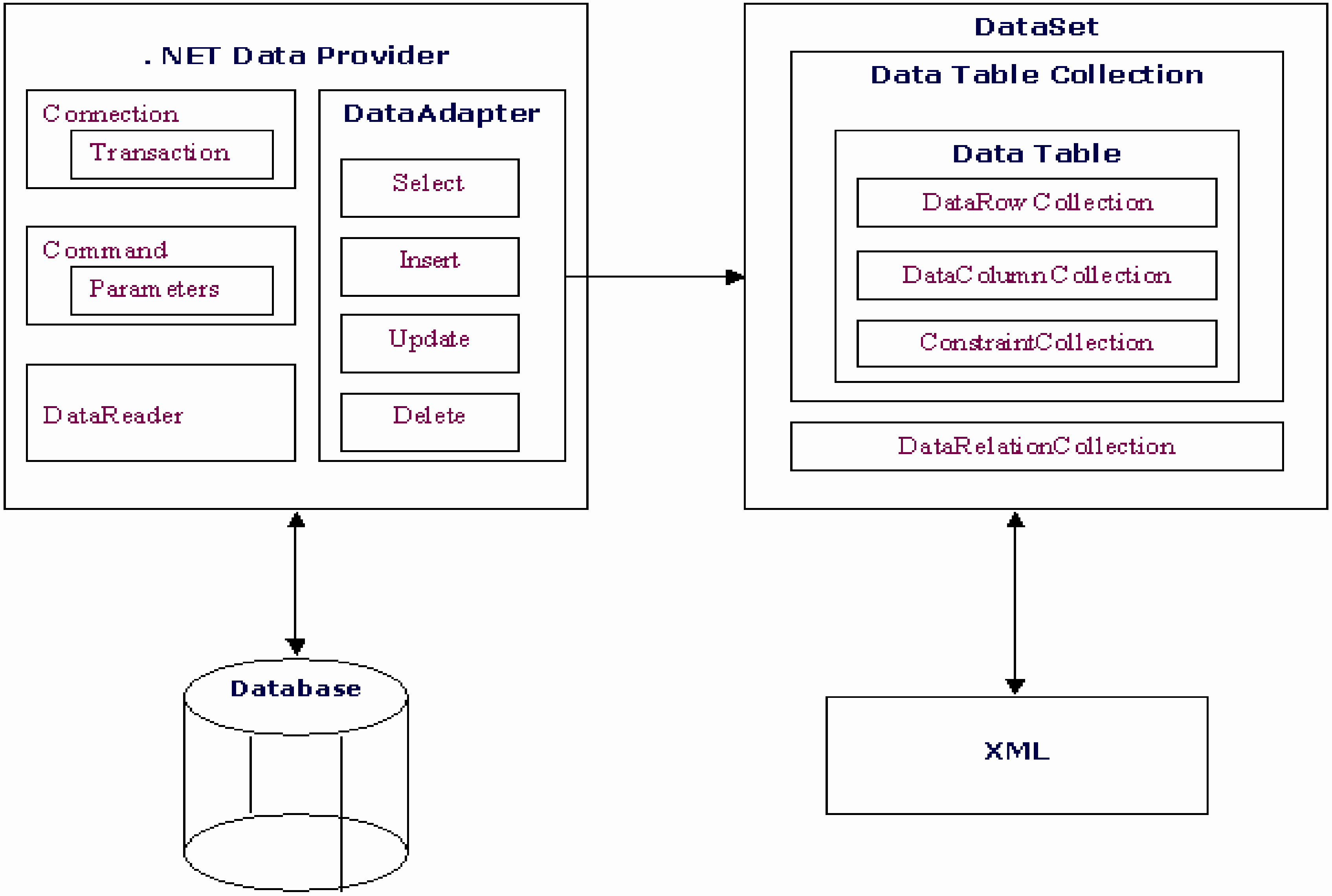
ADO.NET

Introduction to ADO.NET

- Microsoft ADO.NET is part of the Microsoft .NET Framework: a set of tools and layers that allows your application to easily manage and communicate with its file-based or server-based data store.

- In the .NET Framework, the ADO.NET libraries appear under the *System.Data* namespace.



ADO .NET Data Architecture

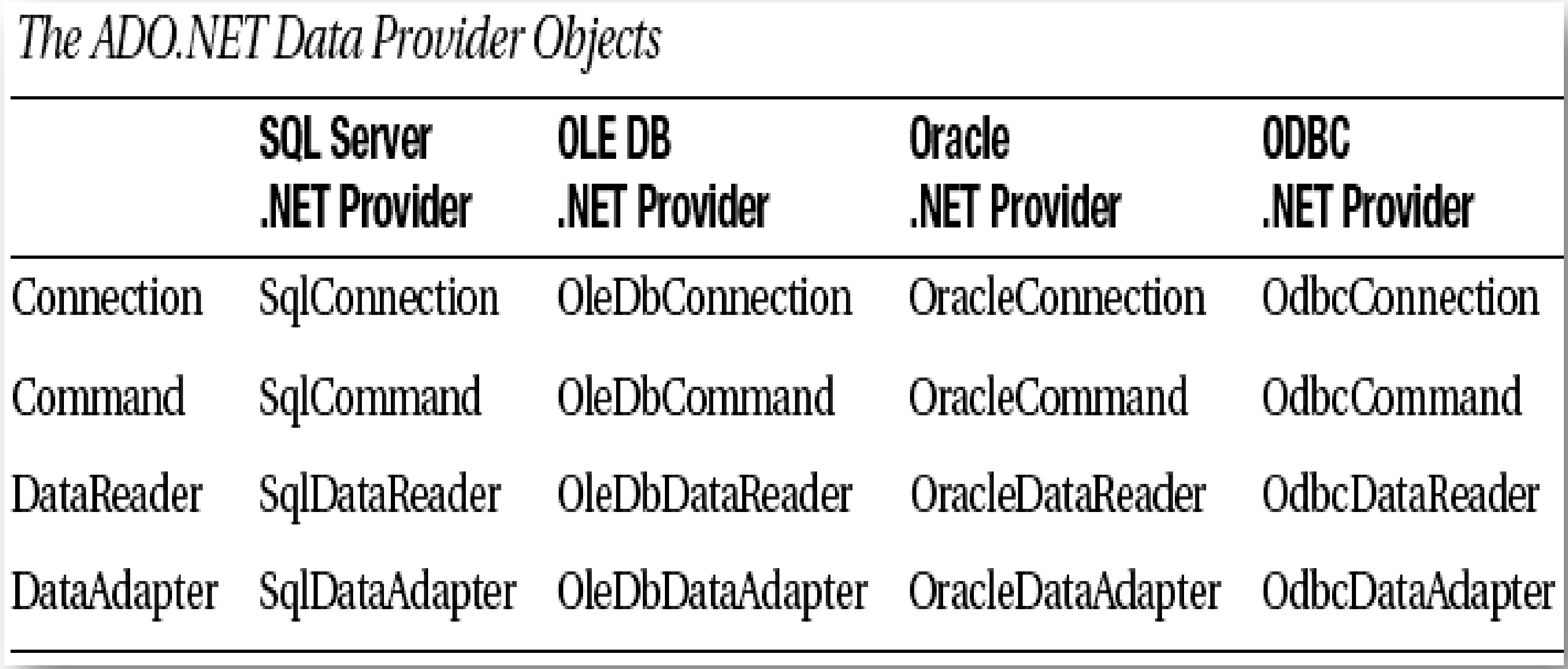
ADO.NET Architecture

- Data Access in ADO.NET relies on two components: *DataSet* and *Data Provider*.
- The dataset is a disconnected, in-memory representation of data.
- The Data Provider is responsible for providing and maintaining the connection to the database.
- A Data Provider is a set of related components that work together to provide data in an efficient and performance driven manner.

.NET Data Provider

- Constituent objects of Data Provider
  - Connection
  - Command
  - DataReader
  - DataAdapter
- Micrososft .NET ships with four data providers
  - SQL Server (for SQL Server 7.0 or above)
  - OLE DB
  - Oracle
  - ODBC

.NET Data Providers



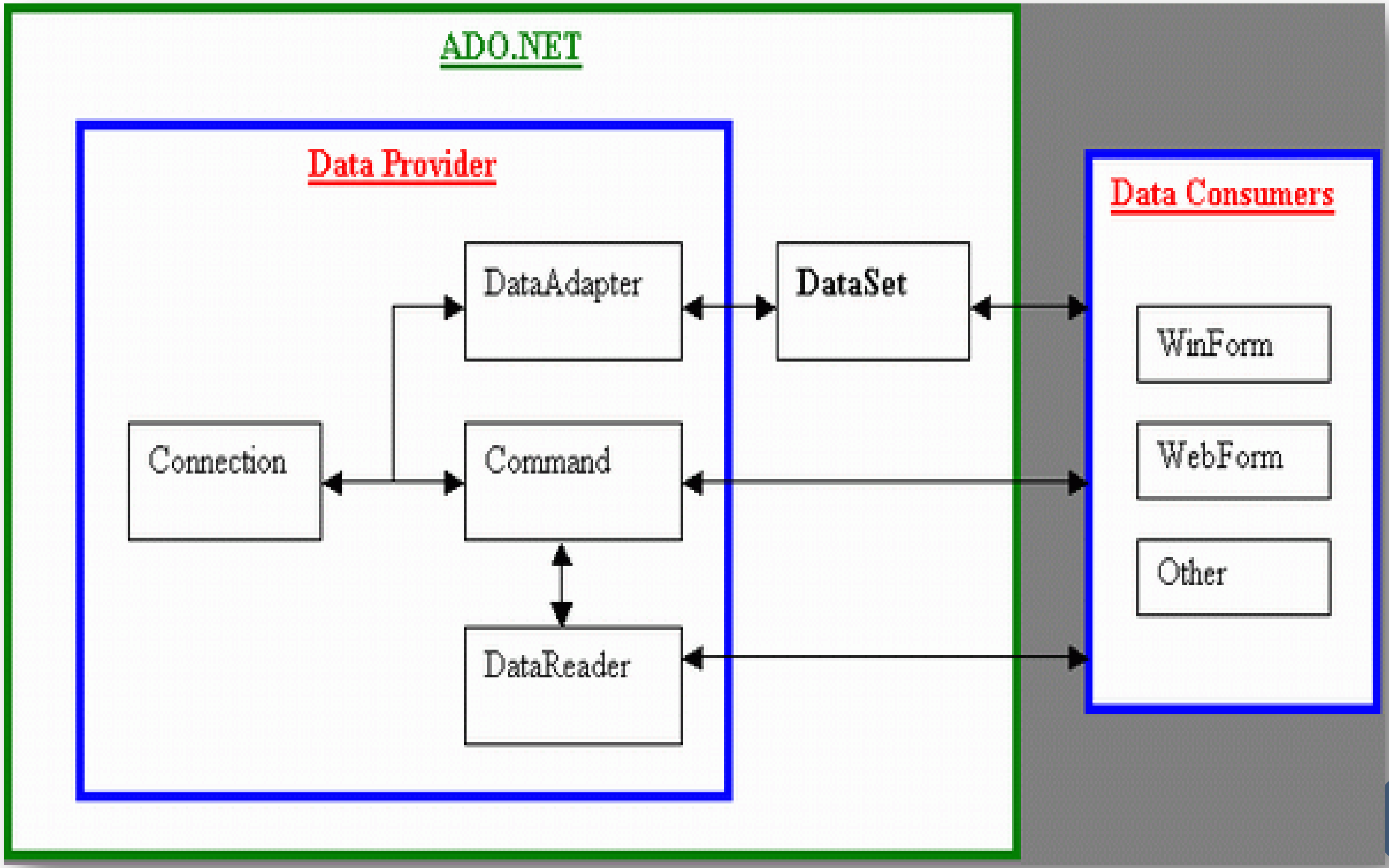
The ADO.NET Data Provider Objects

	SQL Server .NET Provider	OLE DB .NET Provider	Oracle .NET Provider	ODBC .NET Provider
Connection	SqlConnection	OleDbConnection	OracleConnection	OdbcConnection
Command	SqlCommand	OleDbCommand	OracleCommand	OdbcCommand
DataReader	SqlDataReader	OleDbDataReader	OracleDataReader	OdbcDataReader
DataAdapter	SqlDataAdapter	OleDbDataAdapter	OracleDataAdapter	OdbcDataAdapter

Data Access Modes

- Connected
  - One installs a terminal or client-server application on the user's desk. This connects the user directly to the database.
- Disconnected
  - Disconnected access—dealing with sets, graphs, or trees of data.

ADO.NET Object Model



ADO.NET OBJECTS

- Connection
  - Establishes a connection with the data source.
  - Examples of connection objects are *OleDbConnection, SqlConnection etc.*
- Command
  - Command object represents an executable command on the underlying data source.
  - It used to manipulate existing data, query existing data, and update or even delete existing data.
  - e.g. SqlCommand, OleDbCommand etc.

ADO.NET Objects

- DataReader
  - Read-only, forward-only recordset.
  - e.g. SqlDataReader, OleDbDataReader etc.
- DataAdapter
  - A gateway between the disconnected and connected flavors of ADO.NET.
  - e.g. SqlDataAdapter, OleDbDataAdapter etc.
- Parameter
  - A command needs to be able to accept parameters.
  - e.g. SqlParameter

ADO.NET Objects

- Transactions
  - Represents a Transaction object.
  - Allows to execute multi-step operation in one go.
  - e.g.SqlTransaction
- DataSet
  - In-memory database.
  - Collection of *DataTables* & *DataRelations*.
  - Provider-independent.

ADO.NET Objects

- DataTable
  - Represents a table in database.
  - Consists of *DataRows* and *DataColumns*.
- DataRow
  - *Rows* Property of *DataTable* is of *DataRowCollection* type, which represents an enumerable collection of *DataRow* objects.
  - Logical equivalent of a *DataRow* in a database is a row in a table.

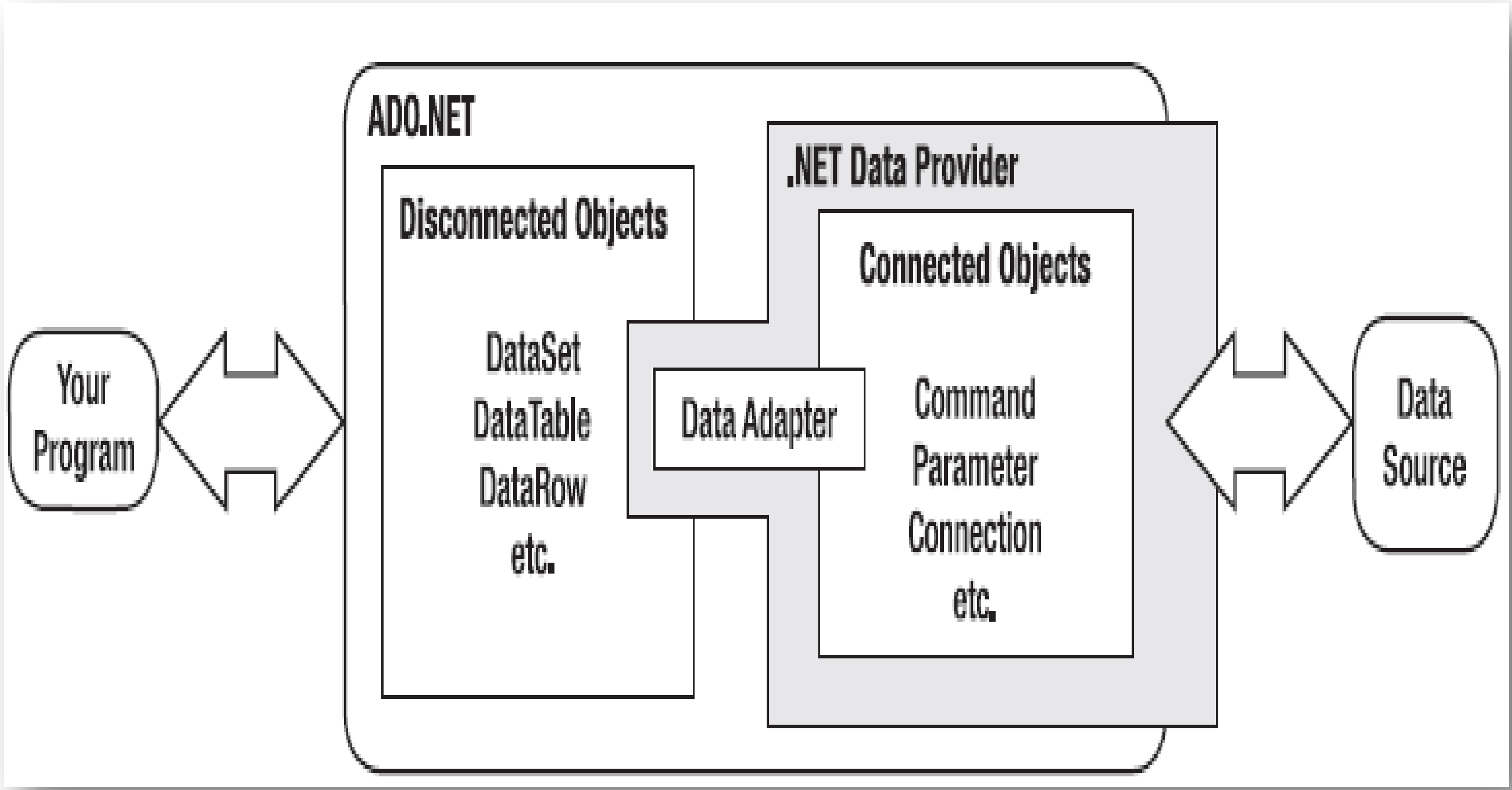
ADO.NET Objects

- DataColumn
  - Represents an individual column in a given table in a database.
- DataView
  - A view in a database.
  - One can create a view of *DataTable*.
- Constraint
  - Represents constraints like UNIQUE, PRIMARY KEY.

ADO.NET Objects

- DataRelation
  - A *DataRelation* object lets you specify relations between various tables that allow one to both validate data across tables and browse parent and child rows in various *DataTable*s.
- DataSet, DataRow, DataColumn, DataView, DataRelation, Constraint are all disconnected objects. Find these in System.Data namespace.

ADO.NET Objects



SqlConnection Class

- Belongs to *System.Data.SqlClient* namspace.

- To create a connection

```
SqlConnection con=new SqlConnection();
```

Or

```
SqlConnection con=new SqlConnection(conString);
```

Connection String

Connection String

- Data Source *or* Server
  - Specifies the SQL server to connect to.
- Initial Catalogue *or* Database
  - Specifies the database to use.
- User ID *or* uid
  - Specifies login id for connection.
- Password *or* pwd
  - Specifies password for the log in.

Connection String

```
SqlConnection con=new SqlConnection();
con.ConnectionString=
        “Server=.;Database=AdventureWorks;user id=sa;password=sa;”;
```

Or

```
SqlConnection con=new SqlConnection
        (“Server=.;Database=AdventureWorks;user id=sa;password=sa;”);
```

To open a connection

```
con.Open();
```

SqlCommand Class

- Namespace: *System.Data.SqlClient*

- Creating a command object

```
SqlCommand cmd=new SqlCommand();
```

Or

```
SqlCommand cmd=new SqlCommand(cmdText,con);
```

Connection to use

Command Text

CommandText & Connection Property

- CommandText
  - Accepts string literal representing T-SQL statement or name of stored procedure or name of the table.
- CommandType
  - Specifies the command type.
  - Text, StoredProcedure, or TableDirect.
- Connection
  - Specifies the connection object to use to access database.

Execute Methods

- ExecuteScalar
  - Returns a single value (e.g. aggregate value) from the database.
  - Mostly used for select queries with aggregate functions.
- ExecuteReader
  - Returns a data reader object with the data queried.
  - Used for SELECT queries.

Execute Methods

- ExecuteNonQuery
  - Used with DML commands only.
  - Returns integer showing number of rows affected.
  - It is also used to create database objects.

SqlDataReader Class

- *ExecuteDataReader* method returns a *SqlDataReader* object if invoked on SqlCommand object.
- Read()
  - Returns true or false.
  - If true, data reader has some more record.
  - If false, no records left.
  - This method chooses next record as current one.
  - At least once, one should invoke this to make the first record as current one before accessing the data.

SqlDataAdapter Class

- Creating a data adapter.

```
SqlDataAdapter da=new SqlDataAdapter();
```

Or

```
SqlDataAdpater da =new SqlDataAdapter(cmd);
```

Command Object

SqlDataAdpater Class

- Fill()
  - Fills the specified data table or dataset with the queried data.
- Update()
  - Updates the changes in data table or dataset back to the database.

```
da.Update(ds);
```

```
da.Fill(ds);
```

Calling Stored Procedure

```
SqlDataAdapter daCategory = new SqlDataAdapter();
daCategory.SelectCommand = new SqlCommand();
daCategory.SelectCommand.Connection = conn;
daCategory.SelectCommand.CommandText = "ProductCategoryList";
daCategory.SelectCommand.CommandType =
            CommandType.StoredProcedure;
```

SqlParameter Class

- Represents a *SqlCommand* parameter.

```
SqlParameter param=new SqlParameter();
```

Or

```
SqlParameter param=new SqlParameter(“id”,SqlDbType.Int);
```

Parameter name

Parameter data type.

SqlParameter Object Properties

- ParameterName
  - Gets or sets the name of the parameter.
- SqlDbType
  - Gets or sets the SQL Server database type of the parameter value.
- Direction
  - Sets or gets the direction of the parameter, such as Input, Output, or Both.

SqlParameter Object Properties

- Size
  - Sets or gets the size of the parameter value.
- Value
  - Sets or gets the value provided to the parameter object.
- SourceColumn
  - Read-write property maps a column from a *DataTable* to the parameter.

Using Parameters

- Identify the available parameters

  - **Input**
  - **Output**
  - **InputOutput**
  - **ReturnValue**

- Include parameters in the parameters collection
- or
- Include parameter values in the command string

Passing Input Parameters

- Create parameter, set direction and value, add to the Parameters collection

```
SqlParameter param = new SqlParameter
        ("@Beginning_Date", SqlDbType.DateTime);
param.Direction = ParameterDirection.Input;
param.Value = Convert.ToDateTime
                        (txtStartDate.Text);
da.SelectCommand.Parameters.Add(param);
```

- Run stored procedure and store returned records

```
ds = New DataSet();
da.Fill(ds, "Products");
```

Using Output Parameters

- Create parameter, set direction, add to the Parameters collection

```
param = new SqlParameter("@ItemCount", SqlDbType.Int);
param.Direction = ParameterDirection.Output;
da.SelectCommand.Parameters.Add(param);
```

- Run stored procedure and store returned records

```
ds = new DataSet();
da.Fill(ds);
```

- Read output parameters

```
iTotal = da.Parameters("@ItemCount").Value;
```

Creating Relationships

- Identify Parent Column
-
```
DataColumn pcol=ds.Tables["Customers"].Columns["CustomerID"];
```

- Create a DataRelation.

```
DataColumn ccol=ds.Tables["Orders"].Columns["CustomerID"];
```

```
DataRelation dr=new DataRelation("CustOrder",pcol,ccol);
```

SqlCommandBuilder Class

- The CommandBuilder examines the DataAdapter object used to create the DataSet, and it adds the additional Command objects for the InsertCommand, DeleteCommand, and UpdateCommand properties.

```
SqlCommandBuilder cmb=new SqlCommandBuilder(da);
da.Update(ds);
```

Data Binding

- ASP.NET has feature to pop data directly into HTML elements and fully formatted controls. Referred as data binding.
- Data binding tells a control where to find data and how one want it displayed, and the control handles the rest of the details.
- Two types of ASP.NET data binding exist: single-value binding and repeated-value binding.

Single Value Binding

- Use it to add information anywhere on an ASP.NET page.
- One can place information into a control property or as plain text inside an HTML tag.
- Single-value data binding allows you to take a variable, property, or expression and insert it dynamically into a page.
- Single-value binding also helps you create templates for the rich data controls.

Repeated Value or List Binding

- It allows to display an entire table or all the values from a single field in a table.

- In repeated-value binding, data binding is configured by setting the appropriate control properties

DataBind() Method

- Basic functionality of data binding supplied in *Control* class.

- Automatically binds a control and any child controls that it contains.

- One can use control-specific *DataBind* method also.

- One can bind the whole page at once by calling the DataBind() method of the current Page object.

Single-Value Binding

- Add special data binding expressions into your .aspx files embedded in <%# and %>

- The expression can be a variable, property of ASP.NET object, a public or protected function defined returning a simple value.

```
<p>Total <%# NoofVisitors %> visitors visited the page. </p>
```

$\uparrow$

Variable whose value retrieved from database. Does not show the value till the page invokes ***DataBind***.

List Data Binding

- List Data Binding requires list controls that supports data binding

- Some of the controls:
  - ListBox, DropDownList, CheckBoxList, RadioButtonList.
    - Provide a list for a single-column of information.
  - GridView, DetailsView, FormView
    - Provide repeating lists or grids that can display more than one column (or field) of information at a time.

List Data Binding

```
ArrayList cars=new ArrayList();
cars.Add("M800");
cars.Add("Alto");
cars.Add("Zen");
cars.Add("Esteem");
lstCars.DataSource=cars;
lstCars.DataBind();
```

$\}$ Page.Load Event Handler

If viewstate of the control is enabled, one don't need to re-create and rebind the control every time the Page.Load event occurs. Use IsPostBack property.

Rebind if the content of data source changes.

Data Source Controls

- Data source controls are sources of data.

- The data source controls include any control that implements the IDataSource interface.

```
<asp:SqlDataSource ProviderName="System.Data.SqlClient"
        ID="SqlDataSource1" runat="server" />
```

Data Source Controls

- SqlDataSource
- ObjectDataSource
- XmlDataSource
- SiteMapDataSource

SqlDataSource Control

- Provides access to any data source that has an ADO.NET Data Provider available; by default, the control has access to the ODBC, OLE DB, SQL Server, Oracle, and SQL Server CE providers.
- The data provider must include data provider factory.
- One can choose a data source by setting the provider name.

Default provider, if omitted no problem

```
<asp:SqlDataSource ProviderName="System.Data.SqlClient"
          ID="SqlDataSource1" runat="server" … />
```

SqlDataSource Connection String

- Connection string usually stored in web.config file.
- To refer connection string in .aspx mark up, use format <%$ ConnectionStrings:<nameofstring> %>

Data Source Connection String

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
                <appSettings/>
  <connectionStrings>
    <add name="Northwind“ connectionString="server=localhost;
                                database=Northwind;user id=sa;password=sa" />
  </connectionStrings>
              …
</configuration>
```

web.config

In .aspx file

```
<asp:SqlDataSource ID=“SqlDataSource1”
                ConnectionString=“<%$ ConnectionStrings:Northwind %>”…/>
```

SqlDataSource Commands

- SqlDataSource control can not only query the data but also update,insert, delete data.

- One can do it through properties

  - SelectCommand
  - InsertCommand
  - DeleteCommand
  - UpdateCommand

- Each property takes a string representing command text or stored procedure name.

- Each has corresponding command type property.

  - SelectCommandType/InsertCommandType
  - DeleteCommandType/UpdateCommandType

Adding SqlDataSource control

```
<asp:SqlDataSource ID=”SqlDataSource1” Runat=”server”
   SelectCommand=”SELECT ProductName, ProductId FROM [Products]”
   ConnectionString=”<%$ ConnectionStrings:Northwind %>”>
</asp:SqlDataSource>
```

```
<asp:DropDownList ID=“ddlProducts” runat=“server”
            DataSourceID=“SqlDataSource1” DataTextField=“ProductName”
            DataValueField=“ProductId” />
```

SqlDataSource with Parameterized Query

```
<asp:SqlDataSource ID=”SqlDataSource1” Runat=”server”
   SelectCommand=”SELECT ProductName, ProductId FROM [Products]
                        where ProductId=@ProductID”
   ConnectionString=”<%$ ConnectionStrings:Northwind %>” >
         <SelectParameters>
         <asp:ControlParameter ControlID=“ddlProducts”
                        Name=“ProductID” PropertyName=“SelectedValue” />
         </SelectParameters>
</asp:SqlDataSource>
```

Parameter

Property of Drop down list

As long as one gives each parameter the same name as the field it affects and preface it with the @ symbol, no need to define parameter.

Parameter Types

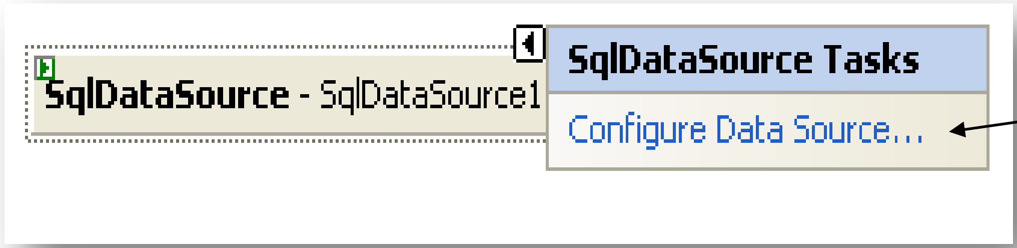
- ControlParameter
  - A property from another control on the page.
- QueryStringParameter
  - A value from the current query string.
- SessionParameter
  - A value stored in the current user’s session.

Parameter Types

- CookieParameter
  - A value from any cookie attached to the current request.
- ProfileParameter
  - A value from the current user’s profile.
- FormParameter
  - A value posted to the page from an input control.

SqlDataSource Events

- *Selected*, *Inserted*, *Updated* and *Deleted* are the events of SqlDataSource.
- All has one parameter of type *SqlDataSourceStatusArgs*.
- *SqlDataSourceStatusArgs* gives access to *Exception* object & *ExceptionHandled* property.
- To prevent error, set *ExceptionHandled=true*.

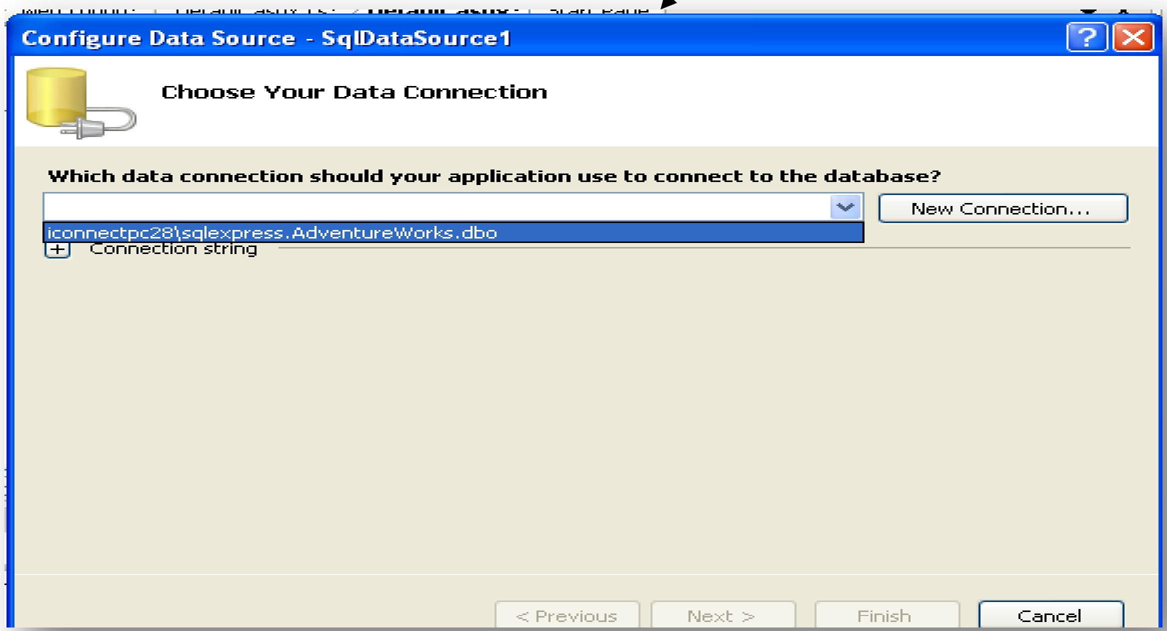


SqlDataSource - SqlDataSource1

SqlDataSource Tasks

Configure Data Source...

Click here, starts wizard



Configure Data Source - SqlDataSource1

Choose Your Data Connection

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

iconnectpc28\sqlexpress.AdventureWorks.dbo

XmlDataSource Control

- The *XmlDataSource* extracts information from an XML file, rather than a database or data access class.
- XML content is hierarchical and can have an unlimited number of levels.
- *SqlDataSource* return flat tables of data.

```
<asp:XmlDataSource ID=”XmlDataSource1” Runat=”server”
            DataFile=”http://msdn.microsoft.com/rss.xml”
            XPath=”rss/channel/item”
</asp:XmlDataSource>
```

ObjectDataSource Control

- Binds data controls to middle-layer business objects.

- Business object or component must

  - Be stateless class.
  - Have a default, no-argument constructor.
  - Contain all logic.
  - Provide the query results when a single method is called.
  - Should have useful methods as non-static.
  - Return query result as dataset, data table or some sort of collection object.

Using ObjectDataSource Control

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    DeleteMethod="Delete" InsertMethod="Insert" SelectMethod="Select"
    TypeName="Customer" UpdateMethod="Update">
        <SelectParameters>
        <asp:QueryStringParameter Name="customerID"
        QueryStringField="ID" Type="Int32" />
        </SelectParameters>
</asp:ObjectDataSource>
```

Component

Parameter for Insert, Update & delete methods.

Programmatically Accessing ConnectionString From web.config

- <connectionStrings> is a special section in which connection strings are stored.

- .NET 2.0 exposes the ConnectionString section using the ConnectionStringSettings class.

- *ConfigurationManager* object has static property *ConnectionStrings* [ConnectionStringSettingsCollection] to expose all connectionstrings in web.config.

```
SqlConnection con=
        ConfigurationManager.Connectionstrings[“Northwind”];
```

Data Controls

- GridView
- DetailsView
- FormView
- These control allows to bind entire tables of data.

Data Controls

- GridView
  - The GridView is an all-purpose grid control for showing large tables of information.
  - Supports editing.
- DetailsView
  - The DetailsView is ideal for showing a single record at a time, in a table. Supports Editing.
- FormView
  - FormView shows a single record at a time and supports editing. FormView is based on templates.

GridView

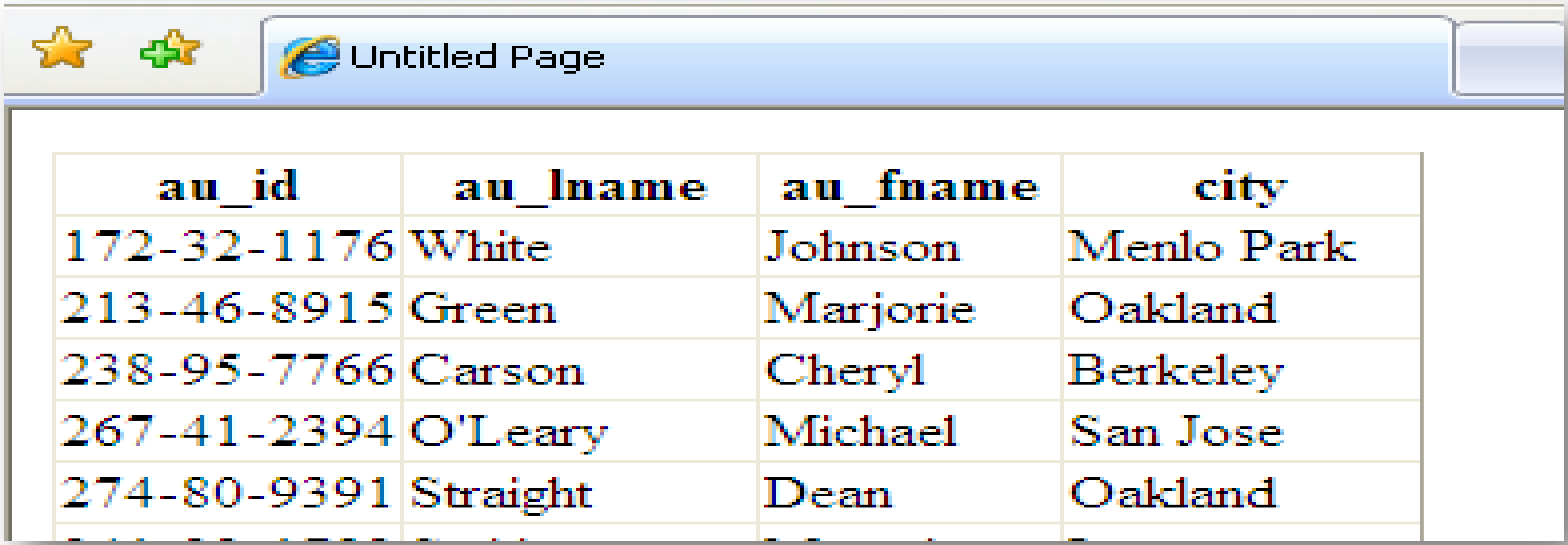
- Flexible grid control that displays a multicolumn table.
- Each record in your data source becomes a separate row. Each field in the record becomes a separate column.
- Functionality includes features for automatic paging, sorting, selecting, and editing.

Displaying Data in GridView

- The GridView provides a DataSource property for the data object to be displayed.
- Set *DataSource* property to data source control & invoke *DataBind* method on GridView.
- Set *AutoGenerateColumns* property to true (default).

Displaying Data in GridView

```
<asp:GridView ID="GridView1" runat="server“
            DataSourceID="SqlDataSource1">
      </asp:GridView>
```



au_id	au_lname	au_fname	city
172-32-1176	White	Johnson	Menlo Park
213-46-8915	Green	Marjorie	Oakland
238-95-7766	Carson	Cheryl	Berkeley
267-41-2394	O'Leary	Michael	San Jose
274-80-9391	Straight	Dean	Oakland

Programmticально Displaying Data

```
GridView1.DataSource=ds.Tables[0];
GridView1.DataBind();
```

DataSet

Specifying Columns in GridView

- GridView allows to hide columns, change their order, or configure some aspect of their display, such as the formatting or heading text.
- Set *AutoGenerateColumns* to false.
- For specifying column, one has to specify column type.

Column Types

- BoundField
  - The column displays text from a field in the data source.
- ButtonField
  - The column displays a button for each item in the list.
- CheckBoxField
  - The column displays a check box for each item in the list.

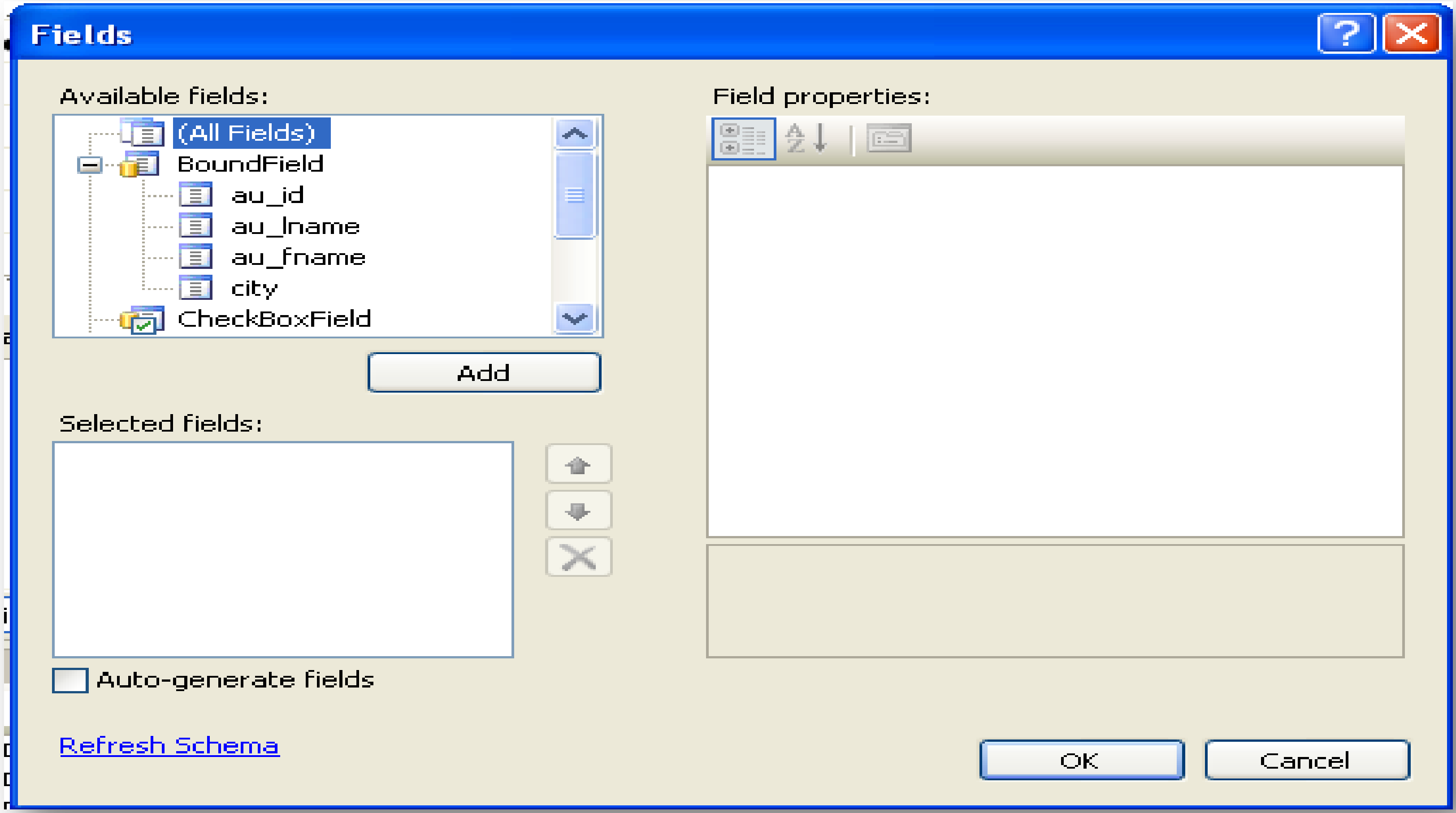
Column Types

- CommandField
  - The column provides selection or editing buttons.
- HyperLinkField
  - The column displays its contents (a field from the data source or static text) as a hyperlink.
- ImageField
  - The column displays image data from a binary field.

Column Types

- TemplateField
  - The column allows you to specify multiple fields, custom controls, and arbitrary HTML using a custom template.
- Using *Columns* property, one can specify the columns.
- Property browser presents GUI to specify columns.

Specifying Columns in GridView



Specifying Columns in GridView

```
<asp:GridView ID="GridView1" runat="server“
    DataSourceID="SqlDataSource1" AutoGenerateColumns="False">
            <Columns>
                <asp:BoundField DataField="au_id“
    HeaderText="Author ID" ReadOnly="True" SortExpression="au_id" />
                <asp:BoundField DataField="au_fname“
     HeaderText="First Name" SortExpression="au_fname" />
                <asp:ButtonField Text="Select" />
            </Columns>
        </asp:GridView>
```

GridView Column Sorting

```
<asp:GridView ID="GridView1" runat="server“
    DataSourceID="SqlDataSource1" AutoGenerateColumns="False“
    AllowSorting=“true” >
            <Columns>
                <asp:BoundField DataField="au_id“
    HeaderText="Author ID" ReadOnly="True" SortExpression="au_id" />
                <asp:BoundField DataField="au_fname“
     HeaderText="First Name" SortExpression="au_fname" />
                <asp:ButtonField Text="Select" />
            </Columns>
        </asp:GridView>
```

After enabling sorting, all column heading become hyperlinks, clicking on that sorts the column.

GridView Column Sorting

- The GridView has *Sort()* method which accepts multile SortExpressions to enable multicolumn sorting.

- One can also implement multicolumn sorting using *Sorting* event.

GridView Column Sorting

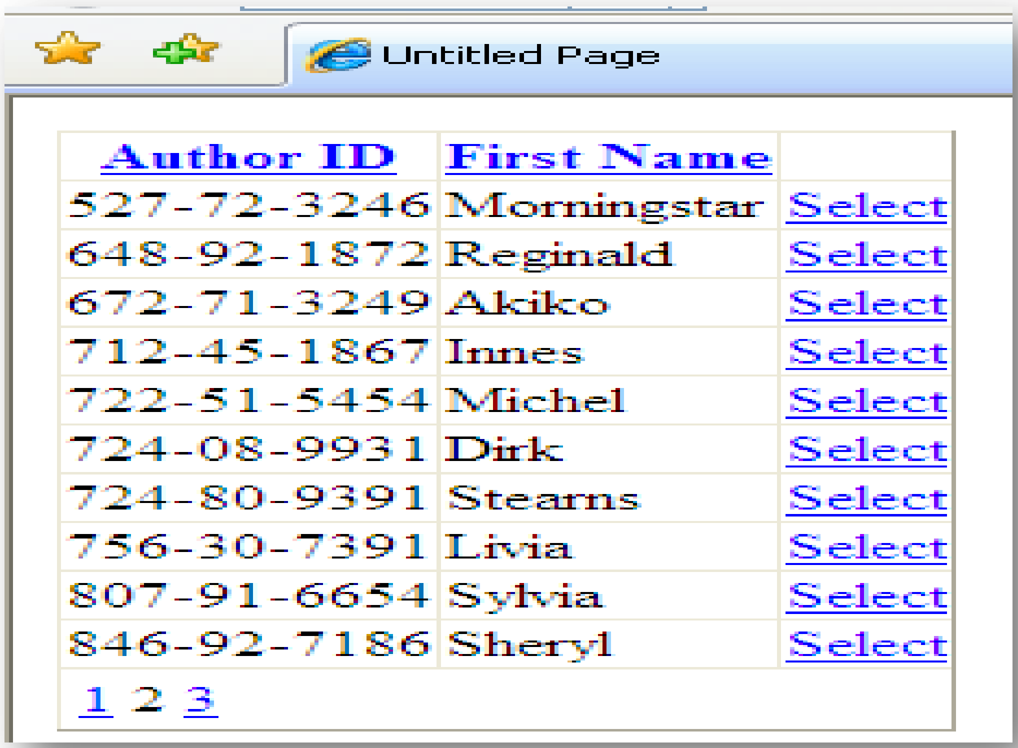
```
protected void GridView1_Sorting(object sender, GridViewSortEventArgs e)
   {
      string oldexpr = GridView1.SortExpression;
      string newexpr = e.SortExpression;
      if (oldexpr.IndexOf(newexpr) < 0)
      {
         if (oldexpr.Length > 0)
            e.SortExpression = newexpr + "," + oldexpr;
         else
            e.SortExpression = newexpr;
      }
      else
      {
         e.SortExpression = oldexpr;
      }
   }
```

GridView Paging

```
<asp:GridView ID="GridView1" runat="server“
    DataSourceID="SqlDataSource1" AutoGenerateColumns="False“
    AllowSorting=“true” AllowPaging=“true” >
            <Columns>
                <asp:BoundField DataField="au_id“
    HeaderText="Author ID" ReadOnly="True" SortExpression="au_id" />
                <asp:BoundField DataField="au_fname“
     HeaderText="First Name" SortExpression="au_fname" />
                <asp:ButtonField Text="Select" />
            </Columns>
        </asp:GridView>
```

Default page size is 10.

GridView Paging



Author ID	First Name	
527-72-3246	Morningstar	Select
648-92-1872	Reginald	Select
672-71-3249	Akiko	Select
712-45-1867	Innes	Select
722-51-5454	Michel	Select
724-08-9931	Dirk	Select
724-80-9391	Stearns	Select
756-30-7391	Livia	Select
807-91-6654	Sylvia	Select
846-92-7186	Sheryl	Select

1 2 3

Customizing Paging

- *PagerSetting* allows to dictate how the grid's paging is displayed.

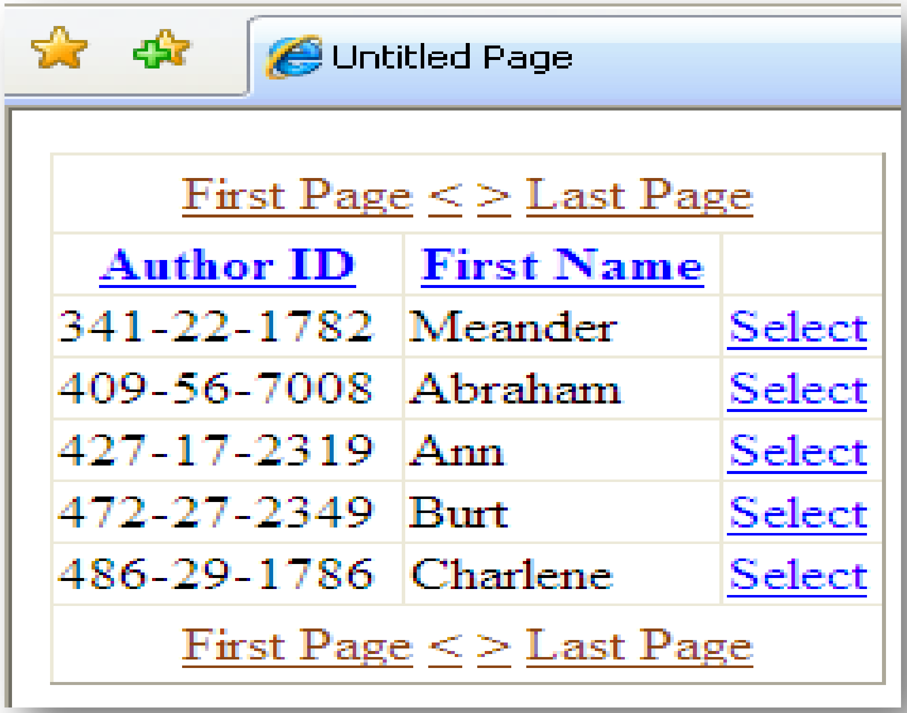
- Specifying *PagerStyle* element in grid, one can customize how the grid displays the Pager text, including font color, size, and type, as well as text alignment.

- *PageSize* property of grid allows to specify page size.

CUSTOMIZING PAGING

```
<asp:GridView ID="GridView1" runat="server“
   DataSourceID="SqlDataSource1" AutoGenerateColumns="False"
   AllowPaging="True" AllowSorting="True“
  OnSorting="GridView1_Sorting“ PageSize="5" Width="221px">
          <Columns>
              <asp:BoundField DataField="au_id" HeaderText="Author ID“
                              ReadOnly="True“ SortExpression="au_id" />
              <asp:BoundField DataField="au_fname" HeaderText="First Name“
                             SortExpression="au_fname" />
              <asp:ButtonField Text="Select" />
          </Columns>
          <PagerSettings FirstPageText="First Page“
                             LastPageText="Last Page“  Mode="NextPreviousFirstLast"
              Position="TopAndBottom" />
          <PagerStyle ForeColor="SaddleBrown" HorizontalAlign="Center" />
      </asp:GridView>
```

Customizing Paging



First Page < > Last Page		
Author ID	**First Name**	
341-22-1782	Meander	Select
409-56-7008	Abraham	Select
427-17-2319	Ann	Select
472-27-2349	Burt	Select
486-29-1786	Charlene	Select
First Page < > Last Page		

How to display null in GridView?

- One can specify what the GridView should display when it encounters a null value within the column.

```
<asp:BoundField DataField="City" HeaderText="City"
    NullDisplayText="N/A" SortExpression="City" ></asp:BoundField>
```

Adding Edit Button to GridView

- First set *UpdateCommand* of data source control (e.g. SqlDataSource).

- Set *AutoGenerateEditButton* attribute of GridView to true.It adds a ButtonField column with an Edit button for each data row.

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1“
AutoGenerateColumns="False“ AllowSorting="True“
OnSorting="GridView1_Sorting“  Width="221px" AutoGenerateEditButton="True">
…
</asp:GridView>
```

```
 <asp:SqlDataSource ID="SqlDataSource1" runat="server“
ConnectionString="<%$ ConnectionStrings:pubsConnectionString %>“
SelectCommand="SELECT [au_id], [au_lname], [au_fname], [city]
            FROM [authors]“
UpdateCommand="UPDATE authors SET au_fname = @au_fname
            WHERE (au_id = @au_id)">
         <UpdateParameters>
            <asp:QueryStringParameter Name="au_fname“
            QueryStringField="fname" />
            <asp:QueryStringParameter Name="au_id“
            QueryStringField="id" />
         </UpdateParameters>
      </asp:SqlDataSource>
```

Adding Edit Button to GridView

- Second way to add edit button s to add a CommandField column.

```
<asp:CommandField HeaderText="Command" ShowEditButton="True"
        ShowHeader=“true” />
```

Editing GridView Data

- Click on edit button, puts grid in edit mode.
- We can control which columns the grid allows to be edited by adding the *ReadOnly* attribute to the columns.

Updating New Data to DataBase

- For updating to database, tell the grid which SQL columns are serving as primary keys.
- Use *DataKeyNames* attribute of Grid to specify key.
- *RowUpdated* event is used to check for any errors when updating data.
- Click on Update button, to update. It uses the *UpdateCommand* of data source control.

Deleting Data from Grid and DB

- Set *AutoGenerateDeleteButton* to true.
- Set *DeleteCommand* of data source control.
- Grid's *RowDeleted* event or *Deleted* event of data source control can be used to deal with errors during delete.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server“
ConnectionString="<%$ ConnectionStrings:pubsConnectionString %>“
SelectCommand="SELECT [au_id], [au_lname], [au_fname], [city]
FROM [authors]“
DeleteCommand="DELETE FROM authors WHERE (au_id = @au_id)">
          <DeleteParameters>
             <asp:Parameter Name="au_id" Type=“String” />
          </DeleteParameters></asp:SqlDataSource
```

Just click on the Delete button, it executes delete command on database.

DetailsView

```
<asp:DetailsView ID="DetailsView1" runat="server"
AutoGenerateRows="False" Height="50px" Width="125px“
DataKeyNames="au_id" DataSourceID="SqlDataSource1">
            <Fields>         }   Autogenerates bound fields for columns.
                …
            </Fields>
        </asp:DetailsView>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server“
ConnectionString="<%$ ConnectionStrings:pubsConnectionString %>"
            SelectCommand="SELECT
 [au_id], [au_lname], [au_fname], [phone], [address], [city], [state], [zip]
FROM [authors]">
        </asp:SqlDataSource>
```

Paging in DetailsView

- Paging allows to browse through the each record.

```
<asp:DetailsView ID="DetailsView1" runat="server"
AutoGenerateRows="False" Height="50px" Width="125px“
DataKeyNames="au_id" DataSourceID="SqlDataSource1“
AllowPaging="True">…</asp:DetailsView>
```

Customizing DeatilsView

- The control, by default, displays each column from the table it is working with.
- One can specify which columns to display.

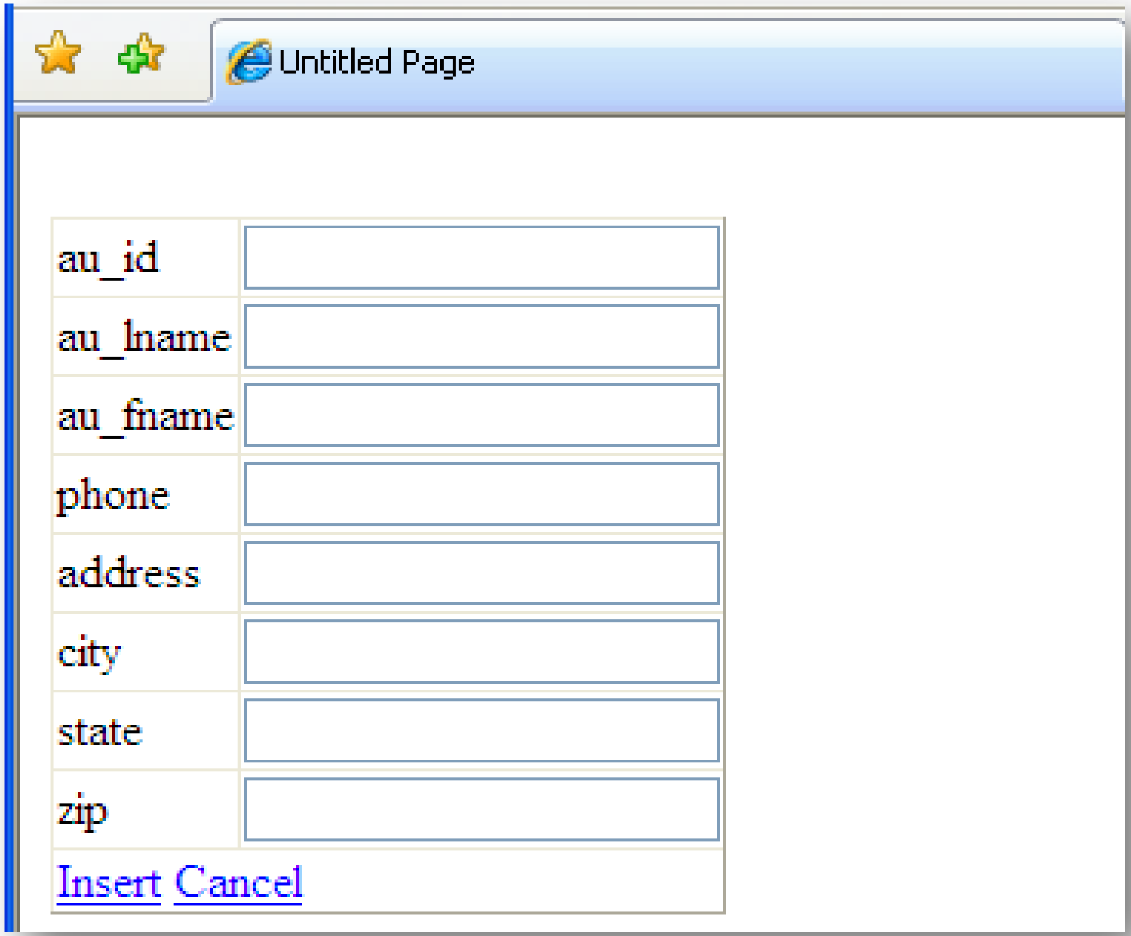
```
<asp:DetailsView ID="DetailsView1" runat="server"
AutoGenerateRows="False" Height="50px" Width="125px“
DataKeyNames="au_id" DataSourceID="SqlDataSource1">
          <Fields>
             <asp:BoundField DataField="au_id" HeaderText="au_id“
             ReadOnly="True" SortExpression="au_id" />
             <asp:BoundField DataField="au_lname" HeaderText="au_lname“
             SortExpression="au_lname" />
             <asp:BoundField DataField="au_fname" HeaderText="au_fname“
             SortExpression="au_fname" />
          </Fields>
       </asp:DetailsView>
```

Insert, Update & Delete using DetailsView

- Set *AutoGenerateInsertButton* to true.

- Add *InsertCommand* & *InsertParameters* for data source control.

- Repeat same for Update & Delete just replace word "Insert" with "Update" or "Delete".


Untitled Page

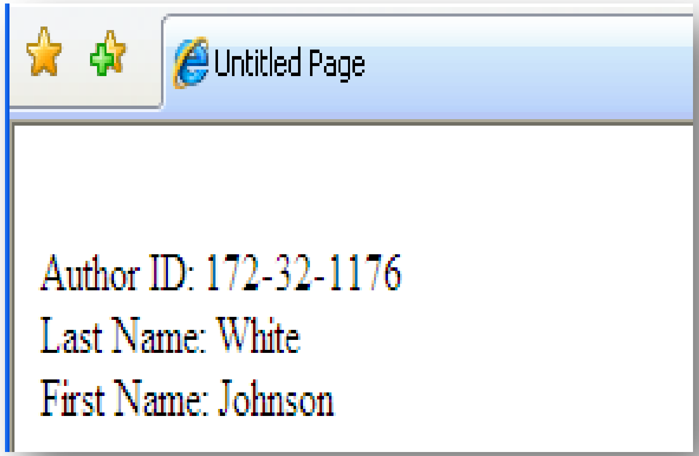
au_id	
au_lname	
au_fname	
phone	
address	
city	
state	
zip	
Insert Cancel	

FormView

- Same like DetailsView Control.
- But it uses templates giving more control to format data.

```
<asp:FormView ID="FormView1" runat="server“
DataKeyNames="au_id" DataSourceID="SqlDataSource1">
          <ItemTemplate>
             Author ID:
             <asp:Label ID="au_idLabel" runat="server“
                          Text='<%# Eval("au_id") %>'></asp:Label><br />
             Last Name:
             <asp:Label ID="au_lnameLabel" runat="server“
             Text='<%# Bind("au_lname") %>'></asp:Label><br />
             First Name:
             <asp:Label ID="au_fnameLabel" runat="server“
             Text='<%# Bind("au_fname") %>'></asp:Label><br />
             <br />
          </ItemTemplate>…</asp:FormView>
```

FormView



Author ID: 172-32-1176

Last Name: White

First Name: Johnson

One can specify templates for edit, insert, pager, header, empty data and footer. **Template allows to specify multiple fields, custom controls, and arbitrary HTML to format UI for data.**

Templates can be used with GridView also.
If one want to edit with validation, use custom edit templates.

DataBinder class

- The *DataBinder* class supports generating and parsing data-binding expressions.

- The syntax of *DataBinder.Eval* method:

  <%# DataBinder.Eval(Container.DataItem, expression) %>

- The *Container.DataItem* expression references the object on which the expression is evaluated.

- The expression is a string with the name of the field to access on the data item object.

- In ASP.NET 2.0, one can use following form.

  <%# Eval(expression) %>

Eval & Bind methods

- *Eval* provides one-way data binding i.e. only reading.
- *Bind* supports two-way data binding i.e. the capability to bind data to controls and submit changes back to the database.
- Else both methods are same. *Bind* can replace *Eval* method.

XPathBinder Class

- Data-bound controls can be associated with raw XML data .

- Data-bound controls are using templates & individual XML fragments can be bound inside the template using the *XPathBinder* object.

- The *XPathBinder.Eval* method accepts an *XmlNode* object along with an XPath expression.

XPathBinder Class

- *XPathBinder* class supports a simplified syntax for its evaluator method.

```
<%# XPathBinder.Eval(Container.DataItem,
“employees/employee/Name”) %>
```

- The *XPathBinder* returns a single node using the XPath query provided.

```
<%# XPath(“employees/employee/Name”) %>
```

SELECT METHOD

- *Select* method returns a list of nodes that match the supplied XPath query.

```
<%# XPathBinder.Select(Container.DataItem,"employees/employee") %>
```

```
<%# XpathSelect("employees/employee") %>
```