

# Visual Basic and Databases

For this tutorial, we're going to create a simple Address Book project. The names and addresses will come from a Microsoft Access database. Download the database before starting these lessons. Once you have saved the database to your own computer, you can begin.

VB.Net allows you many ways to connect to a database or a data source. The technology used to interact with a database or data source is called ADO.NET. The ADO parts stands for Active Data Objects which, admittedly, doesn't explain much. But just like System was a Base Class (leader of a hierarchy, if you like), so is ADO. Forming the foundation of the ADO Base Class are five other major objects:

**Connection**  
**Command**  
**DataReader**  
**DataSet**  
**DataAdapter**

We'll see just what these objects are, and how to use them, in a later section. But we can make a start on the ADO.NET trail by creating a simple Address Book project. All we'll do is see how to use ADO to open up the database you downloaded, and scroll through each entry.

What we're going to be doing is to use a Wizard to create a programme that reads the database and allows us to scroll through it. The wizard will do most of the work for us, and create the controls that allow users to move through the database. The Form we create will look like this when it's finished:

Form1

1 of 5

First Name: John

Surname: Smith

Address1: 12 High Street

Address2: Town District

Address3: Evercrease

Postcode: EV1 2CR

Phone: 222

Email: smith@evercrease.c

Notes: Recently split up with wife. Not coping well.

By clicking the buttons at the top, you can scroll through the database in the image above. We'll make a start in the next part.

NOTE: 2019 users may need to download some data components. (But try the tutorial, first, to see if it works.) Click on **Tools** from the menu at the top, then **Get Tools and Features**. When you see the dialog box, put a tick in the box for **Data storage and processing**. Click the **Modify** button to install.

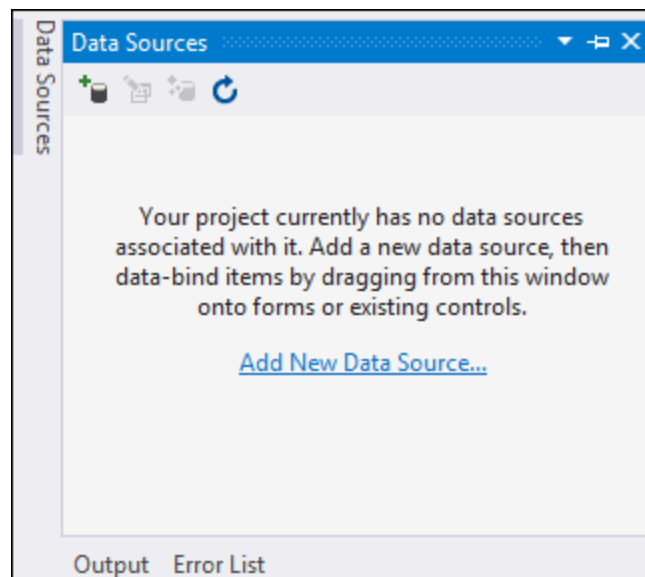
## The Database Wizard in VB NET Express

Let's make a start on our Database project. Create a new project and give it the name **AddressBook**.

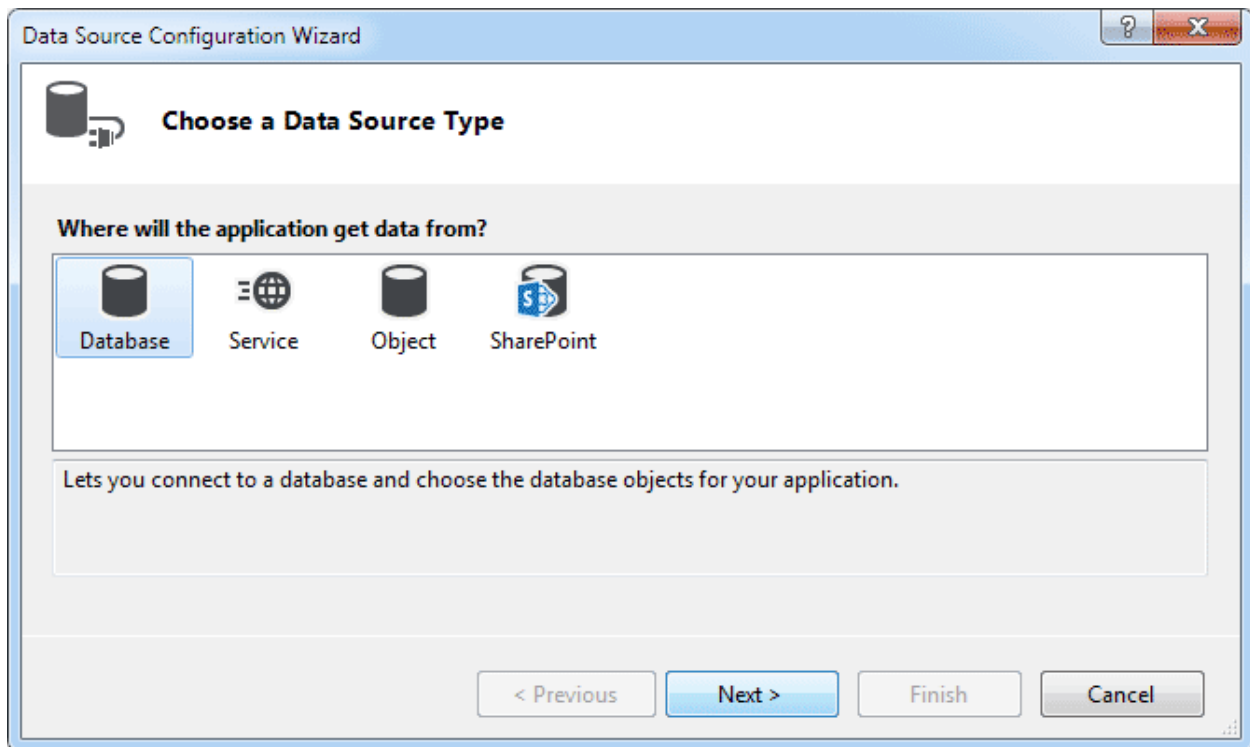
We need to select a Data Source. So click on Data Sources on the left of the Toolbox (If you can't see the tab, click **View > Other Windows > Data Sources**):



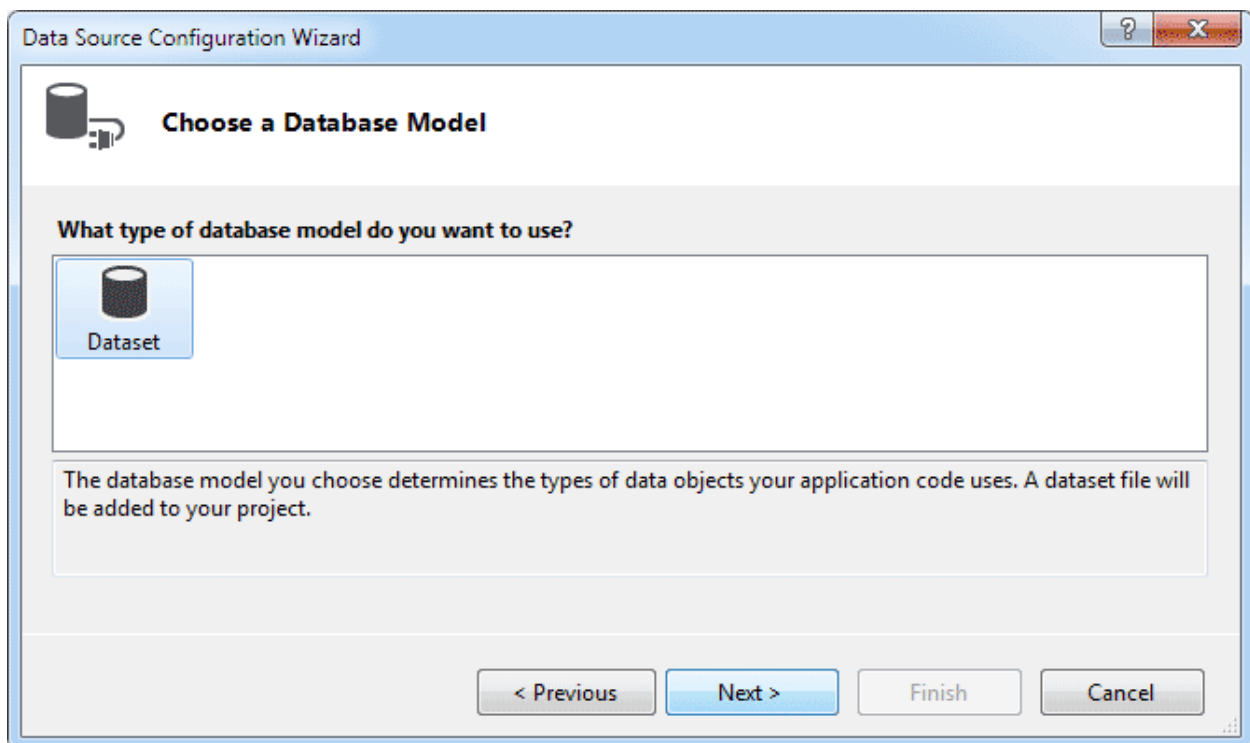
The Data Sources tab should look like this:



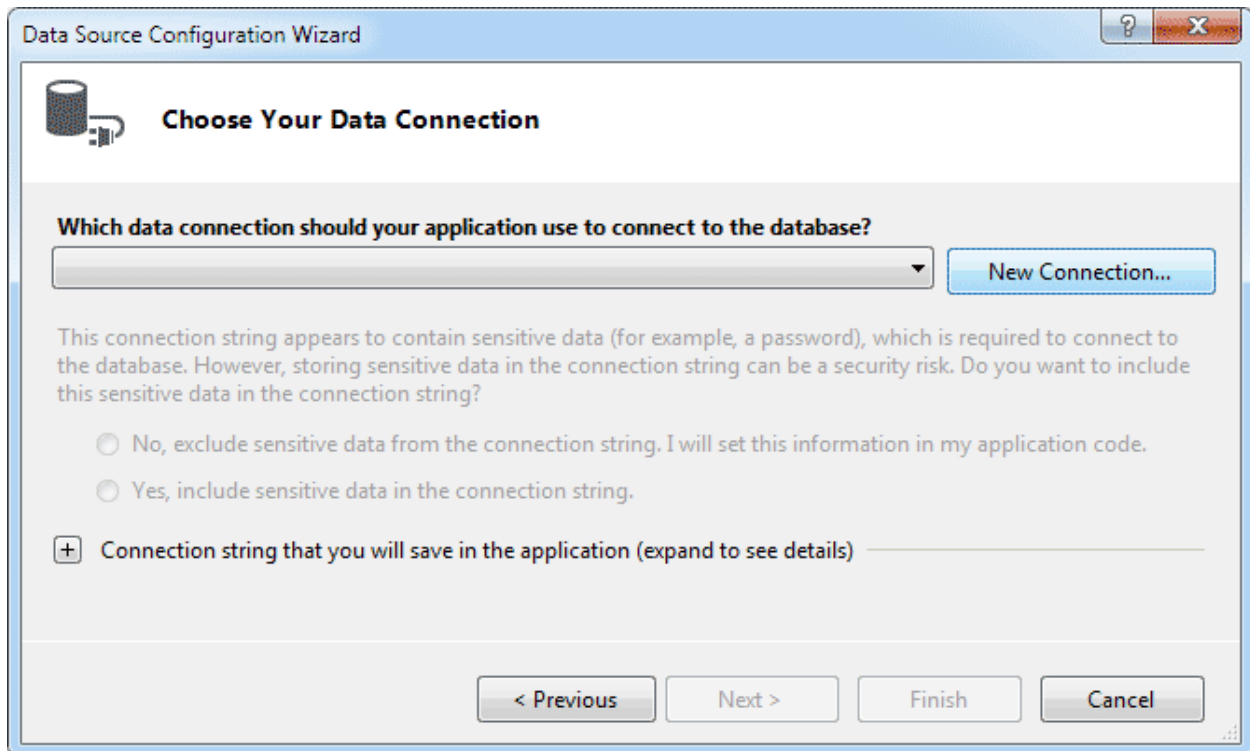
To Add a New Data Source, click on the link "Add New Data Source". When you do, you'll see the following:



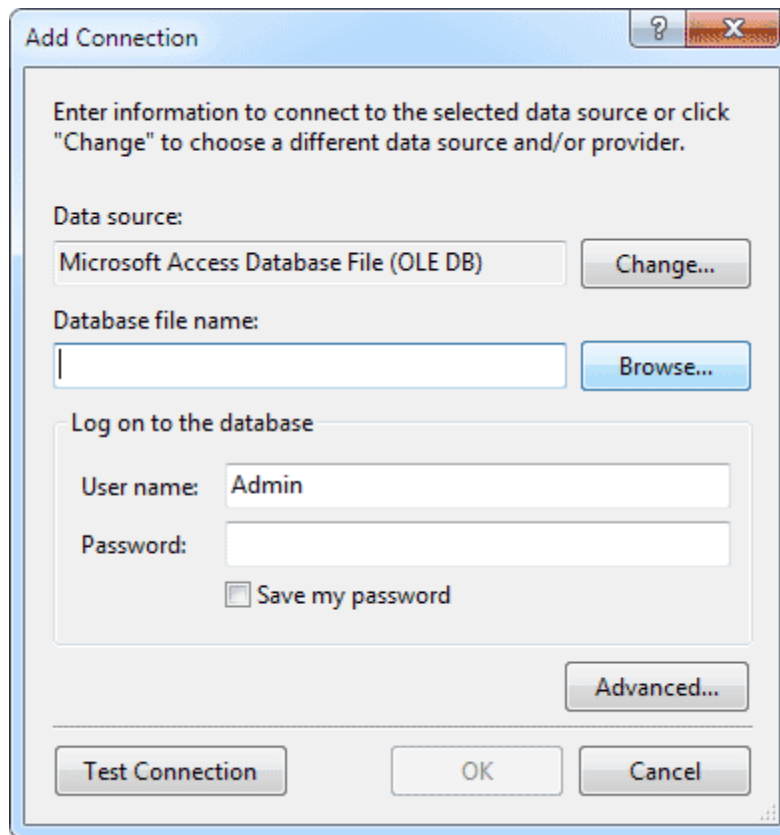
You want to connect to a Database. So select this option, and click Next. You'll see this screen appear:



Select DataSet and click Next. You'll then see a Choose Your Data Connection screen:

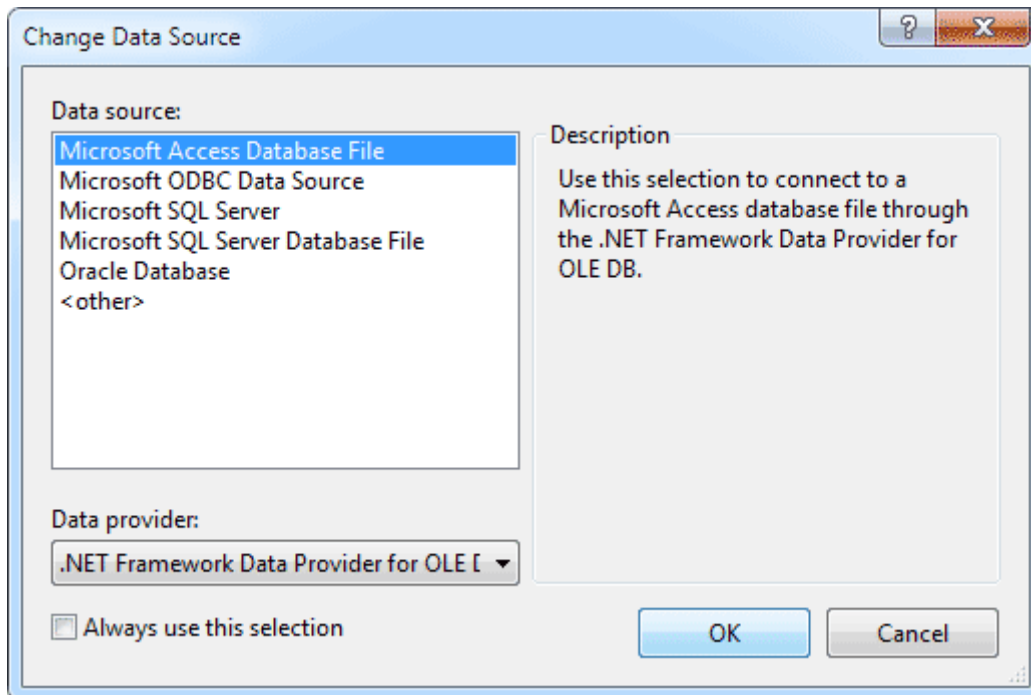


Click the **New Connection** button and another dialogue box pops up - - the Add Connection dialogue box:



The 'Add Connection' dialog box is shown. It has a title bar with a question mark and a close button. The main text says: 'Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.' Below this, there are three sections: 1. 'Data source:' with a text box containing 'Microsoft Access Database File (OLE DB)' and a 'Change...' button. 2. 'Database file name:' with an empty text box and a 'Browse...' button. 3. 'Log on to the database' with a 'User name:' field containing 'Admin', a 'Password:' field, and a checkbox labeled 'Save my password'. At the bottom right is an 'Advanced...' button. At the bottom are three buttons: 'Test Connection', 'OK', and 'Cancel'.

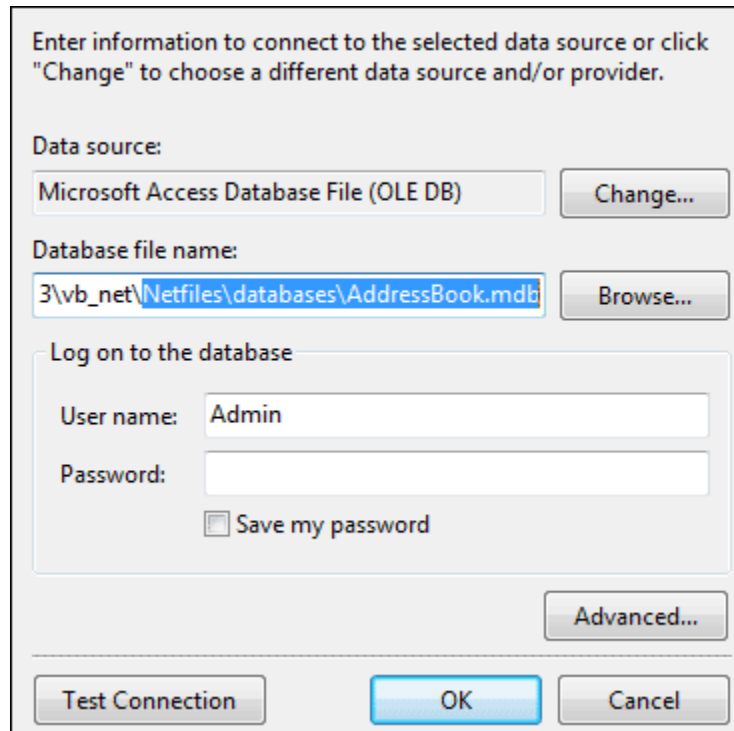
If your Data Source doesn't say Microsoft Access Dataabase File (OLE DB) then click the **Change** button to see this screen:



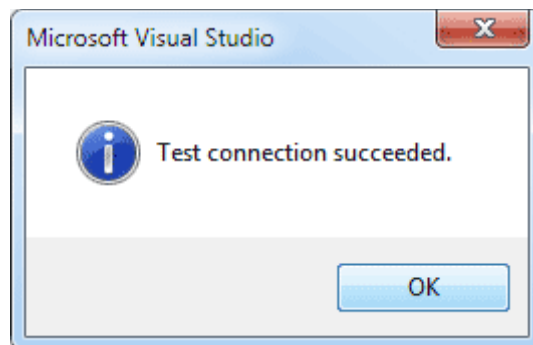
The 'Change Data Source' dialog box is shown. It has a title bar with a question mark and a close button. The main area is divided into two panes. The left pane, titled 'Data source:', contains a list box with the following items: 'Microsoft Access Database File' (highlighted), 'Microsoft ODBC Data Source', 'Microsoft SQL Server', 'Microsoft SQL Server Database File', 'Oracle Database', and '<other>'. The right pane, titled 'Description', contains the text: 'Use this selection to connect to a Microsoft Access database file through the .NET Framework Data Provider for OLE DB.' Below the list box is a 'Data provider:' dropdown menu showing '.NET Framework Data Provider for OLE I'. At the bottom left is a checkbox labeled 'Always use this selection'. At the bottom right are 'OK' and 'Cancel' buttons.

Select Microsoft Access Database File, then click Continue (or OK in some version of Visual Studio). You'll be returned to the Add connection dialogue box.

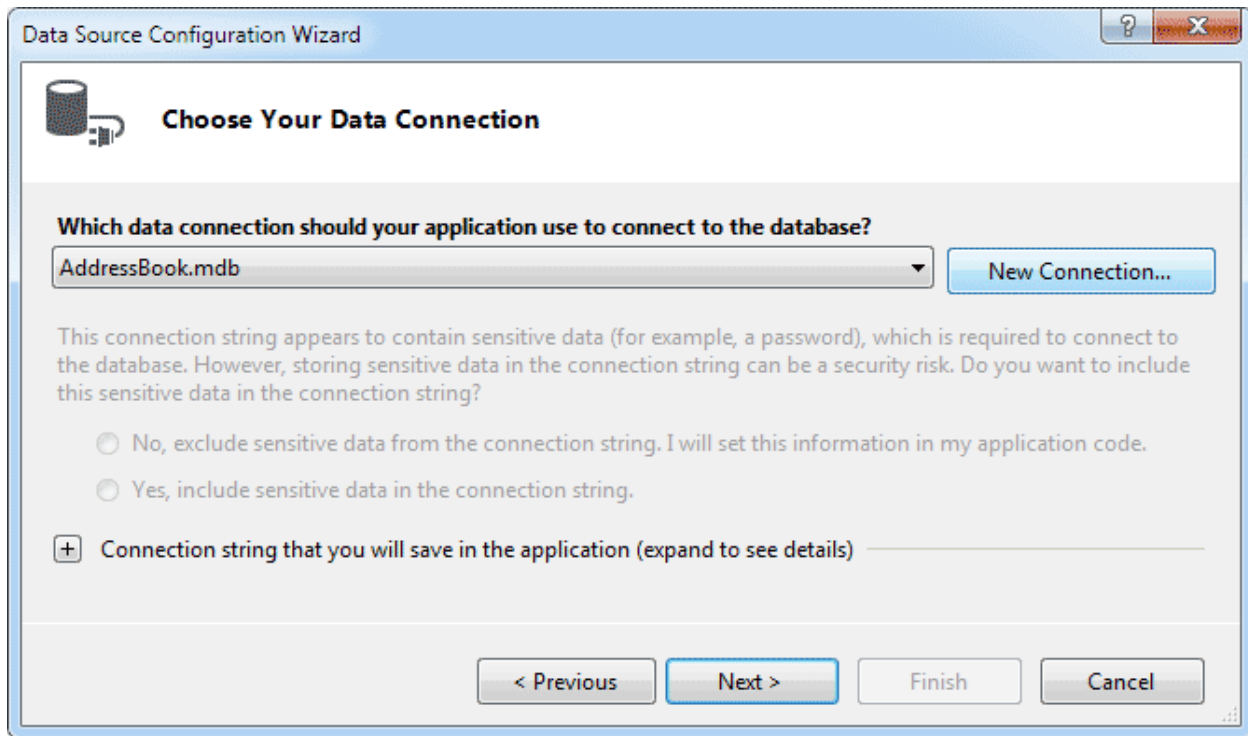
Click the Browse button and navigate to where on your computer you downloaded our Access Database called AddressBook.mdb:



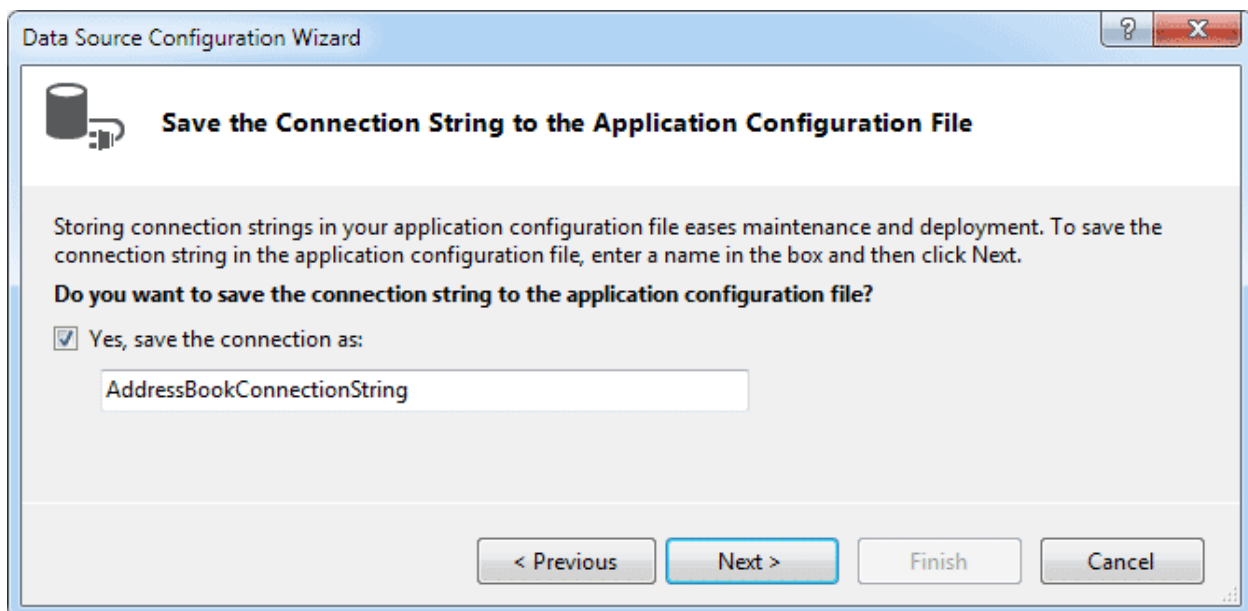
Click **Test Connection** to see if everything is OK, and you'll hopefully see this:



Click the OK button, then click the OK button on the Add Connection dialogue box as well. You will be returned to the Data Source Configuration Wizard, which should now look like this:

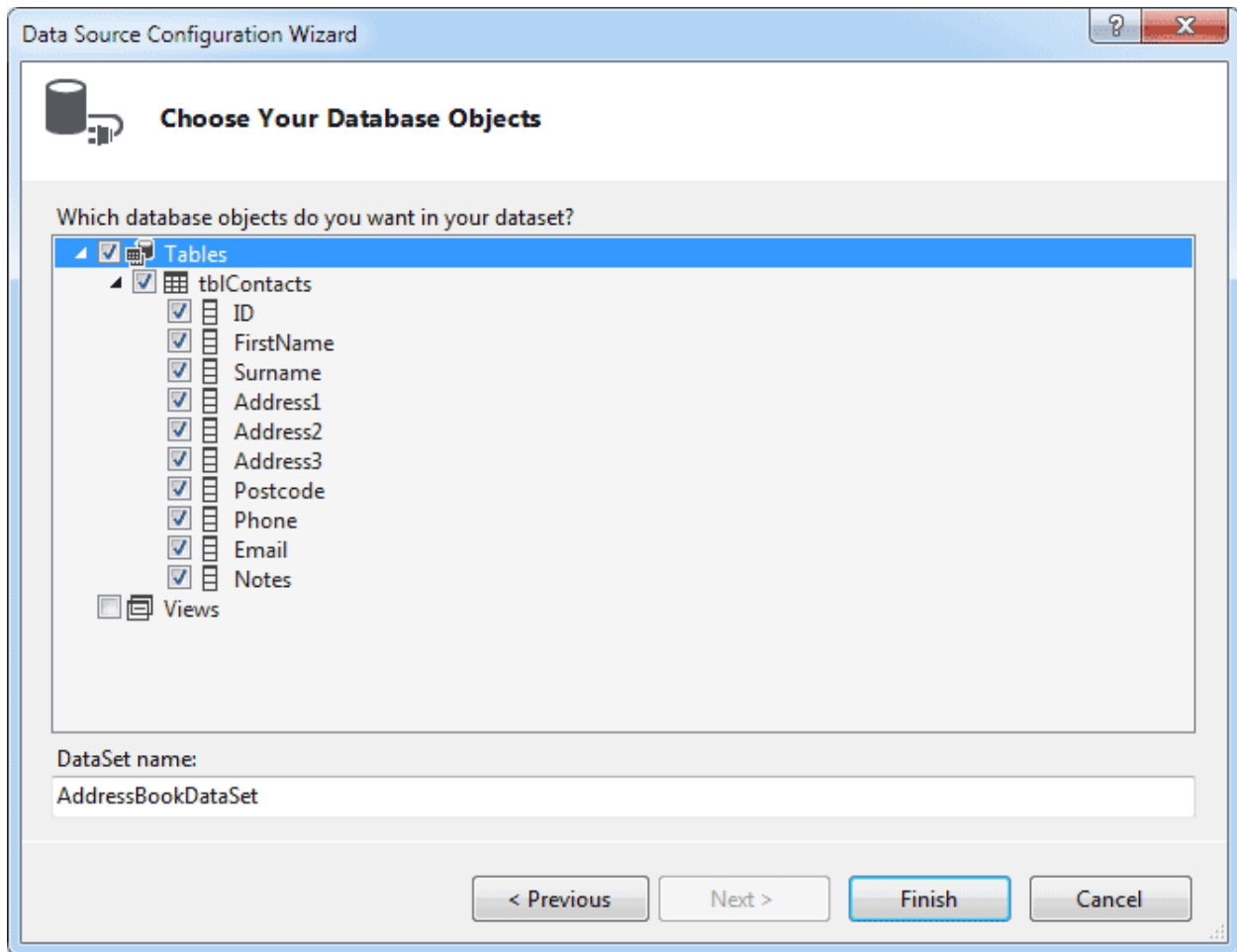


Click Next to move to the next step of the Wizard. You may see a message box appear, however. Click No on the message box to stop VB copying the database each time it runs. You should then see this:



Make sure there's a tick in the box for "Save the connection", and then click Next:

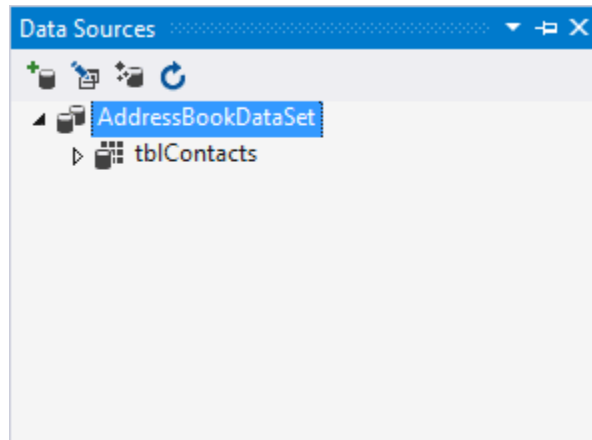




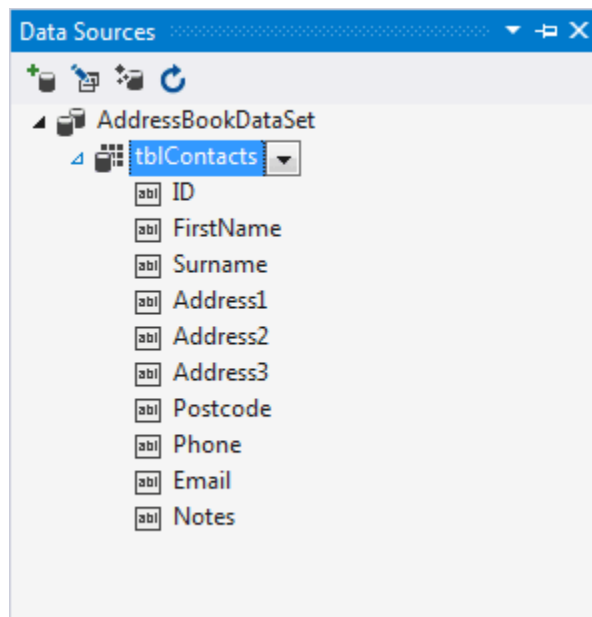
(If you can't see all the fields, click the arrow symbol on the left, next to Tables.)

Here, you can select which tables and fields you want. Tick the **Tables** box to include them all. (You can give your DataSet a different name, if you prefer. The name AddressBookDataSet is one the wizard comes up with.) Click Finish and you're done.

When you are returned to your form, you should notice your new Data Source has been added:

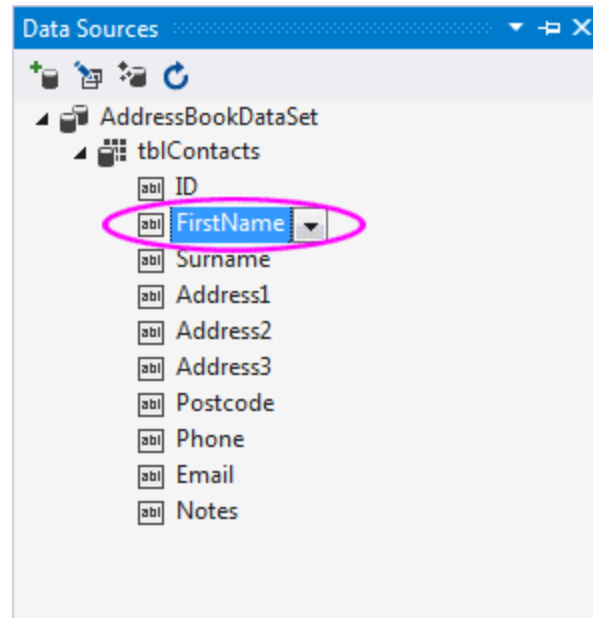


The Data Sources area now displays information about your database. Click the arrow symbol next to **tblContacts**:



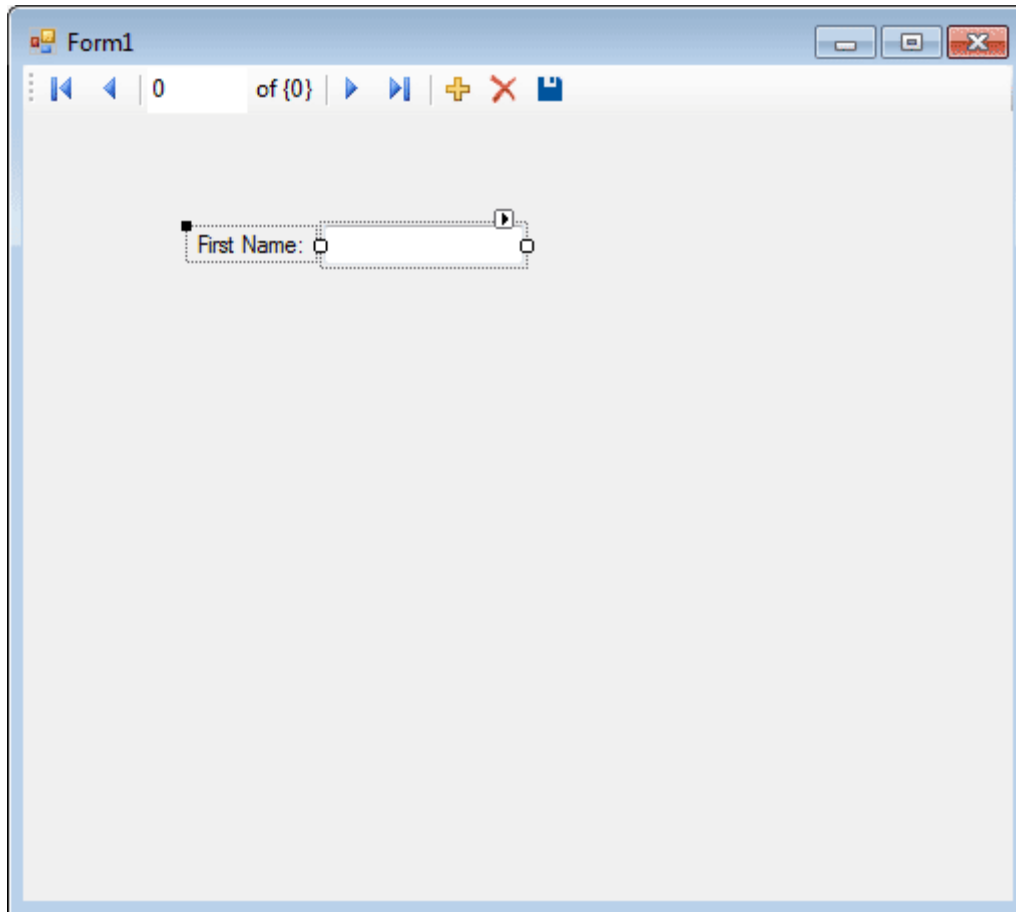
All the Fields in the Address Book database are now showing.

To add a Field to your Form, click on one in the list. Hold down your left mouse button, and drag it over to your form:

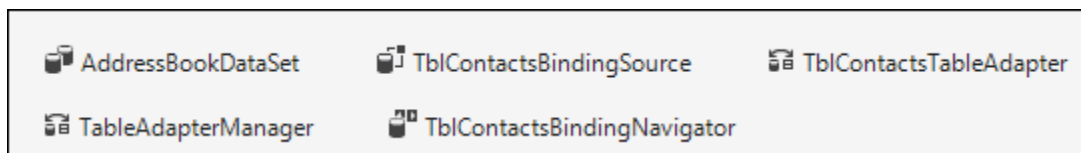


In the image above, the **FN**ame field is being dragged on the Form. Your mouse cursor will change shape.

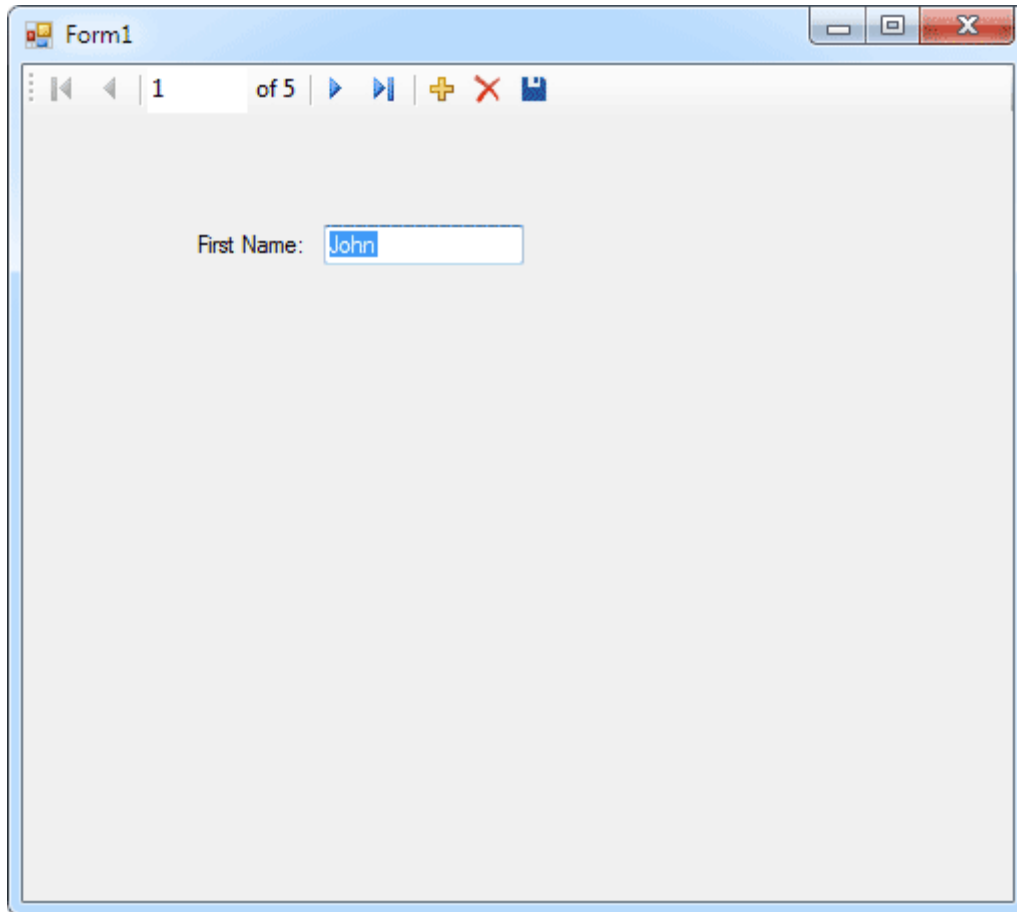
When your Field is over the Form, let go of your left mouse button. A textbox and a label will be added:



There are two other things to notice: a navigation bar appears at the top of the form, and a lot of strange objects have appeared in the object area at the bottom:



We'll explore the Objects in a later section. But notice the Navigation bar at the top of the form. Run your programme by hitting the F5 key on your keyboard. You should see this:



Click the Navigation arrows to scroll through the database, where it says 1 of 5. When you've played around with the controls, stop the form from running, and return to Design View.

Drag and Drop more Fields to your form. But don't align them yet. We'll see an easy way to do this. But once you've dragged the fields to your form, it might look like this:

I'm sure you'll agree - that's a very untidy form. But there's a very easy way to align all your controls. Try this:

1. Click on a label with your left mouse button
2. Hold down the Ctrl key on your keyboard, and select a second label
3. With the Ctrl key still held down, click each label in turn (leave the Notes label unselected)
4. When all the labels are selected, click on the **Format** menu at the top
5. From the Format menu select **Align > Lefts**. The left edges of the labels will align themselves
6. From the Format menu select **Vertical Spacing > Make Equal**. The space between each textbox will then be the same
7. Now do the same with the textboxes (but leave the Notes textbox unselected)

For the Notes Textbox, set the **MultiLine** property to **True** and resize the textbox. With your new controls added, and nicely aligned, press F5 to run your form. Your form might then be something like this:

Form1

1 of 5

First Name: John

Surname: Smith

Address1: 12 High Street

Address2: Town District

Address3: Evercrease

Postcode: EV1 2CR

Phone: 222

Email: smith@evercrease.c

Notes: Recently split up with wife. Not coping well.

Click the Navigation icons to move backwards and forwards through your database.

In the next part, you'll move away from the Wizards and learn how to add your own programming code to open up and manipulate databases.

## Write your own Database code in VB .NET

In this next section, we'll take a look at the objects that you can use to open and read data from a Database. We'll stick with our Access database, the AddressBook.mdb one, and recreate what the Wizard has done. That way, you'll see for yourself just what is going on behind the scenes.

So close any open projects, and create a new one. Give it whatever name you like, and let's begin.

If you haven't yet downloaded the Address Book database, you can get it here:

[Download the Address Book Database](#)

## The Connection Object

The Connection Object is what you need if you want to connect to a database. There are a number of different connection objects, and the one you use depends largely on the type of database you're connecting to. Because we're connecting to an Access database, we'll need something called the OLE DB connection object.

OLE stands for Object Linking and Embedding, and its basically a lot of objects (COM objects) bundled together that allow you to connect to data sources in general, and not just databases. You can use it, for example, to connect to text files, SQL Server, email, and a whole lot more.

There are a number of different OLE DB objects (called data providers), but the one we'll use is called "**Jet**". Others are SQL Server and Oracle.

So place a button on your form. Change the **Name** property to **btnLoad**. Double click your button to open up the code window. Add the following line:

**Dim con As New OleDb.OleDbConnection**

The variable **con** will now hold the **Connection Object**. Notice that there is a full stop after the OleDb part. You'll then get a pop up box from where you can select OleDbConnection. We're also creating a **New** object on this line. This is the object that you use to connect to an Access database.

## Setting a Connection String

There are Properties and Methods associated with the Connection Object, of course. We want to start with theConnectionString property. This can take MANY parameters . Fortunately, we only need a few of these.

We need to pass two things to our new **Connection Object**: the technology we want to use to do the connecting to our database; and where the database is. (If your database was password and user name protected, you would add these two parameters as well. Ours isn't, so we only need the two.)

The technology is called the **Provider**; and you use **Data Source** to specify where your database is. So add this to your code:

```
Dim dbProvider As String
Dim dbSource As String
Dim MyDocumentsFolder As String
Dim TheDatabase As String
Dim FullDatabasePath As String
```

```
dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"
```



First, we set up five string variables. The last line specifies the provider technology we're going to use to do the connecting, in this case Jet.OLEDB.4.0:

```
"PROVIDER=Microsoft.Jet.OLEDB.4.0;"
```

Notice the semicolon at the end of the line. This is needed.

If you're trying to connect to a modern Access database, however, then you may need a different provider, one called ACE:

```
dbProvider = "PROVIDER=Microsoft.ACE.OLEDB.12.0;"
```

There's even a version 15.0:

```
dbProvider = "PROVIDER=Microsoft.ACE.OLEDB.15.0;"
```

Next, you need the name of your database, and where it is. This could be a location on your computer, or on a server. What we'll do is to copy our database to the Documents folder of Windows. First, add a line for the name of your database:

```
TheDatabase = "/AddressBook.mdb"
```

To point to the Documents folder, add this line:

```
MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
```

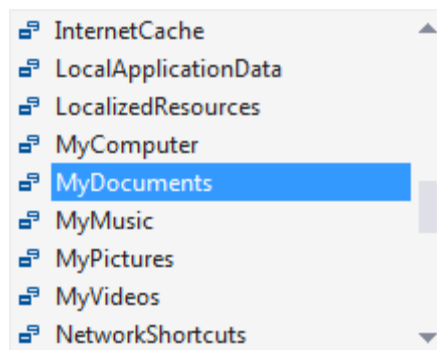
The line is a bit complex. After the equal sign, though, we have this:

```
Environment.GetFolderPath()
```

The folder path you're getting goes between the round brackets of GetFolderPath:

```
Environment.SpecialFolder.MyDocuments
```

The Special Folder in this case is the MyDocuments folder (MyDocuments is the name of the Documents folder.) Other special folders are:



After setting a database name and pointing to the Documents folder, we can combine it for a data source:

```
dbSource = "Data Source = " & FullDatabasePath
```

What we're doing here is adding a path and a database to the text string "Data Source = ".

Your coding window should now look like like this:

```
Private Sub btnLoad_Click(sender As Object, e As EventArgs) Handles btnLoad.Click

    Dim con As New OleDb.OleDbConnection

    Dim dbProvider As String
    Dim dbSource As String
    Dim MyDocumentsFolder As String
    Dim TheDatabase As String
    Dim FullDatabasePath As String

    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    dbSource = "Data Source = " & FullDatabasePath

End Sub
```

We now need a connection string.

At the top of the code, we set up this line:

```
Dim con As New OleDb.OleDbConnection
```

The variable con now holds an OleDbConnection object. This con object needs a database provider, and a data source. You do this with the ConnectionString property:

```
con.ConnectionString = PROVIDER_AND_PATH_HERE
```

After the equal sign, you need a database provider and a data source (the data source is the path to your database).

We could have put these two things all on one rather long line:

```
con.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source =  
'C:/databases/AddressBook.mdb'"
```

Here, we have the PROVIDER and the DATA SOURCE together. Separating the two is a semicolon. The Source is a file path pointing to a folder on our hard drive. This works perfectly well for a simple connection string.

But the path to our database (the Documents folder) is rather long, so we've spread out our code a bit more.

It does the same thing, though: passes the `ConnectionString` property the name of a data provider, and a path to the database.

So add this line to your code:

```
con.ConnectionString = dbProvider & dbSource
```

Here, we're using an ampersand character to combine the provider and data source.

Now that we have a `ConnectionString`, we can go ahead and open the database. This is quite easy - just use the `Open` method of the `Connection` Object:

```
con.Open( )
```

Once open, the connection has to be closed again. This time, just use the `Close` method:

```
con.Close( )
```

Add the following four lines to your code:

```
con.Open( )  
MessageBox.Show("Database is now open")  
  
con.Close()  
MessageBox.Show("Database is now Closed")
```

Your coding window will then look like this:

```

Private Sub btnLoad_Click(sender As Object, e As EventArgs) Handles btnLoad.Click

    Dim con As New OleDb.OleDbConnection

    Dim dbProvider As String
    Dim dbSource As String
    Dim MyDocumentsFolder As String
    Dim TheDatabase As String
    Dim FullDatabasePath As String

    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    dbSource = "Data Source = " & FullDatabasePath

    con.ConnectionString = dbProvider & dbSource

    con.Open()
    MessageBox.Show("Database is now open")

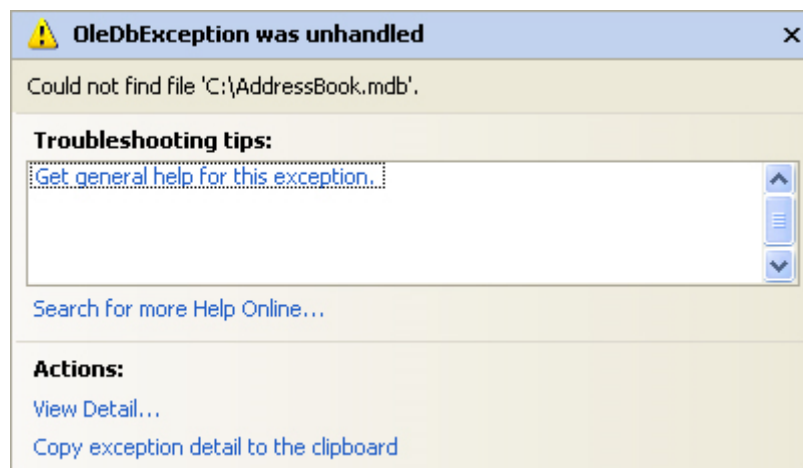
    con.Close()
    MessageBox.Show("Database is now Closed")

End Sub

```

Before testing out your code, make sure you copy the AddressBook database file over to the Documents folder on your computer.

Test out your new code by running your programme. Click your button and the two message boxes should display. If they don't, make sure your Data Source path is correct. If it isn't, you might see a **OleDbException was unhandled** error message:



The error message is a bit on the vague and mysterious side. But what it's saying is that it can't find the path to the database, so it can't Open the connection. The line `con.Open` in your code will then be highlighted in green. You need to specify the correct path to your database. When you do, you'll see the message boxes from our code, and not the big one above.

Now that we've opened a connection to the database, we need to read the information from it. This is where the `DataSet` and the `DataAdapter` come in.

## Data Sets and Data Adapters

In the [previous part](#), you learned how to set up a Connection Object. This was so that you could open a connection to the database itself. But that's not the end of it. The data from the database needs to be stored somewhere, so that we can manipulate it.

ADO.NET uses something called a **DataSet** to hold all of your information from the database (you can also use a `DataTable`, if all you want to do is read information, and not have people write to your database.). But the **DataSet** (and Data Table) will hold a copy of the information from the database.

The `DataSet` is not something you can draw on your form, like a Button or a Textbox. The `DataSet` is something that is hidden from you, and just stored in memory. Imagine a grid with rows and columns. Each imaginary row of the `DataSet` represents a Row of information in your Access database. And each imaginary column represents a Column of information in your Access database (called a Field in database terminology).

This, then, is a `DataSet`. But what's a Data Adapter?

The Connection Object and the `DataSet` can't see each other. They need a go-between so that they can communicate. This go-between is called a Data Adapter. The Data Adapter contacts your Connection Object, and then executes a query that you set up. The results of that query are then stored in the `DataSet`.

The Data Adapter and `DataSet` are objects. You set them up like this:

```
Dim ds As New DataSet
Dim da As OleDb.OleDbDataAdapter

da = New OleDb.OleDbDataAdapter( sql, con )
```

The code needs a little explaining, though. First, the Data Adapter.

## The Data Adapter

The Data Adapter is a property of the OLEDB object, hence the full stop between the two:

### **OleDb.OleDbDataAdapter**

We're passing this object to the variable called **da**. This variable will then hold a reference to the Data Adapter.

While the second line in the code above sets up a reference to the Data Adapter, the third line creates a new Data Adapter object. You need to put two things in the round brackets of the Object declaration: Your SQL string (which we'll get to shortly), and your connection object. Our Connection Object is stored in the variable which we've called **con**. (Like all variable you can call it practically anything you like. We've gone for something short and memorable.) You then pass the New Data Adapter to your variable (**da** for us):

```
da = New OleDb.OleDbDataAdapter(sql, con )
```

We need something else, though. The **sql** in between the round brackets is the name of a variable. We haven't yet set this up. We'll have a look at SQL in a moment. But bear in mind what the Data Adaptor is doing: *Acting as a go-between for the Connection Object and the Data Set*

## Structured Query Language

SQL (pronounced SeeKwel), is short for Structured Query Language, and is a way to query and write to databases (not just Access). The basics are quite easy to learn. If you want to grab all of the records from a table in a database, you use the SELECT word. Like this:

```
SELECT * FROM Table_Name
```

SQL is not case sensitive, so the above line could be written:

```
Select * from Table_Name
```

But your SQL statements are easier to read if you type the keywords in uppercase letters. The keywords in the lines above are **SELECT** and **FROM**. The asterisk means "All Records". Table\_Name is the name of a table in your database. So the whole line reads:

```
"SELECT all the records FROM the table called Table_Name"
```

You don't need to select all (\*) the records from your database. You can just select the columns that you need. The name of the table in our database is **tblContacts**. If we wanted to select just the first name and surname columns from this table, we can specify that in our SQL String:

```
SELECT tblContacts.FirstName, tblContacts.Surname FROM tblContacts
```

When this SQL statement is executed, only the FirstName and Surname columns from the database will be returned.

There are a lot more SQL commands, but for our purposes this is enough.

Because we want to **SELECT** all (\*) the records from the table called **tblContacts**, we pass this string to the string variable we have called **sql**:

```
sql = "SELECT * FROM tblContacts"
```

Your code window should now look like this (though the file path to your database might be different):

```

Private Sub btnLoad_Click(sender As Object, e As EventArgs) Handles btnLoad.Click

    Dim con As New OleDb.OleDbConnection    'THE CONNECTION OBJECT

    Dim dbProvider As String                'HOLDS THE PROVIDER
    Dim dbSource As String                  'HOLDS THE DATA SOURCE
    Dim MyDocumentsFolder As String         'HOLDS THE DOCUMENTS FOLDER
    Dim TheDatabase As String               'HOLDS THE DATABASE NAME
    Dim FullDatabasePath As String          'HOLDS THE DATABASE PATH

    Dim ds As New DataSet                  'HOLDS A DataSet OBJECT
    Dim da As OleDb.OleDbDataAdapter       'HOLDS A DataAdapter OBJECT
    Dim sql As String                      'HOLDS A SQL STRING

    'SET UP THE PROVIDER
    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    'SET THE DATABASE AND WHERE THE DATABASE IS
    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    'SET THE DATA SOURCE
    dbSource = "Data Source = " & FullDatabasePath

    'SET THE CONNECTION STRING
    con.ConnectionString = dbProvider & dbSource

    'OPEN THE DATABASE
    con.Open()

    'STORE THE SQL STRING
    sql = "SELECT * FROM tblContacts"

    'PASS THE SQL STRING AND CONNECTION OBJECT TO THE DATA_ADAPTER
    da = New OleDb.OleDbDataAdapter(sql, con)

    MessageBox.Show("Database is now open")

    'CLOSE THE DATABASE
    con.Close()
    MessageBox.Show("Database is now Closed")

End Sub

```

Now that the Data Adapter has selected all of the records from the table in our database, we need somewhere to put those records - in the **DataSet**.



## Filling the DataSet

The Data Adapter can Fill a DataSet with records from a Table. You only need a single line of code to do this, after the line **da = New OleDb.OleDbDataAdapter(sql, con)**:

```
da.Fill(ds, "AddressBook")
```

As soon as you type the name of your Data Adapter (**da** for us), you'll get a pop up box of properties and methods. Select Fill from the list, then type a pair of round brackets. In between the round brackets, you need two things: the **Name** of your DataSet (**ds**, in our case), and an identifying name. This identifying name can be anything you like. But it is just used to identify this particular Data Adapter Fill. We could have called it "Bacon Sandwich", if we wanted:

```
da.Fill(ds, "Bacon Sandwich ")
```

The code above still works. But it's better to stick to something a little more descriptive than "Bacon Sandwich"!

Add the new line after the creation of the Data Adaptor:

```
da = New OleDb.OleDbDataAdapter(sql, con)  
da.Fill(ds, "AddressBook")
```

And that's it. The DataSet (**ds**) will now be filled with the records we selected from the table called **tblContact**. There's only one slight problem - nobody can see the data yet! We'll tackle that in the next part.

## Displaying the Data in the DataSet

In the [previous section](#), we saw what Data Adaptors and DataSets were. We created a Data Adaptor so that it could fill a DataSet with records from our database. What we want to do now is to display the records on a Form, so that people can see them. So so this:

1. Add two textboxes to your form
2. Change the **Name** properties of your textboxes to **txtFirstName** and **txtSurname**
3. Go back to your code window
4. Add the following two lines:

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(0).Item(1)  
txtSurname.Text = ds.Tables("AddressBook").Rows(0).Item(2)
```

You can add them after the line that closes the connection to the database. Once the DataSet has been filled, a connection to a database can be closed.

Your code should now look like this:

```

Private Sub btnLoad_Click(sender As Object, e As EventArgs) Handles btnLoad.Click

    Dim con As New OleDb.OleDbConnection    'THE CONNECTION OBJECT

    Dim dbProvider As String                'HOLDS THE PROVIDER
    Dim dbSource As String                  'HOLDS THE DATA SOURCE
    Dim MyDocumentsFolder As String         'HOLDS THE DOCUMENTS FOLDER
    Dim TheDatabase As String               'HOLDS THE DATABASE NAME
    Dim FullDatabasePath As String          'HOLDS THE DATABASE PATH

    Dim ds As New DataSet                   'HOLDS A DataSet OBJECT
    Dim da As OleDb.OleDbDataAdapter       'HOLDS A DataAdapter OBJECT
    Dim sql As String                       'HOLDS A SQL STRING

    'SET UP THE PROVIDER
    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    'SET THE DATABASE AND WHERE THE DATABASE IS
    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    'SET THE DATA SOURCE
    dbSource = "Data Source = " & FullDatabasePath

    'SET THE CONNECTION STRING
    con.ConnectionString = dbProvider & dbSource

    'OPEN THE DATABASE
    con.Open()

    'STORE THE SQL STRING
    sql = "SELECT * FROM tblContacts"

    'PASS THE SQL STRING AND CONNECTION OBJECT TO THE DATA_ADAPTER
    da = New OleDb.OleDbDataAdapter(sql, con)

    'FILL THE DATASET WITH RECORDS FROM THE DATABASE TABLE
    da.Fill(ds, "AddressBook")

    MessageBox.Show("Database is now open")

    'CLOSE THE DATABASE
    con.Close()
    MessageBox.Show("Database is now Closed")

    txtFirstName.Text = ds.Tables("AddressBook").Rows(0).Item(1)
    txtSurname.Text = ds.Tables("AddressBook").Rows(0).Item(2)

End Sub

```

Before the code is explained, run your programme and click the button. You should see "John Smith" displayed in your two textboxes.

So let's examine the code that assigns the data from the DataSet to the textboxes. The first line was this:

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(0).Item(1)
```

It's rather a long line! But after the equals sign, you type the name of your DataSet (**ds** for us). After a full stop, select **Tables** from the popup list. The **Tables** property needs something in between round brackets. Quite bizarrely, this is NOT the name of your database table! It's that identifier you used with the Data Adapter Fill. We used the identifier "**AddressBook**". If we had used "Bacon Sandwich" then we'd put this:

```
ds.Tables("Bacon Sandwich")
```

But we didn't, so our code is:

```
ds.Tables("AddressBook")
```

Type a full stop and you'll see another list popping up at you. Select **Rows** from the list. In between round brackets, you need a number. This is a Row number from the DataSet. We want the first row, which is row zero in the DataSet:

```
ds.Tables("AddressBook").Rows( 0 )
```

Type full stop after Rows(0) and the popup list appears again. To identify a **Column** from the DataSet, you use **Item**. In between round brackets, you type which column you want:

```
ds.Tables("AddressBook").Rows(0).Item( 1 )
```

In our Access database, column zero is used for an ID field. The **FirstName** column is the second column in our Access database. Because the Item collection is zero based, this is item 1 in the DataSet.

You can also refer to the column name itself for the Item property, rather than a number. So you can do this:

```
ds.Tables("AddressBook").Rows(0).Item("FirstName")  
ds.Tables("AddressBook").Rows(0).Item("Surname")
```

If you get the name of the column wrong, then VB throws up an error. But an image might clear things up. The image below shows what the items and rows are in the database.

	0	1	2	3	4	5
Row	ID	FirstName	Surname	Address1	Address2	Address3
0	22	John	Smith	12 High Street	Town District	Evercrease
1	23	Jane	Smith	11 High Street	Town District	Evercrease
2	24	Ira	Irate	2 Cold Pond La	Pond District	Evercrease
	25	Sienna	Blue	1 Council Stree	Council District	Evercrease
	26	John	Tucker	2 Copper Lane	Police District	Evercrease
	(AutoNumber)					

The image shows which are the **Rows** and which are the **Items** in the Access database Table. So the **Items** go down and the **Rows** go across.

However, we want to be able to scroll through the table. We want to be able to click a button and see the next record. Or click another button and see the previous record. You can do this by incrementing the Row number. To see the next record, we'd want this:

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(1).Item(1)
txtSurname.Text = ds.Tables("AddressBook").Rows(1).Item(2)
```

The record after that would then be:

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(2).Item(1)
txtSurname.Text = ds.Tables("AddressBook").Rows(2).Item(2)
```

So by incrementing and decrementing the Row number, you can navigate through the records. Let's see how that's done.

## Navigate a Database with VB .NET

You saw in the [previous section](#) that you can navigate through the records of a database by incrementing or decrementing the Row number of the DataSet. In this section, we're going to see a more practical example of how to do that.

To navigate through the dataset, let's change our form. By adding some navigation buttons, we can duplicate [what the wizard did](#). We'll also need to move the code we already have. So let's start with that.

At the moment, all our code is in the Button we added to the form. We're going to delete this button, so we need to move it out of there. The variable declarations can be moved right to the top of the coding window. That way, any button can see the variables. So move your variables declarations to the top, as in the image below (don't forget to add the **Dim inc As Integer** line):

```

Public Class Form1

    Dim inc As Integer
    Dim con As New OleDb.OleDbConnection 'THE CONNECTION OBJECT

    Dim dbProvider As String 'HOLDS THE PROVIDER
    Dim dbSource As String 'HOLDS THE DATA SOURCE
    Dim MyDocumentsFolder As String 'HOLDS THE DOCUMENTS FOLDER
    Dim TheDatabase As String 'HOLDS THE DATABASE NAME
    Dim FullDatabasePath As String 'HOLDS THE DATABASE PATH

    Dim ds As New DataSet 'HOLDS A DataSet OBJECT
    Dim da As OleDb.OleDbDataAdapter 'HOLDS A DataAdapter OBJECT
    Dim sql As String 'HOLDS A SQL STRING

```

We can move a few lines to the Form Load event. So, create a Form Load event, as you did in a previous section. Now move all but the textbox lines to there. Your coding window should then look like this (you can delete the message box lines, or just comment them out):

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load

    'SET UP THE PROVIDER
    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    'SET THE DATABASE AND WHERE THE DATABASE IS
    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    'SET THE DATA SOURCE
    dbSource = "Data Source = " & FullDatabasePath

    'SET THE CONNECTION STRING
    con.ConnectionString = dbProvider & dbSource

    'OPEN THE DATABASE
    con.Open()

    'STORE THE SQL STRING
    sql = "SELECT * FROM tblContacts"

    'PASS THE SQL STRING AND CONNECTION OBJECT TO THE DATA_ADAPTER
    da = New OleDb.OleDbDataAdapter(sql, con)

    'FILL THE DATASET WITH RECORDS FROM THE DATABASE TABLE
    da.Fill(ds, "AddressBook")

    'CLOSE THE DATABASE
    con.Close()

End Sub

```

For your button, all you should have left are these two lines:

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)
```

Since we're going to be deleting this button, this code can be moved. Because all the buttons need to put something into the textboxes, the two lines we have left are an ideal candidate for a Subroutine. So add the following Sub to your code:

```
Private Sub NavigateRecords()

txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)

End Sub
```

When we navigate through the DataSet, we'll call this subroutine.

Now that all of your code has gone from your button, you can delete the button code altogether (including the **Private Sub ... End Sub** parts). Return to your form, click on the button to select it, then press the delete key on your keyboard. This will remove the button itself from your form. (You can also right click on the button, and then select Delete from the menu.)

Here's what your coding window should like:

```

Public Class Form1

    Dim inc As Integer
    Dim con As New OleDb.OleDbConnection    'THE CONNECTION OBJECT

    Dim dbProvider As String                'HOLDS THE PROVIDER
    Dim dbSource As String                  'HOLDS THE DATA SOURCE
    Dim MyDocumentsFolder As String         'HOLDS THE DOCUMENTS FOLDER
    Dim TheDatabase As String               'HOLDS THE DATABASE NAME
    Dim FullDatabasePath As String          'HOLDS THE DATABASE PATH

    Dim ds As New DataSet                  'HOLDS A DataSet OBJECT
    Dim da As OleDb.OleDbDataAdapter        'HOLDS A DataAdapter OBJECT
    Dim sql As String                      'HOLDS A SQL STRING

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load

        'SET UP THE PROVIDER
        dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

        'SET THE DATABASE AND WHERE THE DATABASE IS
        TheDatabase = "/AddressBook.mdb"
        MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        FullDatabasePath = MyDocumentsFolder & TheDatabase

        'SET THE DATA SOURCE
        dbSource = "Data Source = " & FullDatabasePath

        'SET THE CONNECTION STRING
        con.ConnectionString = dbProvider & dbSource

        'OPEN THE DATABASE
        con.Open()

        'STORE THE SQL STRING
        sql = "SELECT * FROM tblContacts"

        'PASS THE SQL STRING AND CONNECTION OBJECT TO THE DATA_ADAPTER
        da = New OleDb.OleDbDataAdapter(sql, con)

        'FILL THE DATASET WITH RECORDS FROM THE DATABASE TABLE
        da.Fill(ds, "AddressBook")

        'CLOSE THE DATABASE
        con.Close()

    End Sub

    Private Sub NavigateRecords()

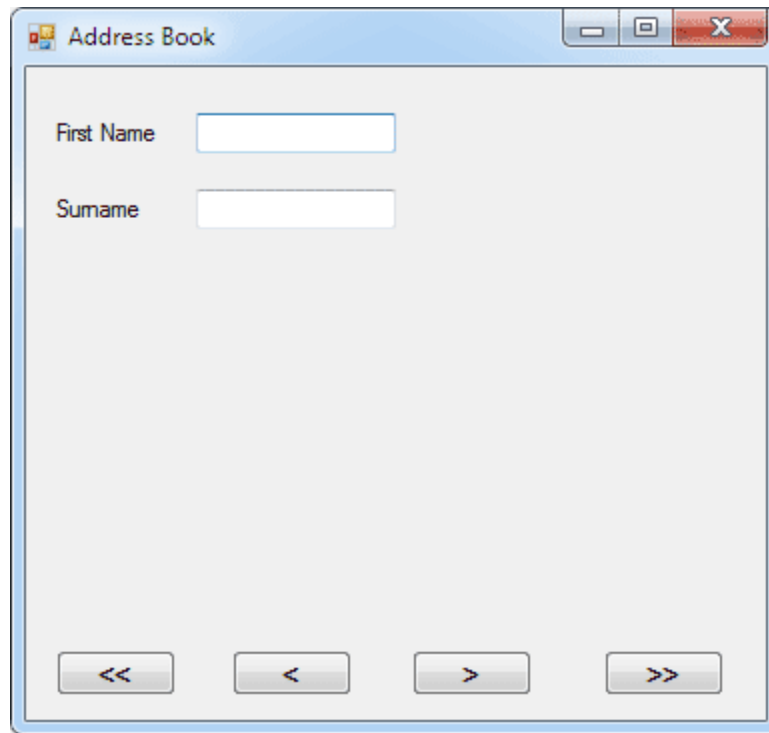
        txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)
        txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)

    End Sub

End Class

```

Now you can re-design the form. Add four new buttons, and change the Name properties to: btnNext, btnPrevious, btnFirst, and btnLast. Change the Text properties to >, <, <<, and >>. Your form will then look like this:



Just a couple of more things to set up before we get started. Add a new variable declaration to the top of your code, just under the Dim inc As Integer line. Add this:

```
Dim MaxRows As Integer
```

We can store how many rows are in the DataSet with this variable. You can get how many rows are in the DataSet with this:

```
MaxRows = ds.Tables("AddressBook").Rows.Count
```

So the Rows property has a Count Method. This simply counts how many rows are in the DataSet. We're passing that number to a variable called MaxRows. You can then test what is in the variable, and see if the inc counter doesn't go past it. You need to do this because VB throws up an error message if try to go past the last row in the DataSet. (Previous versions of VB had some called an EOF and BOF properties. These checked the End of File and Before End of File. These properties have now gone.)

Add the following two lines of code to the Form Load Event of Form1:

```
MaxRows = ds.Tables("AddressBook").Rows.Count  
inc = - 1
```



Your code should then look like this:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles Me.Load

    'SET UP THE PROVIDER
    dbProvider = "PROVIDER=Microsoft.Jet.OLEDB.4.0;"

    'SET THE DATABASE AND WHERE THE DATABASE IS
    TheDatabase = "/AddressBook.mdb"
    MyDocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    FullDatabasePath = MyDocumentsFolder & TheDatabase

    'SET THE DATA SOURCE
    dbSource = "Data Source = " & FullDatabasePath

    'SET THE CONNECTION STRING
    con.ConnectionString = dbProvider & dbSource

    'OPEN THE DATABASE
    con.Open()

    'STORE THE SQL STRING
    sql = "SELECT * FROM tblContacts"

    'PASS THE SQL STRING AND CONNECTION OBJECT TO THE DATA_ADAPTER
    da = New OleDb.OleDbDataAdapter(sql, con)

    'FILL THE DATASET WITH RECORDS FROM THE DATABASE TABLE
    da.Fill(ds, "AddressBook")

    'CLOSE THE DATABASE
    con.Close()

    'GET HOW MANY ROWS ARE IN THE DATABASE TABLE
    MaxRows = ds.Tables("AddressBook").Rows.Count

    'SET A VALUE FOR THE INC VARIABLE
    inc = -1

End Sub
```

Notice the other line of code for the Form Load event:

```
inc = - 1
```

This line sets the inc variable to minus one when the form loads. When the Buttons are clicked, this will ensure that we're moving the counter on by the correct amount.

In the next Part, we'll see how the Buttons on the form work.

# Coding for the Navigate Buttons

In [the last lesson](#), you set up a Form with four buttons and two textboxes. In this lesson, you'll add the code for the buttons.

## How to Move Forward One Record at a Time

Double click your **Next Record** button to access the code. Add the following If ... Else Statement:

```
If inc <> MaxRows - 1 Then
```

```
inc = inc + 1
```

```
NavigateRecords()
```

```
Else
```

```
MessageBox.Show("No More Rows")
```

```
End If
```

We're checking to see if the value in **inc** does not equal the value in **MaxRows** - 1. If they are both equal then we know we've reached the last record in the DataSet. In which case, we just display a message box. If they are not equal, these two lines get executed:

```
inc = inc + 1
```

```
NavigateRecords()
```

First, we move the **inc** counter on by one. Then we call the Sub we set up:

### **NavigateRecords()**

Our Subroutine is where the action takes place, and the values from the DataSet are placed in the textboxes. Here it is again:

```
Private Sub NavigateRecords()
```

```
txtFirstName.Text = ds.Tables("AddressBook").Rows(inc).Item(1)
```

```
txtSurname.Text = ds.Tables("AddressBook").Rows(inc).Item(2)
```

```
End Sub
```

The part that moves the record forward (and backwards soon) is this part:

**Rows( inc )**

Previously, we hard-coded this with:

**Rows( 0 )**

Now the value is coming from the variable called **inc**. Because we're incrementing this variable with code, the value will change each time the button is clicked. And so a different record will be displayed.

You can test out your Next button. Run your programme and click the button. You should now be able to move forward through the DataSet. When you get to the end, you should see the message box display "No More Rows".

None of the other button will work yet, of course. So let's move backwards.

### Move Back One Record at a Time

To move backwards through the DataSet, we need to decrement the **inc** counter. All this means is deducting 1 from whatever is currently in inc.

But we also need to check that inc doesn't go past zero, which is the first record in the DataSet. Here's the code to add to your **btnPrevious**:

```
If inc > 0 Then
```

```
inc = inc - 1
```

```
NavigateRecords()
```

```
Else
```

```
MessageBox.Show("First Record")
```

```
End If
```

So the If statement first checks that **inc** is greater than zero. If it is, inc gets 1 deducted from. Then the NavigateRecords() subroutine gets called. If **inc** is zero or less, then we display a message.

When you've finished adding the code, test your programme out. Click the Previous button first. The message box should display, even though no records have been loaded into the textboxes. This is because the variable **inc** has a value of -1 when the form first loads. It only gets moved on to zero when the Next button is clicked. You could amend your IF Statement to this:

```
If inc > 0 Then
```

```
inc = inc - 1
```

```
NavigateRecords()
```

```
ElseIf inc = -1 Then
```

```
MessageBox.Show("No Records Yet")
```

```
ElseIf inc = 0 Then
```

```
MessageBox.Show("First Record")
```

```
End If
```

This new If Statement now checks to see if inc is equal to minus 1, and displays a message if it does. It also checks if inc is equal to zero, and displays the "First Record" message box.

### Moving to the Last Record in the DataSet

To jump to the last record in the DataSet, you only need to know how many records have been loaded into the DataSet - the **MaxRows** variable in our code. You can then set the **inc** counter to that value, but minus 1. Here's the code to add to your **btnLast**:

```
If inc <> MaxRows - 1 Then
```

```
inc = MaxRows - 1
```

```
NavigateRecords()
```

```
End If
```

The reason we're saying **MaxRows - 1** is that the row count might be 5, say, but the first record in the DataSet starts at zero. So the total number of records would be zero to 4. Inside of the If Statement, we're setting the **inc** counter to MaxRows - 1, then calling the NavigateRecords() subroutine.

That's all we need to do. So run your programme. Click the Last button, and you should see the last record displayed in your textboxes.

### Moving to the First Record in the DataSet

Moving to the first record is fairly straightforward. We only need to set the **inc** counter to zero, if it's not already at that value. Then call the Sub:

```
If inc <> 0 Then
```

inc = 0

NavigateRecords()

End If

Add the code to your **btnFirst**. Run your programme and test out all of your buttons. You should be able to move through the names in the database, and jump to the first and last records.

As yet, though, we don't have a way to add new records, to update records, or to delete them. Let's do that next.

## Add, Update and Delete Records

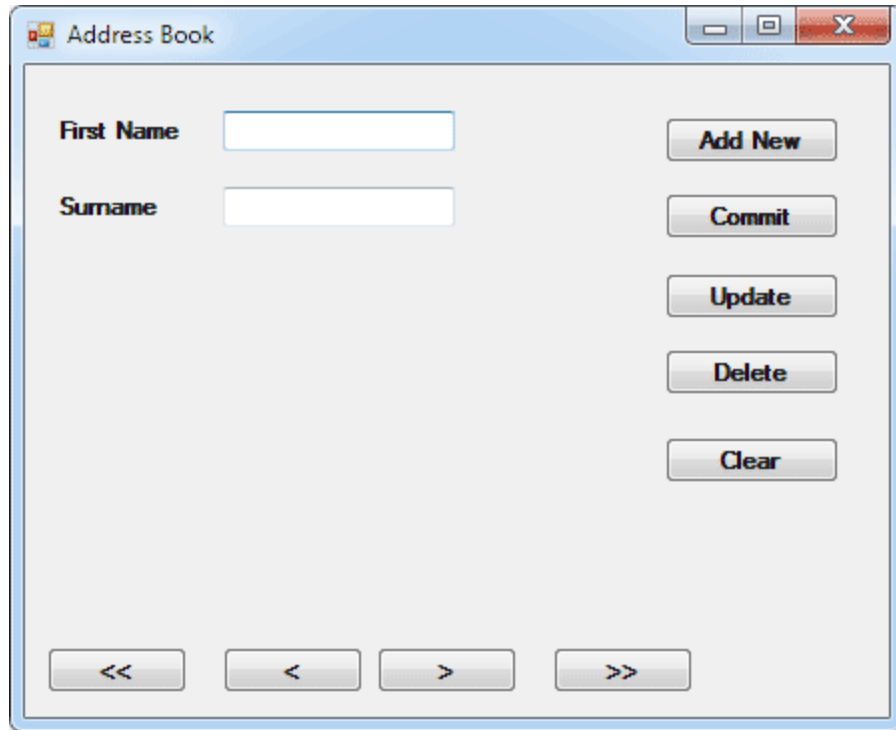
In the [last section](#), you learned how to move through the records in your DataSet, and how to display the records in Textboxes on your form. In this lesson, we'll see how to add new records, how to delete them and how to Update a records.

Before we start the coding for these new buttons, it's important to understand that the DataSet is ***disconnected*** from the database. What this means is that if you're adding a new record, you're not adding it to the database: you're adding it to the **DataSet**! Similarly, if you're updating or Deleting, you doing it to the DataSet, and **NOT** to the database. After you have made all of your changes, you THEN commit these changes to the database. You do this by issuing a separate command. But we'll see how it all works.

You'll need to add a few more buttons to your form - five of them. Change the **Name** properties of the new Buttons to the following:

**btnAddNew**  
**btnCommit**  
**btnUpdate**  
**btnDelete**  
**btnClear**

Change the **Text** properties of the buttons to **Add New Record**, **Commit Changes**, **Update Record**, **Delete Record**, and **Clear/Cancel**. Your form might look something like this:



We'll start with the Update Record button.

### Updating a Record

To reference a particular column (item) in a row of the DataSet, the code is this:

```
ds.Tables("AddressBook").Rows(2).Item(1)
```

That will return whatever is at Item 1 on Row 2.

As well as returning a value, you can also set a value. You do it like this:

```
ds.Tables("AddressBook").Rows(2).Item(1) = "Jane"
```

Now Item 1 Row 2 will contain the text "Jane". This won't, however, effect the database! The changes will just get made to the **DataSet**. To illustrate this, add the following code to your **btnUpdate**:

```
ds.Tables("AddressBook").Rows(inc).Item(1) = txtFirstName.Text  
ds.Tables("AddressBook").Rows(inc).Item(2) = txtSurname.Text
```

```
MessageBox.Show("Data updated")
```

Run your programme, and click the **Next Record** button to move to the first record. "John" should be displayed in your first textbox, and "Smith" in the second textbox. Click inside the

textboxes and change "John" to "Joan" and "Smith" to "Smithy". (Without the quotes). Now click your **Update Record** button. Move to the next record by clicking your **Next Record** button, and then move back to the first record. You should see that the first record is now "Joan Smithy".

Close down your programme, then run it again. Click the **Next Record** button to move to the first record. It will still be "John Smith". The data you updated has been lost! So here, again, is why:

### **"Changes are made to the DataSet, and NOT to the Database"**

To update the database, you need some extra code. Amend your code to this (the new lines are in bold, red text):

```
Dim cb As New OleDb.OleDbCommandBuilder(da)

ds.Tables("AddressBook").Rows(inc).Item(1) = txtFirstName.Text
ds.Tables("AddressBook").Rows(inc).Item(2) = txtSurname.Text

da.Update(ds, "AddressBook")

MessageBox.Show("Data updated")
```

The first new line is this:

**Dim cb As New OleDb.OleDbCommandBuilder(da)**

To update the database itself, you need something called a **Command Builder**. The Command Builder will build a SQL string for you. In between round brackets, you type the name of your Data Adapter, **da** in our case. The command builder is then stored in a variable, which we have called **cb**.

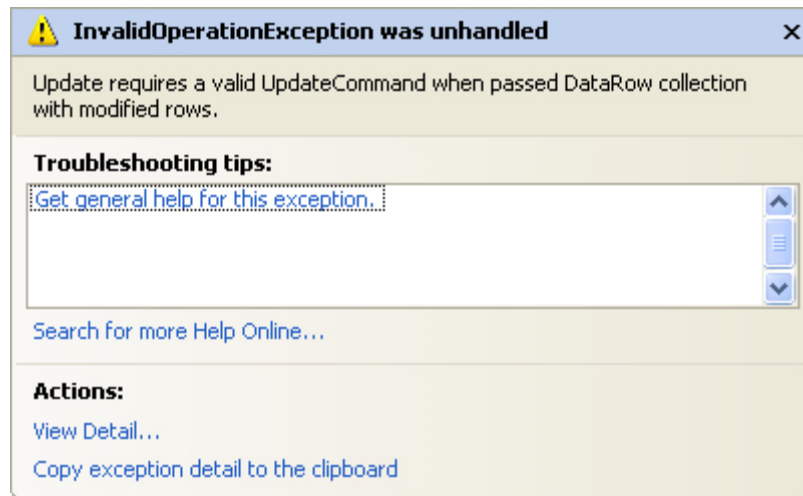
The second new line is where the action is:

**da.Update(ds, "AddressBook")**

The **da** variable is holding our Data Adapter. One of the methods of the Data Adapter is **Update**. In between the round brackets, you need the name of your DataSet (**ds**, for us). The **"AddressBook"** part is optional. It's what we've called our DataSet, and is here to avoid any confusion.

But the Data Adapter will then contact the database. Because we have a Command Builder, the Data Adapter can then update your database with the values from the DataSet.

Without the Command Builder, though, the Data Adapter can't do its job. Try this. Comment out the Command Builder line (put a single quote before the "D" of Dim). Run your programme again, and then try and update a record. You'll get this error message:



The error is because you haven't got a command builder - a Valid Update Command. Delete the comment from your Command Builder line and the error message goes away.

You should now be able to make changes to the database itself (as long as the Access database isn't Read Only).

Try it out. Run your programme, and change one of the records. Click the **Update** button. Then close the programme down, and load it up again. You should see your new changes displayed in the textboxes.

### Exercise

There's one slight problem with the code above, though. Try clicking the **Update** button before clicking the **Next Record** button. What happens? Do you know why you get the error message? Write code to stop this happening

In the next part, we'll see how to add a new record.

## How to Add a New Record

In the [previous part](#), you learned how to Update records in the database. In the part, we'll see how to add a new record to the database using VB .NET code.



## Add a New Record

Adding a new record is slightly more complex. First, you have to add a new Row to the DataSet, then commit the new Row to the Database.

But the **Add New Record** button on our form is quite simple. The only thing it does is to switch off other buttons, and clear the textboxes, ready for a new entry. Here's the code for your **Add New Record** button:

```
btnCommit.Enabled = True btnAddNew.Enabled = False  
btnUpdate.Enabled = False  
btnDelete.Enabled = False
```

```
txtFirstName.Clear()  
txtSurname.Clear()
```

So three buttons are switched off when the **Add New Record** button is clicked, and one is switched on. The button that gets switched on is the Commit Changes button. The Enabled property of **btnCommit** gets set to **True**. But, for this to work, you need to set it to **False** when the form loads. So return to your Form. Click **btnCommit** to select it. Then locate the **Enabled** Property in the Properties box. Set it to **False**. When the Form starts up, the button will be switched off.

The Clear/Cancel button can be used to switch it back on again. So add this code to your btnClear:

```
btnCommit.Enabled = False  
btnAddNew.Enabled = True  
btnUpdate.Enabled = True  
btnDelete.Enabled = True
```

```
inc = 0  
NavigateRecords()
```

We're switching the **Commit Changes** button off, and the other three back on. The other two lines just make sure that we display the first record again, after the Cancel button is clicked. Otherwise the textboxes will all be blank.

To add a new record to the database, we'll use the **Commit Changes** button. So double click your **btnCommit** to access its code. Add the following:

```
If inc <> -1 Then
```

```
Dim cb As New OleDb.OleDbCommandBuilder(da)  
Dim dsNewRow As DataRow
```

```

dsNewRow = ds.Tables("AddressBook").NewRow()

dsNewRow.Item("FirstName") = txtFirstName.Text
dsNewRow.Item("Surname") = txtSurname.Text

ds.Tables("AddressBook").Rows.Add(dsNewRow)

da.Update(ds, "AddressBook")

MessageBox.Show("New Record added to the Database")

btnCommit.Enabled = False
btnAddNew.Enabled = True
btnUpdate.Enabled = True>
btnDelete.Enabled = True

End If

```

The code is somewhat longer than usual, but we'll go through it.

The first line is an If Statement. We're just checking that there is a valid record to add. If there's not, the **inc** variable will be on minus 1. Inside of the If Statement, we first set up a **Command Builder**, [as before](#). The next line is this:

### **Dim dsNewRow As DataRow**

If you want to add a new row to your DataSet, you need a **DataRow** object. This line just sets up a variable called **dsNewRow**. The type of variable is a DataRow.

To create the new DataRow object, this line comes next:

```
dsNewRow = ds.Tables("AddressBook").NewRow()
```

We're just saying, "Create a New Row object in the AddressBook DataSet, and store this in the variable called dsNewRow." As you can see, **NewRow()** is a method of **ds.Tables**. Use this method to add rows to your DataSet.

The actual values we want to store in the rows are coming from the textboxes. So we have these two lines:

```

dsNewRow.Item("FirstName") = txtFirstName.Text
dsNewRow.Item("Surname") = txtSurname.Text

```

The **dsNewRow** object we created has a Property called **Item**. This is like the Item property you used earlier. It represents a column in your DataSet. We could have said this instead:

```
dsNewRow.Item(1) = txtFirstName.Text  
dsNewRow.Item(2) = txtSurname.Text
```

The **Item** property is now using the index number of the DataSet columns, rather than the names. The results is the same, though: to store new values in these properties. We're storing the text from the textboxes to our new Row.

We now only need to call the Method that actually adds the Row to the DataSet:

```
ds.Tables("AddressBook").Rows.Add(dsNewRow)
```

To add the Row, you use the **Add** method of the Rows property of the DataSet. In between the round brackets, you need the name of your DataRow (the variable **dsNewRow**, in our case).

You should know what the rest of the code does. Here's the next line:

```
da.Update(ds, "AddressBook")
```

Again, we're just using the **Update** method of the Data Adapter, just like last time. The rest of the code just displays a message box, and resets the button.

But to add a new Row to a DataSet, here's a recap on what to do:

1. Create a **DataRow** variable
2. Create an Object from this variable by using the **DataRow()** method of the DataSet **Tables** property
3. Assign values to the **Items** in the new Row
4. Use the **Add** method of the DataSet to add the new row

A little more complicated, but it does work! Try your programme out. Click your **Add New Record** button. The textboxes should go blank, and three of the buttons will be switched off. Enter a new First Name and Surname, and then click the **Commit Changes** button. You should see the message box telling you that a new record has been added to the database. To see the new record, close down your programme, and run it again. The new record will be there.

In the next part, you'll learn how to delete a record from the database.

## Delete a Record from a Database

In the [last part](#), you saw how to Add a new record to the database using VB .NET code. In this final part, you'll learn how to delete records.

## Deleting Records from a Database

The code to delete a record is a little easier than last time. Double click your **btnDelete** and add the following:

```
Dim cb As New OleDb.OleDbCommandBuilder(da)

ds.Tables("AddressBook").Rows(inc).Delete()
MaxRows = MaxRows - 1

inc = 0
da.Update(ds, "AddressBook")
NavigateRecords()
```

You've met most of it before. First we set up a Command Builder. Then we have this line:

**ds.Tables("AddressBook").Rows(inc).Delete()**

Just as there is an **Add** method of the DataSet Rows property, so there is a **Delete** method. You don't need anything between the round brackets, this time. We've specified the Row to delete with:

**Rows( inc )**

The **inc** variable is setting which particular Row we're on. When the **Delete** method is called, it is this row that will be deleted.

However, it will only be deleted from the DataSet. To delete the row from the underlying database, we have this again:

**da.Update(ds, "AddressBook")**

The Command Builder, in conjunction with the Data Adapter, will take care of the deleting. All you need to is call the **Update** method of the Data Adapter.

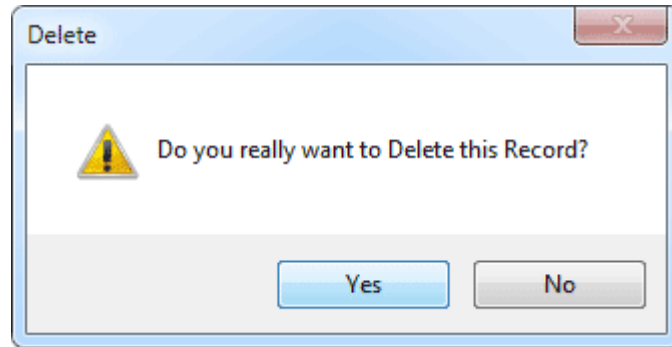
The **MaxRows** line in the code just deducts 1 from the variable. This just ensures that the number of rows in the DataSet matches the number we have in the MaxRows variable.

We also reset the **inc** variable to zero, and call the **NavigateRecords()** subroutine. This will mean that the first record is displayed, after a record has been deleted.

Try out your programme. Click the **Next Record** button a few times to move to a valid record. Then click the **Delete Record** button. The record will be deleted from the DataSet AND the database. The record that is then displayed will be the first one.

There's another problem, though: if you click the **Delete Record** button before the **Next Record** button, you'll get an error message. You can add an If Statement to check that the inc variable does not equal minus 1.

Another thing you can do is to display a message box asking users if they really want to delete this record. Here's one in action:



To get this in your own programme, add the following code to the very top of your Delete button code:

```
If MessageBox.Show("Do you really want to Delete this Record?", "Delete",  
MessageBoxButtons.YesNo, MessageBoxIcon.Warning) = DialogResult.No Then
```

```
    MessageBox.Show("Operation Cancelled")  
    Exit Sub
```

```
End If
```

The first two lines of the code are really one line, spread out so as to fit on this page.

In between the round brackets, we specifying the message to display, followed by a caption for the message box. We then have this:

**MessageBoxButtons.YesNo**

You won't have to type all that out; you'll be able to select it from a popup list. But what it does is give you Yes and No buttons on your message box.

After typing a comma, we selected the **MessageBoxIcon.Warning** icon from the popup list.

But you need to check which button the user clicked. This is done with this:

**= DialogResult.No**

Again, you select from a popup list. We want to check if the user clicked the No button. This will mean a change of mind from the user. A value of No will then be returned, which is what we're checking for in the If Statement.

The code for the If Statement itself is this:

```
MessageBox.Show("Operation Cancelled")  
Exit Sub
```

This will display another message for the user. But most importantly, the subroutine will be exited: we don't want the rest of the Delete code to be executed, if the user clicked the No button.

And that's it for our introduction to database programming. You not only saw how to construct a database programme using the Wizard, but how to write code to do this yourself. There is an awful lot more to database programming, and we've just scratched the surface.