Lab 10: Add, Update and Delete Records

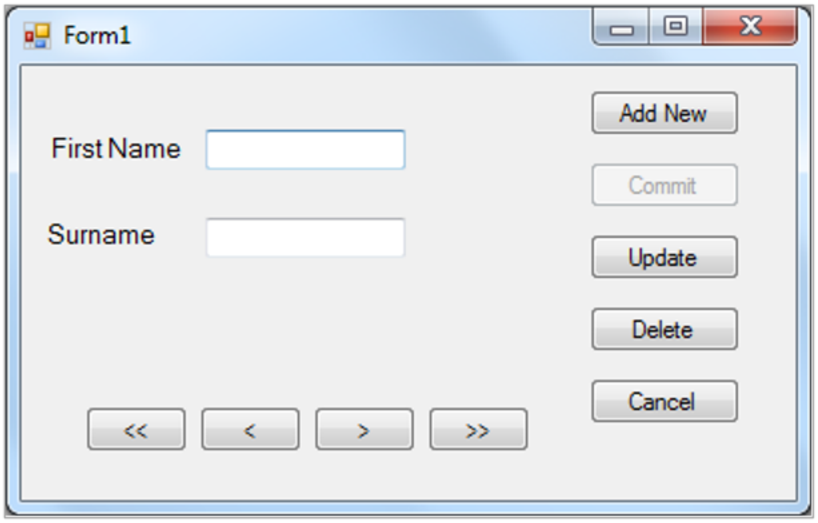
In the last section, you learned how to move through the records in your DataSet, and how to display the records in Textboxes on your form. In this lesson, we'll see how to add new records, how to delete them and how to Update a records.

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 are 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

MsgBox("Data updated")
```

Run your program, 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". 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 program, and 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")

MsgBox("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. Try it out. Run your program, and change one of the records. Click the Update button. Then close the program 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

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")

    MsgBox("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 result 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:

- Create a DataRow variable
- Create an Object from this variable by using the NewRow() method of the DataSet Tables property
- Assign values to the Items in the new Row
- Use the Add method of the DataSet to add the new row

Try your program 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 program, and run it again. The new record will be there.

Deleting Records from a Database

The code to delete a record is a little easier than add new record. 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")
```

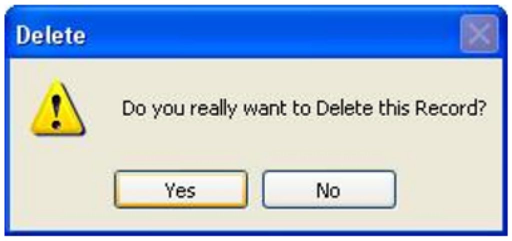
The Command Builder, in conjunction with the Data Adapter, will take care of the deleting. All you need to is call the Update method of the Data Adapter.

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 program. 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:



Do you really want to Delete this Record?

To get this in your own program, 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

        MsgBox("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. We then have this in MessageBox.Show method:

MessageBoxButtons.YesNo

What it does is give you Yes and No buttons on your message box. Therefore 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:

```
MsgBox("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.