Unit 9 Data Access with ADO.NET

Structure:

9.1 Introduction
- Objectives

9.2 Record Navigation

9.3 Add, Update and Delete Records
- Updating record
- Adding new record
- Deleting a record

9.4 SQL Server and ADO .NET

9.5 Data Binding
- Data binding with TextBox
- Data binding with Data grid

9.6 Summary

9.7 Questions and Exercises

9.8 Suggested Readings

9.1 Introduction

In the previous unit, we discussed the data base connectivity and VB .NET support for the data access. We explored the role of connection how to create a connection, establish a connection and close the same after use. We learn to phrase question using SQL to fetch data from the data base. The support from data set and data adapter while accessing the database.

In this unit we will be discussing the ADO .NET in data base access. Manipulation of data means adding, deleting and updating of data in the database also we are going to explore the navigation methods to browse through the data bases. This unit is also discussing the ADO .NET support with SQL server and the data binding technology with the controls.

Objectives:

After studying this unit, you will be able to:

- write the code to navigate the records
- discuss the add, update and delete operations of data.
- explain the class available with SQL Ado.NET
- discuss the technique of data binding with the controls
-

9.2Record Navigation

Recording navigation is nothing but the moving of control across the database to view the record. Generally navigation in any database is possible with the four operations they are, move next, move previous, mover first and move last.

(i) Move Back One Record at a Time

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

But we also need to check that inc does not 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
MsgBox('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 have finished adding the code, you can test your program for output. 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
MsgBox("No Records Yet") ElseIf inc = 0 Then
MsgBox('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.

(ii) 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 program. Click the Last button, and you should see the last record displayed in your textboxes.

(iii) 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 program 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.

9.3 Add, Update and Delete Records

In the previous section, you learned how to move through the records in your DataSet, and how to display the records in Textboxes on your form. We also 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 will 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 as depicted in figure 9.1.

Fig. 9.1: Form design with updating controls

9.3.1 Updating record

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

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

This will retrieve the data available from Item 1 on Row 2. You can also set a value with the following syntax

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

Now Item 1 Row 2 will consist of text "Jane". This will not affect the database. The alterations will happen only in **DataSet**. To demonstrate this, enter the 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")
```

Execute your application, 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 you can 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 the application and 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: it is because

Changes are made to the DataSet, and NOT to the Database

You require some extra code In order to update the database,.

```
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.

Without the Command Builder, though, the Data Adapter can't do it's job. Try this. Comment out the Command Builder line (put a single quote before the *D* of Dim). Run your program again, and then try and update a record. You'll get this error message as shown in figure 9.2.

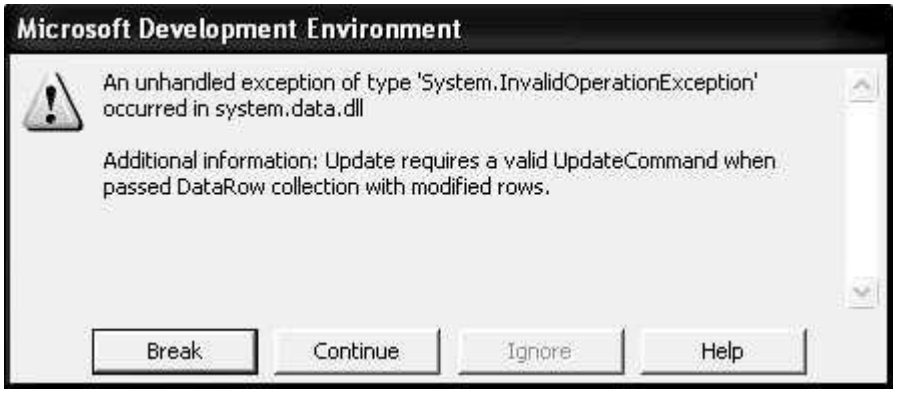

Fig. 9.2: Error Message

The error is because you have not 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 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.

9.3.2 Adding 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 Change**s 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 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:

- 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

A little more complicated, but it does work 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.

9.3.3 Deleting a record

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

Here you should first 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 as shown in figure 9.3.

Fig. 9.3: Delete confirmation window

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

MsgBox("Operation Cancelled")
Exit Sub
End If
```

The first three lines of the code are really one line. The underscore has been used to spread it out, so as to fit on this page.
But we're using the new message box function:

```
MessageBox.Show()
```

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:

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.

And that's it for our introduction to database programming. You not only saw how to construct a database program 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. But in a beginner's course, that's all we have time for.

9.4 SQL Server and ADO .NET

Now we are going to discuss the behavior and features that are related to ADO .NET data provider for SQL server. The basic root namespace for

.NET is the System.Data. This supports the DataTable and DataSet objects to work with any data type. Apart from this .Net supports two more namespaces for retrieving or accessing the data.

System.Data.OleDb: Supports OLE DB data source such as Oracle, Jet etc. This name space includes OleDbCommand, OleDbDataReader, OleDbConnection and OleDbDataAdapter. Using these object we accessed database, that we discussed in the last unit

System.Data.SqlClient: it supports with SQL server 6.5 and above versions. Includes SqlCommand, SqlConnection, SqlDataAdapter and SqlDataReader objects.

SQL Connection class

SqlConnection class cannot be inherited and establish an open connection towards the SQL server database.

```
Public NotInheritable Class SqlConnection _
    Inherits DbConnection _
    Implements ICloneable
```

Has the SqlConnection constructor that initializes the instantiation of the SqlConnection class. This constructor also accepts the connection string to establish a connection.

This class has methods that supports the database transitions like BeginTransaction, changeDatabase, CreateCommand, GetSchema, GetType etc. Following are the list of events supported by the SslConnection Such as Disposed, InfoMessage and StateChange.

SQLDataAdapter

The DataAdapter is an interface between the database and the dataset. It is located between the connected and disconnected parts of Ado.Net as a connector. DataAdapter class will be instantiated by the constructor that are generally overloaded. If we use the default constructor for the DataAdapter, we need to specify the Command object for the actions performed. This means that if we want to retrieve rows from the Data Source, we will have to set the Select command property.

The DataAdapter class is not inheritable Declaration is

*Punlic NotInheritable Class SqlDataAdapter Inherits DbDataAdapter_*

Implements IDbDataAdapter, IDataAdapter,ICloneable

DataSet class

DataSet class consists of information about the set of tables and the relation among those set of tables. Figure 9.4 depicts the DataSet and its related classes. DataSet will not have any idea about the data source or the tables. This will be created dynamically whenever required based on the data source. In SQL server provider, DataSet will be loaded with the help of SqlDataAdapter. Once the data is loaded it is editable and we can manipulate the data without disturbing the data source. Even the data base connectivity is not necessary for this action. When the process is over we can create a new connection and update using the SqlDataAdapter object.

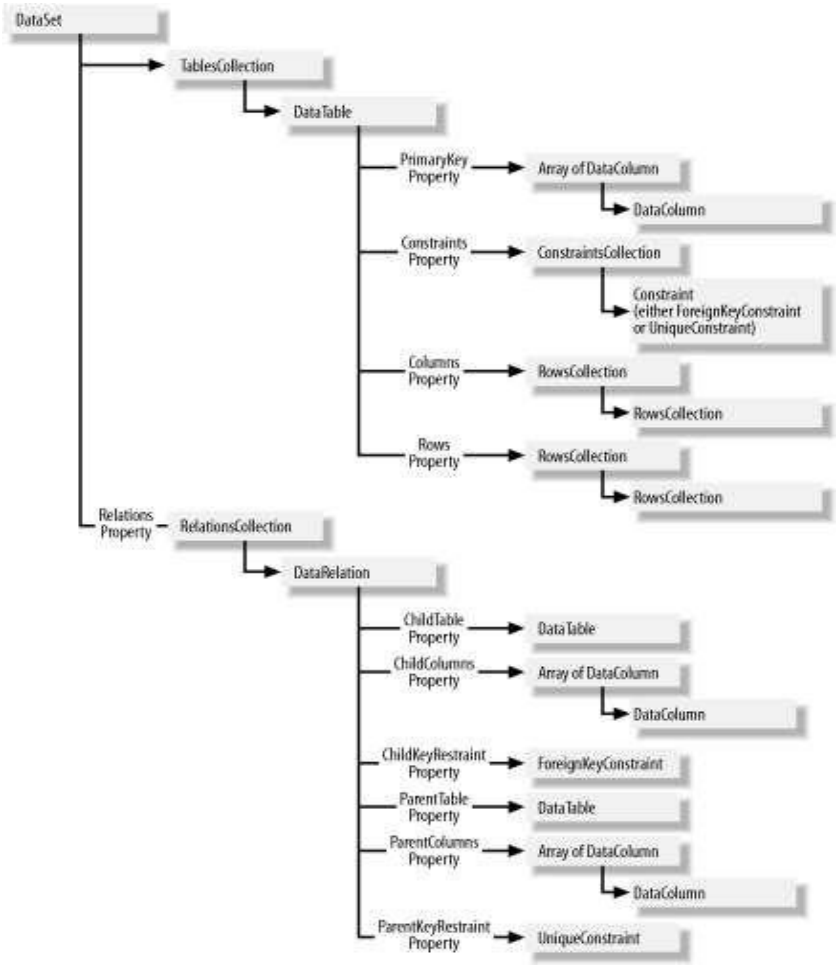

Fig. 9.4: Model diagram for DataSet and related classes.

9.5 Data Binding

The term data binding has a literal meaning when it comes to Windows Form application. It refers to the technique of interfacing elements of a data source with a graphical interface, such as using a TextBox control to bind to a single value from a column. ADO .NET provides a neat data binding infrastructure for graphical controls to seamlessly bind themselves to almost any structure that contains data. This means the data source can be anything from an array value to a set of row. Using data bindings in application essentially reduces the amount of code you have to write for retrieving data from databases. It's true that using data objects in code provides greater control over data, but data binding can achieve the same result if used properly.

9.5.1 Data binding with TextBox

This type of data binding comes under simple binding, here one-to-one association between an individual control property and a single element of a data source happens. You can use it for controls that show only one value at a time. For example, you can bind the Text property of a TextBox control to a DataTable column. If the underlying data source is modified, the control's Refresh method updates the bound value reflecting any changes. We will see small Windows Forms application that uses simple data binding to bind two TextBox controls to two different columns of the Northwind database's Employee table.

In this application we will bind two values to two TextBoxes to display the first and last name of the employee.

1. In visual studio .NET create a new solution.
2. Add a VB. NET Windows application named DataBinding.
3. You should now be in Design view. In the Toolbox, either double-click the TextBox control twice to produce two text boxes that you

can position on the form or drag two TextBox controls on to the form as shown in figure 9.5

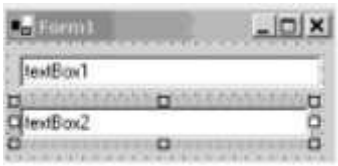

Fig. 9.5: TextBox Controls in form

4. Press F7 to go to Code view. Add the following Imports directives. Imports System.Data.Sqlclient

5. Go back to Design view by clicking the Design tab or by pressing Shift+F7, and double-click the form to go to the underlying code. Your cursor should now be positioned in the Form1_Load method, which was added when you double-clicked the form. Add the following code

   Private Sub Form1_Load(ByVal sender As System.Object, - Byval e As System.EventArgs) Handles MyBase.Load

   *Dim thisConnection As New SqlConnection_*

   ("Server=(local)\netsdk;" &_"integrated security=sspi;"&_ "database=northwind")

   ' Sql Query

   *Dim sql As String = "SELECT * FROM Employees" ' Create Data Adapter*

   Dim da As new SqlDataAdapter(sql, thisconnection)

   ' Create and fill DataSet Da.Fill(ds, "Employees")

   'Bind to first name column of the employee table TextBox1.DataBindings.Add("text", ds, "employees.firstname") 'Bind to lastname column of the employees table TextBox2.DataBindings.Add("text", ds, "employees.lastname")

   End Sub

When you run this program you get output as shown in figure 9.6

Fig. 9.6: DataBinding text boxes bound to data columns

Each data-bound control in a Windows Forms application maintains a list of bindings for all its data-bound properties. The bindings are in a property named DataBindings, which holds a collection of type controlBindingsCollection. This collection can contain a number of individual control property bindings, with each binding added using the Add method.

9.5.2 Data binding with Data grid

This comes under the complex data binding; it is an association between the control and one or more data elements of the data source. You can perform this type of binding with controls that allow more than one value to be displayed at one time, such as a data grid or a data list. Now we are going to discuss this with an example using data grid, which displays all columns of the Northwind database's customers table.

Follow the steps below to develop complex data binding application

1. Add a VB. NET Windows application named ComplexBinding.
2. Add a DataGrid control from the Toolbox. Your form should look similar to the one shown in figure 9.7.

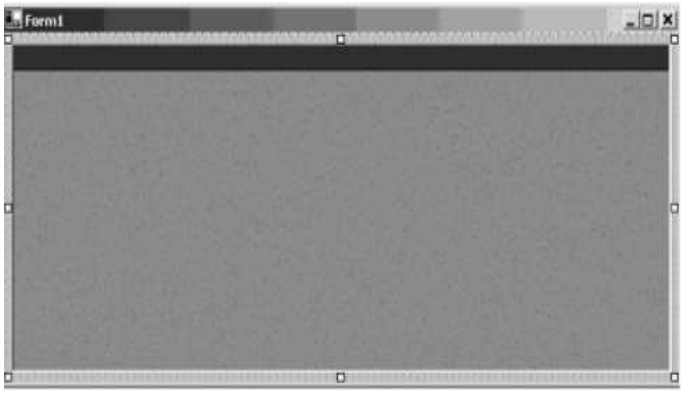

Fig. 9.7: Data grid control

3. Press F7 to reach code window and add the following directive. Imports system.Data.SqlClient

4. Double click the form to go to below code. Your cursor should now be positioned in the Form1_Load method. Add the following code:

```
Private Sub Form1_Load(Byval sender As System.Object, Byval_ e AS
System.EventArgs) Handles MyBase.Load
' Create connection object
("Server=(local)\netsdk;" &_"integrated security=sspi;"&_
"database=northwind")

' Sql Query
Dim sql As String = "SELECT * FROM Employees"

' Create Data Adapter
Dim da As new SqlDataAdapter(sql, thisconnection)

' Create and fill DataSet Da.Fill(ds,
"Employees")

'Bind the data table to the data grid
DataGrid1.SetDataBinding(ds,"Customers")
End Sub
```

5. Make this the startup project, and run the code with Ctrl+F5. You should see the form in figure 9.8

Fig. 9.8: ComplexBinding data grid bound to data table

This data grid also works similar to the Text box control. Once you established a connection with the dataSource, you populate a dataset from the customers table using a data adapter. Next, you need to bind the DataGrid control to the newly populated dataset. We are doing this with the following syntax:

```
DataGrid1.SetDataBinding(ds, "customers")
```

The SetDataBinding method of the DataGrid control accepts two parameters: the data course object and a string literal that describes a data member in the data source. Here, a data source can be any object that's capable of holding data, such us an array or table. The data member describes what element to bind to the control. In the case, you used a dataset as the data source, and the customers data table contained in the data set as a data member. A call to the

SetDataBinding method binds all columns of the customers data table to the DataGrid control at runtime, which renders the data in the style of a spread sheet.

9.6 Summary

- Navigation of records means browsing through the data set. It does four different operations like moving first, moving last, moving next and moving previous.

- Dataset are considered as the temporary buffer to hold the data with or without connection. So the changes whatever happened will not be reflected until you do explicitly.

- System.Data.OleDb and the System.Data.SqlClient are the two .NET namespaces that support the SQL Server data access.

- Simple and complex binding can be done to display the single or group of data in the VB .NET controls.

- Simple data binding can be done through the TextBox controls and the complex data binding can be achieved with grid controls.

9.7 Questions and Exercises:

1. List and explain the navigation techniques with an appropriate example.

2. Explain the various data manipulation operations in the dataset.

3. Discuss the following a. SQLDataAdapter b. DataSet class

4. Explain data binding with TextBox control with example.

5. Brief the complex data binding with the data grid control.

9.8 Suggested Readings

- http://www.vkinfotek.com/dataset.html

- http://msdn.microsoft.com/en-us/library/system.data.sqlclient.qldataadapter. aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1

- http://www.codeproject.com/Articles/20326/ADO-NET-Data-Access-Component-for-SQL-Server-in-Cs

- http://my.safaribooksonline.com/book/web-development/microsoft-aspdotnet/0672323575/introduction-to-adodotnet/ch10lev1sec2