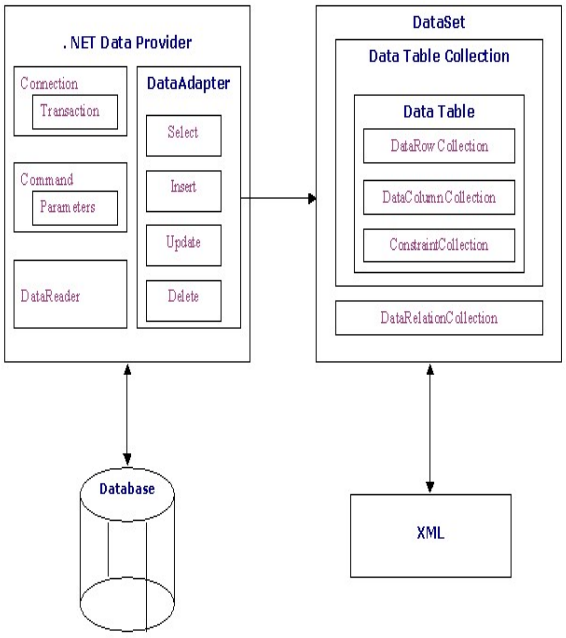
ADO .NET

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access.DOT NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet.

The ADO.NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.



ADO .NET Data Architecture

DataSet

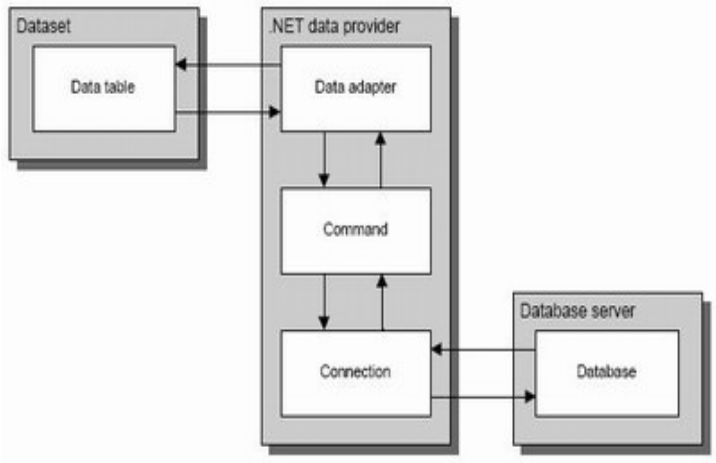
The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

Data Provider:

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

The **Connection object** which provides a connection to the database
The **Command object** which is used to execute a command
The **DataReader object** which provides a forward-only, read only, connected recordset
The **DataAdapter object** which populates a disconnected DataSet with data and performs update

How ADO.NET works



To work with data using ADO.NET, you use a variety of ADO.NET objects. Figure 2-1 shows the primary objects you'll use to develop Windows-based ADO.NET applications.

To start, the data used by an application is stored in a *dataset* that contains one or more *data tables*. To load data into a data table, you use a *data adapter*. The main function of the data adapter is to manage the flow of data between a dataset and a database. To do that, it uses *commands* that define the SQL statements to be issued. The command for retrieving data, for example, typically defines a Select statement. Then, the command connects to the database using a *connection* and passes the Select statement to the database. After the Select statement is executed, the result set it produces is sent back to the data adapter, which stores the results in the data table.

To update the data in a database, the data adapter uses a command that defines an Insert, Update, or Delete statement for a data table. Then, the command connects to the database and performs the requested operation.

Although it's not apparent in this figure, the data in a dataset is independent of the database that the data was retrieved from. In fact, the connection to the database is typically closed after the data is retrieved from the database. Then, the connection is opened again when it's needed. Because of that, the application must work with the copy of the data that's stored in the dataset. The architecture that's used to implement this type of data

processing is referred to as disconnected *data architecture*. Although this is more complicated than a connected architecture, the advantages offset the complexity.

One of the advantages of using disconnected data architecture is improved system performance due to the use of fewer system resources for maintaining connections. Another advantage is that it makes ADO.NET compatible with ASP.NET web applications, which are inherently disconnected. You'll learn more about developing ASP.NET web applications that use ADO.NET in chapters 12 through 14 of this book.

The ADO.NET classes that are responsible for working directly with a database are provided by the *.NET data providers*. These data providers include the classes you use to create data adapters, commands, and connections. As you'll learn later in this chapter, the .NET Framework currently includes two different data providers, but additional providers are available from Microsoft and other third-party vendors such as IBM and Oracle.

Difference between ADO and ADO.net

- ADO works with connected data. This means that when you access data such as viewing and updating data it is real-time with a connection being used all the time. This is barring of course you programming special routines to pull all your data into temporary tables.
- ADO.NET uses data in a disconnected fashion. When you access data ADO.NET makes a copy of the data using XML. ADO.NET only holds the connection open long enough to either pull down the data or to make any requested updates. This makes ADO.NET efficient to use for Web applications. It's also decent for desktop applications.
- ADO has one main object that is used to reference data called the *Recordset object*. This object basically gives you a single table view of your data although you can join tables to create a new set of records. With ADO.NET you have various objects that allow you to access data in various ways. The DataSet object will actually allow you to store the relational model of your database. This allows you to pull up customers and their orders accessing/updating the data in each related table individually.
- ADO allows you to create client-side cursors only whereas ADO.NET gives you the choice of either using client-side or server-side cursors. In ADO.NET classes actually handle the work of cursors. This allows the developer to decide which is best. For Internet development this is crucial in creating efficient applications.

How to Connect to an Access Database with C# .NET

Connecting to an Access database requires a different connection object and connection string.

Access uses something called OLEDB, so you need to create a database object of this type.

Double click your form. Outside of the form load event, type the following:

System.Data.OleDb.OleDbConnection con**;**

Inside of the form load event, type this:

con = new System.Data.OleDb.OleDbConnection();

This sets up a new connection object called con. It is an OLEDB connection object.

The connection string can be found using the same technique as for SQL Server Express, just outlined. Once the string is pasted over, it will look like this:

con.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;
Data Source=C:/Workers.mdb";

(Notice that you can use a single forward slash in a file name, but not a single backslash.)

The Provider you use for Access databases is called Microsoft Jet. We're using version 4.0 in the code above. Again, we need to tell C# where the database is. This is done with **Data Source**. The source above is pointing to a database called Workers.mdb that is on the C drive.

The connection to the database is done with the Open method of your connection object:

con.Open();

Close the connection in a similar way:

con.Close();

You can also issue a Dispose command at the end, if you want. This will do the "tidying up" for you:

con.Dispose();

Add a few message boxes and your coding window should look like this:

```
System.Data.OleDb.OleDbConnection con;

private void Form1_Load(object sender, EventArgs e)
{
    con = new System.Data.OleDb.OleDbConnection();

    con.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;
                            Data Source=C:/Workers.mdb";

    con.Open();

    MessageBox.Show("Database Open");

    con.Close();

    MessageBox.Show("Database Closed");

    con.Dispose();
}
```

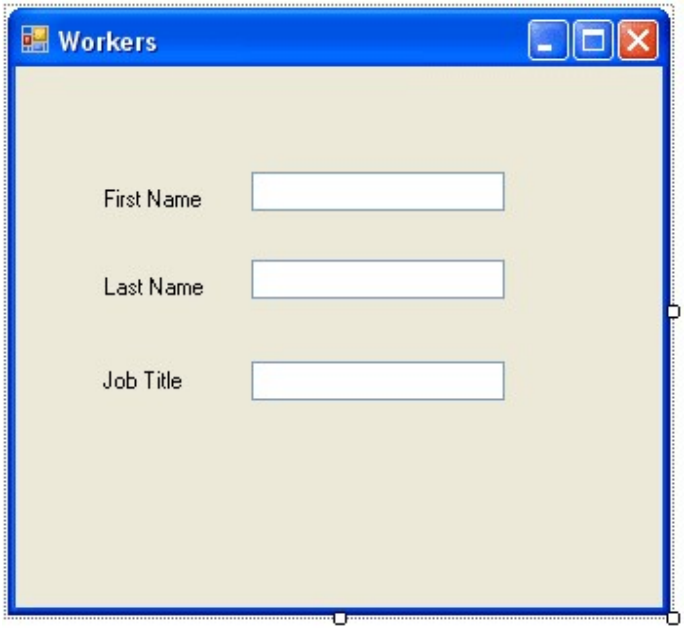
Run your programme and you should see the two message boxes display. The form will display after you click OK on these.

Now that you have made a connection to a Database, you need to learn about Datasets and DataAdapters.

Display Data from a Dataset in C# .NET

At the moment, we have a Dataset filled with records from the database table. But we can't actually see anything. What we'd like to do is to display the records on a form. We'll put the data in textboxes.

Add three text boxes and three labels to your form, so that the design looks something like ours below:



When the form loads, we want the first record from the Dataset to appear in the text boxes.

We'll do all that from a method. So just after your form load code, add a new method called **NavigateRecords**. It's not going to return a value, so you can make it a **void** method (the code below is for an Access database):

```
private void Form1_Load(object sender, EventArgs e)
{
   con = new System.Data.OleDb.OleDbConnection();
   ds1 = new DataSet();

   con.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;
                           Data Source=C:/Workers.mdb";

   string sql = "SELECT * From tblWorkers";
   da = new System.Data.OleDb.OleDbDataAdapter(sql, con);

   con.Open();

   da.Fill(ds1, "Workers");

   con.Close();
   con.Dispose();
}

private void NavigateRecords()
{

}
```

Trying to get at the data from a Dataset can be a torturous business, because there's so many properties and methods to access. The easiest way is to first set up a new **DataRow** variable:

DataRow dRow**;**

This will refer to a row from the Dataset:

DataRow dRow = **ds1.Tables["Workers"].Rows[0];**

So after the equals sign, we have this:

ds1.Tables["Workers"].Rows[0];

You first type the name of your Dataset, which is **ds1** for us. After a dot, select **Tables** from the IntelliSense list. **Tables** is a collection, and stores a list of all the available Tables (a Table is just that grid that we mentioned). To tell C# which Table you want, type its name between square brackets and a pair of double quotes. After another dot, select Rows from the IntelliSense list. In between square brackets, you specify which Row from the Dataset you want. Row zero [0] is the first Row in the Table.

So add that line to your code, and your NavigateRecords method will look like this:

```
private void NavigateRecords()
{
    DataRow dRow = ds1.Tables["Workers"].Rows[0];

}
```

But that's not the end of it! We've only pointed to a Row in the Dataset. We also need to specify a column.

To get at a Column in the Row, the code is this:

dRow.ItemArray.GetValue(1).ToString()

We've started with our Row object, which we've called **dRow**. After a dot, select **ItemArray** from the IntelliSense list. This is an Array of all the Items (Columns) in your Row. We had four columns in our database table: Worker_ID, first_name, last_name and job_title. ItemArray starts at zero, so Worker_ID will be Item 0, first_name will be Item 1, last_name will be Item 2, and job_title will be Item 3.

After another dot, then, select GetValue from the list. As its name suggests, this will Get the Values from your Columns. In between round brackets, you need the Item number from the array. **GetValue(1)** will refer to the first_name column in our Dataset. Finally, you need to convert it to a string with ToString(). Once converted to a string, you can put it straight into a text box.

Putting all that together, add the following code to your NavigateRecords method:

```
private void NavigateRecords()
{
    DataRow dRow = ds1.Tables["Workers"].Rows[0];

    textBox1.Text = dRow.ItemArray.GetValue(1).ToString();
    textBox2.Text = dRow.ItemArray.GetValue(2).ToString();
    textBox3.Text = dRow.ItemArray.GetValue(3).ToString();

}
```

If you prefer, you can put everything on one line. But it will be a very long line. Here it is:

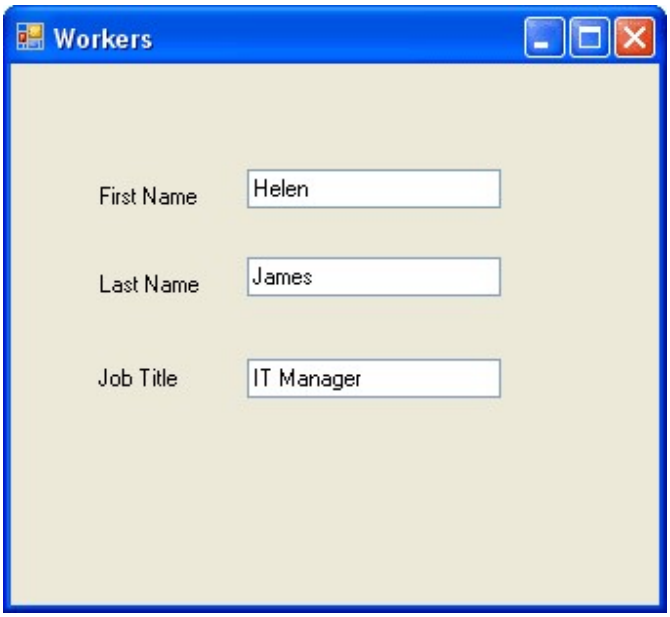
textBox1.Text = ds1.Tables["Workers"].Rows[0].ItemArray.GetValue(1).ToString();

The line is so long we had to make the font size smaller just to fit it on this page!

But you are almost ready to test it all out. The final thing to do is to add a call to your NavigateRecords method. Put the call just before the con.Close line, as in the image below:

```
        con.Open();

        da.Fill(dsl, "Workers");
        NavigateRecords();

        con.Close();
        con.Dispose();
    }

    private void NavigateRecords()
    {
        DataRow dRow = dsl.Tables["Workers"].Rows[0];

        textBox1.Text = dRow.ItemArray.GetValue(1).ToString();
        textBox2.Text = dRow.ItemArray.GetValue(2).ToString();
        textBox3.Text = dRow.ItemArray.GetValue(3).ToString();

    }
```

Now you can test it out. Run your programme and the form should display the first record from your database. It should look like ours:



Now that we have one record displayed, we can add buttons to navigate backwards and forward through all the records in our database.

Add a New Record to the Database

When you add a new record, you'll want to add it to the Dataset and the underlying database. Let's see how. First, if you've added a **con.Dispose** line then comment it out, as it might cause errors further on.

Add two new buttons to the form. Set the following properties for your buttons:

Name: btnAddNew
Text: Add New

Name: btnSave
Text: Save

The **Add New** button won't actually add a new record. The only thing it will do is to clear the text boxes, ready for a new record to be added. The **Save** button is where we'll add the record to the Dataset and to the Database.

Double click your Add New button, and add code to clear the text boxes:

```
private void btnAddNew_Click(object sender, EventArgs e)
{
    textBox1.Clear();
    textBox2.Clear();
    textBox3.Clear();
}
```

That's all we need to do here. You can test it out, if you want. But all the code does is to clear the three text boxes of text. The user can then enter a new record.

After a new record has been entered into the text boxes, we can Save it. So double click your Save button to get at the code.

To save a record, you need to do two things: save it to the Dataset, and save it to the underlying database. You need to do it this way because the Dataset with its copy of the records is disconnected from the database. Saving to the Dataset is NOT the same as saving to the database.

To add a record to the Dataset, you need to create a new Row:

```
DataRow dRow = ds1.Tables["Workers"].NewRow();
```

This creates a New DataRow called **dRow**. But the Row will not have any data. To add data to the row, the format is this:

```
dRow[1] = textBox1.Text;
```

So after your DataRow variable (**dRow** for us) you need a pair of square brackets. In between the square brackets type its position in the Row. This is the Column number. **dRow[1]** refers to the first_name column, for us. After an equals sign, you type whatever it is you want to add to that Column - the text from textBox1, in our case.

Finally, you issue the Add command:

```
ds1.Tables["Workers"].Rows.Add( dRow );
```

After **Add**, and in between a pair of round brackets, you type the name of the Row you want to add, which was dRow in our example. The new Row will then get added to the end of the Dataset.

So add this code to your Save button:

```
private void btnSave_Click(object sender, EventArgs e)
{
    DataRow dRow = ds1.Tables["Workers"].NewRow();

    dRow[1] = textBox1.Text;
    dRow[2] = textBox2.Text;
    dRow[3] = textBox3.Text;

    ds1.Tables["Workers"].Rows.Add(dRow);

    MaxRows = MaxRows + 1;
    inc = MaxRows - 1;
}
```

Notice the last two lines:

MaxRows = MaxRows + 1;
inc = MaxRows - 1;

Because we have added a new Row to the Dataset, we also need to add 1 to the MaxRows variable. The **inc** variable can be set to the last record in the Dataset.

Try it out. When you start your programme, click the Add New button to clear the text boxes. Enter a new record in the blank text boxes and then click your Save button. Click your Previous and Next. You'll see that the new record appears.

(Obviously, you'll want to add error checking code to check that the Save button is not clicked before the Add button. Or simply set the Enabled property to false for the Save button when the form loads. You can then set Enabled to true in your Add button.)

If you close the programme down, and start it back up again you'll find that the new record has disappeared! It's disappeared because we haven't yet added it to the underlying database. We've only added it to the Dataset.

To add a new record to the Database, you need to use the DataAdapter again. This has an **Update** method that will do the job for you. The only thing you need to do is tell it which Dataset holds all the records, and its name:

da.Update(ds1, "Workers");

The code above refers to a DataAdapter called da. In between the round brackets of the Update method, we first have the Dataset (ds1) and then the name we gave to this Dataset (Workers).

However, because the connection to the Database is closed (we closed it during the form load event), we need one more rather curious object - something called a **CommandBuilder**.

The CommandBuilder will reconnect to the database for you. The only thing you need to do is to pass it a DataAdapter. The code to create a CommandBuilder object is this for Access:

System.Data.OleDb.OleDbCommandBuilder cb;
cb = new System.Data.OleDb.OleDbCommandBuilder(da);

And this for SQL Server Express:

System.Data.SqlClient.SqlCommandBuilder cb;
cb = new System.Data.SqlClient.SqlCommandBuilder(da);

In both cases, we have created a CommandBuilder object called **cb**. In between the round brackets in the code above, we have our DataAdapter variable, which was **da**.

You don't need to do anything with the CommandBuilder. It knows what to do! But here's what your code should look like if you're using an Access database:

```
private void btnSave_Click(object sender, EventArgs e)
{
    System.Data.OleDb.OleDbCommandBuilder cb;
    cb = new System.Data.OleDb.OleDbCommandBuilder(da);

    DataRow dRow = dsl.Tables["Workers"].NewRow();

    dRow[1] = textBox1.Text;
    dRow[2] = textBox2.Text;
    dRow[3] = textBox3.Text;

    dsl.Tables["Workers"].Rows.Add(dRow);

    MaxRows = MaxRows + 1;
    inc = MaxRows - 1;

    da.Update(dsl, "Workers");

    MessageBox.Show("Entry Added");
}
```

And here's what your code should look like if you are using a SQL Server Express database:

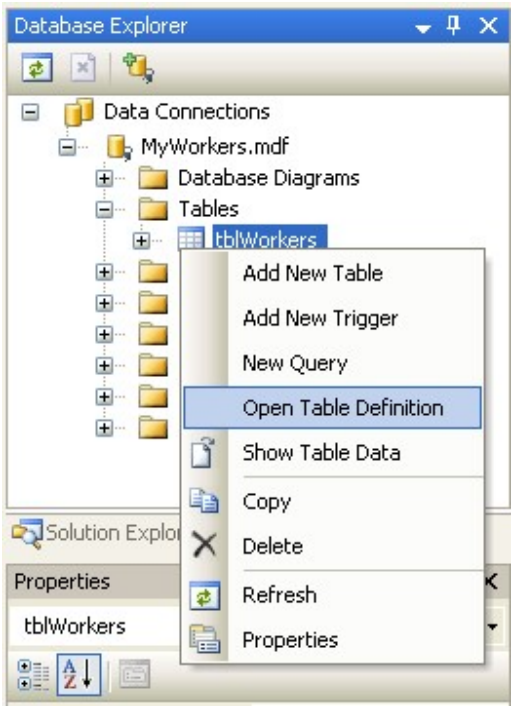
```
private void btnSave_Click(object sender, EventArgs e)
{
    System.Data.SqlClient.SqlCommandBuilder cb;
    cb = new System.Data.SqlClient.SqlCommandBuilder(da);

    DataRow dRow = dsl.Tables["Workers"].NewRow();

    dRow[1] = textBox1.Text;
    dRow[2] = textBox2.Text;
    dRow[3] = textBox3.Text;

    dsl.Tables["Workers"].Rows.Add(dRow);

    MaxRows = MaxRows + 1;
    inc = MaxRows - 1;

    da.Update(dsl, "Workers");

    MessageBox.Show("Entry Added");
}
```

You can now try your programme out. You should find that the new record gets added to the Dataset AND the underlying Database. Close your programme down, reopen it, and check.

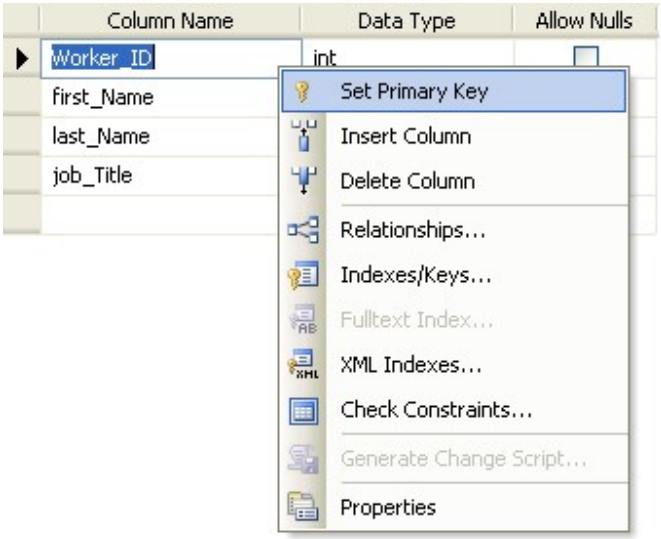
Update and Delete Records

Sometimes, all you want to do is to update a record in the database. This is very similar to Adding a new record.

Before we can update a record, however, we need to set a primary key in the database table, otherwise, C# will throw up an error message. (You can skip this, if you've already added a primary key.) A primary key, remember, is a column of unique data, one that has no duplicates. For us, this is the Worker_ID column. So, open up your table by clicking the Database Explorer tab at the bottom of the Solution Explorer. Expand the Tables entry and right click your table:



Select **Open Table Definition** from the menu. You'll then see the Columns and Data Types in your table. Right click your ID column and select "Set Primary Key".



Save your work in the usual way. We can now add an Update option.

Updating a Record

Examine the following code:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    System.Data.OleDb.OleDbCommandBuilder cb;
    cb = new System.Data.OleDb.OleDbCommandBuilder(da);

    System.Data.DataRow dRow2 = ds1.Tables["Workers"].Rows[inc];

    dRow2[1] = textBox1.Text;
    dRow2[2] = textBox2.Text;
    dRow2[3] = textBox3.Text;

    da.Update(ds1, "Workers");

    MessageBox.Show("Data Updated");
}
```

This is very similar to adding a new record. But the only thing you're not doing is adding a new Row. After creating a new Row called **dRow2**, we set it to the current Row:

= ds1.Tables["Workers"].Rows[inc];

Whatever is in the text boxes then gets transferred to dRow2[1], dRow2[2] and dRow2[3]. These are the Columns in the Row. Then we update the database:

da.Update(ds1, "Workers");

Before trying it out, comment out the line that says **con.Dispose()**, if you added one. Otherwise you'll get a Connection String error.

When you run your form, amend one of your records. Close down the form and open it back up again. You should find that your amendments are still there.

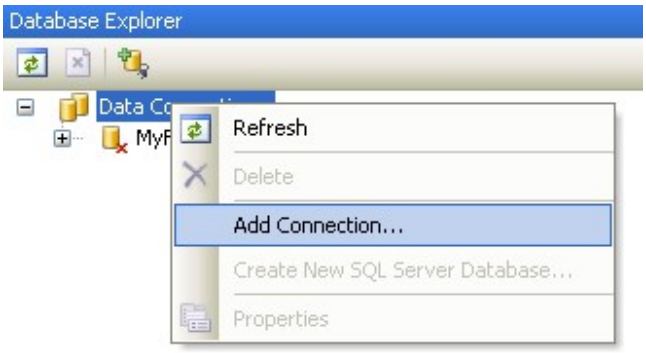
Dynamic SQL Errors

If you are using a SQL Server Express database, and have set a Primary Key, you may still get the following error message:

"Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information."

If so, try the following:

- Delete the database from the Database Explorer by right clicking the database name. From the menu, select Delete (You won't be deleting the database itself, just taking it out of the project.)
- Do the same if there is a database in the Solution Explorer
- Delete any XSD entries as well
- In the Database Explorer window, right click Data Connection and select "Add Connection" from the menu:



From the dialogue box that appears, click the **Browse** button and search for your database:

Click the **Test Connection** button just to make sure it's all right. Then click OK. This will open the original database. Now you can set the primary key as above. Save and test your programme out.

Delete a Record

To delete a record from the Dataset, you use the **Delete** method:

```
ds1.Tables["Workers"].Rows[inc].Delete();
```

This is enough to Delete the entire Row (**Rows[inc]**). But it is only deleted from the Dataset. Here's the code to delete the record from the database, as well:

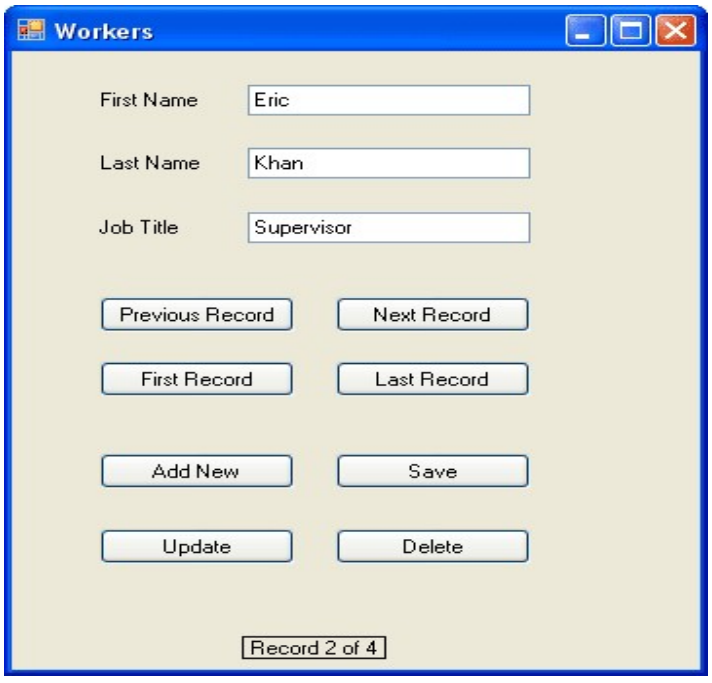
```
private void btnDelete_Click(object sender, EventArgs e)
{
    System.Data.OleDb.OleDbCommandBuilder cb;
    cb = new System.Data.OleDb.OleDbCommandBuilder(da);

    dsl.Tables["Workers"].Rows[inc].Delete();
    MaxRows--;
    inc = 0;
    NavigateRecords();

    da.Update(dsl, "Workers");

    MessageBox.Show("Record Deleted");
}
```

Notice that we're deducting 1 from the MaxRows variable (**MaxRows--**), as well as setting **inc** to 0.

Try it out for yourself. Add a new record to your database. Then try to Delete it. You may get an error about concurrency violations. If you close down the programme and open it back up again, you should find that you'll be able to delete it without any errors.

Concurrency violations can happen for many reasons, but in general it's because the Dataset and Database are out of sync with one another. The easiest solution is to set a Boolean value when a new record is added. If you try to delete, check if this value is true. If it is, then let your user known that they can't delete this new record.

Exercise
Examine this version of our form:



If you look at the bottom, you'll see a label that says "Record 2 of 4". Implement this in your own programme. If you set up a method, you can just call it when the form loads, and again from NavigateRecords.