Chapter 7

Database Applications (ADO.NET)

Chapter 7
Database Applications (ADO.NET)

System.Data namespace has all the classes you need to access any type of database.

To perform database operations, you use one of the two core ADO.NET namespaces:

- **System.Data.SqlClient** for data access with SQL Server.
- **System.Data.OleDb** namespace for OLE DB data access to other than SQL Server, such as Access, Excel, dBASE, ….

No matter what namespace you use, **SqlClient** or **OleDb**, you must still perform the same tasks to work with any database. For example, to get a **Connection** object, you use the **System.Data.SqlClient.SqlConnection** class if you're accessing SQL Server. For OLE DB, you use the **System.Data.OleDb.OleDbConnection** class.

Classes :

- **Connection** You set its properties and call **Open** method to connect to data source.
- **Command** Actually holds your SQL statements that select, insert, update, and delete records. It also supports *Parameters* collection that makes it easy to support stored procedures.
- **DataReader** object is for read-only, forward-only data access.
- **DataAdapter** sits between the database and a DataSet object. DataAdapters are used in the **disconnected mode**. DataAdapters have InsertCommand, UpdateCommand, SelectCommand, and DeleteCommand properties that enable you to specify how to get or set values in the database.
- **DataSet** is an in-memory representation of data and are used in conjunction with DataAdapters. The DataAdapter provides the Connection and Command objects, and the DataSet provides a place for the data to go.

Connecting to a Database

To work with any database, the first thing you must do is connect to it, using **Connection** object to connect to a database (**SqlConnection** /**OleDbConnection**). When connecting to a database, you must specify the **server** that the database resides on, the **database name**, and the **authentication information** for the database. There are several variations of the **connection string**.

Examples of each of the following types of connection string options:

- Connecting to a Microsoft Access database using the OleDbConnection class
- Connecting to a .NET Framework SDK sample Microsoft data engine (MSDE) database with integrated security
- Connecting to a SQL Server database with integrated security
- Connecting to a SQL Server database passing the user ID and password

VB.NET

```
' Connect to a Microsoft Access database using
System.Data.OleDb.OleDbConnection class
Dim strOleDb As String =  "Provider=Microsoft.Jet.OLEDB.4.0;" _
   & "Data Source=..\Northwind.mdb"
Dim cnOleDb As New OleDbConnection()
cnOleDb.ConnectionString = strOleDb
cnOleDb.Open()

' Connect to the MSDE SDK database using
System.Data.SqlClient.SqlConnection class
Dim strMSDE As String =
"Server=(local)\NetSDK;DataBase=Northwind;" _
   & "Integrated Security=SSPI"
Dim cnMSDE As New SqlConnection()
cnMSDE.ConnectionString = strMSDE
cnMSDE.Open()

' Connect to a local SQL Server database using
System.Data.SqlClient.SqlConnection class
Dim strSQL As String = "Server=localhost;DataBase=Northwind;
Integrated Security=SSPI"
Dim cnSQL As New SqlConnection()
cnSQL.ConnectionString = strSQL
cnSQL.Open()

' Connect to a local SQL Server database using
System.Data.SqlClient.SqlConnection class
Dim strSQL1 As String = "Server=localhost;DataBase=Northwind;
uid=sa;pwd=password"
Dim cnSQL1 As New SqlConnection()
cnSQL1.ConnectionString = strSQL1
cnSQL1.Open()
```

C#

```
// Connect to a Microsoft Access database using
System.Data.OleDb.OleDbConnection class
string strOleDb =  @"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=..\Northwind.mdb";
OleDbConnection cnOleDb = new OleDbConnection();
cnOleDb.ConnectionString = strOleDb;
cnOleDb.Open();

// Connect to the MSDE SDK database using
System.Data.SqlClient.SqlConnection class
string strMSDE =
@"Server=(local)\NetSDK;DataBase=Northwind;Integrated
Security=SSPI";
SqlConnection cnMSDE = new SqlConnection;()
cnMSDE.ConnectionString = strMSDE;
cnMSDE.Open();

// Connect to a local SQL Server database using
System.Data.SqlClient.SqlConnection class
string strSQL =
@"Server=localhost;DataBase=Northwind;Integrated
Security=SSPI";
sqlConnection cnSQL = new SqlConnection();
cnSQL.ConnectionString = strSQL;
cnSQL.Open();

// Connect to a local SQL Server database using
System.Data.SqlClient.SqlConnection class
string strSQL1 =
@"Server=localhost;DataBase=Northwind;uid=sa;pwd=";
SqlConnection cnSQL1 = new SqlConnection();
cnSQL1.ConnectionString = strSQL1;
cnSQL1.Open();
```

The difference between OleDbConnection and SqlConnection

With SQL Server, you pass the server name (in this case, localhost), the database name, and either the integrated security constant or the actual user ID and password. With **OleDbConnection**, you give the database path, and OLE DB provider name.

Properties of the Connection Object

Property	Description
ConnectionTimeout	Gets wait time before terminating the connect attempt and generating error
Database	Gets the name of the current database
DataSource	Gets the name of the instance of SQL Server to which to connect
PacketSize	Gets size (bytes) of network packets used to communicate with SQL Server
ServerVersion	Gets a string containing the version of the instance of SQL Server
State	Gets the current state of the connection
WorkstationId	Gets a string that identifies the database client

When you open a Connection object, you must explicitly close it, by calling **Close**. **SqlConnection** /**OleDbConnection** classes are overloaded, so you can also pass the connection string when you create the connection as the following VB.NET code :

```
Dim cn As New
SqlConnection("Server=localhost;DataBase=Northwind;uid=sa;pwd=password")
```

Using the Command Object and DataReaders

The **Command** object is used to execute SQL statements against a database. The SQL statements can be text or the name of a stored procedure in SQL Server.

You create a **Command** object in one of two ways:

- By calling the **CreateCommand** method of a **Connection** object
- By creating an instance of the **SqlCommand** /**OleDbCommand** class, and passing a valid **Connection** object to the **Command** instance

After you create a **Command** object, you set properties that indicate SQL statement, timeout, connection information, and parameters if there are any .

Common SqlCommand and OleDbCommand Properties

Property Name	Description
CommandText	Gets/sets SQL statement or stored procedure to execute at the data source
CommandTimeout	Gets/sets the wait time before terminating an attempt and generating error
CommandType	Gets/sets a value indicating how the CommandText property is interpreted
Connection	Gets/sets the OleDbConnection or SqlConnection used by this command
Parameters	Gets the OleDbParameterCollection or SqlParameterCollection
Transaction	Gets/sets the transaction in which this command executes

Then you call one of the methods listed below to execute your SQL statement .

Method Name	Description
ExecuteReader	Executes commands that return rows.
ExecuteNonQuery	Executes commands such as T-SQL: **INSERT, DELETE, UPDATE, SET**
ExecuteScalar	Retrieves a **single value** from a database. **Not include aggregate values.**

Using Connection, Command, and DataReader Objects to Retrieve Data

To read the data after you execute **ExecuteReader**, you create a **DataReader** (**SqlDataReader/OleDbDataReader**) that holds the data returned from the database.

C#

```
SqlConnection cn = new SqlConnection
  (@"Server=(local)\NetSDK;DataBase=pubs;Integrated
Security=SSPI");

SqlDataReader dr;                    // Create a SqlDataReader object

SqlCommand cmd = new SqlCommand();        // Create a new
SqlCommand object

cmd.CommandText = "Select au_lname, au_fname from Authors";
cmd.Connection = cn;

cn.Open();              // Open the Connection

dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

string strName;
while (dr.Read())
  {
          strName = dr.GetString(0) + ", " + dr.GetString(1);
          listBox1.Items.Add(strName);
   }
cn.Close();
```

- After you have called the **ExecuteReader** method, you use the **Read** method of the **DataReader** class to loop through the records. Each time a record is read, the position in the **Reader** is advanced to the next record.
- You can use other loop for reading from **DataReader**, as the following VB.NET code:

```
Do Until dr.Read = False
    ' do something with the data
Loop
```

Methods of the SqlDataReader ClassMethod Name

GetSqlBoolean,
GetSqlByte,
GetSqlDateTime,
GetSqlDecimal,
GetSqlDouble,
GetSqlInt16,
GetSqlInt32,
GetSqlInt64,
GetSqlMoney,
GetSqlSingle,
GetSqlString

You'll know if you're using the wrong method when an exception occurs. A safe bet is to use either the numeric ordinal position of the data in the row with the **GetString** method, or the field name and put the field data into a string.

Using ExecuteNonQuery with a Command Object

The ExecuteNonQuery method is used when you aren't returning any data, as in the case of an Insert, Update, or Delete.

VB.NET

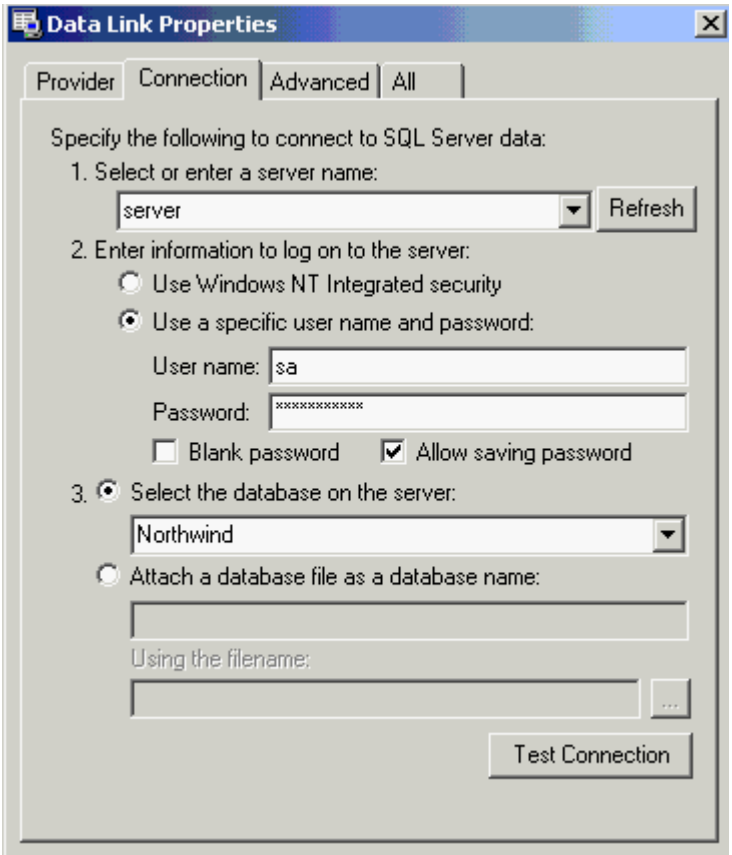
```
Sub DoNonQuery()
   Dim cn As New SqlConnection(
"Server=(local)\NetSDK;DataBase=pubs;" _
     & "Integrated Security=SSPI")

   Dim cmd As New SqlCommand()
```

```
    With cmd
        .CommandText = "Delete from Authors where au_lname = 'Smith'"
        .Connection = cn
        .CommandType = CommandType.Text
    End With

  Try
        cn.Open()
        cmd.ExecuteNonQuery()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    Finally
        If cn.State = ConnectionState.Open Then  cn.Close()
    End Try
End Sub
```

The magic Disconnected Mode



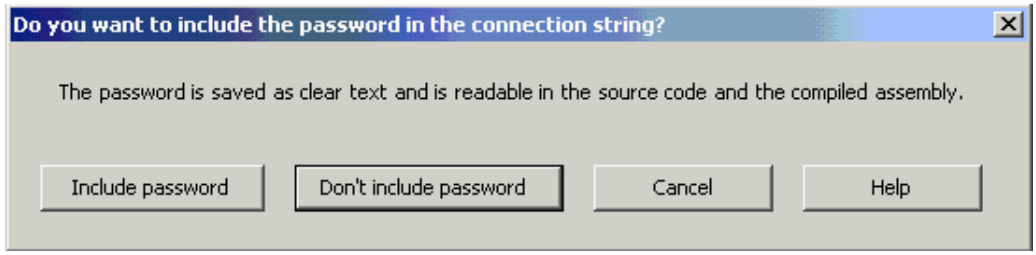
1. create a new C# windows application project say AdoDisconnect

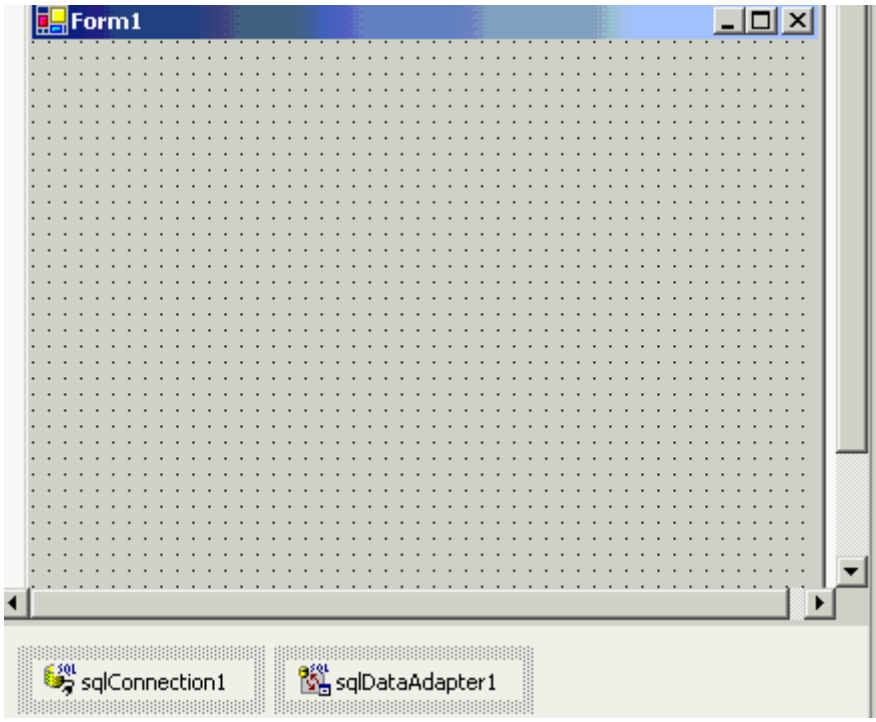
2. From the "server explorer"**, right click on** "data connections" **then choose** "add connection"

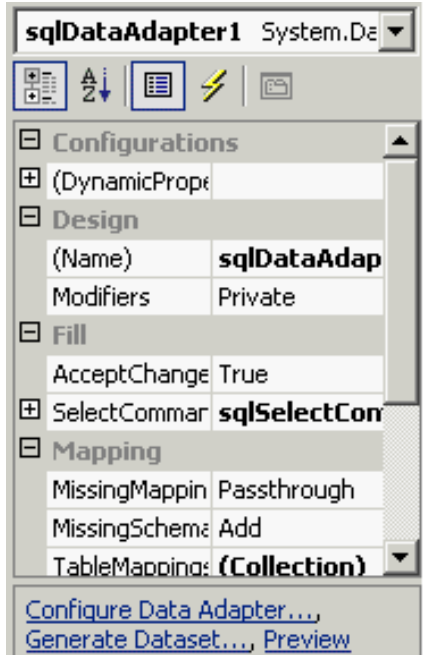
- **on the** "provider" **tab, choose the best provider.**
- **on the** "connection" **tab, enter**
  - **server name (if provider is SQL Server or Oracle)**
  - **user name and password,**
  - **then choose the database .**
- **if you want you test connection, press** "test connection"
- **press** "OK"

3. From the "server explorer"**, right click on** "data connections" **then**

- **from** "tables"**, drag-and-drop the desired table on the target form**
- **click** "include password"





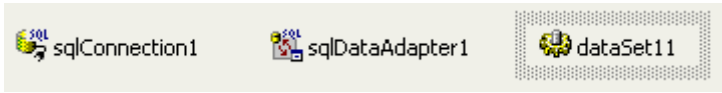


4. notice the added Connection **and** DataAdapter **objects.**

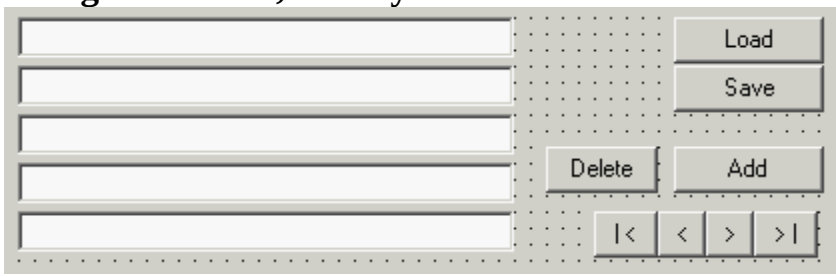
- **you can check their properties, specially** "ConnectionString" **of the** Connection **object**

5. click "Generate Dataset" **link from properties of the** DataAdapter **object, and give it its name.**

- **notice the added** DataSet **object.**



6. design the form , usually as follows:



6. for each relevant field in the table, we create a textbox

- **from the properties of each textbox , set the** "DataBindings" >> "Text" **to the corresponding field from the dataset.**

7. programming the buttons:

- under bttnLoad_Click
  sqlAdapter1.Fill(dataset1);
- under bttnSave_Click
  sqlAdapter1.Update(dataset1);
- under bttnFirstRecord_Click
  this.BindingContext [dataset1, "table_name"].
  Position =0;
- under bttnPreviousRecord_Click
  this.BindingContext [dataset1, "table_name"].
  Position--;
- under bttnNextRecord_Click
  this.BindingContext [dataset1, "table_name"].
  Position++;

- under bttnLastRecord_Click

```
this.BindingContext [ dataset1,  "table_name"].
Position =
        this.BindingContext [ dataset1,
"table_name"].Count -1;
```

- under bttnAdd_Click

```
this.BindingContext [ dataset1,
"table_name"].EndCurrentEdit();
this.BindingContext [ dataset1,
"table_name"].AddNew();
```

- under bttnDelete_Click

```
this.BindingContext [ dataset1,
"table_name"].EndCurrentEdit();
this.BindingContext [ dataset1,
"table_name"].RemoveAt(
        this.BindingContext [ dataset1,  "table_name"].
Position);
```

Using the DataGrid control

```
SqlConnection cn = new SqlConnection
 (@"Server=(local)\NetSDK;DataBase=northwind;
Integrated Security=SSPI");

SqlDataAdapter da = new SqlDataAdapter  ("SELECT * from
Orders", cn);

DataSet ds = new DataSet("Orders");

da.Fill(ds);

// Set the DataSource property of the grid to bind the data
from the DataSet
dataGrid1.DataSource = ds;
```

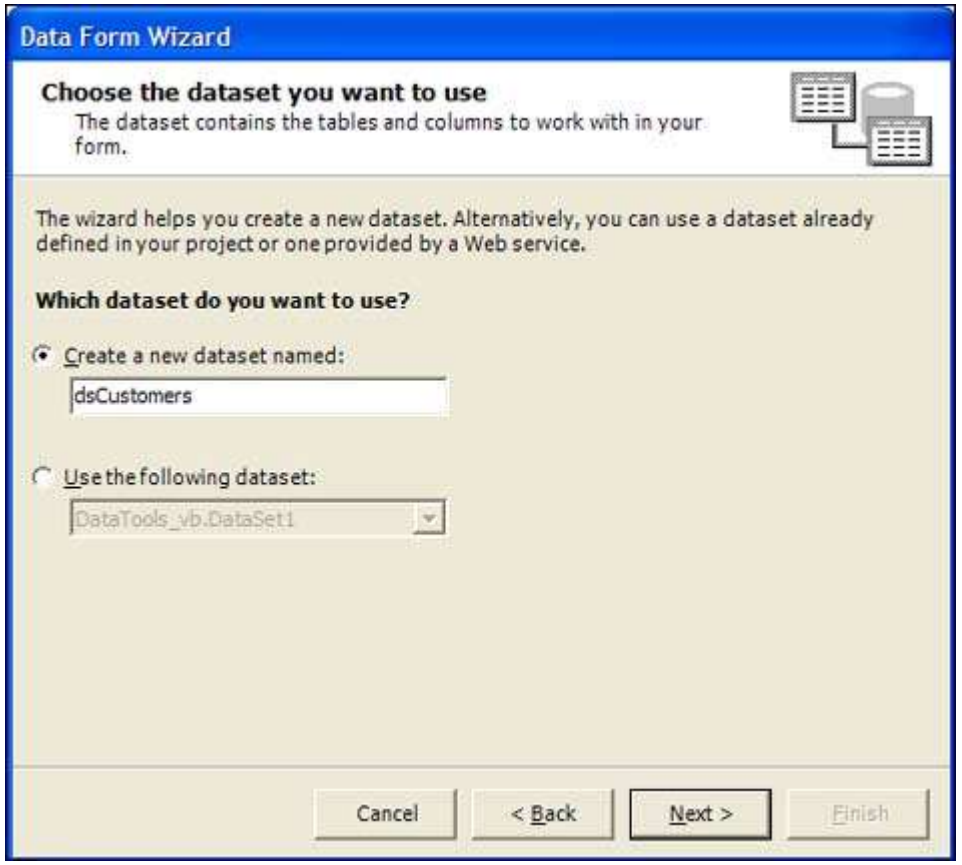
Using the DataForm Wizard to Create Data Entry Forms

1. In your application, right-click the project name in the Solution Explorer and select **Add New Item**. Select **Data Form Wizard** from the Add New Item dialog, and name it CustomersDataEntry,

2. After you click the OK button, the Welcome to the DataForm Wizard dialog appears.

Click the Next button to begin the wizard.

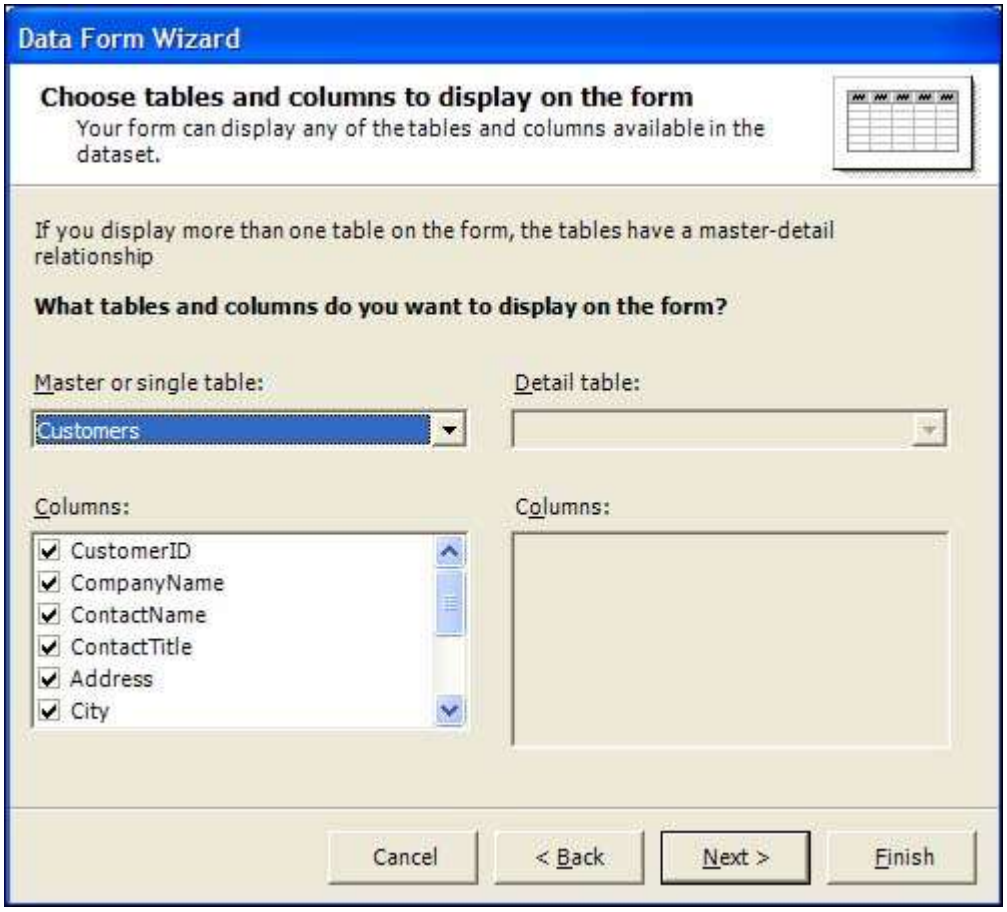
3. The first step in the wizard is to choose the dataset that will hold the databinding. You have two options on this screen: Either create a new dataset or use an existing dataset.



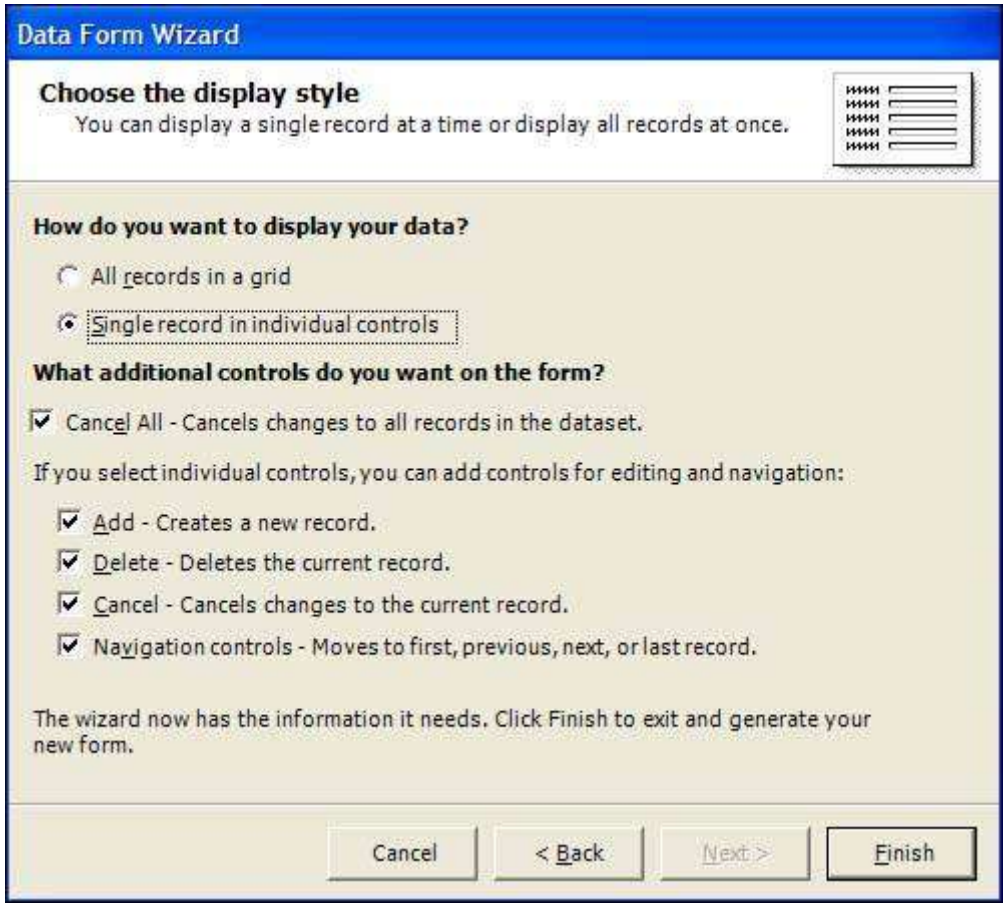
4. Now that you've named the DataSet, click the Next button to select a Connection object.
(DISCUSSED BEFORE !!)

5. Next, you must choose the tables that this form will use. select the Customers table from the left-side tree view and move it to the Selected Items node


Choose tables or views
The tables or views you choose will determine which columns will be available to display on your form.
The wizard creates a data adapter to populate the dataset from available tables or views. If you pick more than one item, you can establish a relationship between them in the next step.
What item (or items) do you want to access?
Available item(s):
Tables
Categories
CustomerCustomerDemo
CustomerDemographics
Employees
EmployeeTerritories
Order Details
Orders
Products
Selected item(s):
Tables
Customers
Views
Cancel
< Back
Next >
Finish



6. Click the Next button to get to the final step of the wizard. At this point, you decide what type of form you want to create.



At this point, you have several options based on what functionality you want to give to the end user. Select the Single Record in Individual Controls option for this form. After you click the Finish button, the form is created.

Calling stored procedures

We didn't delve too much into the Command.Parameters collection today. Being able to send parameters to a stored procedure is extremely important. To get familiar with the syntax, examine the following Visual Basic .NET code:

C#

```
OleDbCommand1.CommandText = "UpdateAuthor";
OleDbCommand1.CommandType =
System.Data.CommandType.StoredProcedure;
OleDbCommand1.Parameters["au_id"].Value = listAuthorID.Text;
OleDbCommand1.Parameters["au_lname"].Value =
txtAuthorLName.Text;
OleDbCommand1.Parameters["au_fname"].Value =
txtAuthorFName.Text;
OleDbConnection1.Open();
OleDbCommand1.ExecuteNonQuery();
OleDbConnection1.Close();
```