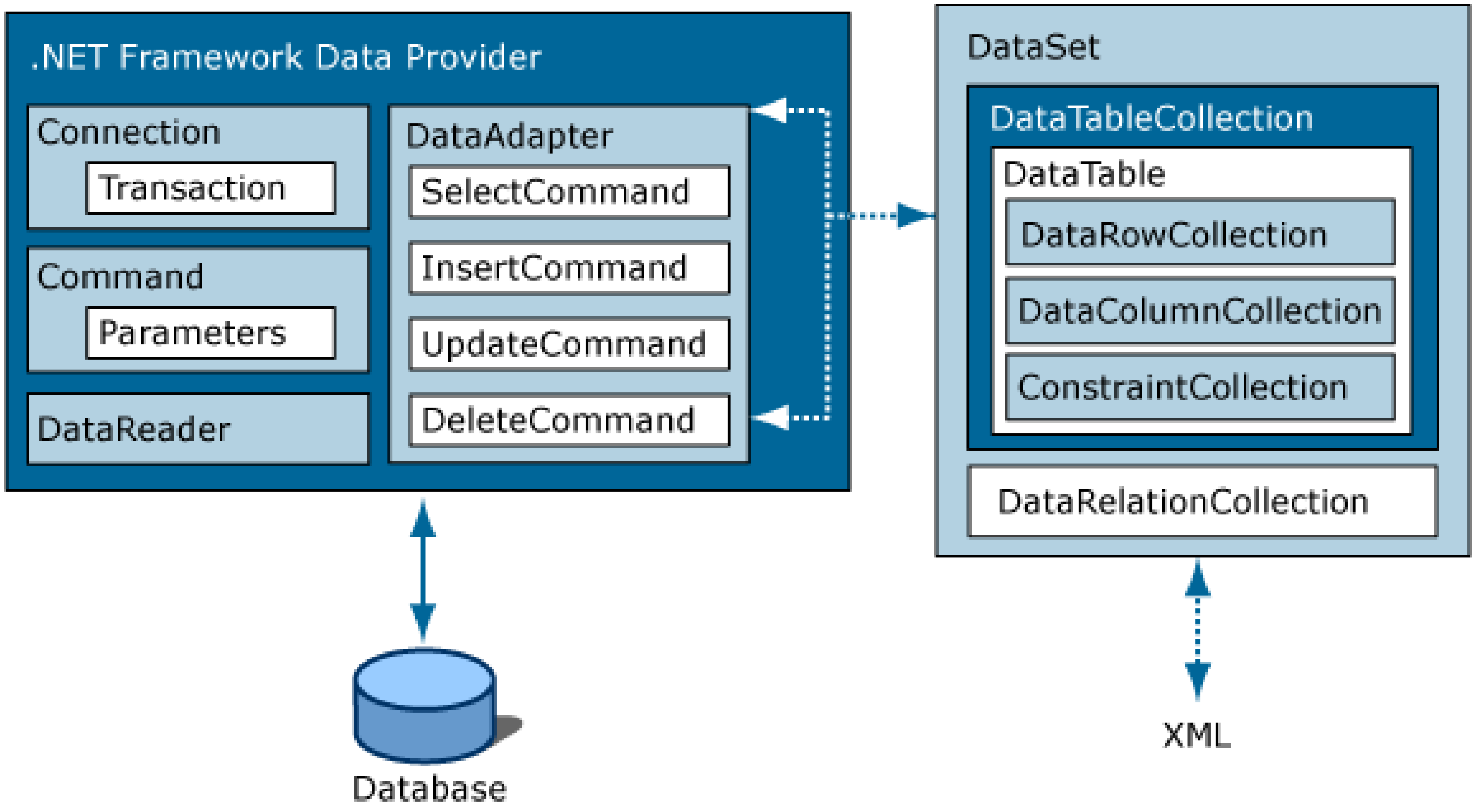
# Data Access for .NET Applications



## ADO.NET

- ADO.NET is a set of classes that expose data access services for .NET applications.
- ADO.NET provides consistent access to data sources such as SQL Server, Oracle, XML, and to data sources exposed through OLEDB and ODBC.

# ADO.NET Architecture

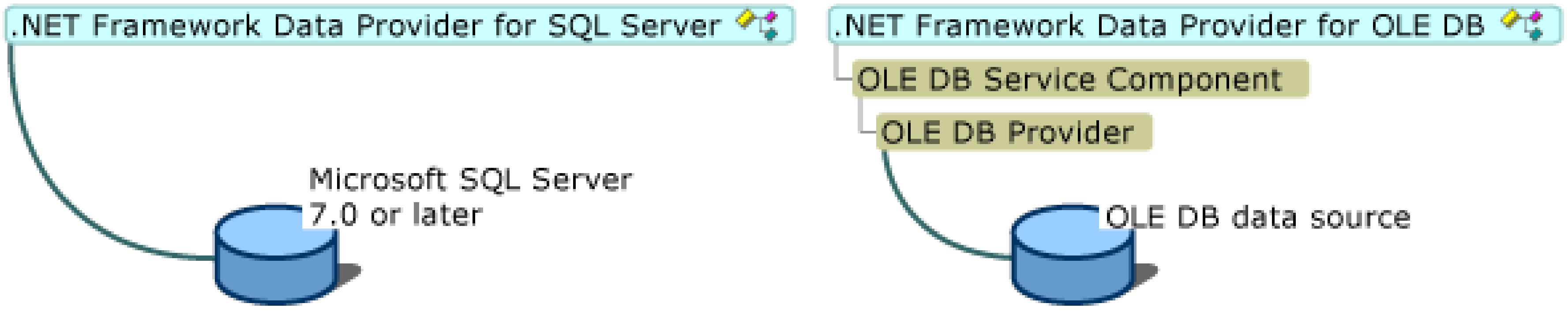


## .NET Data Providers

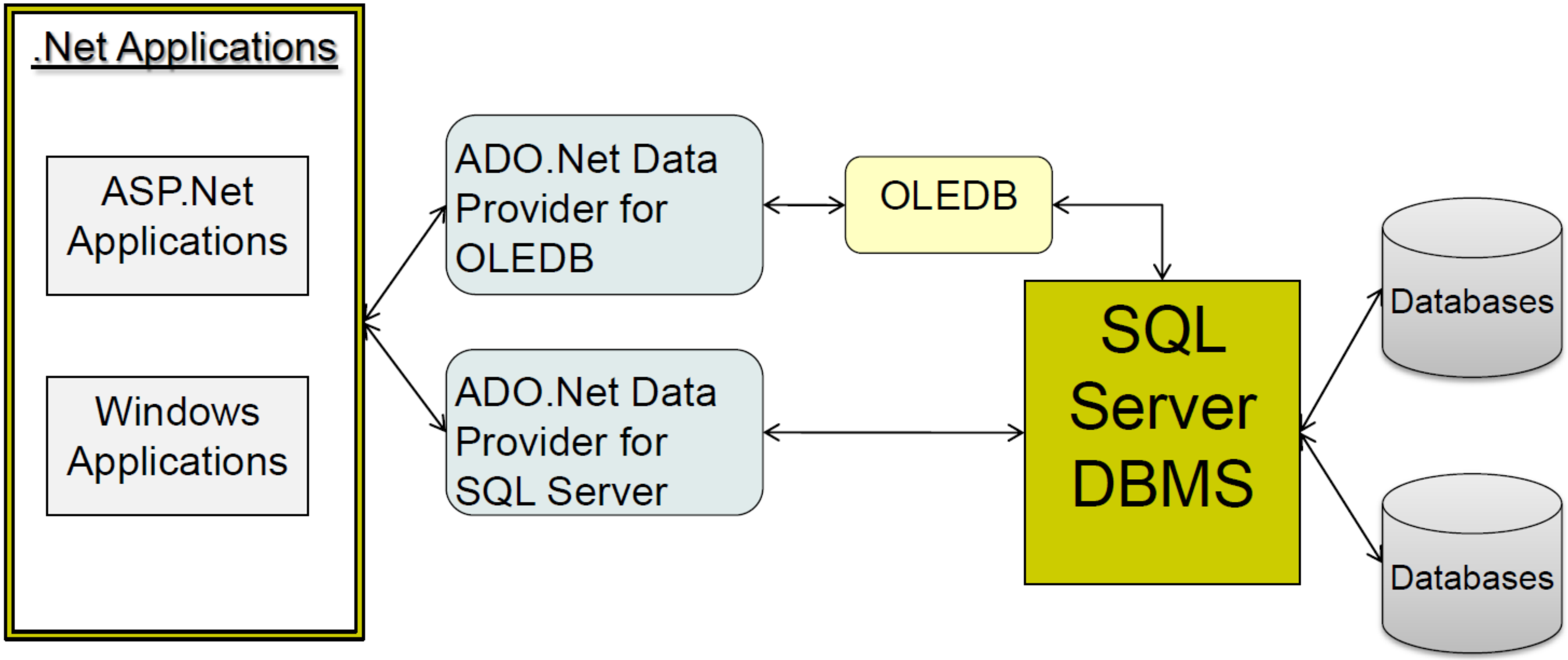
- A .NET Framework data provider is used for connecting to a database, executing commands, and processing results.
- .NET Framework data providers include:
  - Data Provider for SQL Server
  - Data Provider for OLE DB
  - Data Provider for ODBC
  - Data Provider for Oracle
  - EntityClient Provider

# .NET Data Provider for SQL Server

- The .NET Framework Data Provider for SQL Server(System.Data.SqlClient) uses its own protocol to communicate with SQL Server.
- It is optimized to access a SQL Server directly without adding an OLE DB or ODBC layer.


.NET Framework Data Provider for SQL Server
Microsoft SQL Server 7.0 or later
.NET Framework Data Provider for OLE DB
OLE DB Service Component
OLE DB Provider
OLE DB data source

# SQL Server for .NET Applications



## Using Namespaces

- Use the Imports or using statement to import namespaces.

```
using System.Data;
using System.Data.SqlClient;
```

- Namespaces used with ADO.NET include:
  - System.Data
  - System.Data.SqlClient
  - System.Data.OleDb

# Using Database for .NET Apps

- Steps to access a database

| Step | Base Class | Description |
|---|---|---|
| Connecting to database | Connection | Establishes a connection to a specific data source. |
| Preparing commands | Command | Executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection. |
| Processing results | DataReader | Reads a forward-only, read-only stream of data from a data source. |

# A Quick Database Query Example

Step 1: Use SqlConnection class to establish a connection to an SQL Server database. A connection string is supplied.

```
SqlConnection con = new SqlConnection("Server=(local); Data
Source=A212-SLLUO\\SQLEXPRESS;Initial
Catalog=AdventureWorks2014;Integrated Security=True");
```

Step 2: Build a new command with simple and static SQL statement. Associate the command with the connection.

```
SqlCommand cmd = new SqlCommand("SELECT TOP 20 Name
FROM Production.Product ORDER BY Name");
cmd.Connection = con;
```

Step 3: Open the connection and execute the command. Use a DataReader to process the result.

```
con.Open();
SqlDataReader data = cmd.ExecuteReader();
```

Column name

```
while( data.Read() )
{
   Response.Write("<p>" + data["Name"] + "</p>");
}
con.Close();
```

# Command

- Simple SQL command text
  - Supply SQL statements as the command text

```
SqlCommand cmd = new SqlCommand("SELECT TOP 20 Name
FROM Production.Product ORDER BY Name");
cmd.Connection = con;
```

Don't forget the association.

- Dynamic command text
  - Command text can be an expression with constant strings and variables.

```
int number = 10;
SqlCommand cmd = new SqlCommand("SELECT TOP" + number + "Name
FROM Production.Product ORDER BY Name");
cmd.Connection = con;
```

## Command Execution

- Depending on the command type and return value, four methods are provided for a command object.

| Methods | Return Value |
|---|---|
| ExecuteReader() | Returns a DataReader object. |
| ExecuteScalar() | Returns a single scalar value. |
| ExecuteNonQuery() | Executes a command that does not return any rows. |
| ExecuteXMLReader() | Returns an XmlReader. Available for a SqlCommand object only. |

# Command Parameters

- Using parameters is a better practice than using variables.
  - Parameters are restricted to type, size, and other constraints defined for the column value. It’s a way to validate user input.
  - Preventing SQL Injection.
- Example:

Use @ to define a parameter where a value is supplied.

```
SqlCommand cmd = new SqlCommand("SELECT TOP 20 Name FROM
   Production.Product WHERE ListPrice < @price ORDER BY Name");
cmd.Connection = con;

SqlParameter sqlpara = new SqlParameter("@price", 10);
cmd.Parameters.Add(sqlpara);
```

Don’t forget to add the parameter to the command.

## Injection Attacks

- Example: Prompt for user/pass, and do lookup:

```
SELECT * FROM users
WHERE    user = u AND password = p;
```

- We expect to get input of something like:
  - **user**: mjohnson
  - **pass**: topsecret

```
SELECT * FROM users
WHERE user = 'mjohnson' AND password = 'topsecret';
```

# Injection Attacks

- Example: Prompt for user/pass, and do lookup:

```
SELECT * FROM users
WHERE    user = u AND password = p;
```

- Consider another input:
  - **user**: ' OR 1=1 OR user = ‘
  - **pass**: ' OR 1=1 OR pass = '

```
SELECT * FROM users
WHERE  user = ''
       OR 1=1
       OR user = ''
       AND password = ''
       OR 1=1
       OR pass = '';
```

```
SELECT * FROM users
WHERE    user = u AND password = p;
```

- Consider another input:
  - **user**: your-boss' OR 1=1 #
  - **pass**: abc

```
SELECT * FROM users
WHERE user = 'your-boss'
      OR 1=1 #' AND password = 'abc';
```

- Consider another input:
  - **user**: your-boss
  - **pass**: ' OR 1=1 OR pass = '

```
SELECT * FROM users
WHERE user = 'your-boss'
      AND password = ''
      OR 1=1
      OR pass = '';
```

# Multi-Command Injection Attacks

```
SELECT * FROM users
WHERE    user = u AND password = p;
```

- Consider another input:
  - **user**: '; DELETE FROM users WHERE user = 'abc'; SELECT FROM users WHERE password = ‘
  - **pass**: abc

```
SELECT * FROM users WHERE user = '';
DELETE FROM users WHERE user = 'abc';
SELECT FROM users WHERE password = '' AND password = 'abc';
```

  - **user**: '; DROP TABLE users; SELECT FROM users WHERE password = '

```
SELECT * FROM users WHERE user = '';
DROP TABLE users;
SELECT FROM users WHERE password = '' AND password = 'abc';
```

## Preventing Injection Attacks

- Ultimate source of problem: quotes
- Soln 1: don’t allow quotes!
  - Reject any entered data containing single quotes
  - Q: Is this satisfactory?
  - Does Amazon need to sell O’Reilly books?
- Soln 2: escape any single quotes
  - Replace any ' with a '' or \'

# DataReader

- Getting values of each row
  - Use Getter methods: GetInt32(), GetString(), etc.
  - Use collections: reader[1], reader["Name"]
- Example:

Use ExecuteScalar if only one value is returned.

```
SqlDataReader data = cmd.ExecuteReader();
while( data.Read() )
{
   Response.Write("<p>" + data["Name"] + "</p>");
}
con.Close();
```

The Read() method will retrieve one row at a time.

Use either column index number or column name to get a value for the current row..

# Database Modification

- Use ExecuteNonQuery() method for *SQL UPDATE*, *INSERT INTO*, and *DELETE FROM* statements
- Example:

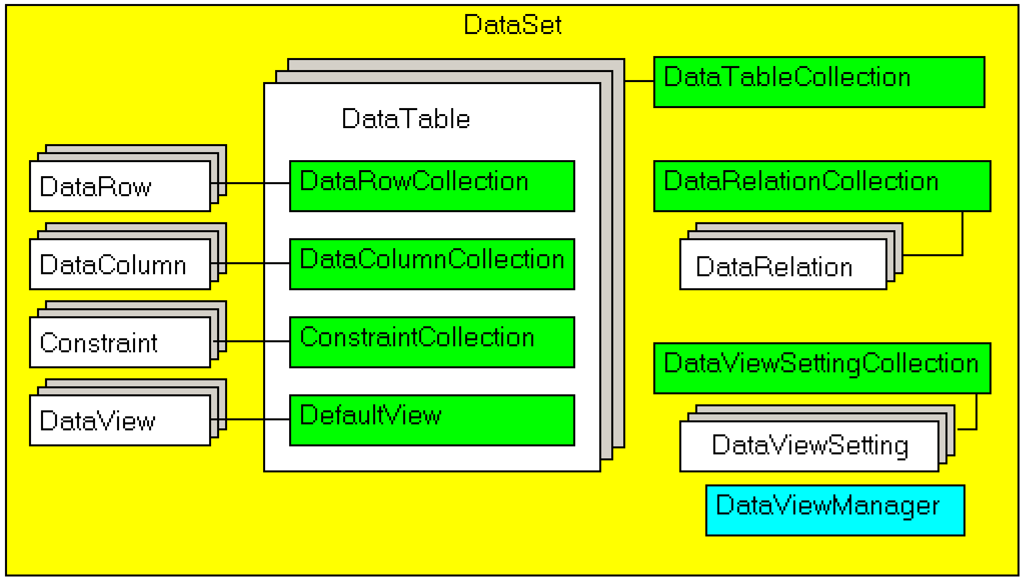
Connection is the same; command is an "Update" SQL

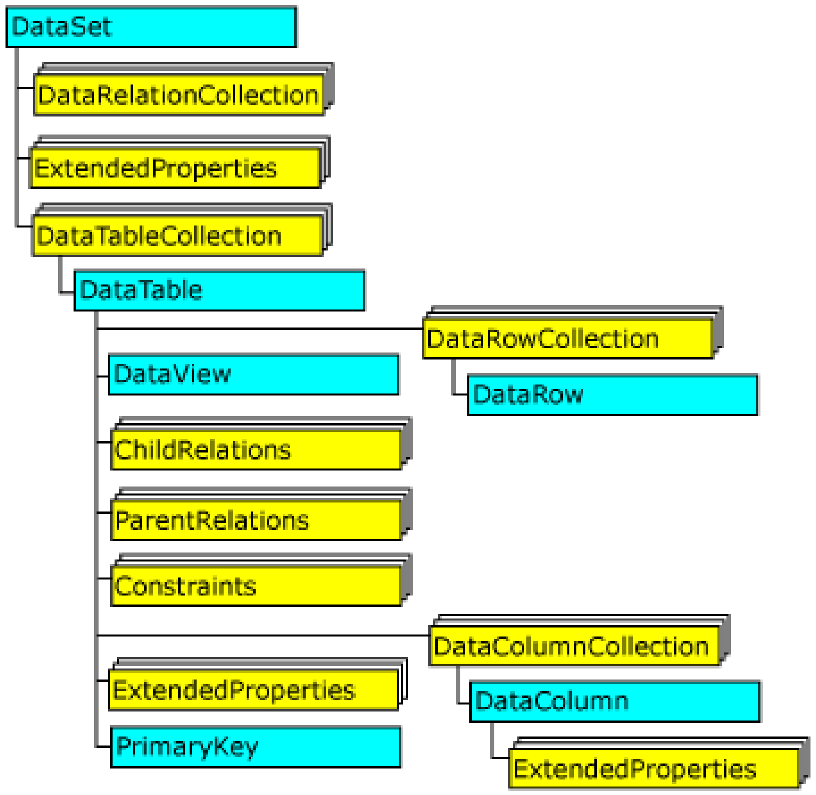
```
SqlCommand cmd  = new SqlCommand();
cmd.CommandText = "UPDATE Production.Product SET ListPrice =
                   @price WHERE ProductID = 1";
cmd.Connection = con;
SqlParameter sqlpara = new SqlParameter("@price", 999.0);
cmd.Parameters.Add(sqlpara);
con.Open();
int result = cmd.ExecuteNonQuery();
con.Close();

if (result == 1) Response.Write("Update succeed");
else Response.Write("Update failed");
```

ExecuteNonQuery() retuens an integer indicating number of results affected. If it is 0, very likely the execution failed.

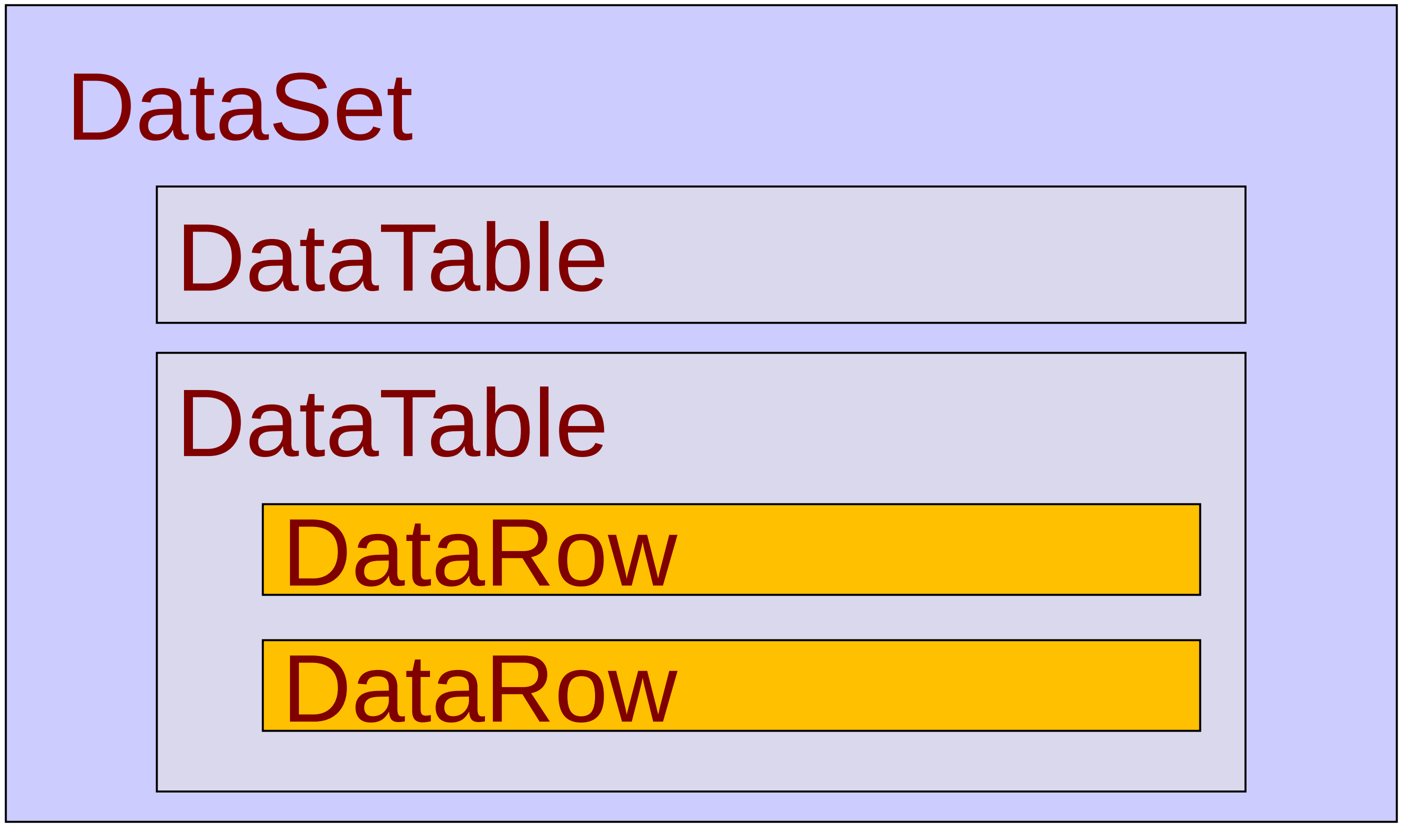
# DataSet

- The DataSet object is a disconnected, in-memory representation of a consistent relational data model.





## Sets, Tables and Rows



# Populating DataSet from Database

- Example: use DataAdapter to fill a DataSet

```
SqlConnection con = new SqlConnection("Server=(local); ...");

SqlCommand cmd = new SqlCommand(@"SELECT TOP 20 Name
 FROM Production.Product ORDER BY Name");
cmd.Connection = con;
con.Open();

SqlDataAdapter sda = new SqlDataAdapter(cmd);

DataSet  ds = new DataSet();
sda.Fill(ds, "Product");

foreach (DataRow dr in ds.Tables["Product"].Rows)
{ Response.Write("<p>" + dr[0] + "</p>"); }

con.Close();
```

Connection and command remain the same.

The fill method will automatically create the default table structure and fill the data.

Read each row in the data table, referring values by number index or column name.

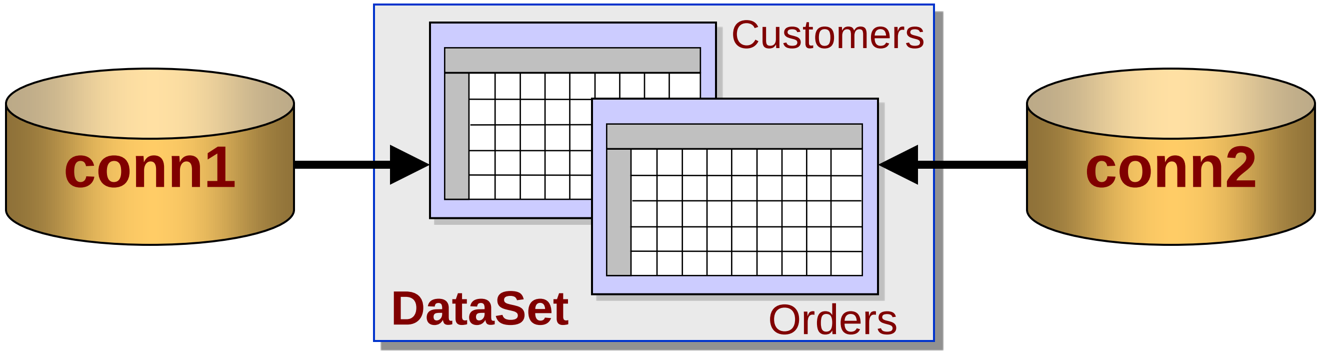
## Storing Multiple Tables

- **Add the first table**

```
daCustomers = new SqlDataAdapter
    ("select * from Customers", conn1)
daCustomers.Fill(ds, "Customers")
```

- **Add the subsequent table(s)**

```
daOrders = new SqlDataAdapter
    ("select * from Orders", conn2)
daOrders.Fill(ds, "Orders")
```



# Reading Data from a DataSet

- Use a DataView object to further manipulating the data in a DataSet

```
...
SqlDataAdapter sda = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
sda.Fill(ds, "Product");
```

Get a default view of all data.

```
DataView dv = ds.Tables[0].DefaultView;
```

Apply sorting and filtering constraints. The format of these constraints resemble SQL ORDER BY and WHERE clauses.

```
dv.Sort = "Name Desc";
dv.RowFilter = "Name LIKE 'B%'";
```

Use a 2-D array like format to refer to the data. The first index is the row number and the second one is the column number or name.

```
for(int i = 0; i < dv.Count; i++)
   Response.Write("<p>" + dv[i][0] + "</p>");
```

# Insert a Record into DataTable

```
...

SqlDataAdapter sda = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
sda.Fill(ds, "Product");

DataTable table ds.Tables["Product"];
DataRow row = table.NewRow();
row["Name"] = ...;
row["ProductID"] = ...;
row["Price"] = ...;
...
Table.Rows.Add(row);

sda.Update(table);
```

only writes the ones that were changed

# DataReader or DataSet

- DataReader returns data in a forward-only, read-only manner.
  - Fast processing, improving application performance.
- Use a DataSet to do the following:
  - *Cache data* locally in your application so that you can manipulate it. If you only need to read the results of a query, the DataReader is the better choice.
  - Interact with data dynamically such as binding to a web server control or combining and relating data from multiple sources.
  - Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

# ASP.Net Data Binding

## Data Binding Controls

- Simple data controls (ListControl based)
  - DropDownList
  - CheckBoxList, RadioButtonList
  - ListBox
- Composite data controls
  - ListView
  - DataList
  - GridView
  - Repeater, DetailsView, FormView

## Data Sources

- Generally, any class or component that implements the IList interface is a valid data source.
- Classes that support the IList interface in the .NET:
  - Collections
    - Array, ArraList, List<>, HashTable, Dictionary, etc.
  - ADO.NET
    - DataReader, DataSet, DataTable, DataView, DataColumn
- DataSource Controls
  - Linq, Entity, Object, SQL Server, XML, etc.

## Advantages and Disadvantages

- Advantages
  - To write data driven applications quickly, with less code and fast execution.
  - .NET automatically generate data binding code in the background.
  - Control over the data binding process by using events.
- Disadvantages
  - More optimized code can be written by using the unbound or traditional methods.
  - Complete flexibility can only be achieved by using the unbound approach.

# General Binding Process

| Step | List Controls + Binding Code | Composite Controls + Binding Code | List Controls + Data Source Controls | Composite Controls + Data Source Controls |
|---|---|---|---|---|
| 1. Preparing a data source | Programmatically get the data source object. | | Declaratively configure a data source control. | |
| 2. Defining a web control | Define control styles | Define control styles, templates and data binding fields | Define control styles | Define control styles, templates and data binding fields |
| 3. Linking data source to a control | Programmatically set the control's DataSource property (and other relevant binding properties, such as DataMember, DataTextField, DataValueField) | | Declaratively set the control's DataSourceID property | |
| 4. Binding data | Programmatically call the DataBind() method | | (N/A) | |

## Displaying DataSet Data in List-Bound Controls

- Set the properties

| Property | Description |
|---|---|
| DataSource | The DataSet containing the data |
| DataMember | The DataTable in the DataSet |
| DataTextField | The field in the DataTable that is displayed |
| DataValueField | The field in the DataTable that becomes the value of the selected item in the list |

- Fill the DataSet, then call the DataBind method

```
DataAdapter.Fill(ds);
Employees.DataBind();
```

# DropDownList + Collection

```
protected void Page_Load(object sender, EventArgse)
{

   List<String> list = new List<string>();
   list.Add("Ford");
   list.Add("GM");
   list.Add("Chrysler");



   this.DropDownList2.DataSource = list;
   this.DropDownList2.DataBind();

   this.RadioButtonList2.DataSource = list;
   this.RadioButtonList2.DataBind();
}
```

Create a list collection of strings.

Set the control’s DataSource property to the collection object.

Call the DataBind method.

The same data source can be bound to multiple controls.

# DropDownList + ADO.NET Data Reader

```
SqlConnection con = ...;
SqlDataReader source;

using(con)
{
    con.Open();
    SqlCommandcmd   = new SqlCommand();
    cmd.CommandText = "SELECT DISTINCT Name FROM ...";
    cmd.Connection  = con;

    source = cmd.ExecuteReader();
    this.DropDownList1.DataSource = source;
    this.DropDownList1.DataTextField  = "Name";
    this.DropDownList1.DataValueField = "Name";
    this.DropDownList1.DataBind();
    source.Close();
}
```

database connection.

DataReader can a data source.

Column names

DataTextFieldis what's being displayed in the dropdown list; DataValueFieldis the real value of a list item.

# List Controls + ADO.NET DataSet

```
DataSet source = ...;

this.DropDownList1.DataSource = source;
this.DropDownList1.DataMember = "Product";
this.DropDownList1.DataTextField = "Name";
this.DropDownList1.DataValueField = "ProductID";
this.DropDownList1.DataBind();

this.RadioButtonList1.DataSource = source.Tables[0];
this.RadioButtonList1.DataTextField = "Name";
this.RadioButtonList1.DataValueField = "ProductID";
this.RadioButtonList1.DataBind();

this.BulletedList1.DataSource = source.Tables[0].DefaultView;
this.BulletedList1.DataTextField = "Name";
this.BulletedList1.DataBind();
```

load the database data into a DataSet.

When a Dataset is the data source; set the DataMember property to a *table* in the DataSet.

DataTable can be directly set as a data source.

Column names

DataView can be used as a data source.

## Composite Data Controls

- Composite data controls are not directly transformed to a single HTML element
  - Consist of different templates which is used to define layout and style flexibly
  - More data fields can be bound to controls
- Major composite data controls
  - Repeater: the simplest data control that repeats for each data item
  - DataList: can repeat data items in a flexible layout and style
  - ListView: the most powerful data control
  - GridView: presents data in a table

## Templates and Data Binding Fields

- Composite controls use templates to define its layout and style.
- Common templates include:
  - ItemTemplate: for each data item in the collection or table
  - AlternatingTemplate: defined to dinstinguising odd and even number data items
- Data binding fields: to bind a data field in templates; the value of the data field (an expression) changes for each item in the data source.
  - <%# Eval() %>: output binding
  - <%# Bind() %>: bi-directional binding, usually used for user input controls like textbox.

# Example: Repeater Control

```
<asp:RepeaterID = "Repeater1" runat = "server">
  <HeaderTemplate><h1>Books</h1></HeaderTemplate>
  <FooterTemplate><h1>End</h1></FooterTemplate>

  <ItemTemplate>
    <%# Container.ItemIndex + 1 %>. <%# Eval("Name") %> :
    <%# Eval("ListPrice") %>
  </ItemTemplate>

  <AlternatingItemTemplate>
   -<%# "Color : " + Eval("Color") %>
  </AlternatingItemTemplate>

  <SeparatorTemplate><br/></SeparatorTemplate>
</asp:Repeater>
```

Templates

ItemTemplateis the major template for data item output.

A mixture of text, static HTML tags and data binding fields in templates.

This template defines anything goes between each data item output.

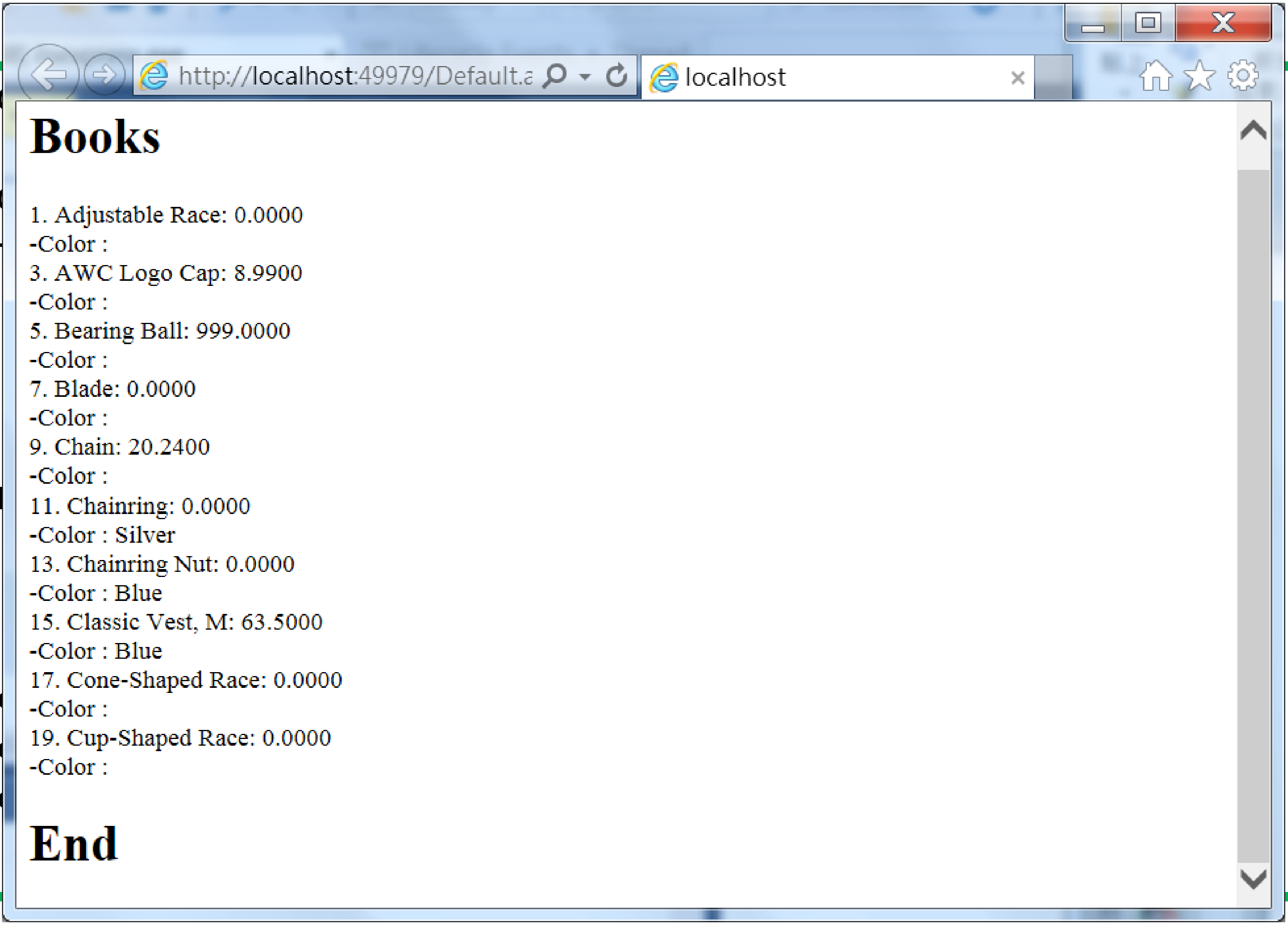
Don’t forget to set the DataSource property and call the DataBind() method in the code-behind page.

# Example: Repeater Control

```
SqlConnection con = new SqlConnection("Server=(local); ...

SqlCommand cmd = new SqlCommand(@"SELECT TOP 20 Name,
ListPrice, Color FROM Production.Product ORDER BY Name");
cmd.Connection = con;
con.Open();

SqlDataAdapter sda = new SqlDataAdapter(cmd);
        DataSet ds = new DataSet();
sda.Fill(ds, "Product");

Repeater1.DataSource = ds;
Repeater1.DataMember = "Product";
Repeater1.DataBind();

con.Close();
```

# Example: Repeater Control

```
Sql
Sql
Lis
cmd
con
Sql
sda
Rep
Rep
Rep
con
```



http://localhost:49979/Default.a

localhost

## Books

1. Adjustable Race: 0.0000
-Color :
3. AWC Logo Cap: 8.9900
-Color :
5. Bearing Ball: 999.0000
-Color :
7. Blade: 0.0000
-Color :
9. Chain: 20.2400
-Color :
11. Chainring: 0.0000
-Color : Silver
13. Chainring Nut: 0.0000
-Color : Blue
15. Classic Vest, M: 63.5000
-Color : Blue
17. Cone-Shaped Race: 0.0000
-Color :
19. Cup-Shaped Race: 0.0000
-Color :

## End

## Example: DataList Control

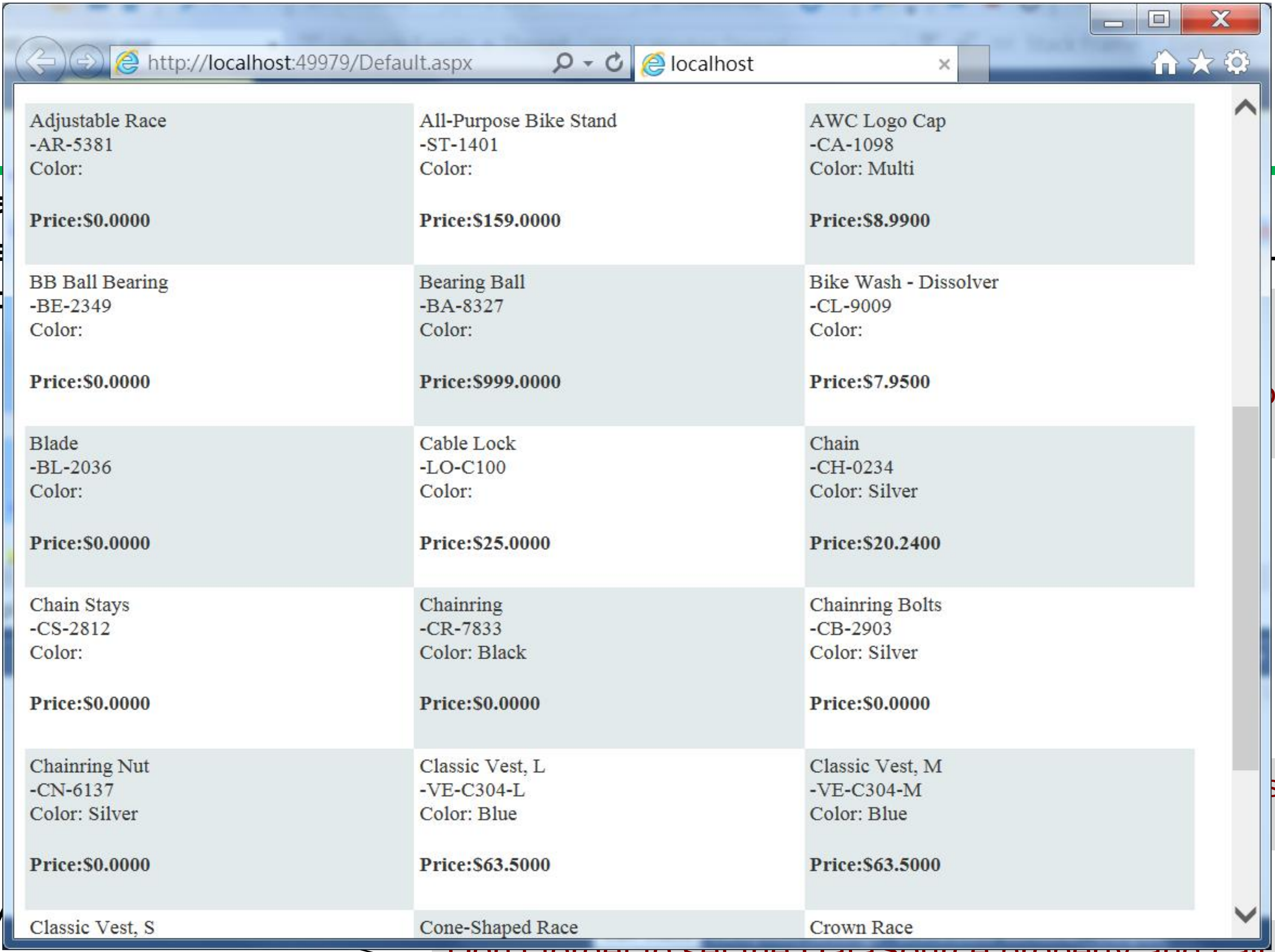
```
<asp:DataList ID="DataList1" runat="server" Width="900px"
CellPadding="4" RepeatColumns="3" RepeatDirection="Horizontal"
ForeColor="#333333">

   <ItemTemplate>
       <%# Eval("Name") %><br/>
       -<%# Eval("ProductNumber") %><br/>
       Color: <%# Eval("Color") %>
       <h4>Price:$<%# Eval("ListPrice") %></h4>
   </ItemTemplate>

   <ItemStyle BackColor="#E3EAEB" Width="300px"
   VerticalAlign="Top" />

   <AlternatingItemStyle BackColor="White" />
</asp:DataList>
```

DataList control can present data in a grid or flow list layout. Use the “RepeatColumns” and “Repeat Direction” properties to set grid size.

Compared to Repeater, DataListprovides style templates.

Don’t forget to set the DataSource property and call the DataBind() method in the code-behind page.

```
<a
Ce                                                  ."
Fo
```



| | | |
|---|---|---|
| Adjustable Race<br>-AR-5381<br>Color:<br><br>**Price:$0.0000** | All-Purpose Bike Stand<br>-ST-1401<br>Color:<br><br>**Price:$159.0000** | AWC Logo Cap<br>-CA-1098<br>Color: Multi<br><br>**Price:$8.9900** |
| BB Ball Bearing<br>-BE-2349<br>Color:<br><br>**Price:$0.0000** | Bearing Ball<br>-BA-8327<br>Color:<br><br>**Price:$999.0000** | Bike Wash - Dissolver<br>-CL-9009<br>Color:<br><br>**Price:$7.9500** |
| Blade<br>-BL-2036<br>Color:<br><br>**Price:$0.0000** | Cable Lock<br>-LO-C100<br>Color:<br><br>**Price:$25.0000** | Chain<br>-CH-0234<br>Color: Silver<br><br>**Price:$20.2400** |
| Chain Stays<br>-CS-2812<br>Color:<br><br>**Price:$0.0000** | Chainring<br>-CR-7833<br>Color: Black<br><br>**Price:$0.0000** | Chainring Bolts<br>-CB-2903<br>Color: Silver<br><br>**Price:$0.0000** |
| Chainring Nut<br>-CN-6137<br>Color: Silver<br><br>**Price:$0.0000** | Classic Vest, L<br>-VE-C304-L<br>Color: Blue<br><br>**Price:$63.5000** | Classic Vest, M<br>-VE-C304-M<br>Color: Blue<br><br>**Price:$63.5000** |
| Classic Vest, S | Cone-Shaped Race | Crown Race |

on”

s

```
</
```

Don't forget to set the DataSource property and call the DataBind() method in the code-behind page.

## Example: ListView Control

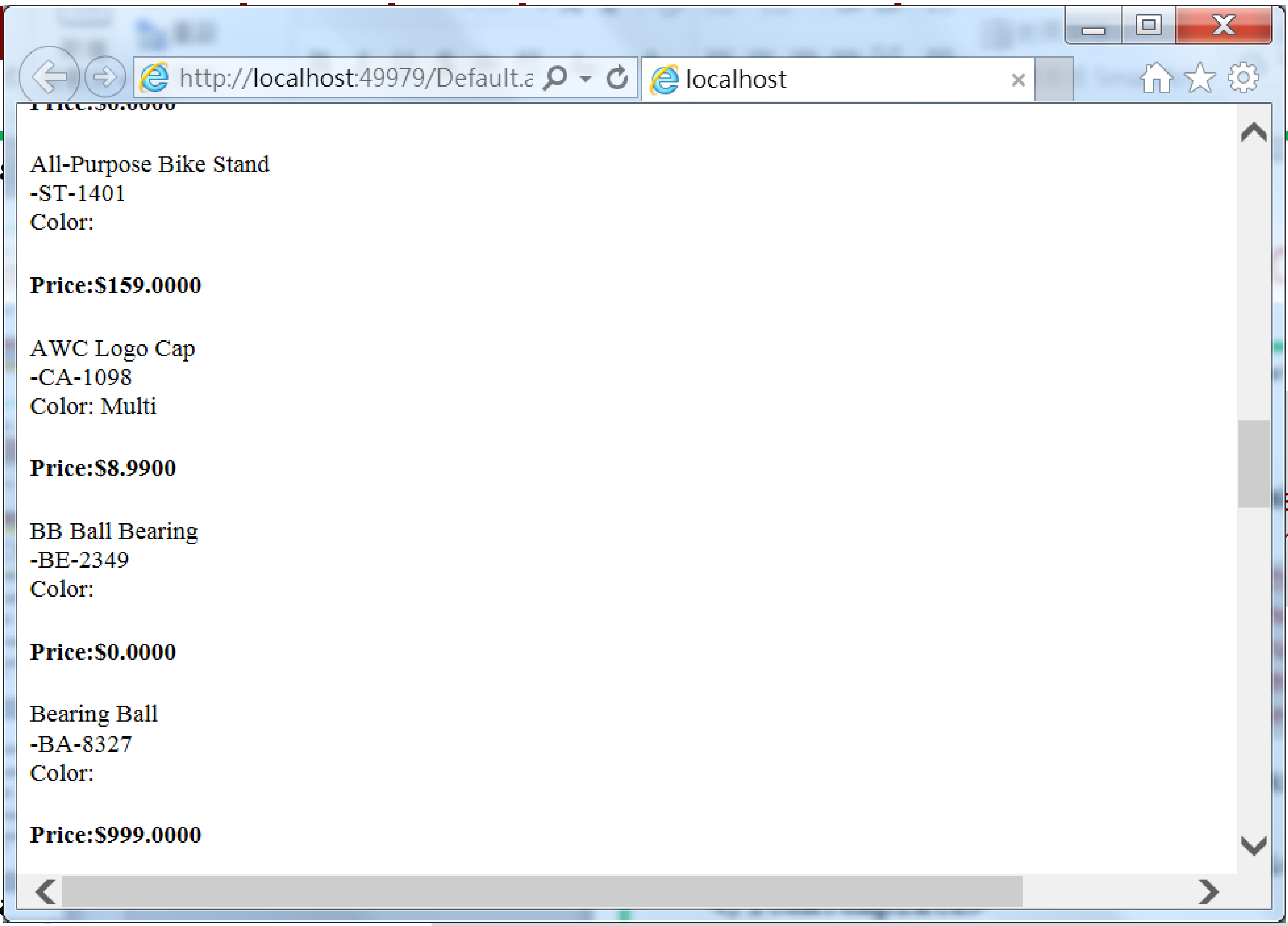
```
<asp:ListView ID="ListView1" runat="server">

    <LayoutTemplate>
       <div runat="server" id="itemPlaceholder" /><hr />
    </LayoutTemplate>




    <ItemTemplate>
       <%# Eval("Name") %><br/>
       -<%# Eval("ProductNumber") %><br/>
       Color: <%# Eval("Color") %>
       <h4>Price:$<%# Eval("ListPrice") %></h4>
    </ItemTemplate>

</asp:ListView>
```

This template is required. Use a place holder (div, span, tr, p, etc. with runat="server", id is required) to indicate the content to be replaced by the content in ItemTemplate.

Don't forget to set the DataSource property and call the DataBind() method in the code-behind page.

<a

</



```
Price:$0.0000

All-Purpose Bike Stand
-ST-1401
Color:

Price:$159.0000

AWC Logo Cap
-CA-1098
Color: Multi

Price:$8.9900

BB Ball Bearing
-BE-2349
Color:

Price:$0.0000

Bearing Ball
-BA-8327
Color:

Price:$999.0000
```

e
nt

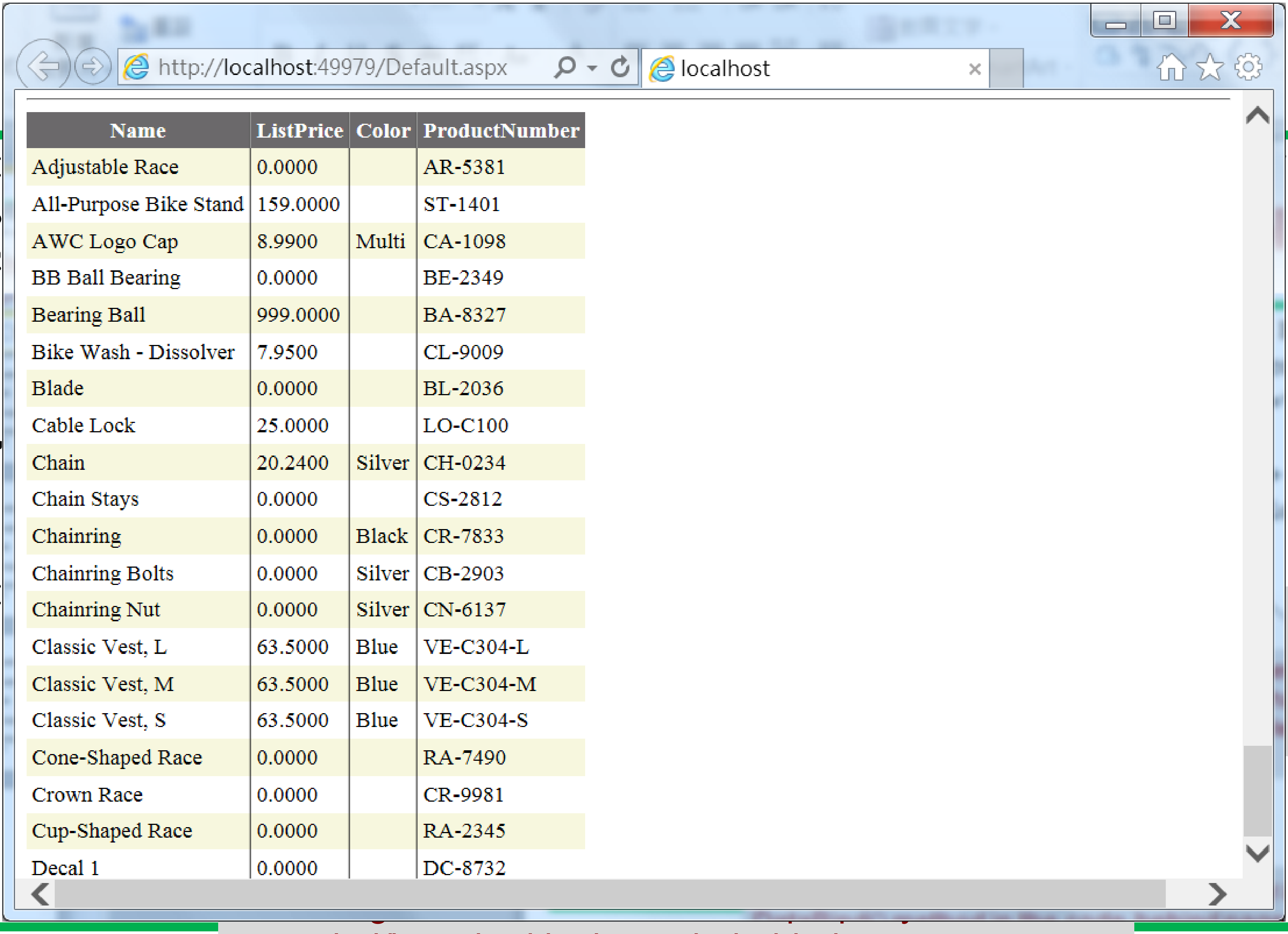
Don't forget to set the DataSource property and call the DataBind() method in the code-behind page.

# Example: GridView Control

```
<asp:GridView ID="GridView1" runat="server" BackColor="White"
BorderColor="#DEDFDE" BorderStyle="None" BorderWidth="1px"
CellPadding="4" ForeColor="Black" GridLines="Vertical">

   <AlternatingRowStyle BackColor="White" />
   <HeaderStyle BackColor="#6B696B" Font-Bold="True"
                                          ForeColor="White" />
   <RowStyle BackColor="#F7F7DE" />

</asp:GridView>
```

GridView presents data in a table layout, so it does not have flexible layout templates. Style templates can be defined.

Don’t forget to set the DataSource property and call the DataBind() method in the code-behind page.



http://localhost:49979/Default.aspx

localhost

| Name | ListPrice | Color | ProductNumber |
|---|---|---|---|
| Adjustable Race | 0.0000 | | AR-5381 |
| All-Purpose Bike Stand | 159.0000 | | ST-1401 |
| AWC Logo Cap | 8.9900 | Multi | CA-1098 |
| BB Ball Bearing | 0.0000 | | BE-2349 |
| Bearing Ball | 999.0000 | | BA-8327 |
| Bike Wash - Dissolver | 7.9500 | | CL-9009 |
| Blade | 0.0000 | | BL-2036 |
| Cable Lock | 25.0000 | | LO-C100 |
| Chain | 20.2400 | Silver | CH-0234 |
| Chain Stays | 0.0000 | | CS-2812 |
| Chainring | 0.0000 | Black | CR-7833 |
| Chainring Bolts | 0.0000 | Silver | CB-2903 |
| Chainring Nut | 0.0000 | Silver | CN-6137 |
| Classic Vest, L | 63.5000 | Blue | VE-C304-L |
| Classic Vest, M | 63.5000 | Blue | VE-C304-M |
| Classic Vest, S | 63.5000 | Blue | VE-C304-S |
| Cone-Shaped Race | 0.0000 | | RA-7490 |
| Crown Race | 0.0000 | | CR-9981 |
| Cup-Shaped Race | 0.0000 | | RA-2345 |
| Decal 1 | 0.0000 | | DC-8732 |

<
B
C
F
<

DataBind() method in the code-behind page.

## Composite Controls Comparison

| | Functionalities | | | | | |
|---|---|---|---|---|---|---|
| **Control Type** | **Flexible Layout** | **Data Grouping** | **Sorting** | **Paging** | **Update and Delete** | **Insert** |
| **ListView** | supported | supported | supported | supported | supported | supported |
| **GridView** | X | X | supported | supported | supported | X |
| **DataList** | supported | supported | X | X | X | X |
| **Repeater** | supported | X | X | X | X | X |

## Data Source Control

- Data source control is a declarative way to define a data source in the .aspx page and make it available for other controls to bind to, without requiring code.
  - They can connect to and retrieve data from a data source
  - They can also support modifying data.

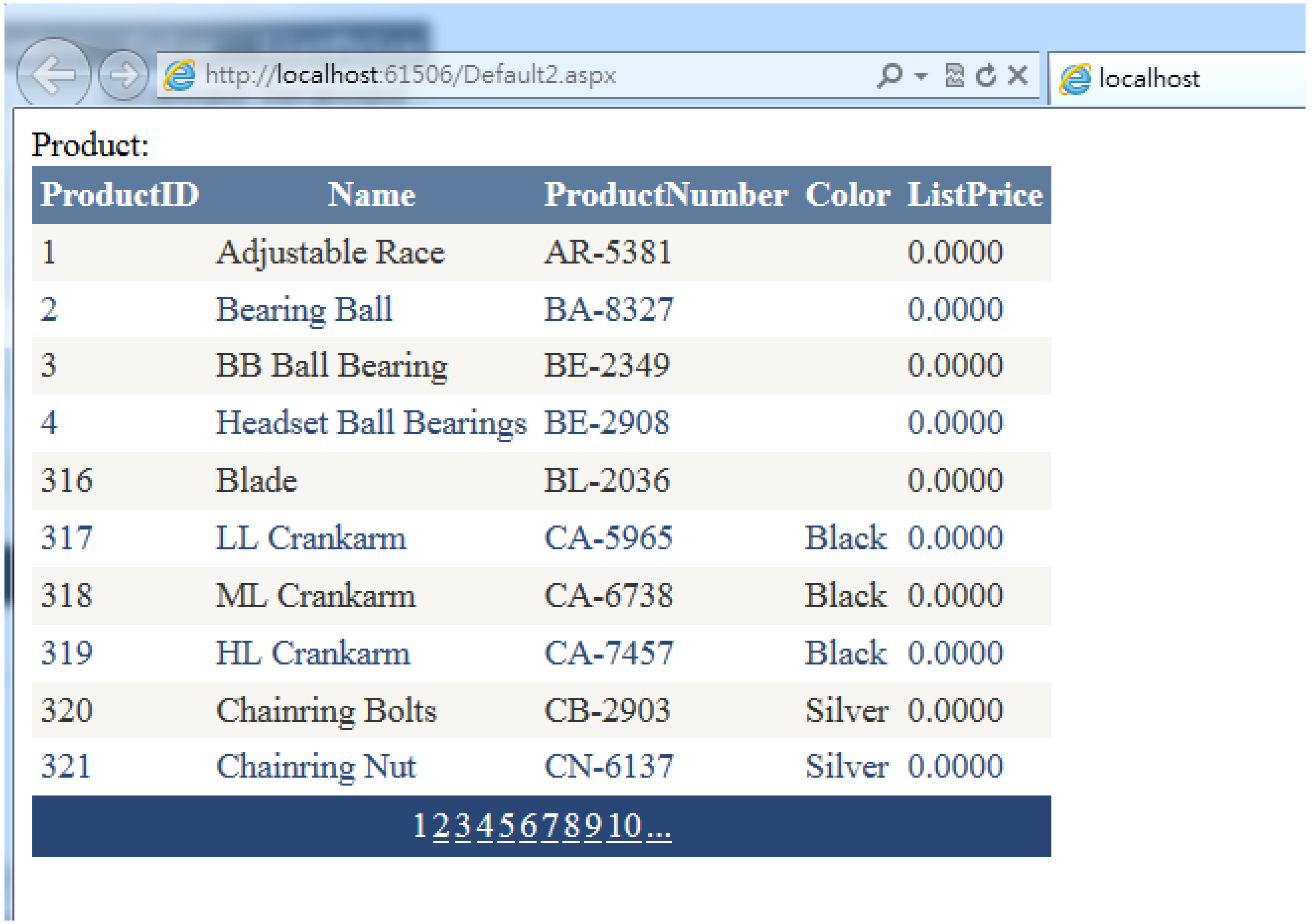
| Major data source control | Description |
|---|---|
| SqlDataSource | Enables you to work with Microsoft SQL Server, OLE DB, ODBC, or Oracle databases. When used with SQL Server, supports advanced caching capabilities. The control also supports sorting, filtering, and paging when data is returned as a DataSet object. |

## Exercise

- Using the GridView and SqlDataSource Controls
- Add a GridView Control and go to AutoFormat dialog and setup the style to 'Professional'.
- Add a SqlDataSource Controls, connect to AdventureWorks database select the Name, ProductID, ProductNumber, Color, ListPrice from Production.Product table

# Exercise

- Using the GridView and SqlDataSource Controls



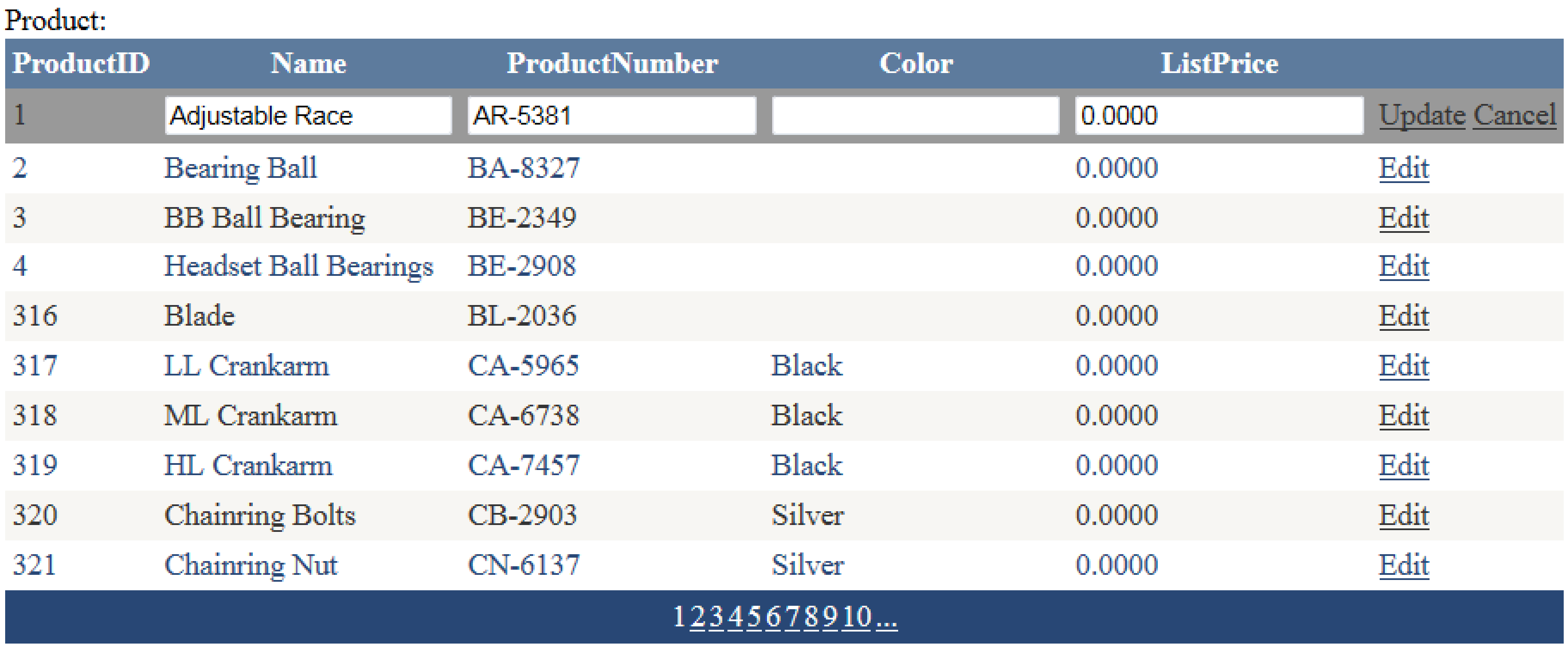
Product:

| ProductID | Name | ProductNumber | Color | ListPrice |
|---|---|---|---|---|
| 1 | Adjustable Race | AR-5381 | | 0.0000 |
| 2 | Bearing Ball | BA-8327 | | 0.0000 |
| 3 | BB Ball Bearing | BE-2349 | | 0.0000 |
| 4 | Headset Ball Bearings | BE-2908 | | 0.0000 |
| 316 | Blade | BL-2036 | | 0.0000 |
| 317 | LL Crankarm | CA-5965 | Black | 0.0000 |
| 318 | ML Crankarm | CA-6738 | Black | 0.0000 |
| 319 | HL Crankarm | CA-7457 | Black | 0.0000 |
| 320 | Chainring Bolts | CB-2903 | Silver | 0.0000 |
| 321 | Chainring Nut | CN-6137 | Silver | 0.0000 |

1 2 3 4 5 6 7 8 9 10 ...

# Exercise

- Make the GridView control enable edit and setup the edit command of the SqlDataSource Control.

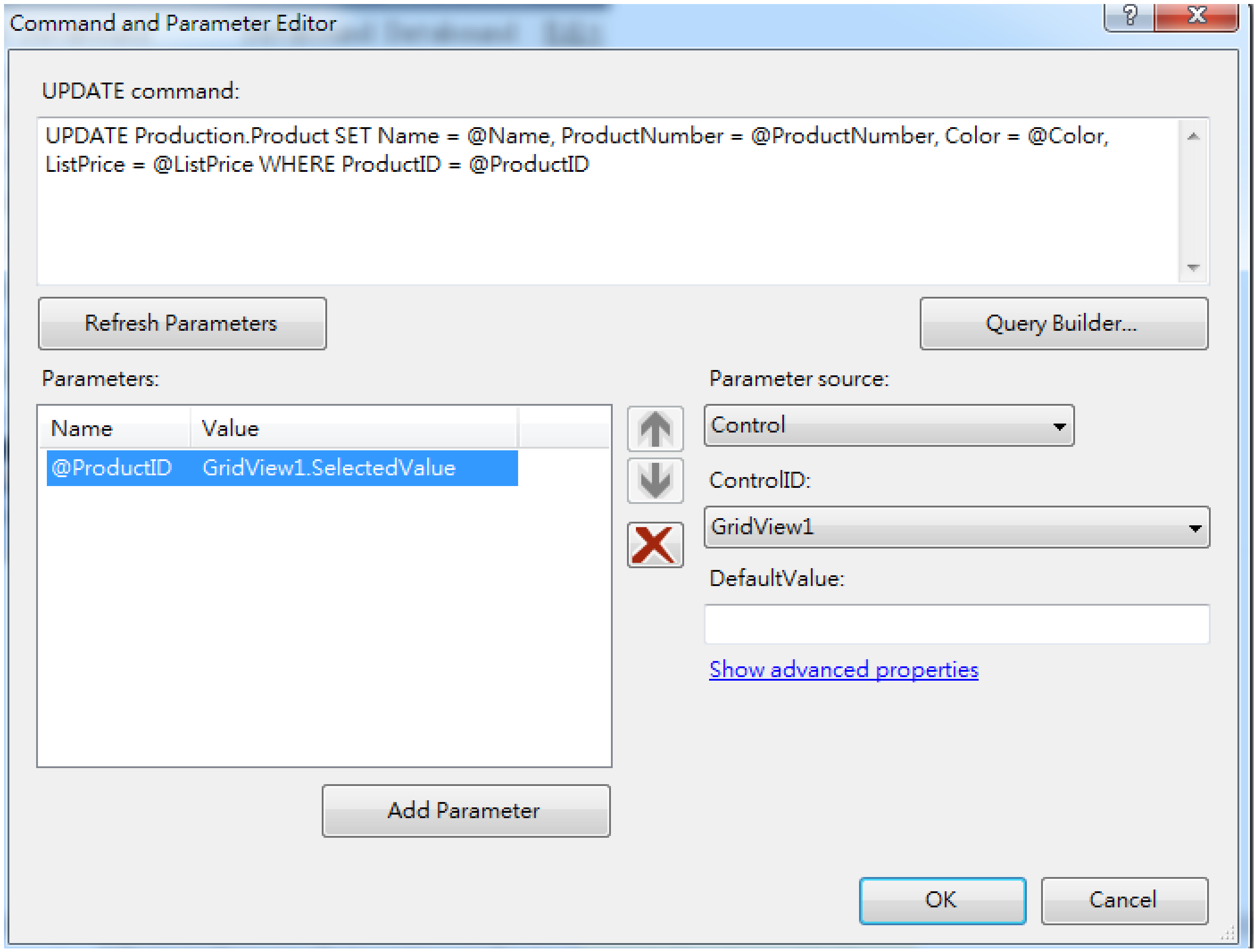


Product:

| ProductID | Name | ProductNumber | Color | ListPrice | |
|---|---|---|---|---|---|
| 1 | Adjustable Race | AR-5381 | | 0.0000 | Update Cancel |
| 2 | Bearing Ball | BA-8327 | | 0.0000 | Edit |
| 3 | BB Ball Bearing | BE-2349 | | 0.0000 | Edit |
| 4 | Headset Ball Bearings | BE-2908 | | 0.0000 | Edit |
| 316 | Blade | BL-2036 | | 0.0000 | Edit |
| 317 | LL Crankarm | CA-5965 | Black | 0.0000 | Edit |
| 318 | ML Crankarm | CA-6738 | Black | 0.0000 | Edit |
| 319 | HL Crankarm | CA-7457 | Black | 0.0000 | Edit |
| 320 | Chainring Bolts | CB-2903 | Silver | 0.0000 | Edit |
| 321 | Chainring Nut | CN-6137 | Silver | 0.0000 | Edit |

1 2 3 4 5 6 7 8 9 10 ...

# Exercise


Command and Parameter Editor

UPDATE command:

```
UPDATE Production.Product SET Name = @Name, ProductNumber = @ProductNumber, Color = @Color,
ListPrice = @ListPrice WHERE ProductID = @ProductID
```

Parameters:

| Name | Value |
|---|---|
| @ProductID | GridView1.SelectedValue |

Parameter source: Control

ControlID: GridView1

DefaultValue: