NEAR EAST UNIVERSITY

Faculty of Computer Engineering

Department of Computer Engineering

STUDENT REGISTRATION USING VB.NET PROGRAMMING LANGUAGE

Graduating Project
COM-400

Student: **Serhat EROĞLU**

Supervisor: **Mr. ÜMİT İLHAN**

Nicosia - 2006

ACKNOWLEDGEMENTS

"First, I would like to thank my supervisor **Mr. Ümit İLHAN** for
his invaluable advice and belief in my work and myself over
the course of this Graduating Project..

Second, I would like to express my gratitude to Near East
University for giving lots of opportunity that made the work possible,

Third, I thank my family first of all my father **Rasim EROĞLU**
for their constant encouragement
And support during the preparation of this project.

Finally, I would also like to thank all my friends **Bilal ŞİŞMAN, Ünal KURT, Gökan Kaçmaz** and **Ali Serdar Terlemez**
for their advice and support."

ABSTRACT

The aim of my project is to using database applications by using vb.net programming language.

The project processes student details which are registered to a database. The work includes the following basic operations; add, delete, modify, student records.

I prepared this project in vb.net 2003 which Vb.net has a number of unique features that make it a great choice for building .NET applications. VB.NET is still the only language in VS.NET that includes background compilation, which means that it can flag errors immediately, while you type.

Great deal of gains were collected during the study of the technologies that I have used in this project.

TABLE OF CONTENTS

LIST OF ABBREVIATION

VB.NET: Visual Basic.NET
VS.NET: Visual Studio.NET
OO: Object Oriented
MHz: mega hers
MB: mega bayt
GB: giga byte
GUI: Graphical User Interface
ASP: Active Server Pages
DBMS: Database Management System

INTRODUCTION

This Project is for the dilettante, and aimed at anyone who is interested in learning VB.NET. The approach that has been followed here is that applications are built first, and then, the code is deciphered to unveil the internal workings of the product.

However, in order to thoroughly appreciate the internal execution of the programs, there are certain significant concepts of the language that need to be discerned first. I have ferreted out the relevant concepts and presented them in the most elementary manner.

This Project converges predominantly around the language aspect, to provide enhanced insights to the programmer into the innovative and improved features of VB.Net. We are of the opinion that on learning the language, large applications can be designed with effortless ease.

I have ensured that your interest does not evaporate, by making the project as entertaining and informative as possible. I have done my best.

The Thesis consists of the introduction, four chapter and conclusion.

Chapter1 presents information about how to use vb.NET.

Chapter2 presents information about basic database applications in vb.NET

Chapter3 presents descriptions and codes. I explain every component about program.

Chapter4 presents information about Microsoft Access which I used for database.

CHAPTER1:

1. Learning Visual Basic .NET

1.1 What is .NET?

Microsoft® .NET (released 2001) is Microsoft's new generation "software platform." It's a language-neutral environment for writing programs. Program in any language, run on one operating system. Includes comprehensive class libraries that provide rich features.

.NET represents an advanced new generation of software that will drive the Next Generation Internet. Its purpose is to make information available any time, any place, and on any device. Quick definition of .NET is an initiative to integrate all Microsoft products with the "Next Generation".

1.2 Why VB.NET

The world of applications is changing. They move to Web. It need for reusability, centralization and scalability. MTS, COM+, and Component Services cannot be fully taken advantage of by VB. Features can be implemented more completely with .NET.

To get the benefit of .NET framework and its core execution engine:

- Garbage collection
- OO mechanism
- Standard security services
- Integrated debugging tools

1.3 Installation Requirements for VB.NET

- Operating System
  - Windows 2000 Service Pack 4
  - Windows XP Service Pack 2
  - Windows Server 2003 Service Pack 1
  - Windows x64 editions
  - Windows Vista
- Processor
  - Computer with a 600 MHz or faster processor
  - (1 GHz or higher recommended)
- RAM
  - Minimum: 192 MB

- Recommended: 256 MB (512 MB or more with SQL Express)

- Hard Drive
  - Minimum: 500 MB
  - Includes Visual Basic Express and the .NET Framework 2.0
  - Full Installation: 1.3 GB
  - Also includes MSDN Express Library 2005 and Microsoft SQL Server 2005 Express Edition

1.4. Getting Started with VB.NET

You can use VB.NET to create three different types of programs:

- **Web applications**
- **Windows applications**
- **Console applications**

The .NET platform is web-centric. The VB.NET language was developed to allow .NET programmers to create very large, powerful, high-quality web applications quickly and easily. The .NET technology for creating web applications is called ASP.NET.

ASP.NET, the next generation from ASP (Active Server Pages), is composed of two Microsoft development technologies: Web Forms and Web Services. While the development of fully realized web applications using these technologies is beyond the scope of this book, learning the basics of the VB.NET language will certainly get you started in the right direction. VB.NET is generally acknowledged to be one of the languages of choice for ASP.NET development.

Typically, you'll create an ASP.NET application when you want your program to be available to end users on any platform (e.g., Windows, Mac, Unix). By serving your application over the Web, end users can access your program with any browser. When you want the richness and power of a native application running directly on the Windows platform, alternatively you might create a desktop-bound Windows application. The .NET tools for building Windows applications are called Windows Forms; a detailed analysis of this technology is also beyond the scope of this book. However, if you don't need a Graphical User Interface (GUI) and just want to write a simple application that talks to a console window (i.e., what we used to call a DOS box), you might consider creating a console application. This book makes extensive use of console applications to illustrate the basics of the VB.NET language. Web, Windows, and console applications are described and illustrated in the following pages.

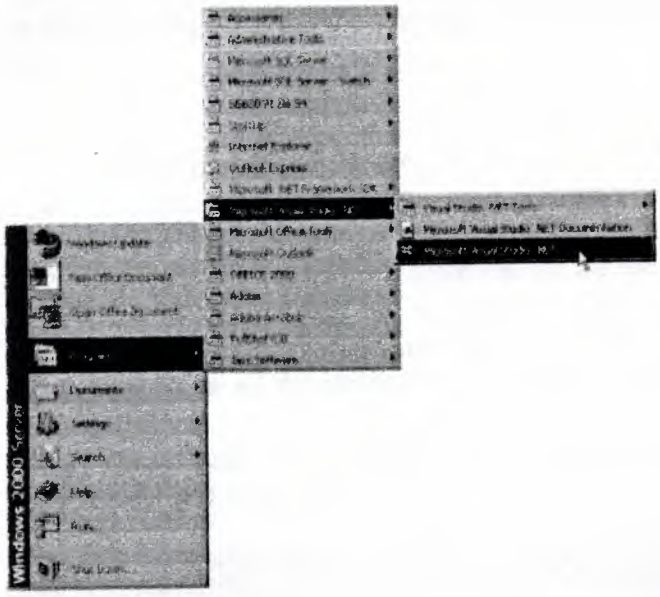
1.1.1. Windows applications

I use windows application in my project so, I am going to explain this application for you;

A Windows application runs on a PC's desktop. You are already familiar with Windows applications such as Microsoft Word or Excel. Windows applications are much more complex than console applications and can take advantage of the full suite of menus, controls, and other widgets you've come to expect in a modern desktop application. Figure 2-2 shows the output of a simple windows application.

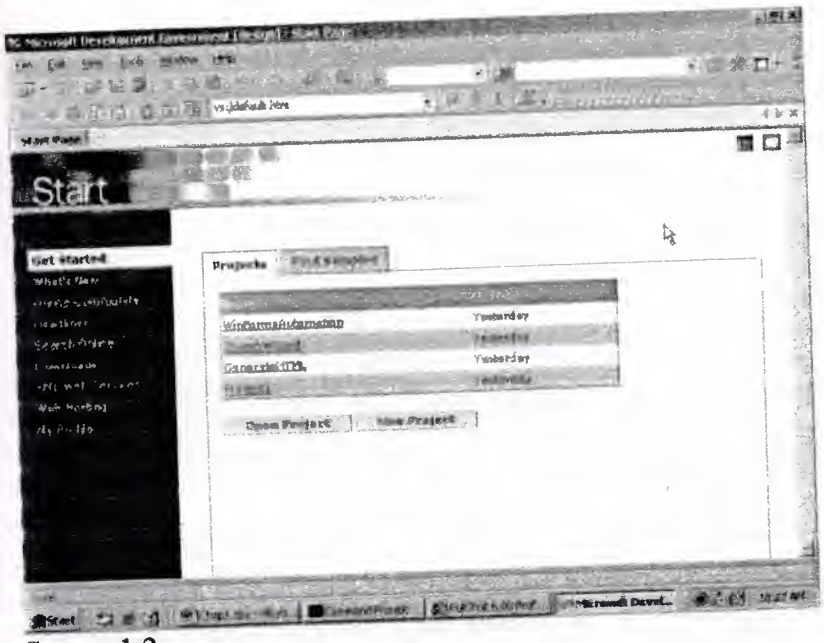
We assume that you have a copy of Visual Studio.Net installed on your machine.

To launch Visual Studio, click on the ubiquitous Start Button in Windows. (In our case, we have installed Windows 2000). Then, click on Programs, followed by Microsoft Visual Studio .Net. Finally, select the option of Microsoft Visual Studio .Net. This is shown in the Screen 1.1.



Screen 1.1

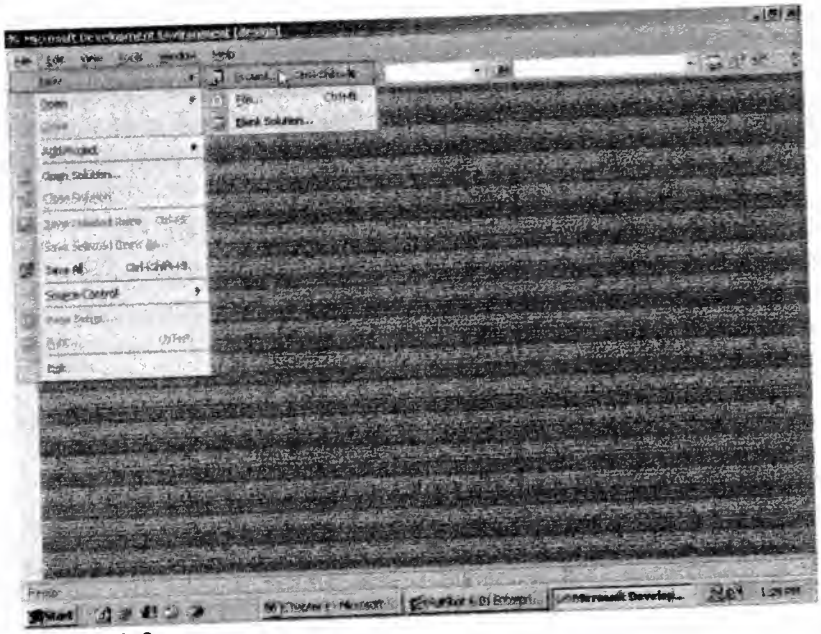
Shortly thereafter, a window pops up, showing Visual Studio.Net in all its glory, as shown in screen 1.2.



Screen 1.2

One of the salient features of present day software is that it can be customized to a large extent. This is why the final appearance of a product could be drastically different from what it originally looked like, when freshly taken out of the box. Thus, the screen 1.2 will never look identical on all the machines. We shall explicate these differences in due course, but for the moment, we are exceedingly inclined towards embarking upon the development of a small application or project. But before venturing any further, close all the child windows by clicking on the x sign.

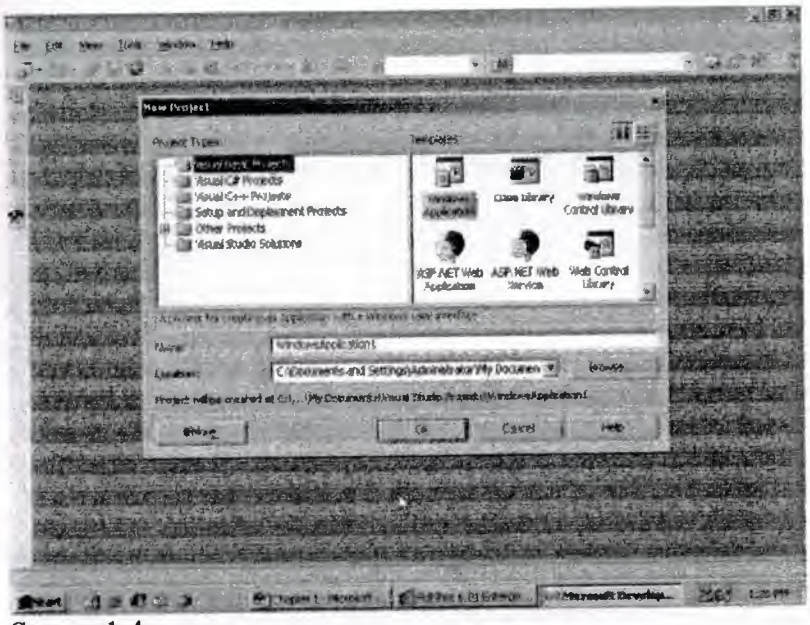
Then, click on the menu option File, followed by New, and finally Project. This becomes evident in screen 1.3.


Screen 1.3

The above actions will result in the display of the Dialog box titled "New Project". This is shown in screen 1.4.

The advantage of using Visual Studio .Net is that diverse applications, each using a different programming language, can be developed simultaneous, under a single streamer.

This explains why three language options are present in the first pane of the dialog box, viz. Visual Basic, Visual C# and Visual C++.



Screen 1.4

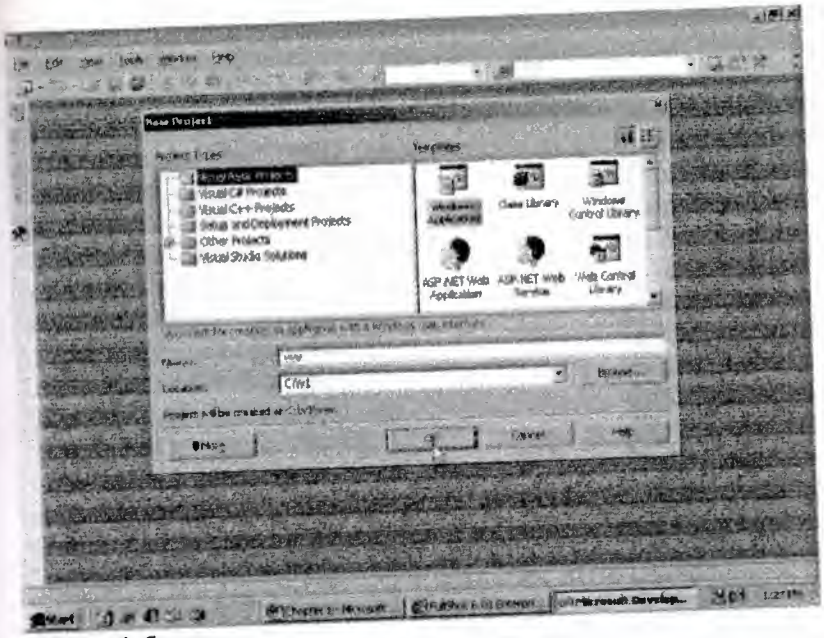
The option of Visual Basic project is listed first. Hence, it is highlighted by default. This signifies the priorities of Microsoft, and is the very raison de etré of this book!

A perfect analogy for Visual Studio .Net is that of a glass, which can be used to drink either water or juice or wine. In a similar fashion, under the single roof of Visual Studio .Net, various programs developed in different languages, can be written. The language used is simply not an issue.

The second pane titled 'Templates' determines the type of application that would eventually be created. This could be an executable file running under Windows, or a Web Application running a web server.

Since we are naïve about this product, we select the option of Windows Application. This option is selected by default.

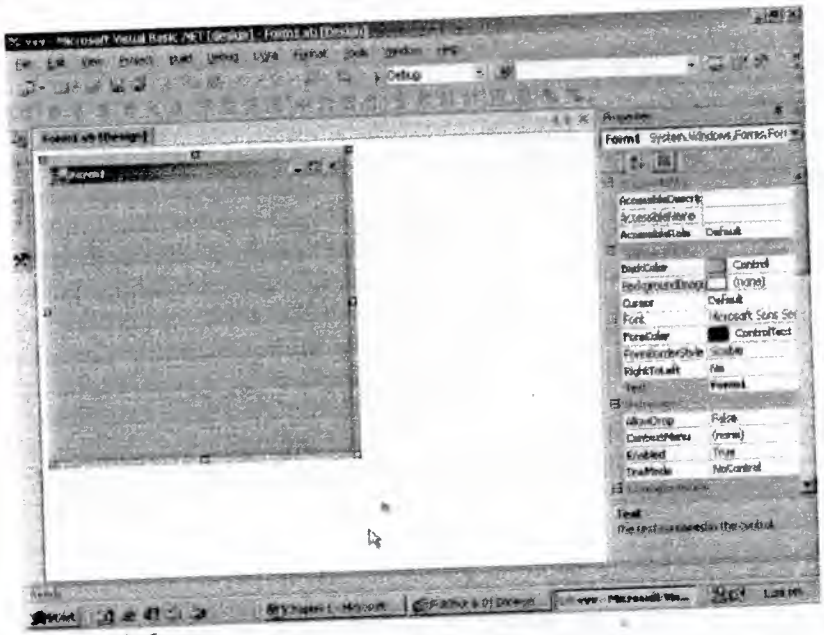
The next task is to specify a name for the project. We have chosen the name 'vvv', since we sincerely believe that it will bring us good tidings on our very first application. We have used our newly created folder called 'v1' as the location for creating the application. You are at liberty to choose any folder that you fancy.



Screen 1.5

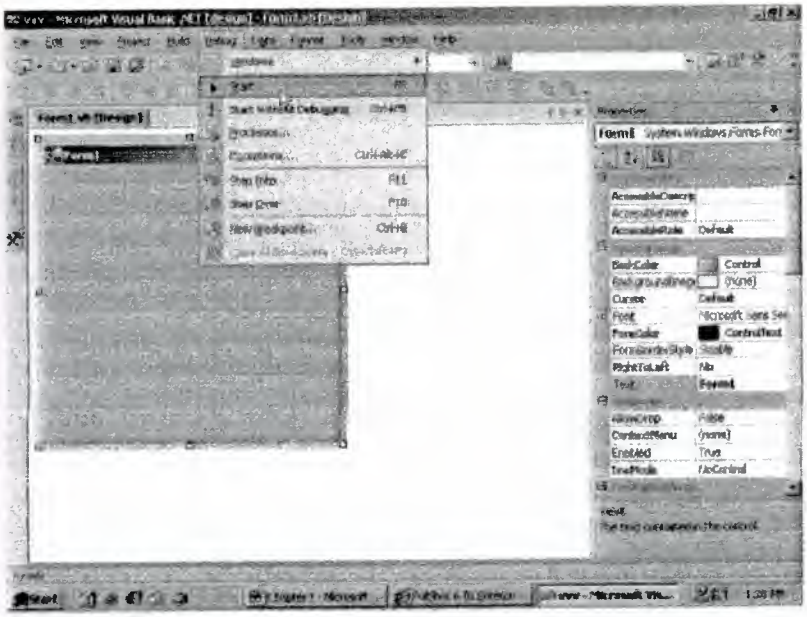
After specifying all the details as shown in screen 1.5, click on the OK button.

This brings us to Screen 1.6, where we encounter an empty form.



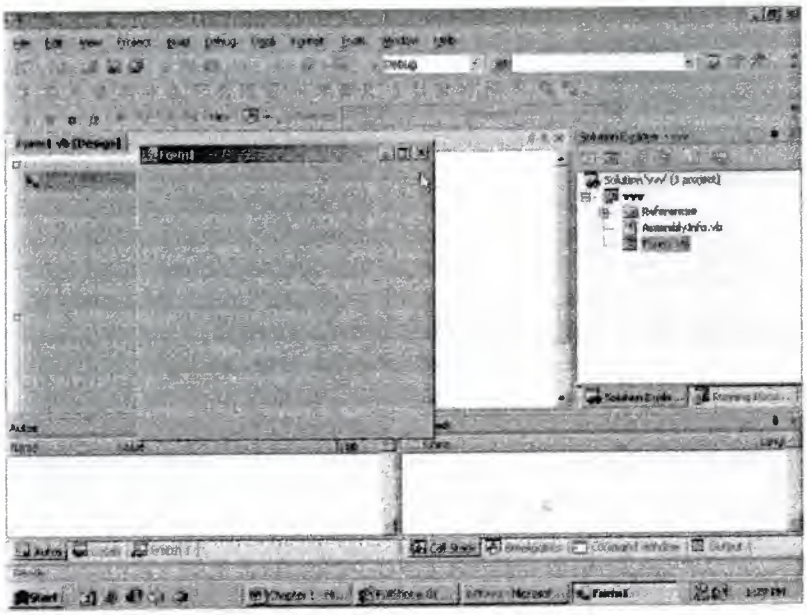
Screen 1.6

Now, in order to view the output of our handiwork, click on the menu option Debug, and then on Start. This is shown in screen 1.7.



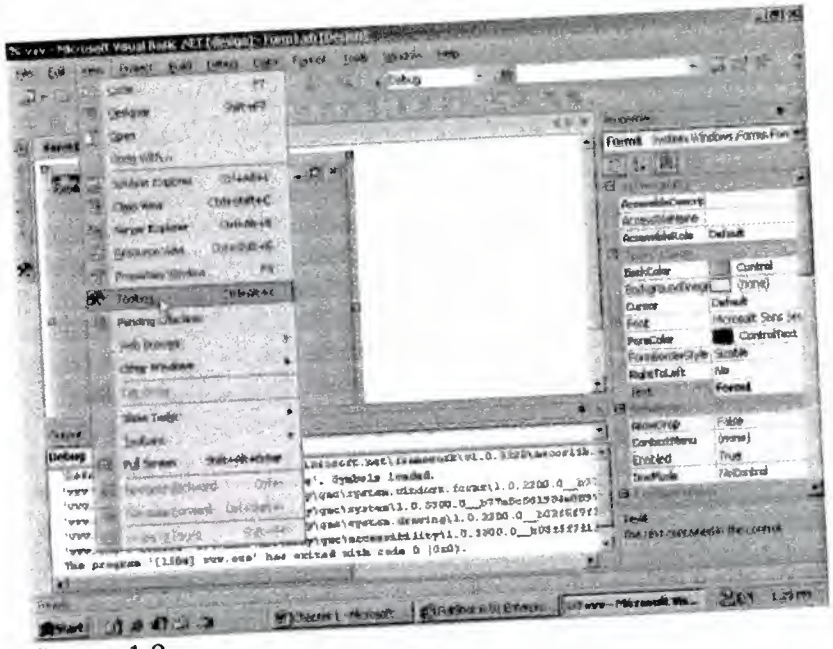
Screen 1.7

This ignites the excitement of Visual Studio .Net, and it starts processing the request. Finally, as the outcome of the operation, it displays an empty form, as seen in screen 1.8. This is no mean achievement for people like us, considering the fact that we have not written even a single line of code yet!



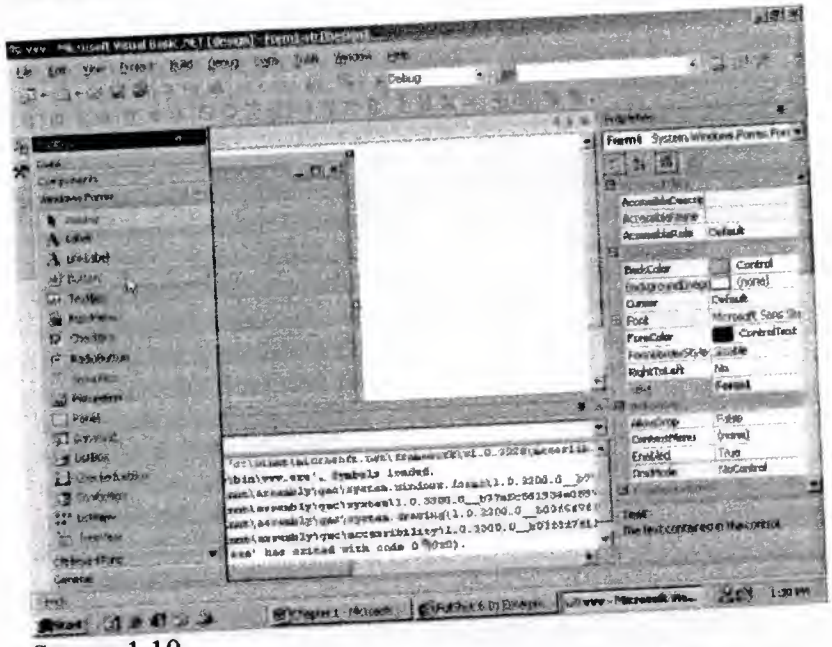
Screen 1.8

Close this form window by clicking on the x sign on the right hand side. This will revert us back to the form in Visual Studio .Net. The next task that we venture upon is, to display a button on the form. To accomplish this, click on the menu option View, and then, on the option of ToolBox, as shown in screen 1.9.



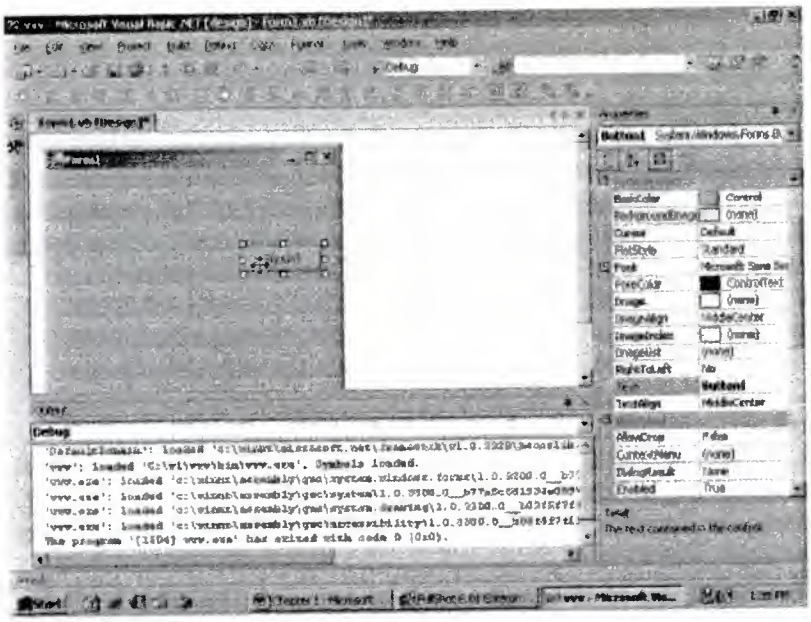
Screen 1.9

This option brings up a toolbox containing a list of objects or controls. The toolbox window is visible on the left hand side of the screen, as revealed in screen 1.10.



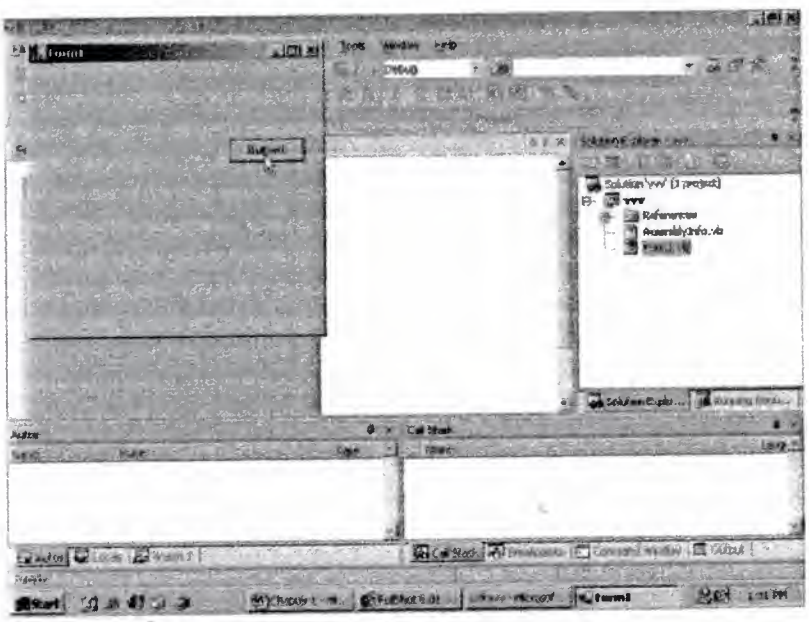
Screen 1.10

Now, simply click on the control labeled as Button to select it, and then, drag and drop it into the form. Screen 1.11 displays the position where we have dropped our button.



Screen 1.11

Then, run the application in the customary manner by clicking on the menu Debug, and then, click on Start. Screen 1.12 confirms that the form now contains the button in it. Pat on the back! Having covered just a couple of pages, we have already placed a button on the form.



Screen 1.12

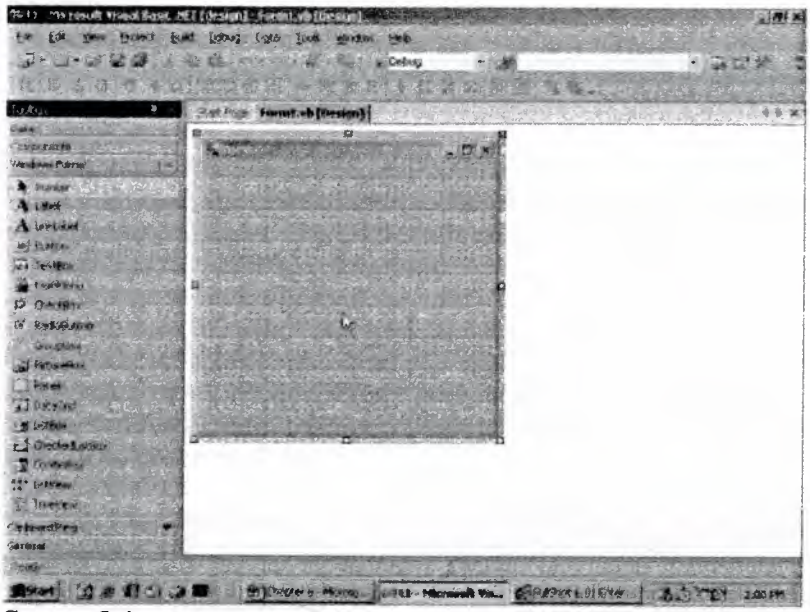
The shortcoming of our present endeavour is that, we are clueless about what is going on behind the scenes. Clicking on the OK button in the New Project dialog box, results in the generation of considerable amount of code by the Visual Studio .Net framework. Apparently, we have merely clicked on the ToolBox, which is a User Interface tool; and then, we have dragged and dropped a button in the form. However, in reality, ample code has been generated in the background to accomplish this seemingly simple task.

We are convinced that prior to forging ahead with the task of building complex applications using Visual Studio .Net, there is a need for discerning the code that it generates. Also, along the way, we shall keep reverting back to the Visual Studio .Net framework, to demonstrate as to how it has made a programmer's life more easygoing. However, you have to learn a programming language that is to be used with it, since it is a fundamental fact that Visual Studio .Net cannot build customized applications by itself. It is the task of the programmer to program it to satisfy the specific requirements.

CAPTER 2:

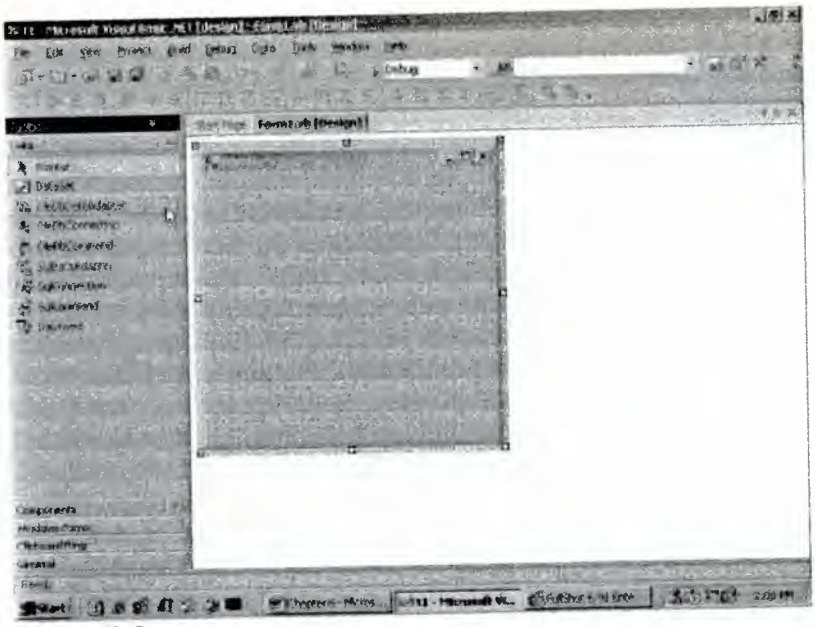
2. Database Applications

You should arrive at screen 2.1, having a mammoth Form and a ToolBox to the left. If for some reason, you are unable to see the toolbox on the left, click on the menu View, and select ToolBox. But, if you have toolbox that hides automatically, unselect the option of AutoHide from the Windows menu option.



Screen 2.1

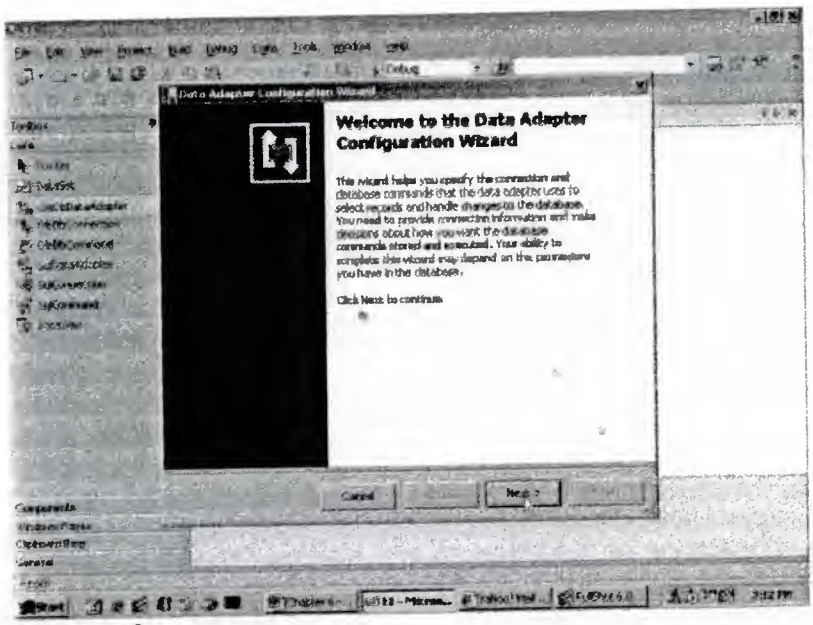
The toolbox has millions of controls. Hence, they are located under separate categories or tabs. The category of Data encompasses all the controls that work with data. Since this chapter deals with databases, select the Data subhead, and examine the controls stipulated below it. The screen that we arrive at, is shown in screen 2.2.



Screen 2.2

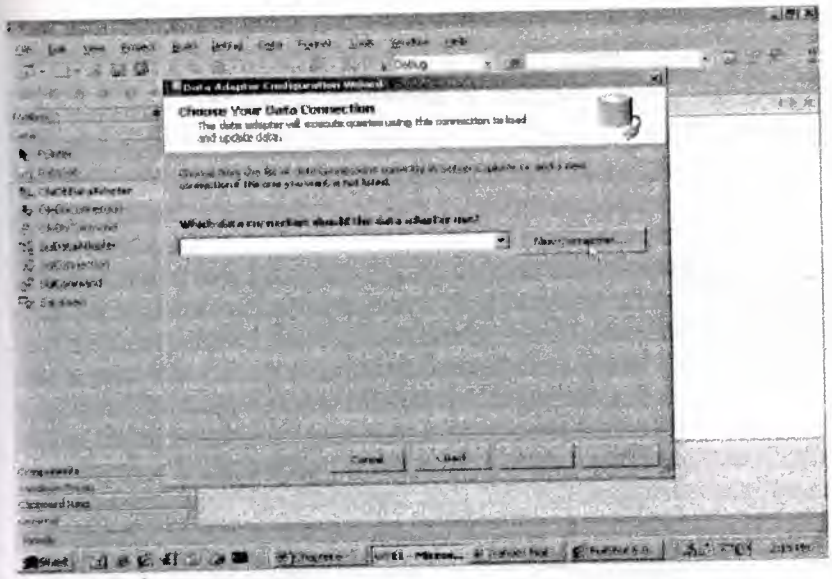
Choose the control called OleDbDataAdapter and place it on the Form. It should be done in a manner akin to the one employed in the previous chapter, to place a button on the Form.

The procedure remains the same, i.e. click on the control to select it, and then, keeping the left mouse button pressed, drag and drop it onto the Form. Surprisingly, unlike a button, the control is not positioned on the Form. Instead, a dialog box pops up, as shown in screen 2.3.



Screen 2.3

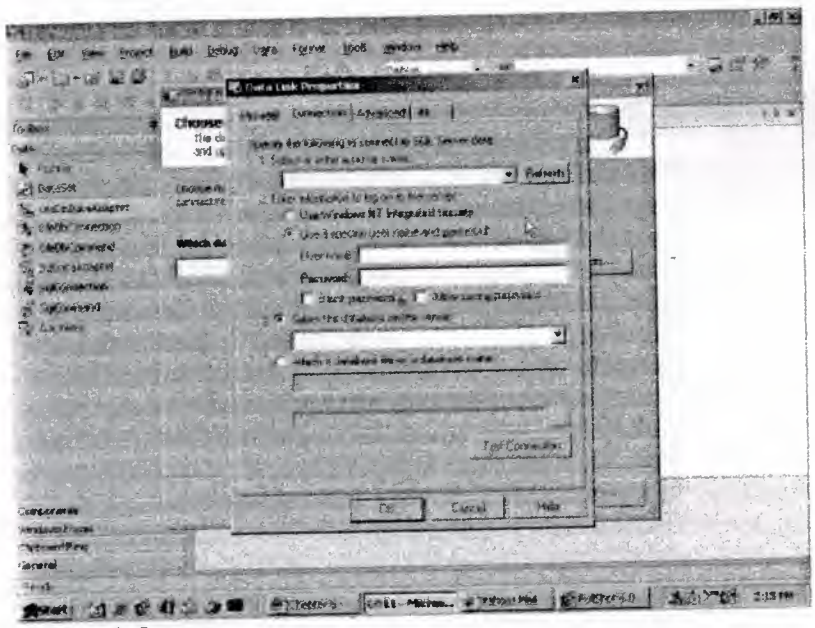
Here, we are informed in no uncertain terms, that we are in the august presence of Merlin's Wizard! The job of a wizard is to make our lives easier by using black magic. Thus, a Microsoft wizard throws a volley of questions at us. Based on the answers furnished by us, it toils towards ameliorating our lives by writing tons of code to achieve the desired outcome.



Screen 2.4

By convention, the first screen of a wizard provides information about the type of the wizard. Presently, we are not very keen on reading it. So, we click on the Next button. This transports us to the next step in the wizard, as shown in screen 2.4.

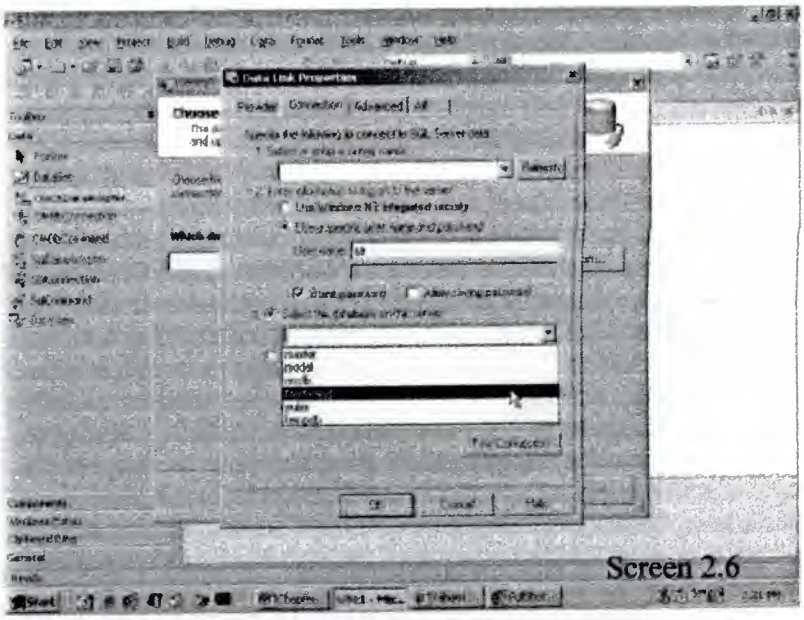
Here, we are asked to confirm which database we prefer to connect to. Since we have not created any connection in the past, we click on the 'New connection' button to arrive screen 2.5.



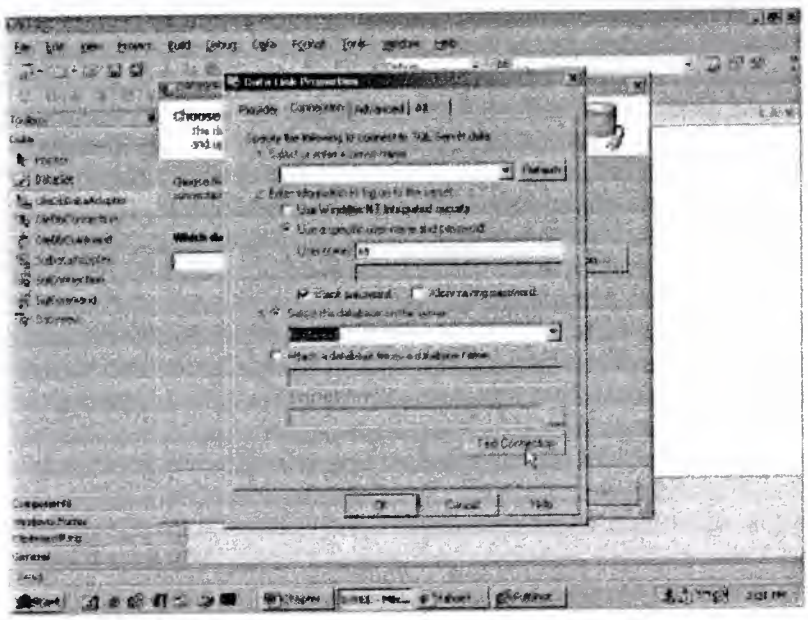
Screen 2.5

The Data Link Properties Dialog Box requires considerable amount of information, before it can create a new connection. However, all the boxes do not require to be filled up. The server name is left blank, since the database server runs on the same machine as the application.

A connection is made to the server, employing the user name 'sa'. Therefore, 'sa' is entered in the textbox labeled 'User Name'. Since the password is set to blank, the Check box for Blank password is selected. Then, we click on the down arrow of the drop down listbox for displaying the option of select database on server. The screen 2.6 depicts the outcome of our action.



The list provided is that of the databases, which have been created on the server. After having selected the Northwind database, the dialog box looks similar to what is shown in screen 2.7.

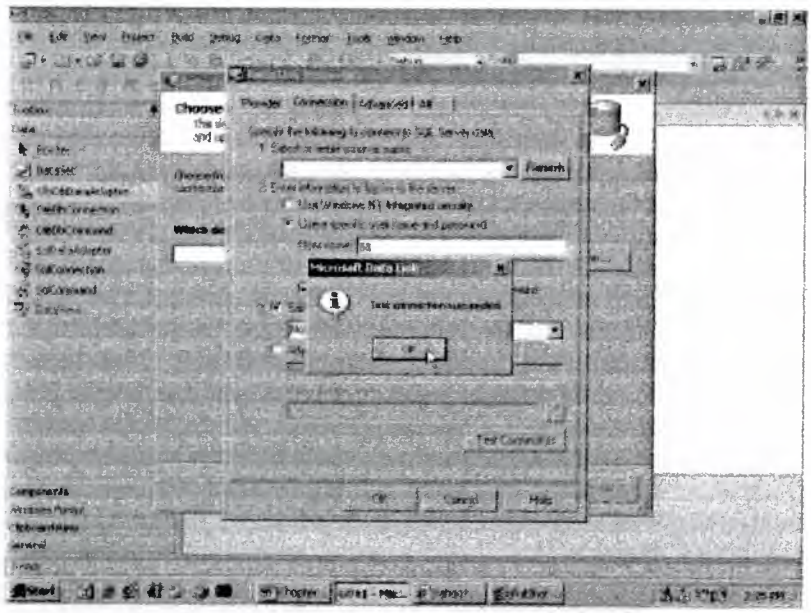


Screen 2.7

The program that we had written earlier had incorporated data from the 'Northwind' database, which is visible in the list. Although we are at liberty to select the database, we will stick to Northwind for the time being.

An added advantage of a drop down listbox is that, it averts the possibility of a wrong database name being entered by the user, since a name has to be selected from the list that is provided.

Now, click on the button labeled Test Connection. This finally ascertains whether all the values entered are valid from the standpoint of the database server.



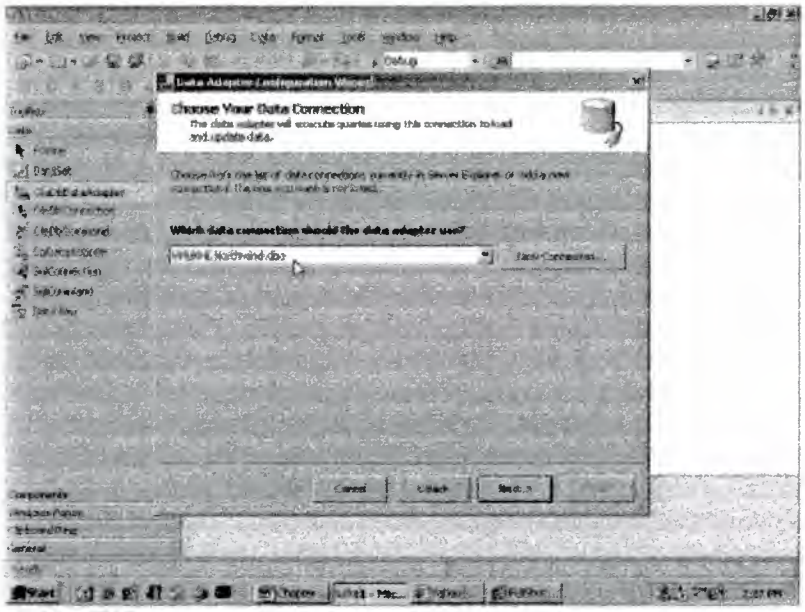
Screen 2.8

The Screen 2.8 returns with a message of success.

Some processing had taken place when the Test connection button was clicked.

This involved connecting to SQL Server using the user name 'sa' and using a blank password, and thereafter, accessing the Northwind database.

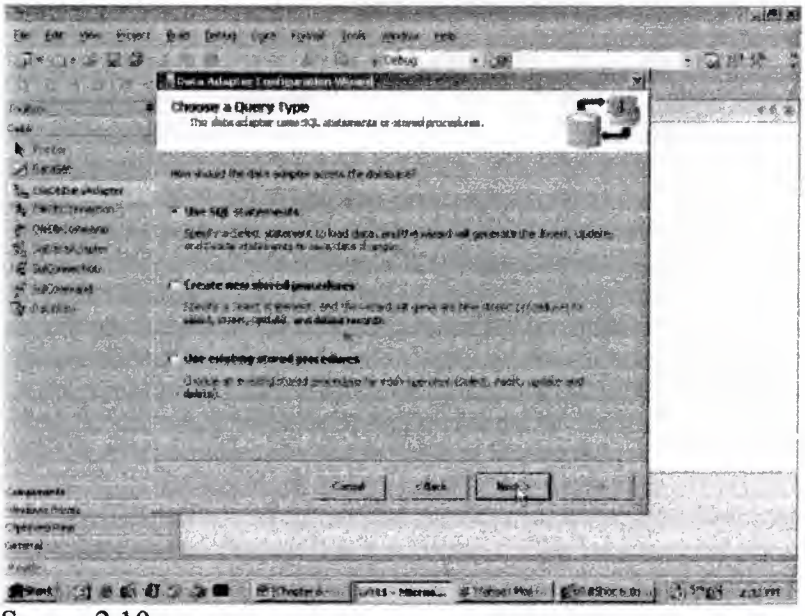
When you click on OK, you will be reverted back to one of the previous screens, with the name of VMUKHI.Northwind.dbo in the listbox, as shown in screen 2.9.



Screen 2.9

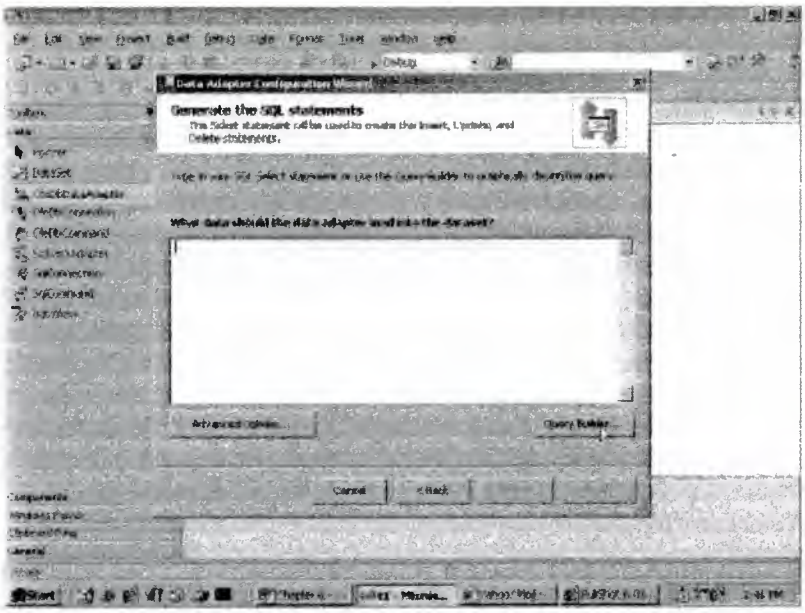
Vmukhi is the name assigned to our computer. It is followed by the name of the database Northwind, with an extension of dbo.

Now, click on the Next button to arrive at the screen 2.10. A 'stored procedure' is a program that resides and executes on the server. We feel quite contented to use the trusted SQL Select statements to fetch data. So, we simply click on the Next button, without making any amendments to screen 2.10.



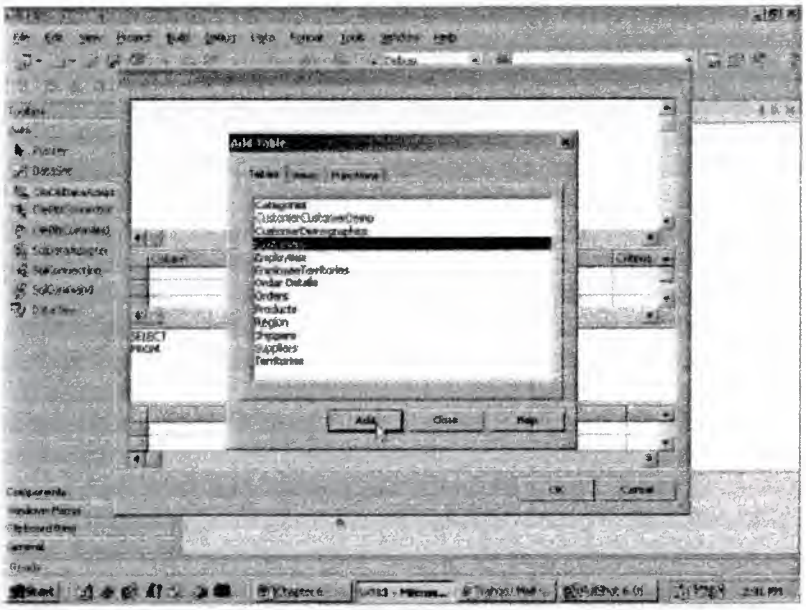
Screen 2.10

In screen 2.11, we see a cursor blinking in the text area, which is an indication that an SQL Select statement has to be entered.



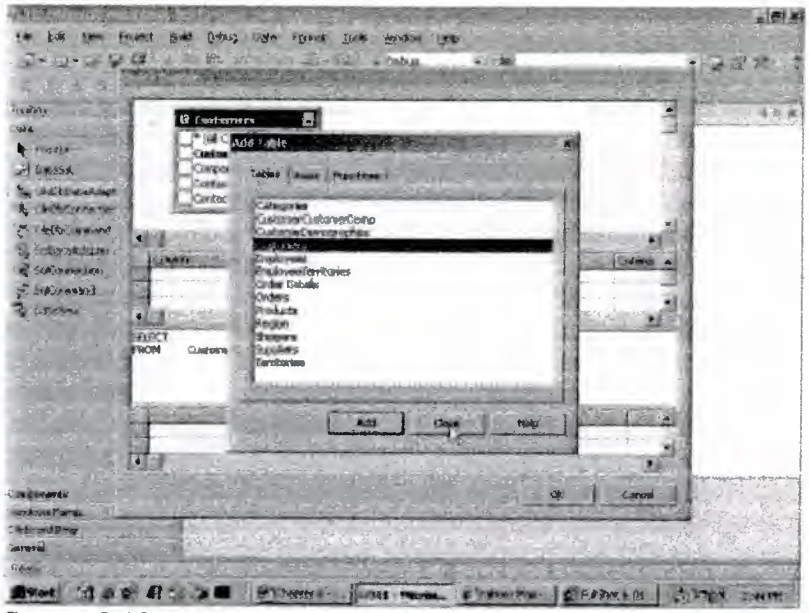
Screen 2.11

In case you are not conversant with the options, you can click on Query Builder, as we have done, to arrive at screen 2.12.



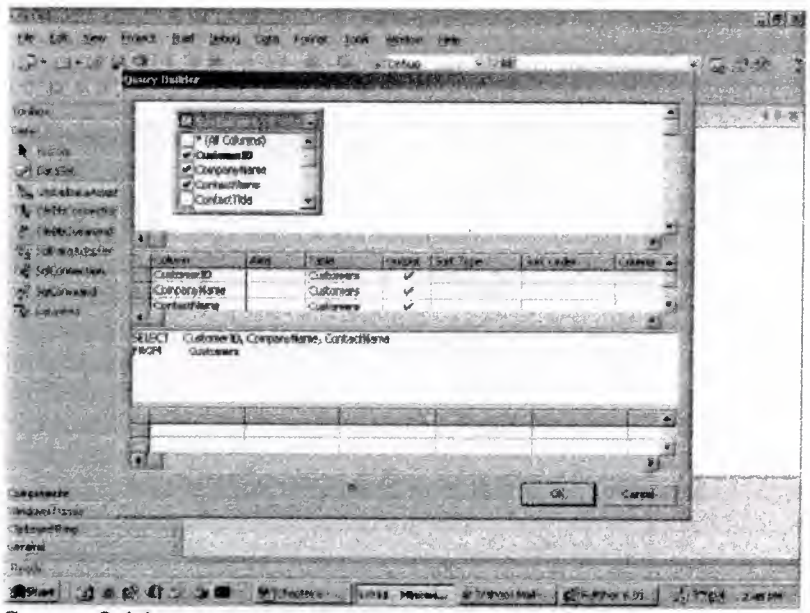
Screen 2.12

The listbox displays the tables that constitute the Northwind database. Click on Customers, and then on the Add button. The background area in screen 2.12 changes, to display a box containing the word 'customers', with a list of fields within it. Since there are no more tables to select from, we click on the Close button.



Screen 2.13

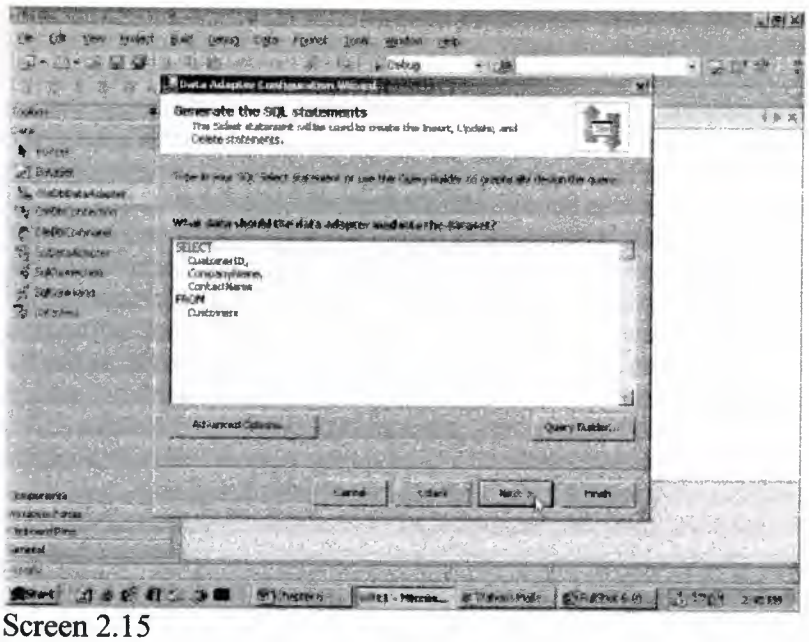
In screen 2.14, the first three fields are selected. This has been achieved by clicking on the check box, due to which, the selected field names get added to the Select statement.



Screen 2.14

Since we are satisfied with what we have achieved, we click on the OK button.

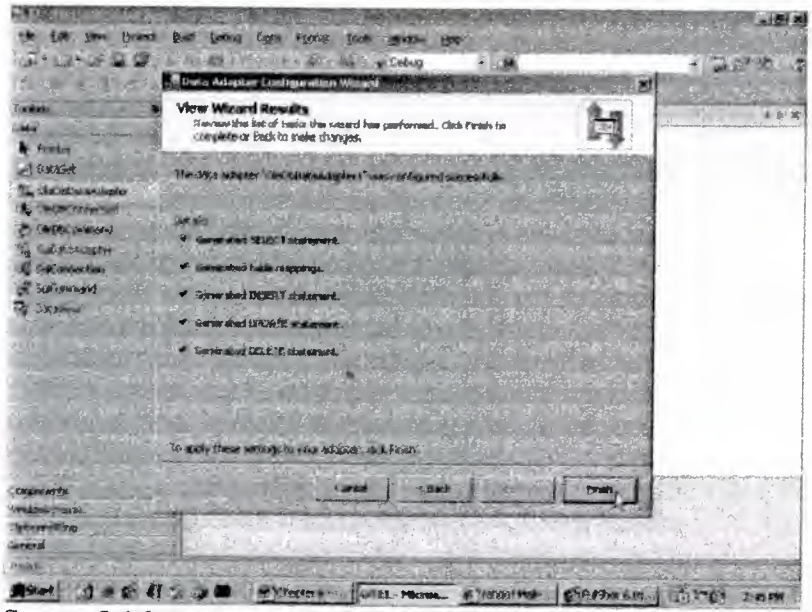
This takes us to screen 2.15, which displays the Select statement that is generated. The Select statement shows the three selected fields of CustomerID, Company Name and ContactName for which data is to be retrieved from the Customers table.



Screen 2.15

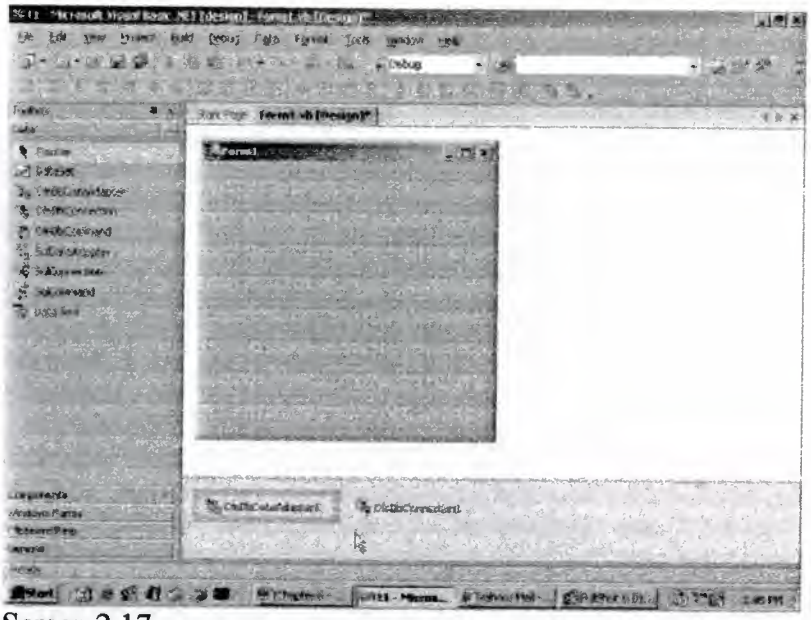
This is where a wizard really flaunts its mettle, since it is easier to select from a list, rather than to write things out manually. The beauty of SQL is revealed in all its glory, when we have to work with multiple tables.

When we click on the Next button, the screen 2.16 reveals that the wizard has internally performed a number of tasks for us. Finally, click on Finish to end the wizard.



Screen 2.16

As shown in screen 2.17, the screen does not show the object on the Form; instead, it has two objects, i.e. OleDbDataAdapter1 and OleDbConnection1, which are located at the bottom of the screen.

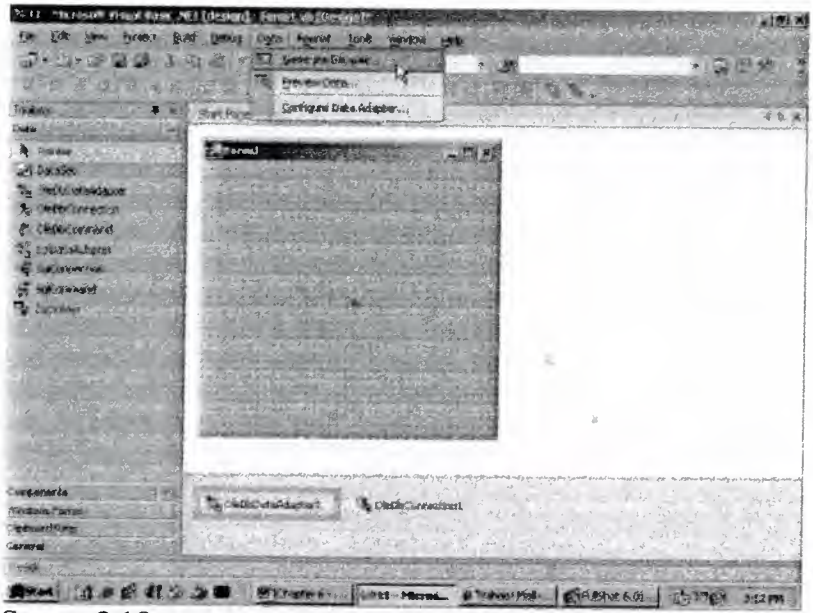


Screen 2.17

This highlights the fact that there are two types of controls.

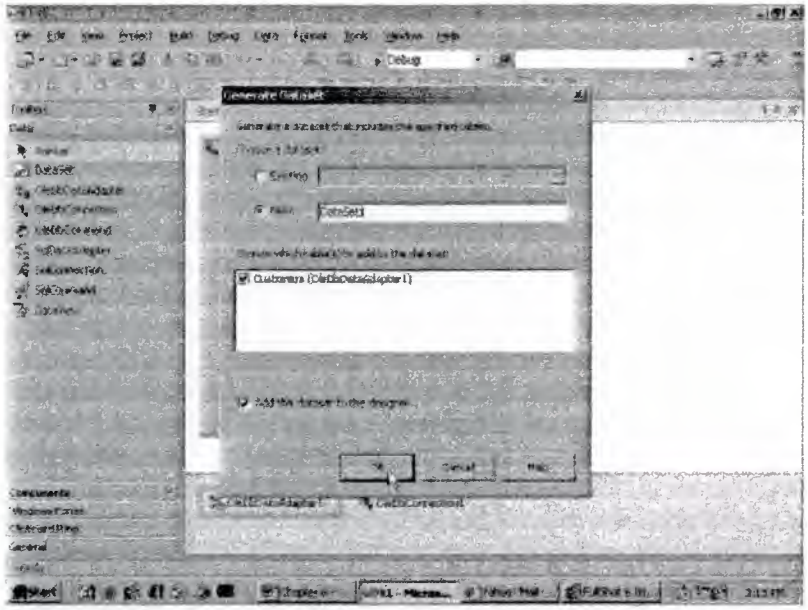
- The first type is similar to a button, which gets displayed in the Form.
- The second type is that of OleDbConnection, which does not have an User Interface, and which gets placed at the bottom of the Form.

The next object required is a DataSet object, which simply relates to a collection of tables. The Visual Studio.Net interface has a menu option Data. So, click on Data and select the option of 'Generate Dataset', as shown in Screen 2.18.



Screen 2.18

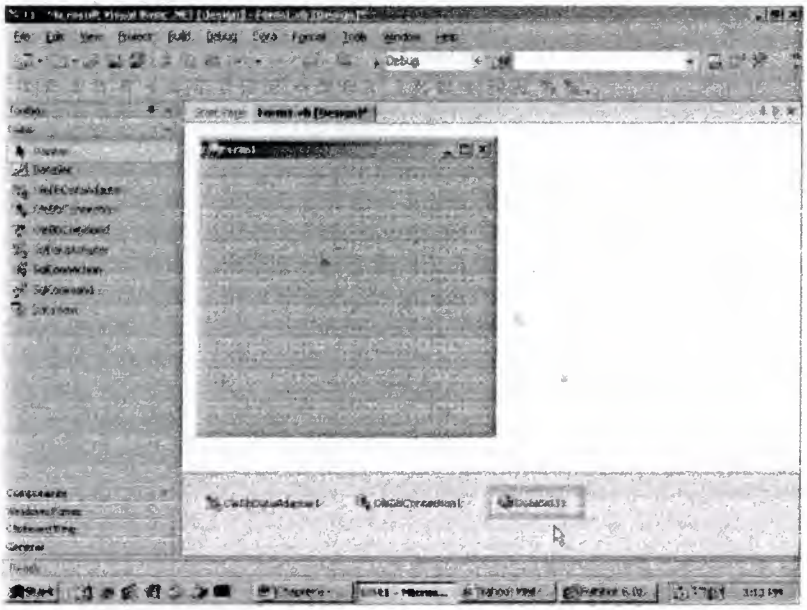
This generates a screen, as shown in screen 2.19.



Screen 2.19

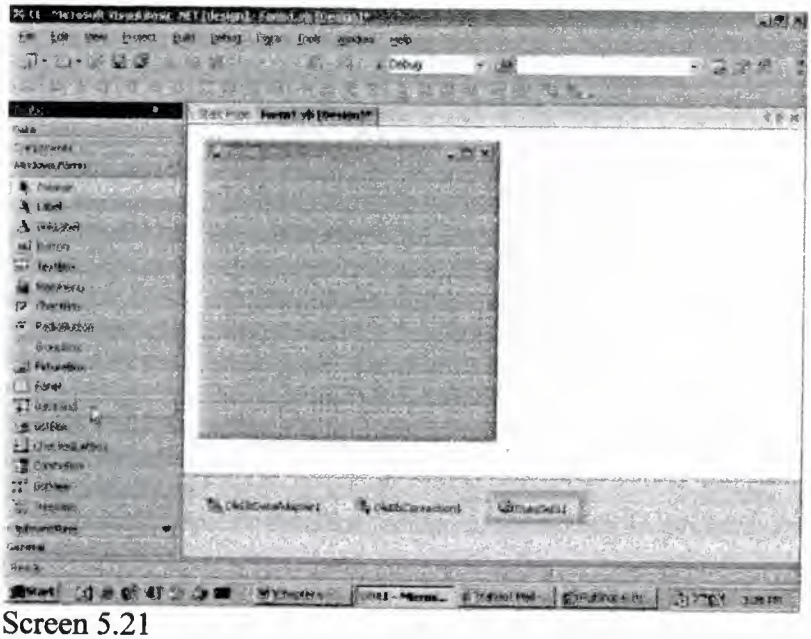
We are contented with the default options assigned to our dataset, where the name indicated is DataSet1. The framework is extremely astute, and hence, it adds the Customers table to the Dataset, as shown in the screen. Finally, since the check box at the bottom is selected, the dataset gets added to the Designer.

When the OK button is clicked, the dataset DataSet1 will be displayed at the bottom of the non User Interface controls. This is shown in Screen 2.20

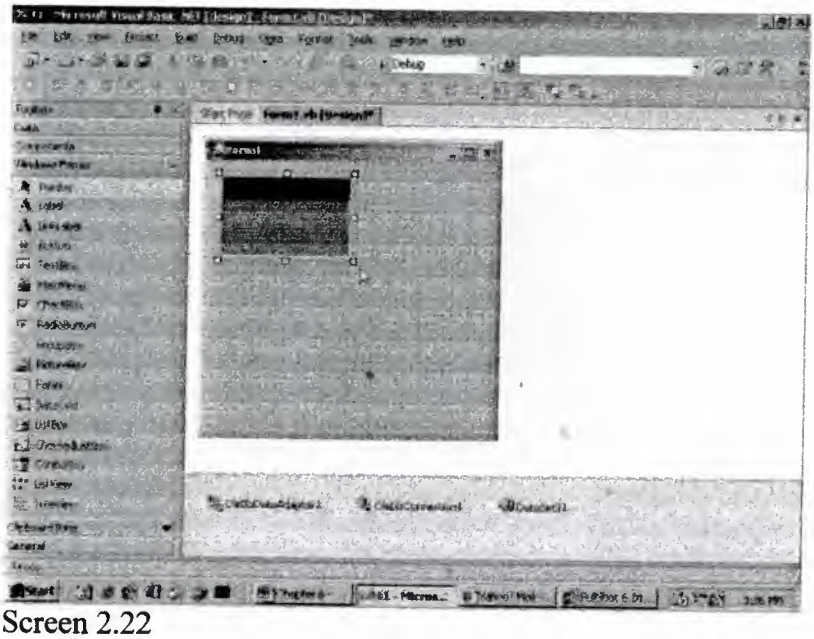


Screen 2.20

Next, a DataGrid has to be added to the Form. So, we click on the Windows Form Tab in the toolbox, select DataGrid and drag it onto the Form window. This is shown in Screen 2.21 and Screen 2.22.

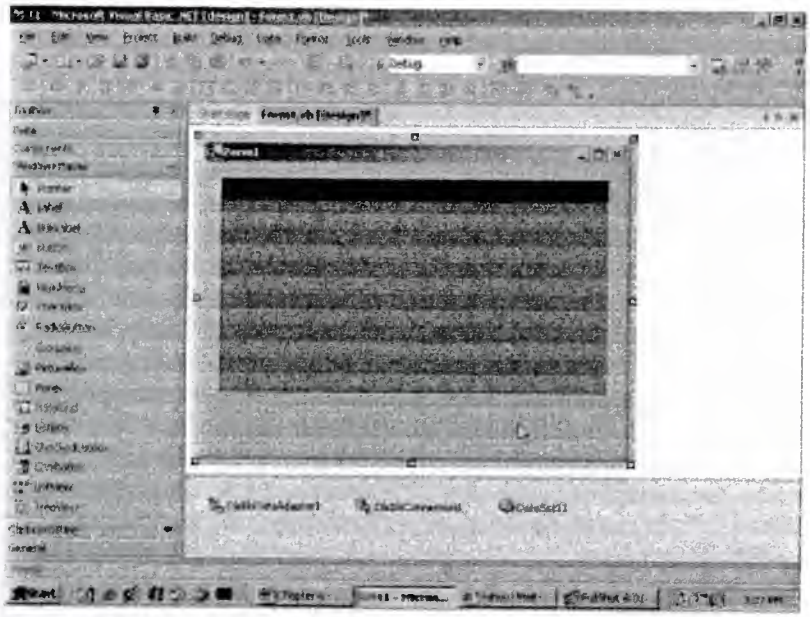


Screen 5.21



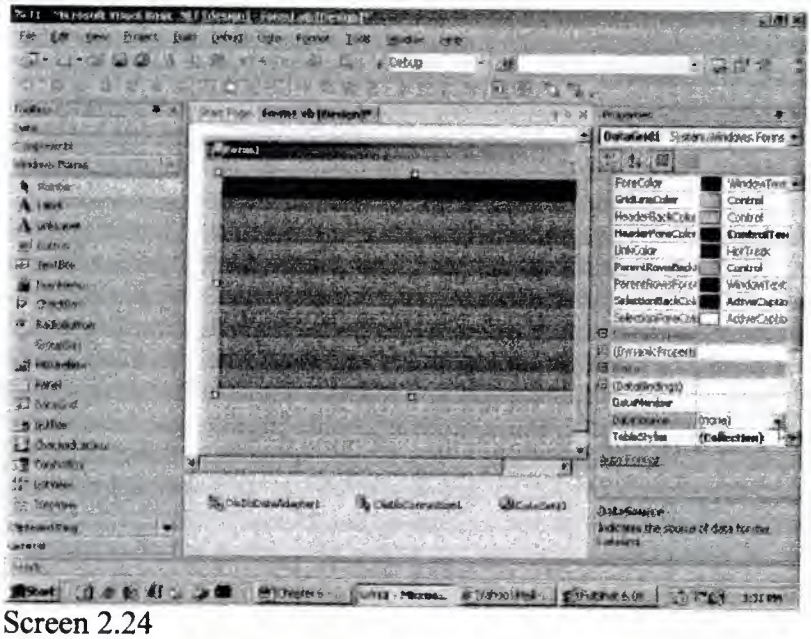
Screen 2.22

The DataGrid and Form are disconcertingly cramped. So, we click on their edges and drag them diagonally to enlarge their sizes, as in evident in screen 2.23.



Screen 2.23

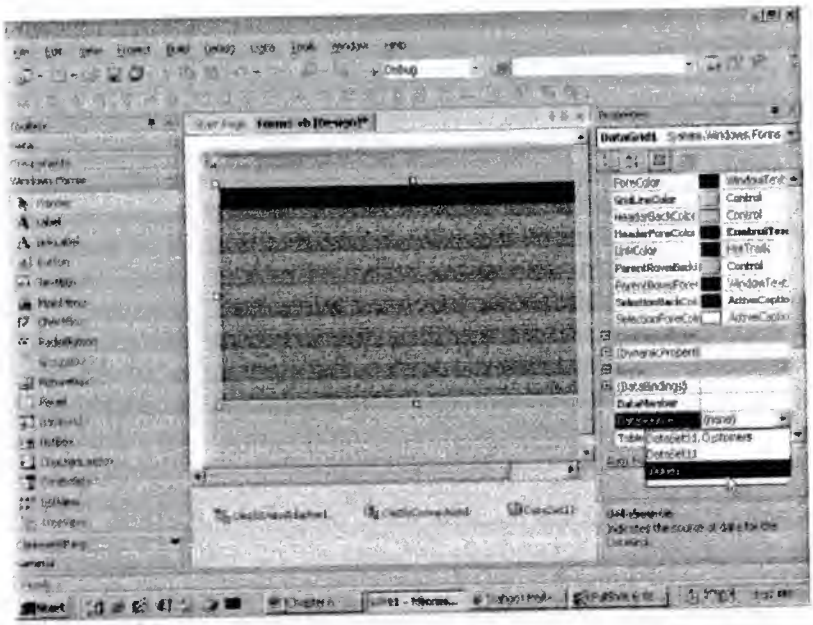
Our program had initialized various properties such as ClientSize, Background color etc. To modify these properties, click on the menu option of View-Properties Window.



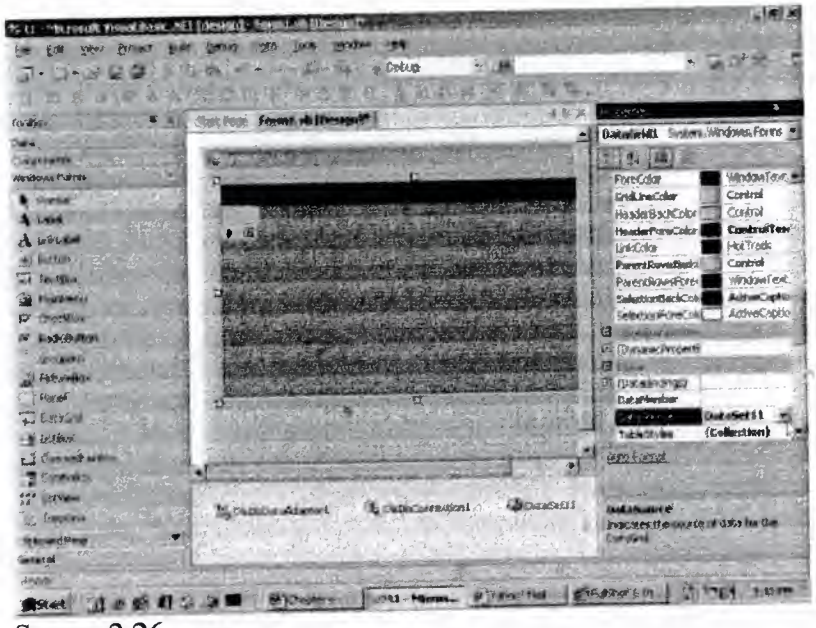
Screen 2.24

This brings up the Properties window on the right hand side of the screen. Since the DataGrid control is selected, the properties related to the Grid are displayed. When you select the Form, its properties get displayed in the Properties Window.

Click on the drop down listbox for the DataSource property. It displays a list containing two items, as seen in screen 2.25. Select the option of DataSet1, as shown in screen 2.26.

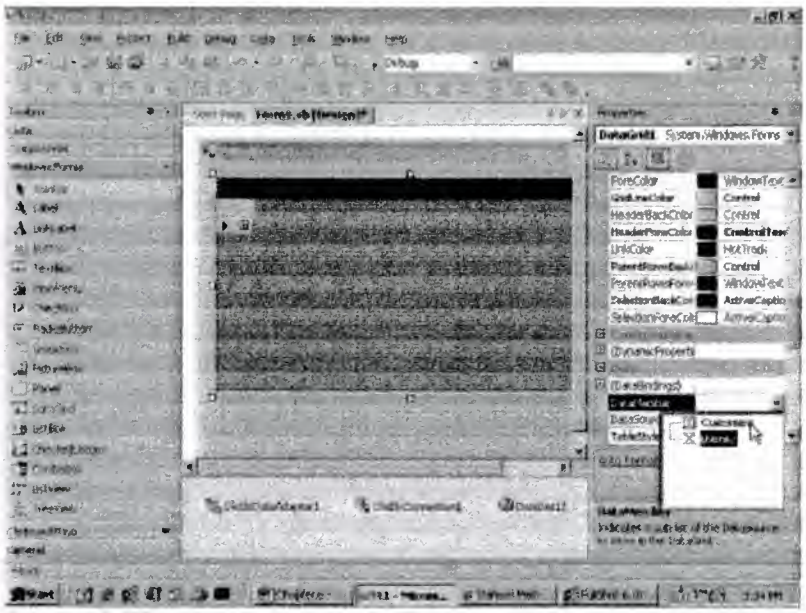


Screen 2.25



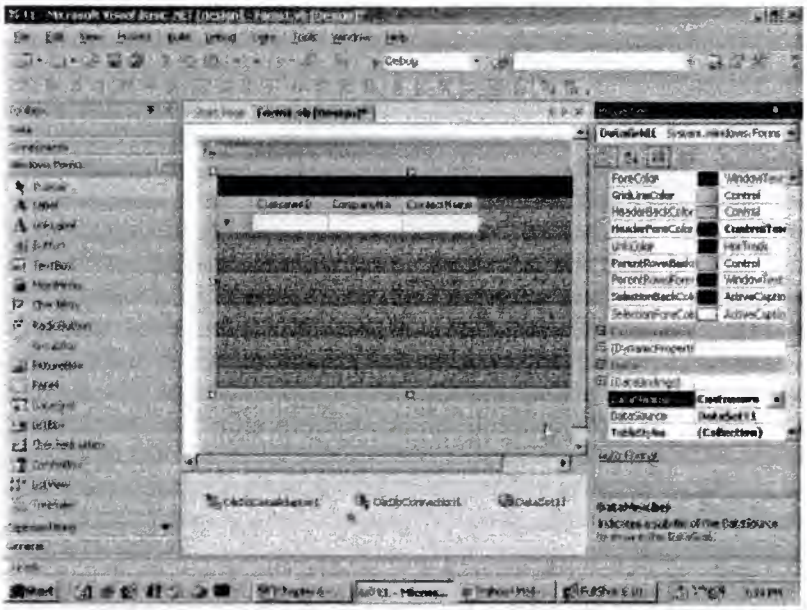
Screen 2.26

The DataMember property that is positioned immediately above the DataSource, is also required to be Set. So, click on the down arrow, and from the list, select the table named Customers.



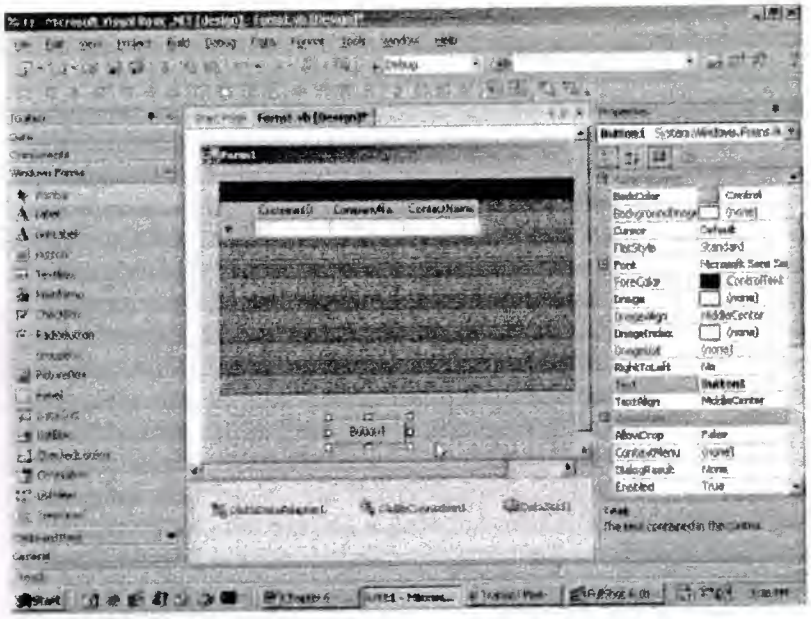
Screen 2.27

In screen 2.28, we have selected the Customers table. On doing so, the DataGrid automatically gets transformed to display three fields in the table.



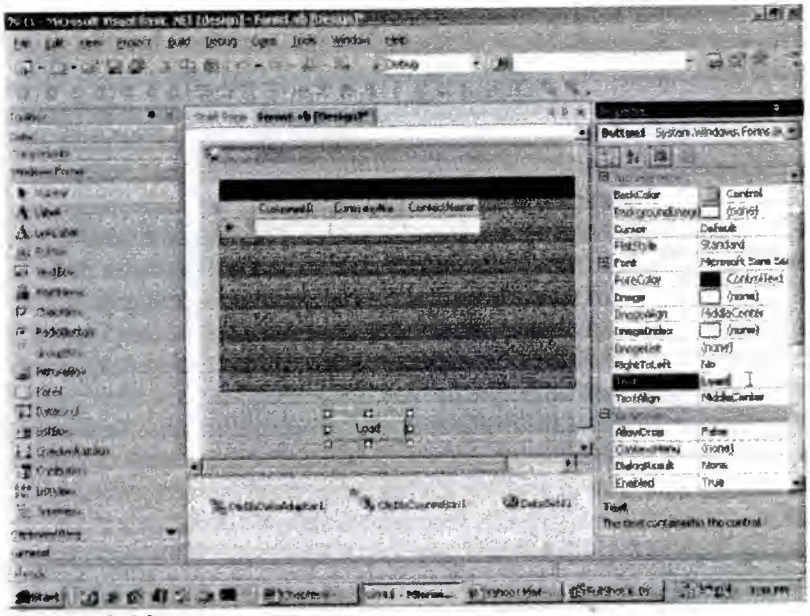
Screen 2.28

Now, incorporate another button control into the Form, as is shown in screen 2.28. The Properties window exhibits the properties for the button, since it is the selected object.



Screen 2.29

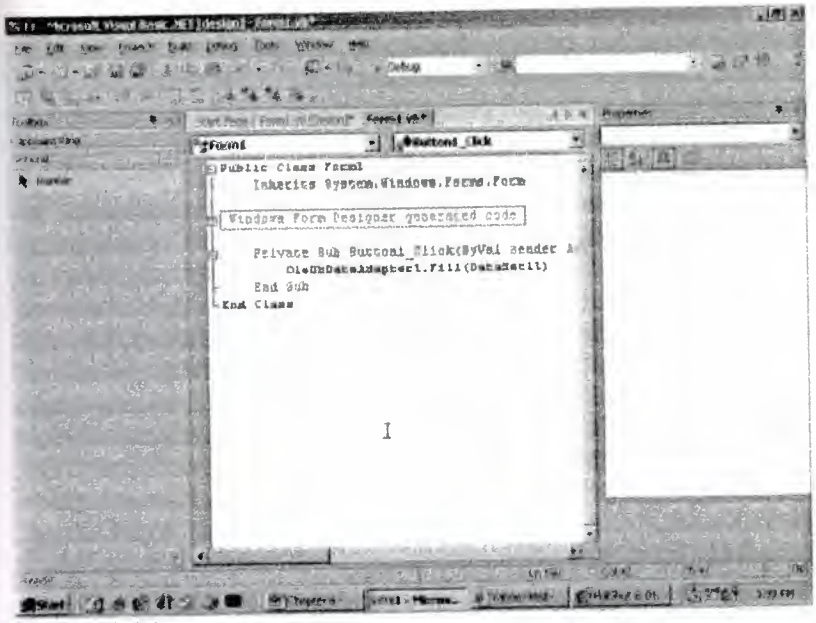
In the Properties window, enter the word 'Load' in the Text property of the button. Then, press Enter, and you will instantly witness the text on the button change to 'Load', as shown in screen 2.30.



Screen 2.30

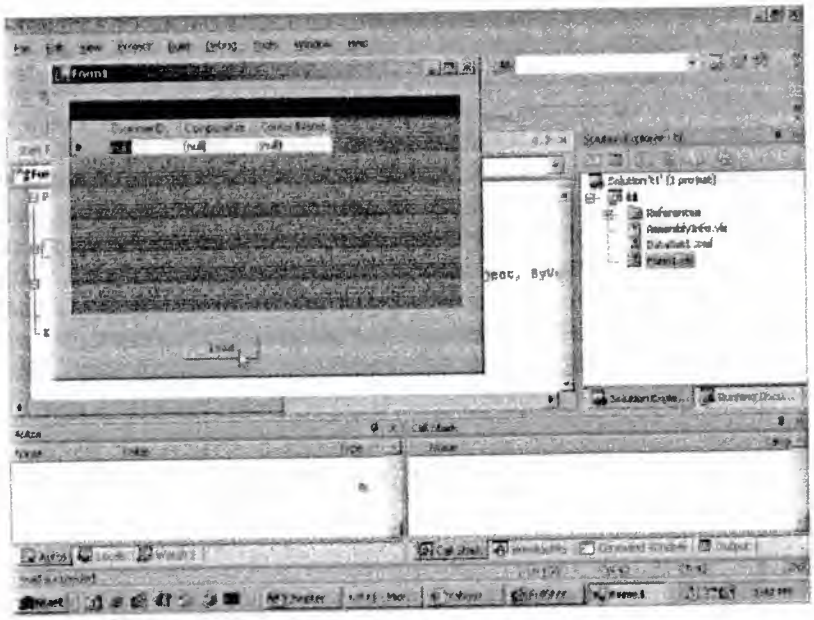
Now, double click on the button, and enter the following line at the cursor position:

```
OleDbDataAdapter1.Fill(DataSet11)
```

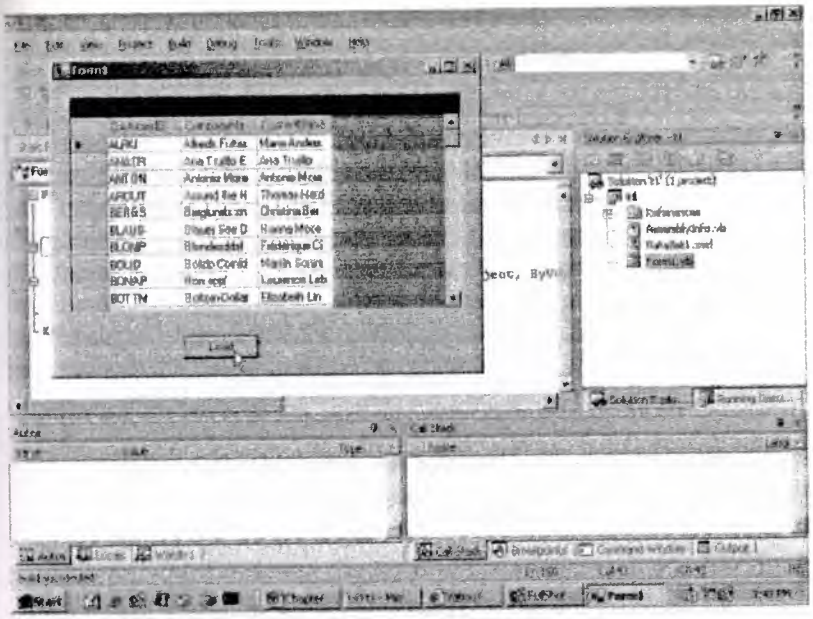


Screen 2.31

In order to verify whether everything has been entered accurately or not, click on the menu option Debug, and then, click on Start. The screen that emerges is shown in screen 2.32.


Screen 2.32

An empty DataGrid and a button labeled 'Load' are displayed. Now, click on the button. Lo & behold! The screen transforms, displaying the Data Grid containing the data retrieved from the Customer table, as is evident in screen 2.33.



Screen 2.33

Let us now snoop behind the scenes to check on the activities that have taken place. We would be explaining each new step incorporated above, by interpreting the code that has been generated. You will never be able to grasp the applications without learning the language. Therefore, we insist that you learn the language before diving straight into the depths of the applications. We have copied the generated code and presented it below

We have converged our attention around the code that has been introduced recently, so that our explanation is confined to the newly added code only, in order to avoid repetition. Furthermore, once the code has been explicated, it will not be exhibited again.

Since the process of code generation has been automated, a lot of code has been included to handle situations that may possibly occur only once in a million years! Thus, a lot of redundant code is generated. We have displaced all comments, since they impede our understanding of the code. We have also done away with all the blank lines, since they occupy too much space.

```
Public Class Form1
Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
Public Sub New()
MyBase.New()
InitializeComponent()
End Sub
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub
```

```
Friend WithEvents OleDbDataAdapter1 As
System.Data.OleDb.OleDbDataAdapter
Friend WithEvents OleDbSelectCommand1 As
System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbInsertCommand1 As
System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbUpdateCommand1 As
System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbDeleteCommand1 As
System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbConnection1 As
System.Data.OleDb.OleDbConnection
Private components As System.ComponentModel.IContainer
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
Me.OleDbDataAdapter1 = New System.Data.OleDb.OleDbDataAdapter()
Me.OleDbSelectCommand1 = New System.Data.OleDb.OleDbCommand()
Me.OleDbInsertCommand1 = New System.Data.OleDb.OleDbCommand()
Me.OleDbUpdateCommand1 = New System.Data.OleDb.OleDbCommand()
Me.OleDbDeleteCommand1 = New System.Data.OleDb.OleDbCommand()
Me.OleDbConnection1 = New System.Data.OleDb.OleDbConnection()
Me.OleDbDataAdapter1.DeleteCommand = Me.OleDbDeleteCommand1
Me.OleDbDataAdapter1.InsertCommand = Me.OleDbInsertCommand1
Me.OleDbDataAdapter1.SelectCommand = Me.OleDbSelectCommand1
Me.OleDbDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Customers", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("CustomerID", "CustomerID"),
New System.Data.Common.DataColumnMapping("CompanyName",
"CompanyName"), New
System.Data.Common.DataColumnMapping("ContactName",
"ContactName")})})
Me.OleDbDataAdapter1.UpdateCommand = Me.OleDbUpdateCommand1
Me.OleDbSelectCommand1.CommandText = "SELECT CustomerID,
CompanyName, ContactName FROM Customers"
Me.OleDbSelectCommand1.Connection = Me.OleDbConnection1
Me.OleDbInsertCommand1.CommandText = "INSERT INTO
Customers(CustomerID, CompanyName, ContactName) VALUES (?, ?, ?);
SEL" & _
"ECT CustomerID, CompanyName, ContactName FROM Customers WHERE
(CustomerID = ?)"
Me.OleDbInsertCommand1.Connection = Me.OleDbConnection1
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("CompanyName",
System.Data.OleDb.OleDbType.VarWChar, 40, "CompanyName"))
```

```
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("ContactName",
System.Data.OleDb.OleDbType.VarWChar, 30, "ContactName"))
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Select_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
Me.OleDbUpdateCommand1.CommandText = "UPDATE Customers SET
CustomerID = ?, CompanyName = ?, ContactName = ? WHERE (Cust" & _
"omerID = ?) AND (CompanyName = ?) AND (ContactName = ? OR ? IS
NULL AND ContactN" & _
"ame IS NULL); SELECT CustomerID, CompanyName, ContactName FROM
Customers WHERE (" & _
"CustomerID = ?)"
Me.OleDbUpdateCommand1.Connection = Me.OleDbConnection1
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("CompanyName",
System.Data.OleDb.OleDbType.VarWChar, 40, "CompanyName"))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("ContactName",
System.Data.OleDb.OleDbType.VarWChar, 30, "ContactName"))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"CustomerID", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_CompanyName",
System.Data.OleDb.OleDbType.VarWChar, 40,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"CompanyName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_ContactName",
System.Data.OleDb.OleDbType.VarWChar, 30,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"ContactName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_ContactName1",
System.Data.OleDb.OleDbType.VarWChar, 30,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"ContactName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Select_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
Me.OleDbDeleteCommand1.CommandText = "DELETE FROM Customers
WHERE (CustomerID = ?) AND (CompanyName = ?) AND (ContactNa" & _
"me = ? OR ? IS NULL AND ContactName IS NULL)"
Me.OleDbDeleteCommand1.Connection = Me.OleDbConnection1
```

```
Me.OleDbDeleteCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"CustomerID", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbDeleteCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_CompanyName",
System.Data.OleDb.OleDbType.VarWChar, 40,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"CompanyName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbDeleteCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_ContactName",
System.Data.OleDb.OleDbType.VarWChar, 30,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"ContactName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbDeleteCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_ContactName1",
System.Data.OleDb.OleDbType.VarWChar, 30,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"ContactName", System.Data.DataRowVersion.Original, Nothing))
Me.OleDbConnection1.ConnectionString = "Provider=SQLOLEDB.1;Persist
Security Info=False;User ID=sa;Initial Catalog=Northw" & _
"ind;Use Procedure for Prepare=1;Auto Translate=True;Packet
Size=4096;Workstation" & _
" ID=VMUKHI;Use Encryption for Data=False;Tag with column collation when
poss" & _
"ible=False"
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Name = "Form1"
Me.Text = "Form1"
End Sub
#End Region
End Class
```

In the Form, we first introduced an OleDbDataAdapter object. This resulted in the creation of the DataAdapter and DataConnection objects, which carried the information essential to connect to the database server. Resultantly, the above code was generated.

Other than calling the InitializeComponent function, the constructor has no other role to play. All the valuable code is placed in the InitializeComponent function. The Dispose function remains unchanged, and can be ignored completely. Normally, all the instance variables are positioned after these housekeeping functions.

We had earlier learnt that two objects of type SqlConnection and SqlDataAdapter were required to connect to any database in SQL Server. Since these objects are restricted to SQL Server, they cannot be used in the context of other database servers.

Thus, the code is not generic in nature. Therefore, the Wizard creates objects of type OleDbConnection and OleDbDataAdapter, since they can then be implemented on multiple database servers, including SQL Server. Barring this difference, both the objects work in a similar manner.

Every corresponding object has a number appended to the name of the class. This number gets incremented when another object of the same kind is added to the Form. This helps in assigning unique names to each object.

The wizard generates SQL statements for adding, updating and deleting records from the database. The OleDbCommand object is created for this purpose, since it is conversant in dealing with SQL.

In the InitializeComponent function, all the instance variables are initialized using the new keyword with the constructor that does not take any parameters.

The DeleteCommand is a property of type OleDbCommand, which represents an SQL statement or a stored procedure employed for deleting records. Similarly, the object also contains properties to represent Insert, Select and Update commands.

The property of TableMappings is of type DataTableMappingCollection. It is a collection objects that provides the master mapping between the source table and a DataTable. The AddRange function accepts an array of DataTableMapping objects.

```
Me.OleDbDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Customers", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("CustomerID", "CustomerID"),
New System.Data.Common.DataColumnMapping("CompanyName",
"CompanyName"), New
System.Data.Common.DataColumnMapping("ContactName",
"ContactName")})})
```

The first parameter to the constructor of a DataTable Mapping is a string that represents a table. The second parameter is the name of the table selected in the Select statement, i.e. Customers. Following this array, there is another array named Columns, which is a DataColumnMapping collection.

The DataColumnMapping object takes two strings:

- The first is the column name from the data source.
- The second is the column name from the DataSet that it must map to.

While building the query, we had chosen three columns. Therefore, there are three members in the array. The parameter names and the column names have been kept identical.

```
Me.OleDbSelectCommand1.CommandText = "SELECT CustomerID,
CompanyName, ContactName FROM Customers"
```

Do bear in mind that most of the above code is superfluous, and hence, dispensable. The OleDbCommand has a property called CommandText, using which, the OleDBSelectCommand is initialized to the SQL statement associated with the Command object. The SQL statement that is generated with the Query Builder, is assigned to this property.

```
Me.OleDbSelectCommand1.Connection = Me.OleDbConnection1
```

Every Adapter object needs a Connection handle. Therefore, the Connection property is set to the OleDbConnection object.

The CommandText property in the Insert Object is set to a SQL Insert statement. We can very easily add records while the DataGrid is displaying its set of records.

```
Me.OleDbInsertCommand1.CommandText = "INSERT INTO
Customers(CustomerID, CompanyName, ContactName) VALUES (?, ?, ?);
SEL" & _
"ECT CustomerID, CompanyName, ContactName FROM Customers WHERE
(CustomerID = ?)"
```

Note, that the syntax for the Insert statement starts with the reserved words 'Insert into', followed by the name of the table, i.e. Customers, followed by the list of field names in round brackets. This is followed by the reserved word 'values', which is followed by the actual values that need to be inserted, placed within round brackets. These values are not known at this stage. Hence, they are represented by a ? symbol, which represents a parameter or a place holder. It will be substituted by a value in due course of time.

The SELECT statement with the Insert command identifies a unique customer record. Since the line is broken on two lines, the word 'SELECT' has the continuation character of &_ in double quotes.
One line below the insert command is the Parameters Collection object, which keeps track of all the parameters.

```
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
```

To this collection, we add an instance of an OleDbParameter object, whose constructor takes the following parameters: First is a parameter name, followed by the data type, followed by the width of the column, and finally, followed by the name of the source column. The parameters have been assigned the same names as those of the fields.

```
Me.OleDbInsertCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Select_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
```

The parameter that identifies the CustomerID in the where clause is given the name of Select_CustonerID as shown above.

```
Me.OleDbUpdateCommand1.CommandText = "UPDATE Customers SET
CustomerID = ?, CompanyName = ?, ContactName = ? WHERE (C" & _
"ustomerID = ?) AND (CompanyName = ?) AND (ContactName = ? OR ? IS
NULL AND ContactN" & _
"ame IS NULL); SELECT CustomerID, CompanyName, ContactName FROM
Customers WHERE (" & _
"CustomerID = ?)"
```

The Update command follows next, which contains the word Update, followed by the table name Customers, and finally, followed by the fields that are to be changed.

Only three of the fields have been chosen. Therefore, the SET command has exactly three field names as three parameters. By default, the Update statement acts on all records. Therefore, to modify only specific records, the 'where' clause has to be used, thereby restricting access to the number of records it can act upon.

The CustomerID is the primary key, which has a unique value for each record. Therefore, if the Primary key is used in the 'where' clause, it will affect only one record.

```
Me.OleDbUpdateCommand1.Parameters.Add(New
System.Data.OleDb.OleDbParameter("Original_CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5,
System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0, Byte),
"CustomerID", System.Data.DataRowVersion.Original, Nothing))
```

If you notice the parameter statement, the name of the field has been prefixed with the word Original_. The fourth parameter is an enum of ParameterDirection, which refers to the type of parameter. The enum consists of the four values of Input, Input Output, Output and Return Value.

The value of Input signifies that the value has not been received, but would be furnished in due course. The next value of False is a Boolean, which indicates whether the column will accept null values or not; a value of True signifies it that it will.

The next value is for precision, which determines the number of digits permissible to the left and right of the decimal point. This is followed by the Scale parameter, which determines the total number of decimal places that the field can be resolved to. Initially the above two parameters would appear absurd.

Next, we encounter the real column name, followed by a DataRowVersion enum, which takes four values: Current, Default, Original and Proposed. Original represents original values, as against the current values. Finally, we come to an object, which currently has no value.

```
Me.OleDbDeleteCommand1.CommandText = "DELETE FROM Customers
WHERE (CustomerID = ?) AND (CompanyName = ?) AND (ContactName = ?
OR ? IS NULL AND ContactName IS NULL)"
```

After the Update command, we shall tackle the Delete command. This command starts with the words 'delete from', followed by the table name Customers, and then,

followed by the 'where' clause that identifies the records that can be deleted. The parameters are identical to the Update command.

```
Me.OleDbConnection1.ConnectionString = "Provider=SQLOLEDB.1;Persist
Security Info=False;User ID=sa;Initial Catalog=Northw" & _
"ind;Use Procedure for Prepare=1;Auto Translate=True;Packet
Size=4096;Workstation" & _
" ID=VMUKHI;Use Encryption for Data=False;Tag with column collation when
poss" & _
"ible=False"
```

Now, the focus once again shifts to the OleDbConnection object. Earlier, we had passed the connection string to the constructor. However, as an alternative approach, we could use the ConenctionString property instead. This is what the Constructor eventually does with the string that is passed to it.

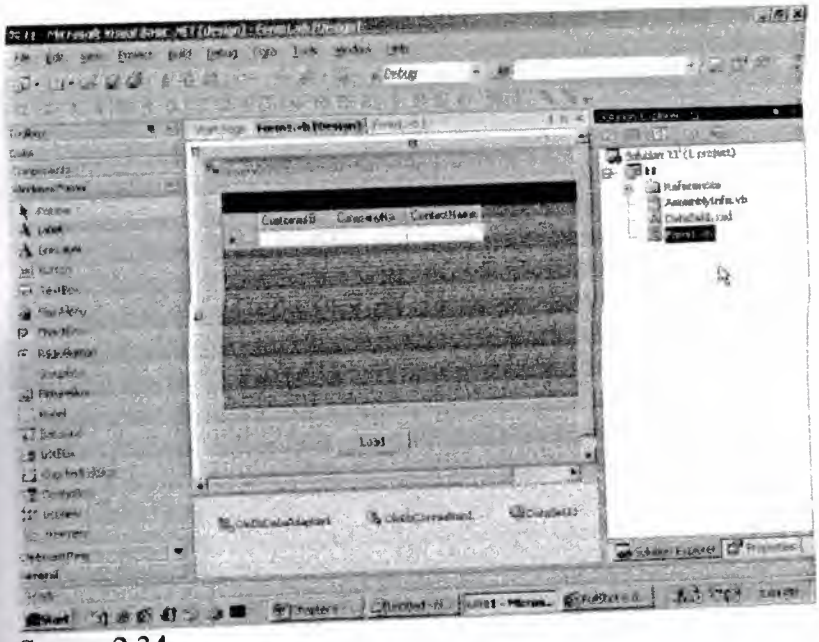
We shall restrict ourselves to only a few of the connection string properties for the moment. When the property Provider refers to the database, a connection gets established. The User ID property is the username, and the Initial Catalog is Northwind. The rest of the code is conventional and mundane.

Before concluding its tasks, the wizard takes all the information that has been entered in the textboxes, and it generates the above code. After the wizard generated the SQL statements, we clicked on the menu-option Generate Dataset under Data.

```
Friend WithEvents DataSet11 As t1.DataSet1
```

The dataset name DataSet1, which is the default name, is used to identify the dataset. This results in the generation of an instance variable called DataSet11 of type t1. Its name is formed by joining the name of our project with the name of the dataset, i.e. DataSet1.

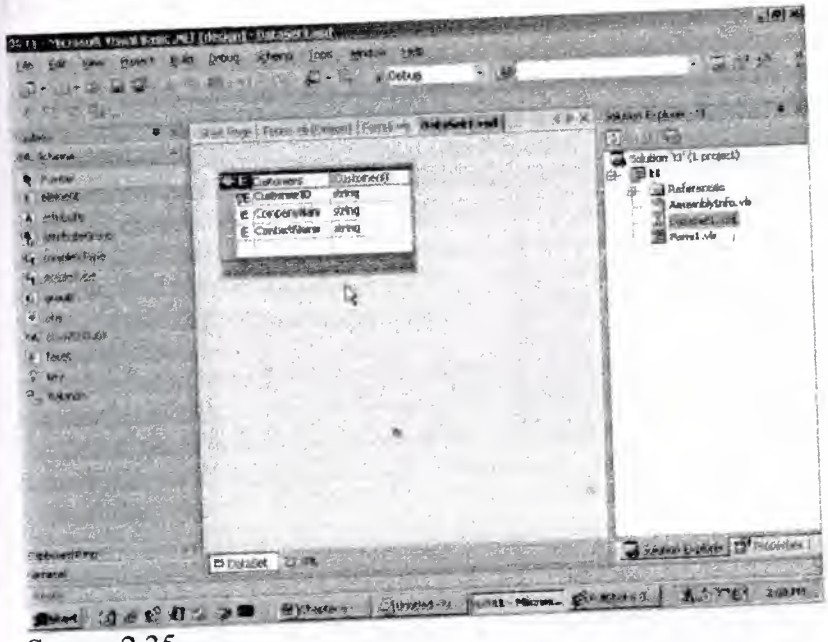
Before proceeding any further, click on the menu option View, and then, on the Solution Explorer. This brings us to screen 2.34, which has a window that lists the files constituting our project.



Screen 2.34

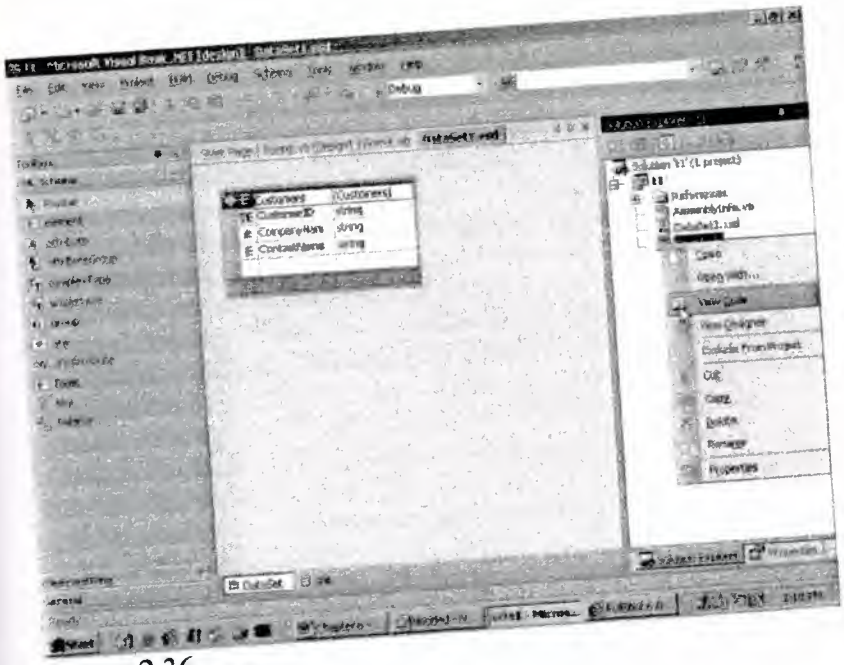
A Solution or a project for the moment will be used interchangeably.

The file Form1.vb contains the code for the Form Design. Double click on the DataSet1.xsd file, which is generated on creation of the DataSet object. This brings us to the screen 2.35, where the relationship has been depicted in a visual form.



Screen 2.35

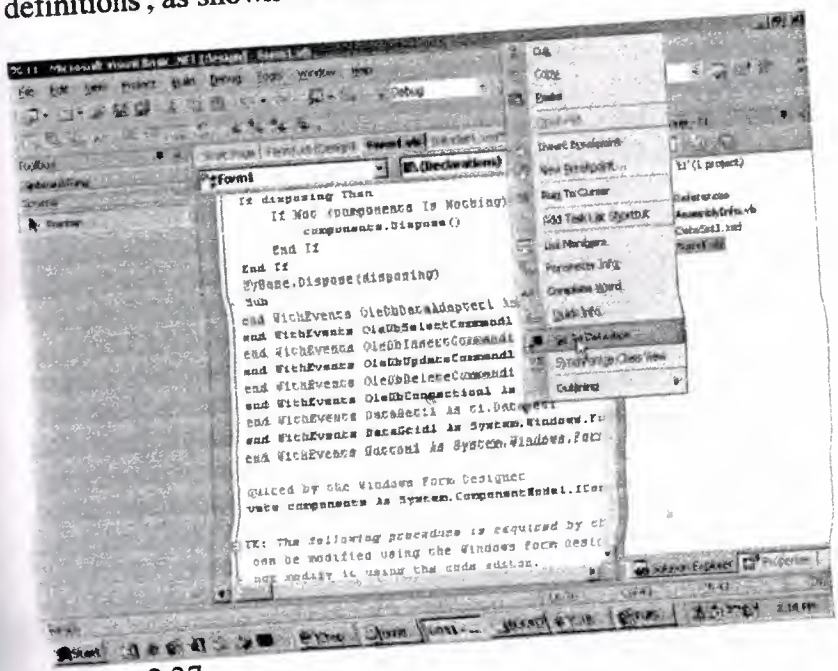
In the Solution Explorer, right click on Form1.vb item. The popup menu that emerges is shown in screen 2.36. It reveals the options that are available on this item.



Screen 2.36

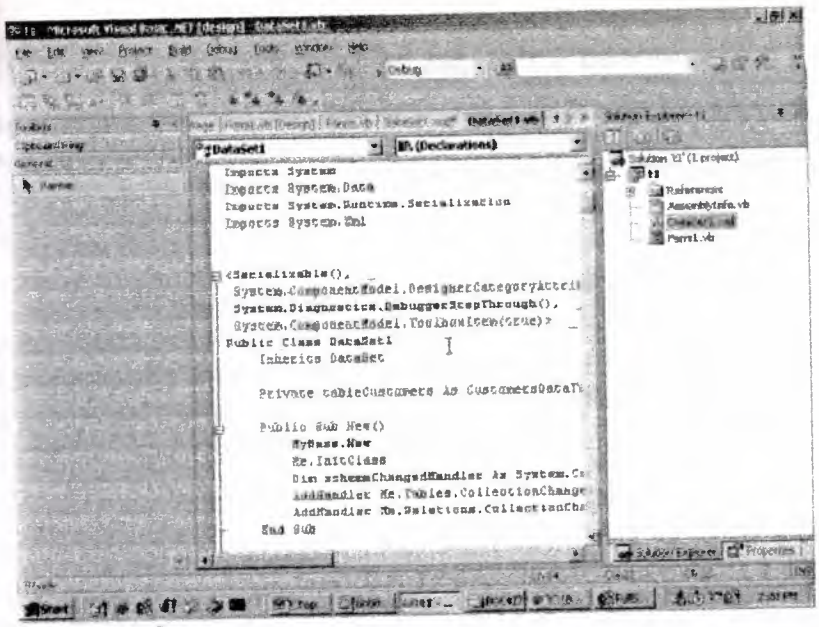
On selecting View Code, we arrive at the Code Generator. The other route to reach the Code Generator is by clicking on the filename Form1.vb on the panel, just below the toolbars.

In the code that is generated within Form1.vb, we place the cursor on the line DataSet11 at t1.DataSet1, and then, right click on it. Select the option of 'Go to definitions', as shown in screen 2.37.



Screen 2.37

This option would transport you to screen 2.38, where the class of DataSet1 is defined. It divulges the fact that DataSet1 is defined in a separate class of DataSet1.vb, and it is derived from the class DataSet.



Screen 2.38

In the beginning of the file DataSet1.vb, we come across the word Autogenerated, which merely informs us that a program has generated this file. Someday we intend to lift the veil off the name of the program that generates this code, and also the code that it generates. However, at this moment, we are not interested in the code that is written for the DataSet, since we shall not be using any of it.

The Visual Basic.Net language assimilates various types of code, which we shall abstain from analyzing right now. However, we shall explicate the various types of code, as we stumble upon them in due course.

In addition to the instance variable that gets created, the following lines get added to the Form.vb code, when the Generate DataSet menu-option is selected.

```
Me.DataSet11 = New t1.DataSet1()
CType(Me.DataSet11, System.ComponentModel.ISupportInitialize). BeginInit()
Me.DataSet11.DataSetName = "DataSet1"
Me.DataSet11.Locale = New System.Globalization.CultureInfo("en-US")
Me.DataSet11.Namespace = "http://www.tempuri.org/DataSet1.xsd"
CType(Me.DataSet11, System.ComponentModel.ISupportInitialize).EndInit()
```

The first line simply creates an instance of a DataSet1 object and stores it in DataSet11. The next line begins with a function called Ctype.

The sole task of the Ctype function is to convert the data type of the first parameter, to the data type specified by the second parameter. Thus, the first parameter, i.e. DataSet11 object of type DataSet is converted into the ISupportInitialize type, which belongs to the System.ComponentModel namespace.

The resultant object is an ISupportInitialize type, from which the BeginInit function is called. The above line could have been alternately worded as: DataSet11.BeginInit(). This is so because the DataSet class is derived from ISupportInitialize.

The BeginInit function signals to the framework that the DataSet initialization process has commenced, and hence, it should stop all activities related to the DataSet, until the EndInit function gets called. Therefore, the DataSet object is not authorized to carry out any internal initializations, since these functions optimize changes to multiple sets of properties.

However, at design time, we can co-initialize properties that are contingent upon each other. For example, the DataSetName specifies a name for the DataSet, and the locale always deals with the language issues. The namespace property handles the reading and writing of XML Schemas. A point to be noted here is that, the above code is spread out in the sub InitializeComponent.

After the DataSet generation is completed, we had selected the DataGrid control from the Forms Toolbox. This results in the creation of an instance variable called DataGrid1.

```
Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
Me.DataGrid1 = New System.Windows.Forms.DataGrid()

CType(Me.DataGrid1,System.ComponentModel.ISupportInitialize). BeginInit()

Me.DataGrid1.DataMember = ""
Me.DataGrid1.HeaderForeColor = System.Drawing.SystemColors.ControlText
Me.DataGrid1.Location = New System.Drawing.Point(32, 64)
Me.DataGrid1.Name = "DataGrid1"
Me.DataGrid1.TabIndex = 0

Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.DataGrid1})
CType(Me.DataGrid1, System.ComponentModel.ISupportInitialize).EndInit()
```

In general, ushering in of any control leads to the generation of additional code in the above format. An instance variable is created with the name of the class, followed by a number beginning with 1, and not 0.

It is assumed that every control would be dealing with events. Therefore, it has BeginEvent and EndEvent methods, which allow the control to be initialized without any interference. In principle, they serve the purpose of a "Do Not Disturb" sign. The control is at liberty to act in any way that it likes, until the EndEvent method is called. It even enjoys the license not to do anything at all !

The abovementioned two methods are actually precautionary measures, which may or may not be implemented.

The DataMember property is initialized to Null, the HeaderForeColor property is initialized to a predefined color, the tab index is initialized to 0 and the name is initialized to DataGrid1. The only important property is Location, since its co-ordinates determine the position on the Form.

You may wonder as to why Visual Studio.Net provides distinct properties for each control. The logic behind this is that Visual Studio.Net is oblivious to what is being written, since it is the job of the control to usher in the code that it yearns for. The AddRange function is used to add the control to the Controls collection.

When the DataSource property of the DataGrid is set to DataSet, *the line given* below gets inserted. It also affects the design time look of the DataGrid by displaying a plus sign.

```
Me.DataGrid1.DataSource = Me.DataSet11
```

Each time a property is altered, an additional line of code gets added to the Code Painter. Thus, the DataSource property gets set to DataSet11.

Our view-point is that, every software developer should use Visual Studio.Net, since it is sure to make each one of them more efficient programmers.

The different user interfaces of the Properties editor also ensure that, on most occasions, the user does not have to enter a value. This is achieved by providing drop down listboxes encompassing the various possible values, thereby eliminating the slightest probability of the user committing a blunder.

Yet another advantage is that, very often, programming tends to become very tedious and irksome. Therefore, it is a dream come true when the framework generates the code for us, even if it does so only once in a while.

Next, the DataMember property is set to Customers. This does not add any fresh line of code, but merely changes the previously written line to the following:

```
Me.DataGrid1.DataMember = "Customers"
```

We would definitely want you to attempt a small experiment. In the Design Mode, Set the DataSource property to None. On doing so, the DataMember property too gets reset to None, thus echoing the above action.

Visual Studio.Net transmutes itself into a feature of convenience, whereby, the dynamic behaviour gets reflected by the code in the control itself, such as a DataGrid control. Next, we insert the button control. This leads to the addition of the following code in the file:

```
Friend WithEvents Button1 As System.Windows.Forms.Button
Me.Button1 = New System.Windows.Forms.Button()
Me.Button1.Location = New System.Drawing.Point(96, 184)
Me.Button1.Name = "Button1"
Me.Button1.TabIndex = 1
Me.Button1.Text = "Button1"
```

As can be seen above, the outcome is very predictable when the code is generated by a computer program. An instance variable Button1 is created, which can also handle events.

In the InitializeComponent function, the instance variable is actually instantiated, and the properties of Location, Name, TabIndex and Text properties are set. The TabIndex property is set to 1, since it is the second control that is being added.

```
Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.Button1,
Me.DataGrid1})
```

The utility of the AddRange function is demonstrated beyond doubt, when controls get added to the Form. Note, that by using an array, the Button and the DataGrid are added concurrently.

When the Text property of the button is changed to 'Load', the following line gets added:

```
Me.Button1.Text = "Load"
```

We have decided to refrain from explaining the same code repetitively. The above can be achieved by double clicking on the button and writing the code that is shown above. We are nearing a state of considerable ease while working with Visual Studio.Net, since we are now in a position to unravel as to what is happening internally.

The next application that we have chosen to build is that of a Master-Detail, or a Parent-Child, or a One-To-Many relationship, using Visual Studio.Net. Firstly, we intend to display a list of Customers in our data grid, and then, we wish to display the orders placed by each customer.

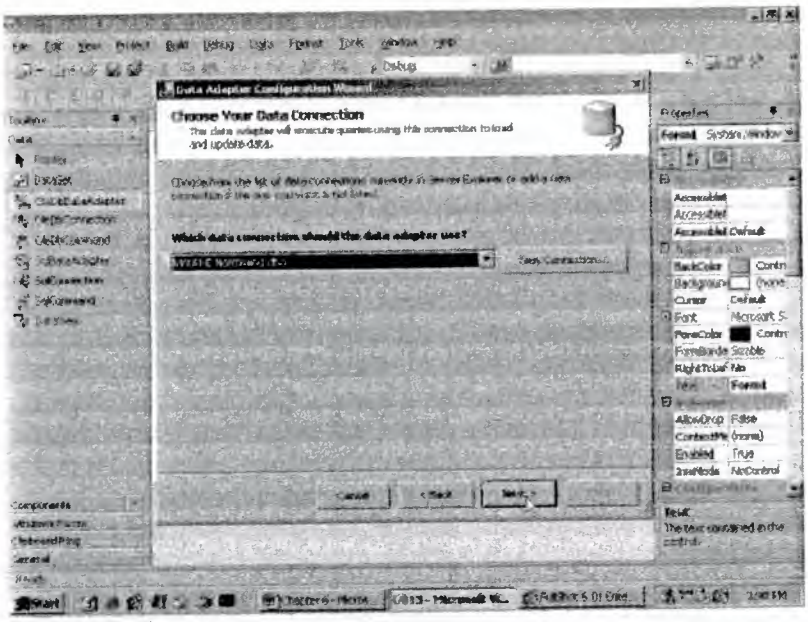
So, watch out! Here we go!

Click on the menu-option, File-New-Project. Now, you shall arrive at the New Project dialog box. As usual, in the first pane, select the option of Visual Basic projects, and in the second pane, select the option of Windows Applications. The name assigned to this project is t3, and the project shall dwell in the directory v1. Then, click on the OK button.

The New Projects dialog box retains information about the projects that were last worked upon. Hence, on many occasions, the previous values remain selected.

Thereafter, we select the DataAdapter control after clicking on the Data Tab in the ToolBox. The wizard begins in the same way as before, i.e. by asking for information that it requires to generate the code.

The screen 2.39 depicts the previous connection that we had made to the Northwind database.



Screen 2.39

And if it does not do so, click on the drop down listbox, which will display a list of connections that have been created previously. So far, we have created only a single connection, thereby rendering this list inconsequential as of now.

Ensure that the Northwind connection is selected, and then, click on the Next button. In the screen for data retrieval, the default option of 'SQL statement' is left selected, since we are not yet well versed in working with 'stored procedures'. Now, we click on the 'Next' button to arrive at the screen where we have to enter the SQL statements. We could have used the Query Builder again, but this time around, we would rather enter the following SQL statement manually:

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address,
City FROM Customers
```

Click on Finish to create the Connection and Adapter objects, which represent the connection to the Customers table.

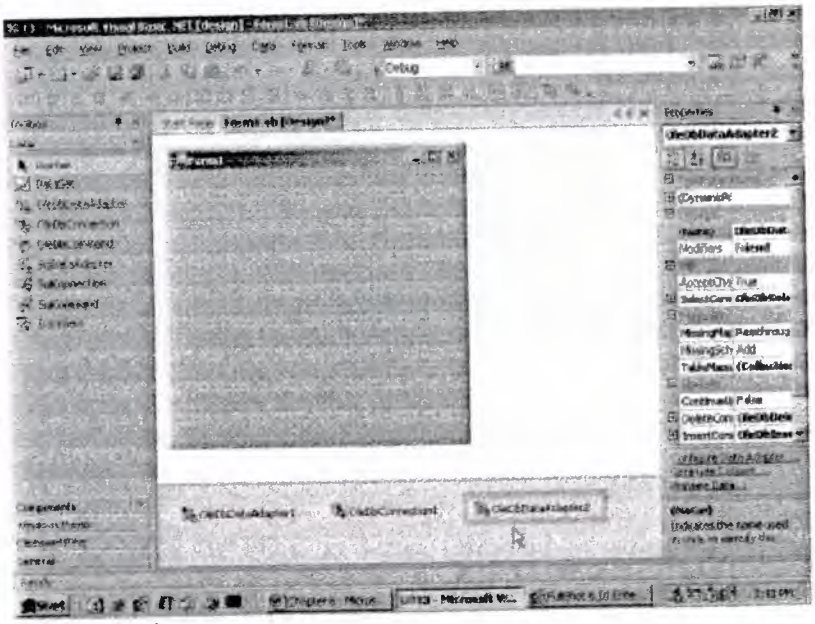
Now, we have to ferret data from the Orders tables. The same process is repeated, using which, one more OleDbDataAdapter object is dragged onto the Form. The connection is once again established with the Northwind database, as it contains both the tables of Customers and Orders.

In the next screen, the data access method option of 'SQL Statement' is left selected. When we click on the Next button, we are asked for the SQL statement. Here, we manually enter the following SQL statement:

```
SELECT OrderID, CustomerID, EmployeeID, OrderDate, ShipCountry,
ShipRegion, ShipCity FROM Orders
```

To proceed further, click on the Finish button.

From the Orders table, we have randomly chosen some fields, keeping in mind that the CustomerID field is of primary importance, since it is the only link between the two tables of Customers and Orders. It is not mandatory for the Primary key and Foreign key to possess the same name. However, both of them must be specified in the list of fields of the Select statement. The outcome of this activity is the creation of a single control named OleDbDataAdapter2. This is shown in screen 2.40.

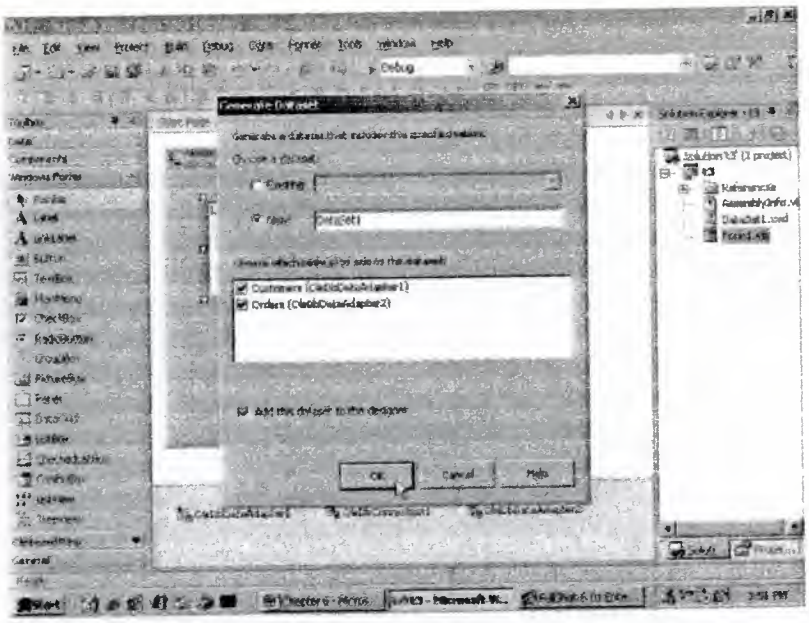


Screen 2.40

The framework is extremely smart. It realizes that an attempt is being made to re-establish a connection to the Northwind database. Therefore, it reuses the existing connection, instead of creating a new one.

The Data is to be displayed in a control. So, at this juncture, instead of using the DataGrid, a listbox from the Windows Form tab in the toolbox is used.

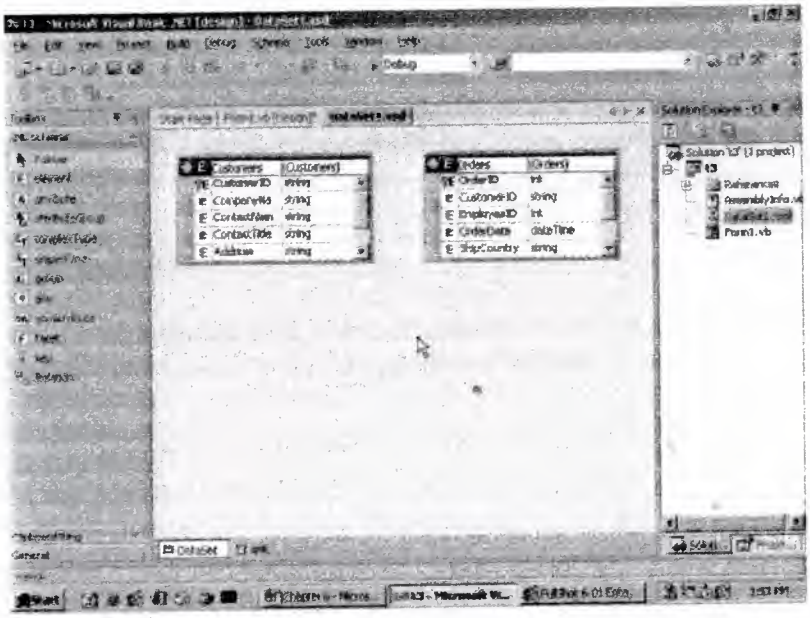
Bring in the listbox control, and then, click on Generate DataSet in the Data menu. The dialog box that gets displayed, has a dataset name that we are quite content with, but the Customers table is not shown as selected.



Screen 2.41

Select Customers, as shown in screen 2.41, and then, click on OK. This results in the addition of one more control named DataSet11, to the list of three invisible controls.

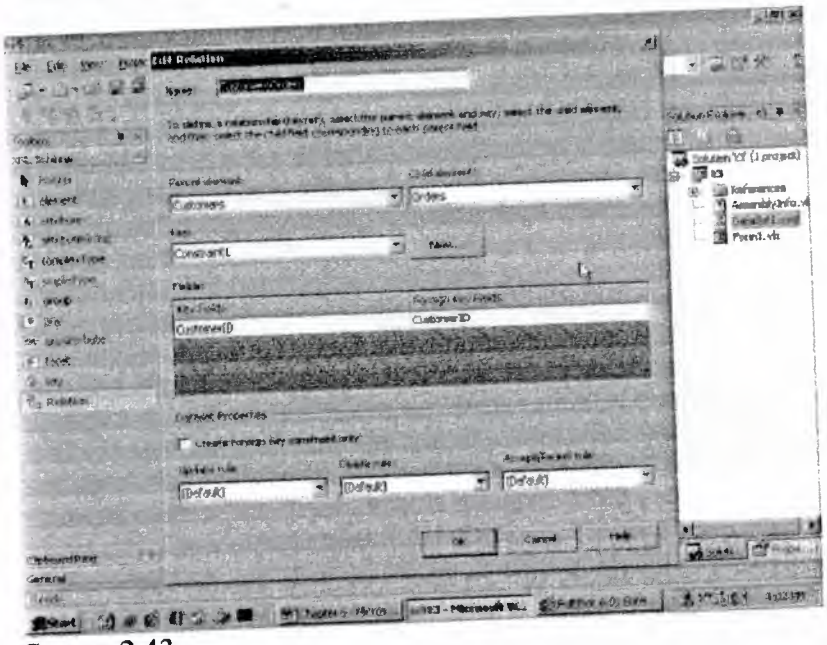
Next, we click on View, followed by Solution Explorer, to activate the window. Then, double click on the file named dataset1.xsd. This activates the XML Designer, as shown in screen 2.42.



Screen 2.42

Here, we come across two tables that constitute our dataset, viz. Customers and Orders. Each table must necessarily have a Primary key, which is normally a single field. Thus, we see two keys listed in front of the fields, which are the primary keys. The second column indicates the data types of the fields.

The toolbox has only one tab named XML Schemas, from where the last control named Relation is introduced into the Form. The control is dropped onto the Orders tables, since it is the Child table. This brings up the Edit Relation dialog box.



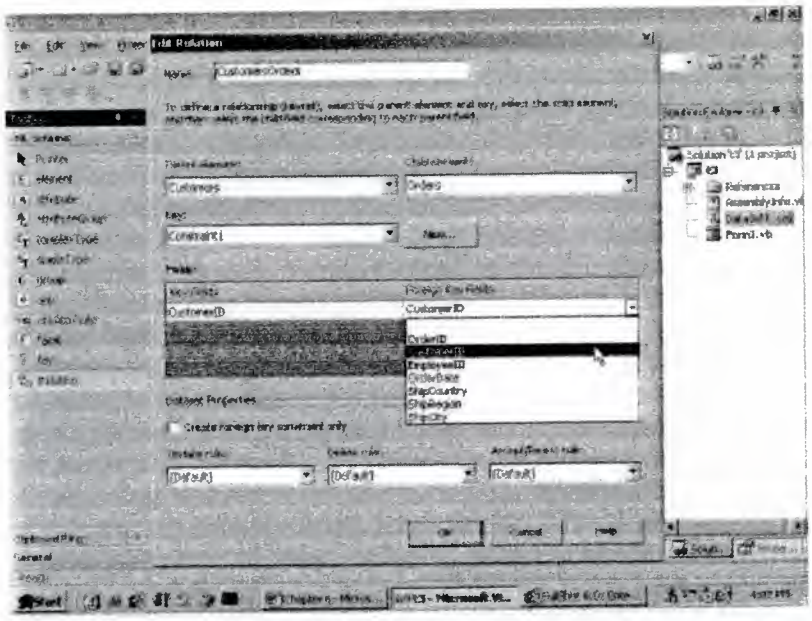
Screen 2.43

The first textbox discloses the name of the relation i.e. CustomersOrders, which would be displayed in the DataGrid. Then, there exist two listboxes, which when clicked upon, exhibit a list of tables in the dataset of Customers and Orders, respectively.

The Parent table is the Customers table, since it contains unique values for the CustomerID, and the child table is Orders, which has the Foreign key, encompassing multiple values for the CustomerID. When we drop the relation object on the child, it picks up most of these values and sets them as defaults. Hence, we do not have to enter them ourselves.

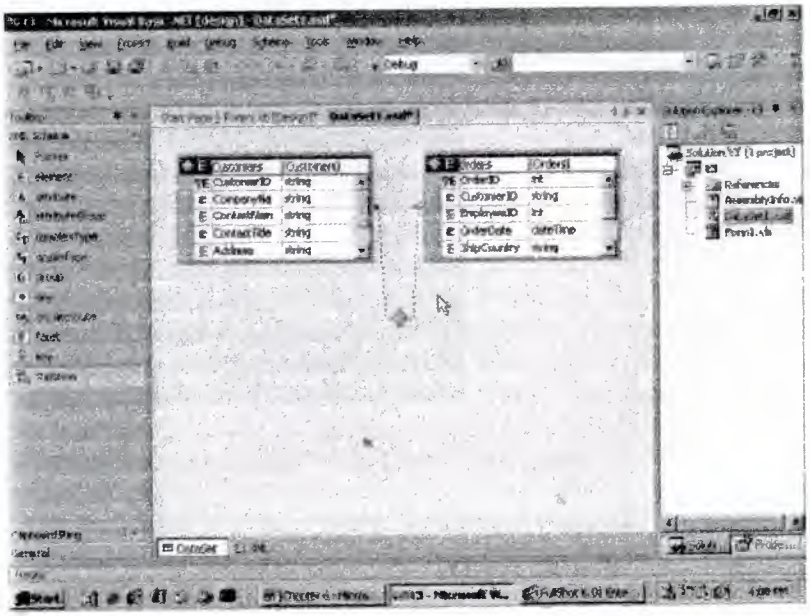
The dialog box is smart enough to select CustomerID as the Parent key, but it expects us to select the Foreign key ourselves.

Therefore, we click on the down arrow, as in screen 2.44, and select the CustomerID column as the Foreign key, and then, we click on OK.



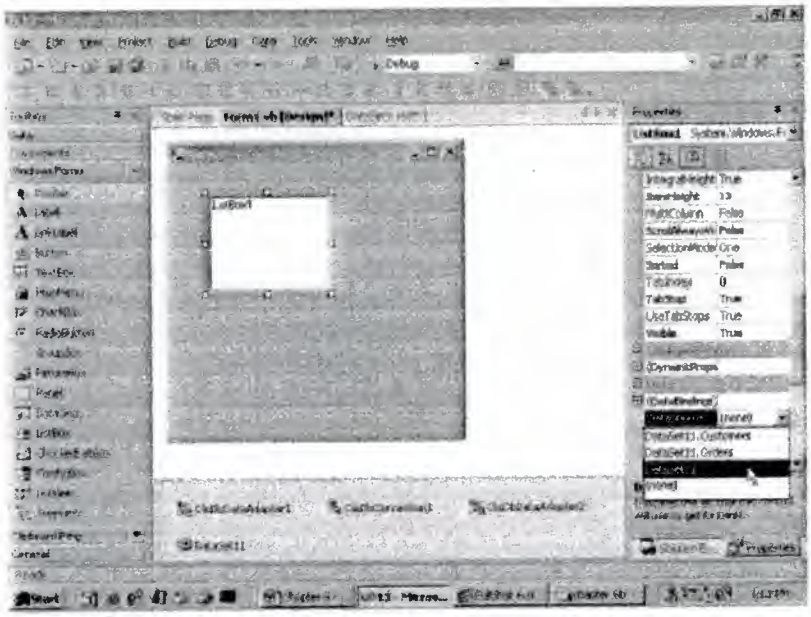
Screen 2.44

The visual representation of this relationship is depicted in screen 2.45. The Customer table has one unique id; and hence, it has a single arrow, whereas the Orders table has many occurrences of the ids. Hence, CustomerID has multiple arrows leading to it.



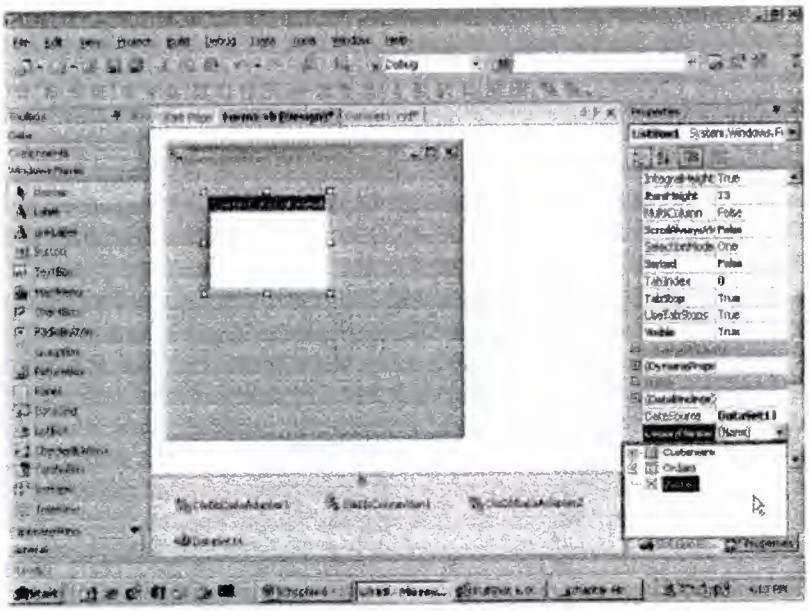
Screen 2.45

Once the relation is set, click on the tab Form1.vb. Then, Drag and Drop a listbox control from the ToolBox on the form, and move on to the property of DataSource, as shown in screen 2.46. Set the value of the DataSource property to DataSet11.



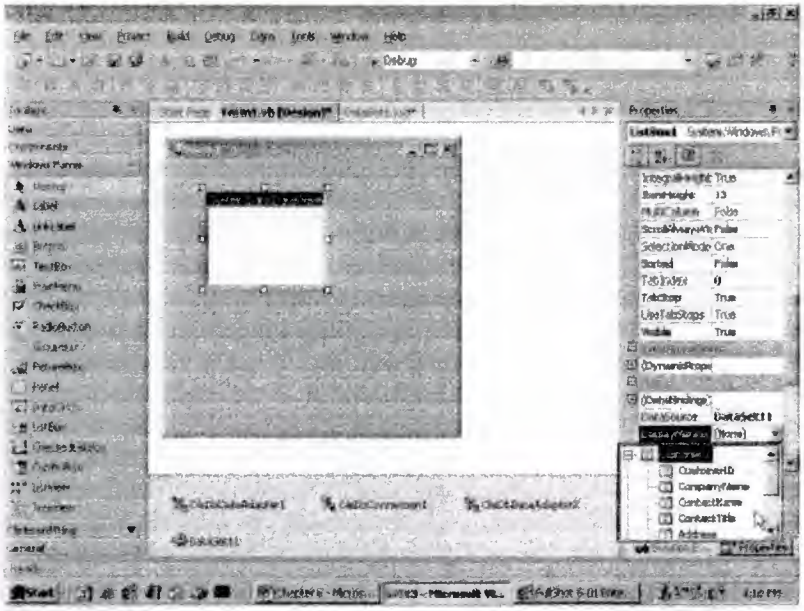
Screen 2.46

The next property of DataMember shows a drop-down listbox, which when clicked on, displays the two tables of Customers and Orders. This is shown in screen 2.47.



Screen 2.47

This occurs because the DataSource property is initialized to the dataset named DataSet11. The two tables named Customers and Orders, which comprise the dataset object, are visible.

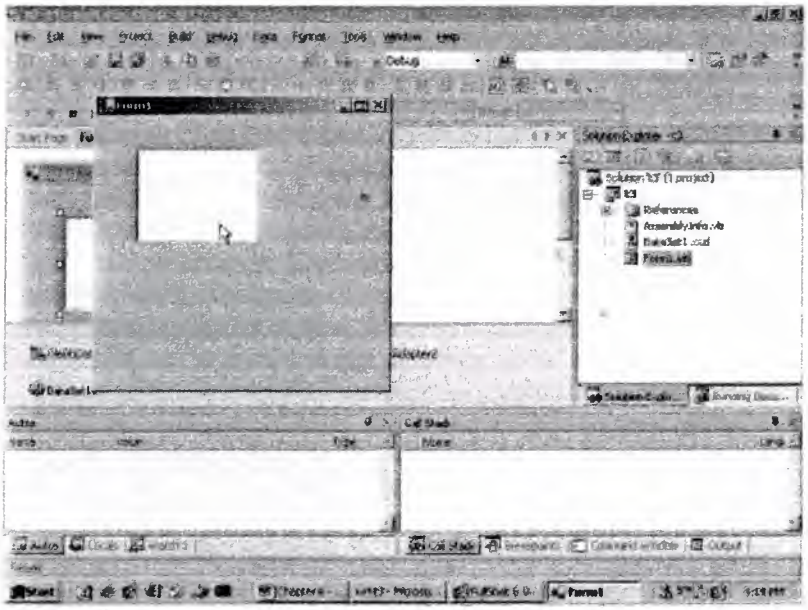


Screen 2.48

As we want to select the field of customer name, we click on the plus sign of the Customers table. This leads to a display of all the fields specified in the Select statement.

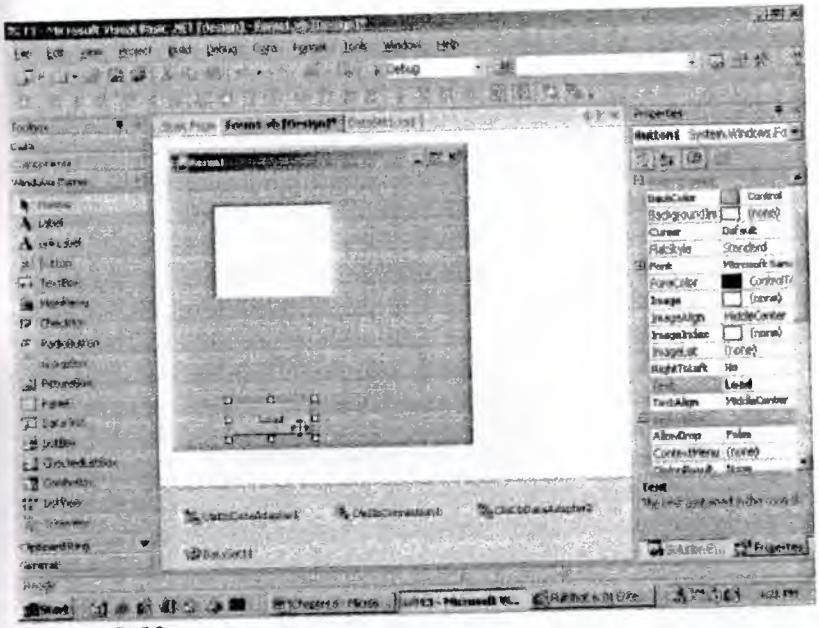
The company name is a lot more intuitive than the customer id. So, the value of the property is initialized to Customers.CompanyName, i.e. the name of the table, followed by a dot, followed by the name of the field.

In order to authenticate the list, press the F5 key. This compiles and runs the project. The output is extremely disappointing, since all that we see is an empty listbox, as shown in screen 2.49.



Screen 2.49

Close both, the Form window and the window labeled as Output. In the designer mode, drag and drop a button from the ToolBox, and change its label to 'Load', as shown in screen 2.50.



Screen 2.50

Double click on the button, and enter the following code in the event handling function:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
OleDbDataAdapter1.Fill(DataSet11)
End Sub
```

Here, as before, we call the Fill function of the OleDbDataAdapter object from the Customers table, since presently, we are not interested in the data contained in the Orders table.

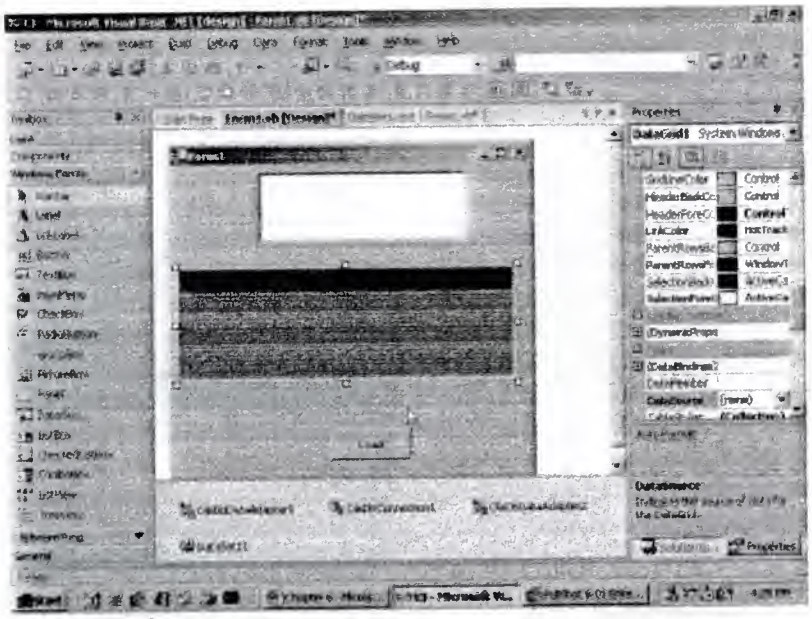
Press F5 to run the application, and then, click on the button labeled 'Load'. This results in the listbox getting filled with data, as shown in the screen 2.51.

Screen 2.51

The Fill function is responsible for filling up the empty dataset, which is eventually displayed by the control. This is because, the property in the control is set to

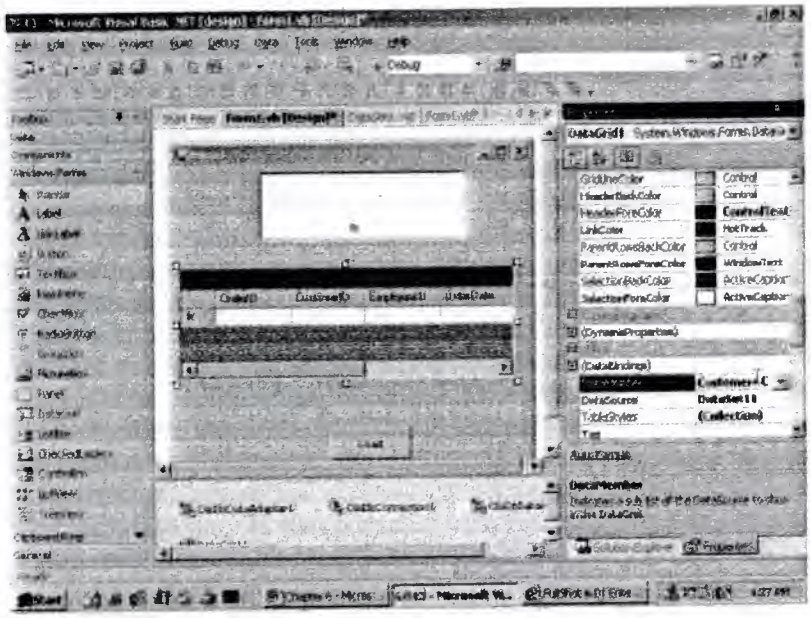
this dataset. At this point in time, the data from the Customers table is on display, but there is no sign of data from the Orders table.

Now, to retrieve data from the Orders table, close the windows that have popped up while running the program, and drag and drop a DataGrid onto the Form. Change the layout of the control, as shown in screen 2.52.



Screen 2.52

The DataSource property in the DataGrid is set to DataSet11. For the DataMember field, the property is set to the name of the Relation object and the table name Customers.CustomersOrders.

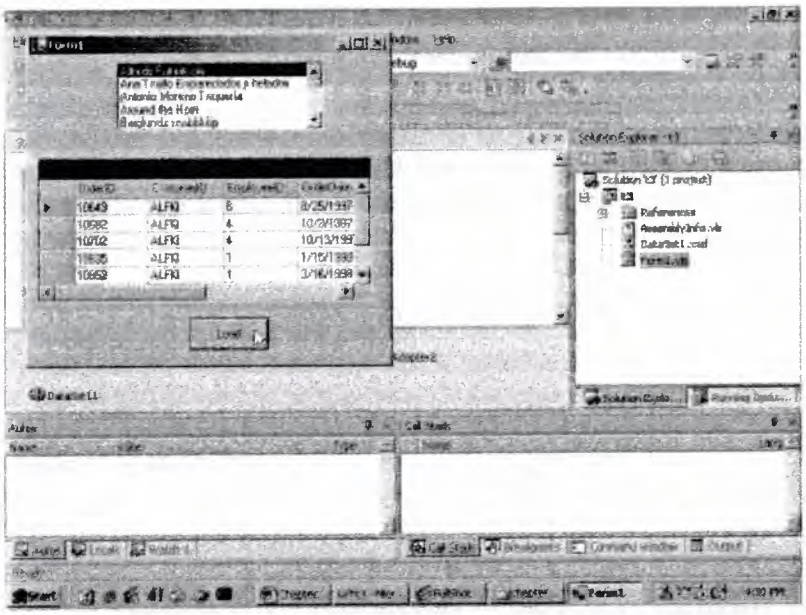


Screen 2.53

Moreover, one line of code is inserted in the event handling code of the button.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
OleDbDataAdapter1.Fill(DataSet11)
OleDbDataAdapter2.Fill(DataSet11)
End Sub
```

Now, run the above program by pressing the F5 key. When the Form loads on, click on Load. The screen that appears, looks like the screen 2.54, wherein a list of customer names is displayed in the listbox, followed by the list of orders in the Orders tables.



Screen 2.54

Now, each time that a new customer is selected, the list of orders of that customer is displayed in the DataGrid. This results from the two changes that we have effected: Firstly, we had filled up the Orders table using the Fill function. Secondly, we had used the name of the relation as a DataMember name.

The DataGrid, on learning that the name specified is not a table name, uses the relation object to determine the records that need to be displayed.

Also, while clicking on a new customer name in the listbox, the DataGrid is apprised about the new Customer that has been selected. Hence, it alters the records displayed in the DataGrid, depending upon the CustomerID in the Orders table.

```
Me.DataGrid1.DataMember = "customers.customersorders"
```

When the DataMember in the datagrid is set to the relation object, the control field merely adds one line, to initialize the property to the value. No other code gets inserted. Furthermore, no DataRelation object gets created in the Form code. It is found in the file DataSet1.vb.

The above example appears to be too simplistic. So let us spice things up by adding a little more complexity. The Orders table summarizes the orders as per the customer id. Moreover, there exists a table named Order Details, which consists of the list of orders per order id.

What we wish to accomplish at this juncture is, to prepare a list of orders as per the order id, when a certain order is clicked on. So, to begin with, we need an SQL statement to fetch records from the Order Details table.

So, from the Data category in the toolbox, the control named OleDbDataAdapter is again inducted into the Form. Then, we click on the Next button thrice:

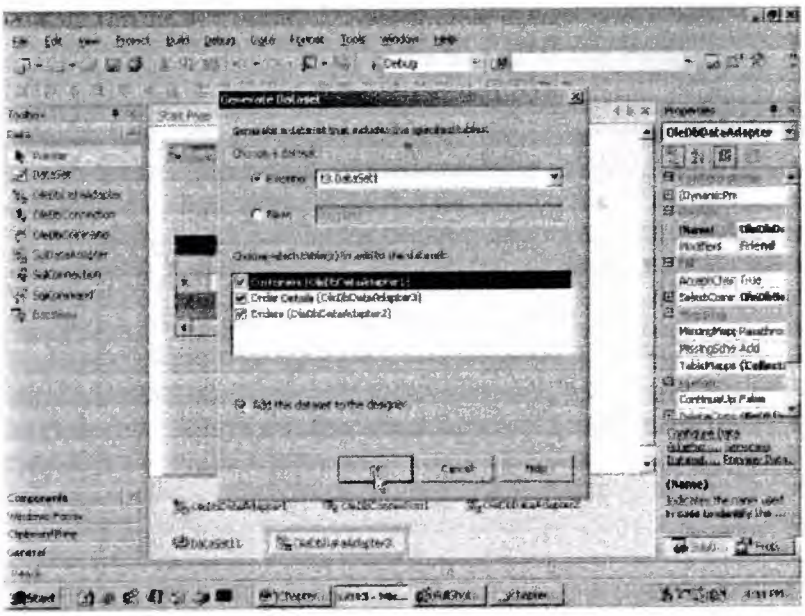
- Firstly, on the opening screen.
- Secondly, on the Connection screen, since the Northwind connection is chosen by default.
- Thirdly, on the screen in which the default of SQL Statement has been chosen.

In the text block, we now have to write the SQL Select statement, which fetches the records for the specified fields from the Order Details table. A point to be noted is that, whenever the name of a table contains a space, the table name must be placed within square brackets.

```
SELECT OrderID, ProductID, UnitPrice, Quantity, Discount FROM [Order
Details]
```

When we click on Finish, one more OleDbDataAdapter object gets added at the bottom of the Form.
The DataSet must be regenerated, for which the Generate option is selected from the Data menu option.

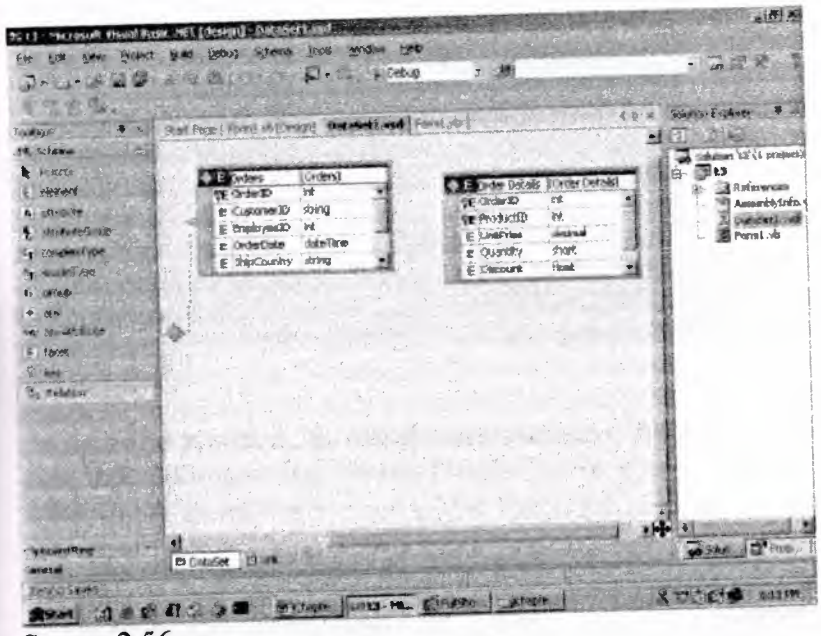
The dialog box that appears contains only the newly added table as selected. But, we need to select all the three tables, as seen in screen 2.55.



Screen 2.55

Finally, click on OK. In case a message box pops up, click on 'Yes to All'. Now, make sure that the Solution explorer is visible.

If it is not, then click on the menu View-Solution Explorer, and then, double click on the DataSet1.xsd file. As we invariably fall short of screen space, we recommend that you scroll to the right to arrive at screen 2.56.



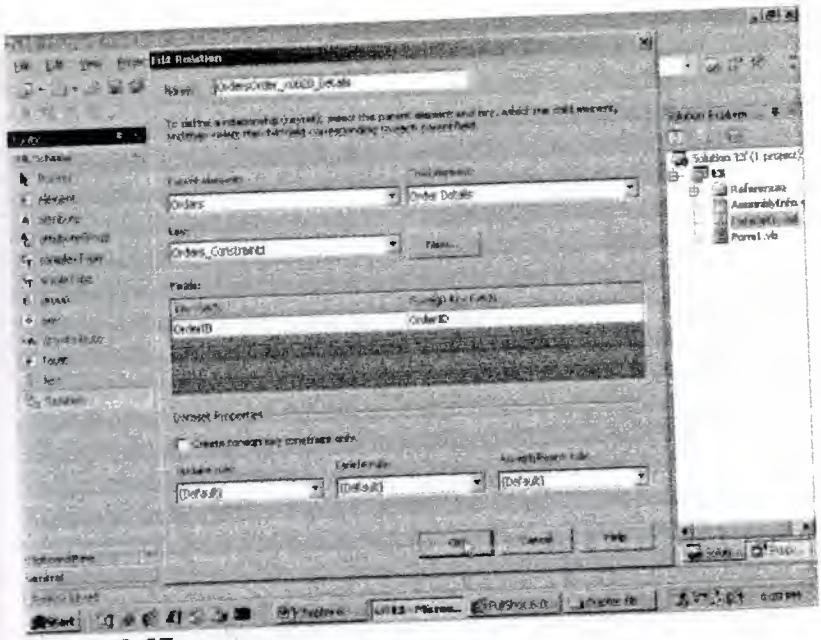
Screen 2.56

Here, we come across the third table named Order Details. After the table comes into sight, drop the Relation object onto the Order Details table. Remember that a single order id in the Orders table will have multiple records of the order id in the Order Details table.

In the Edit Relation dialog box, the name of the relation is shown as CustomersOrder_x0020_Details. We will not amend it, since the framework automatically changes it when the names of the Parent table and the Child table change. If you change the name of the parent table from Customers to Orders, the name of the relation will change to OrdersOrder_x0020_Details. If this does not happen, we request you to amend it manually to OrderssOrder_x0020_Details.

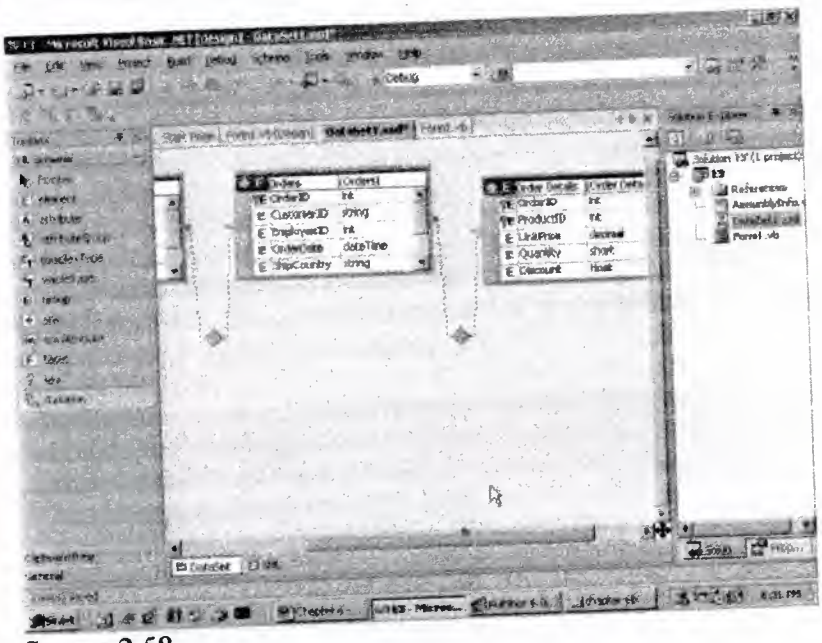
For each order record found in the Orders table, there exists a list of order details in the Order Details table. Therefore, the child element is assigned to Order Details. However, when you click on the down arrow of the dropdown listbox, three tables will be displayed. This is because the DataSet now comprises of three tables. Ensure that the child element contains the table Order Details. In the Fields block, the Parent key field is OrderID. Therefore, the child key field that is currently empty, is also changed to OrderID.

Make sure that your screen appears similar to what is seen in screen 2.57, and then, click on OK.



Screen 2.57

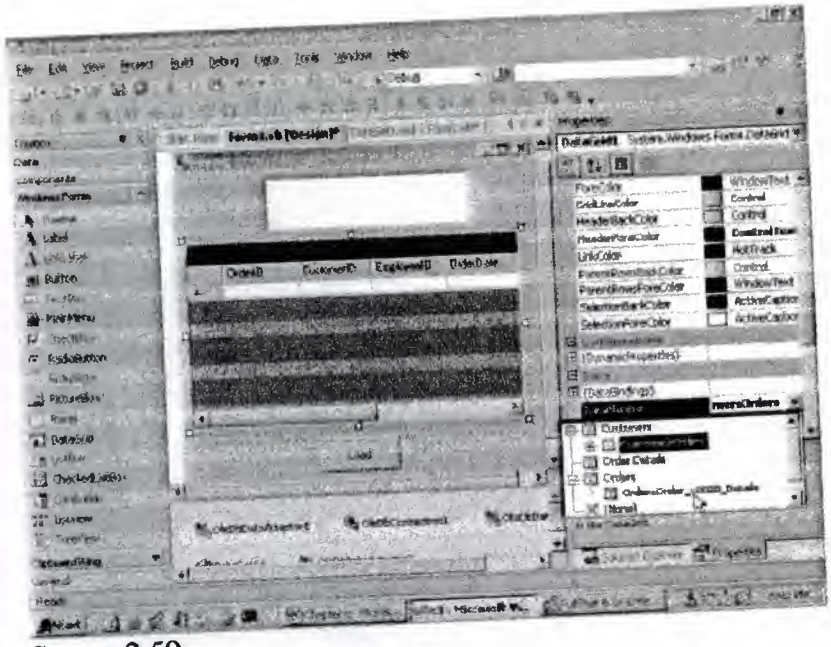
The screen 2.58, which comes up next, displays a 'one to many relation' between the tables Orders and Order Details. It is always advisable to periodically refresh everything, as well as to save all the work. So, click on the menu option File-Save All.



Screen 2.58

In order to verify the fact that the relation object has been inserted, click on the property called DataMember. The drop down listbox displays a plus sign next to the Orders.

Click on the plus sign to see the relation object as shown in screen 2.59.Thus, the DataMember exhibits a total of two relations.

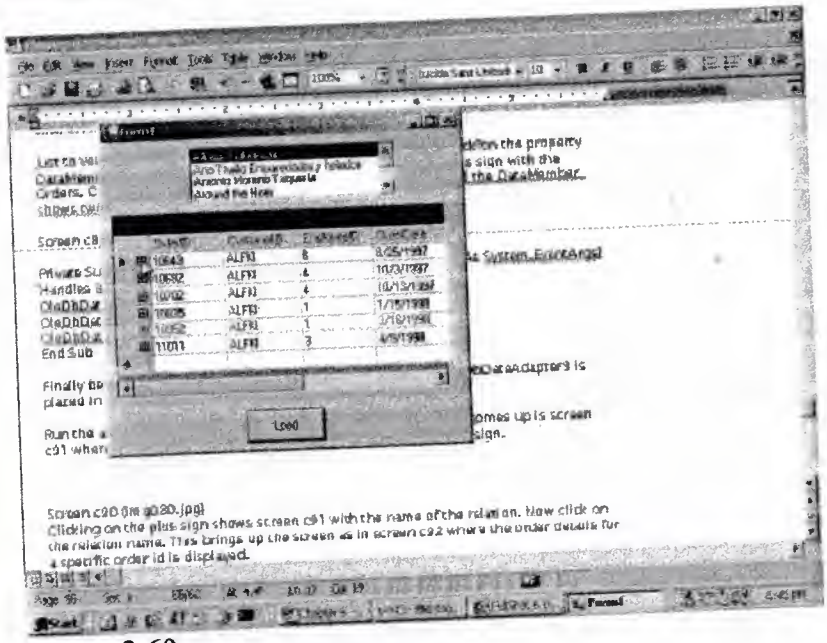


Screen 2.59

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
OleDbDataAdapter1.Fill(DataSet11)
OleDbDataAdapter2.Fill(DataSet11)
OleDbDataAdapter3.Fill(DataSet11)
End Sub
```

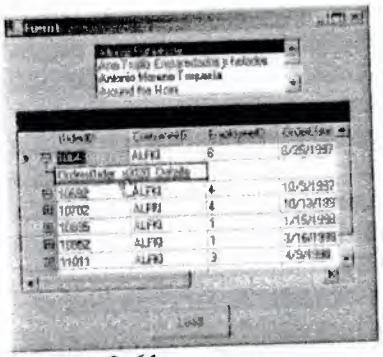
Finally, before the project is executed, the Fill function of the OleDbDataAdapter3 is placed in the Load button.

Run the application and click on the button. The screen that comes up is shown in screen 5.68. Here, we witness a list of Orders in the DataGrid, with each of them having a plus sign placed next to it.

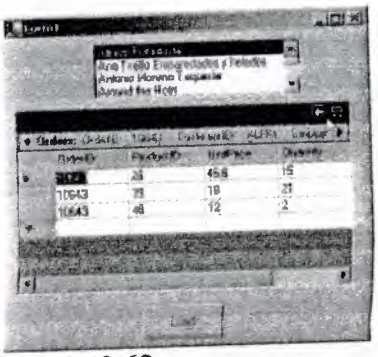


Screen 2.60

If you click on the plus sign, the screen 2.61 containing the name of the relation is seen. Now, click on the relation name. This brings up the screen shown in screen 2.62, where the order details for a specific order id are displayed.



Screen 2.61



Screen 2.62

Now, we present one more database application. In this example, we wish to display one record at a time.

Click on the menu File-New-Project, and in the New Project dialog box, select the option of Visual Basic projects in the first pane, and the option of Windows Application in the second pane. Name the project as t4, and then, click on OK.

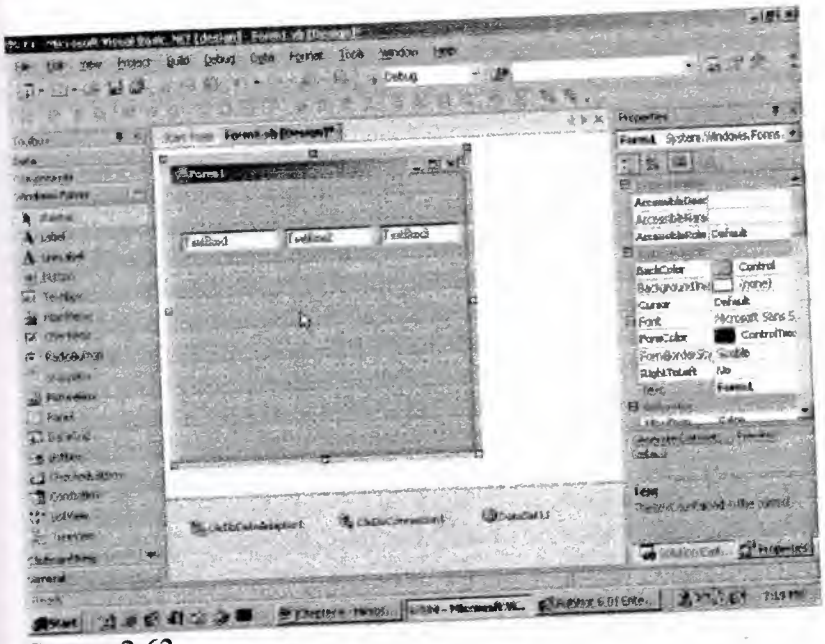
Usher in the OleDbDataAdapter object from the Data tab in the toolbox. This object is brought in to enable us to insert an SQL select statement, which would retrieve data from the database. Click on the Next button on the first three screens, and in the SQL statement textbox, enter the following Select statement:

SELECT CompanyName, ContactName, ContactTitle FROM Customers

Since we are running short of time and space, we shall display only three fields from the Customers table. On clicking the Finish button, the two objects of Adapter and Connection object get created. Then, we generate the DataSet object, by selecting the menu option of Generate DataSet from the Data menu. All the default settings are acceptable to us. Therefore, we click on OK, resulting in the creation of a DataSet called DataSet1.

In this application, we want to display the data that has been retrieved from the database in textboxes. So, three textboxes from the toolbox are dragged onto the Form.

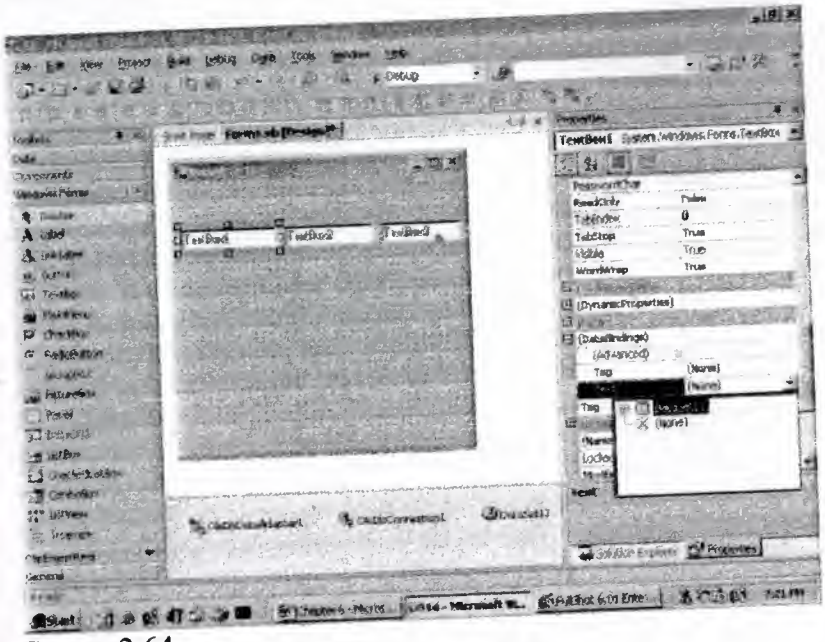
This is shown in screen 2.63. These three textboxes are named as TextBox1, TextBox2 and TextBox3.



Screen 2.63

Now, make sure that the first textbox is selected. Scroll down the Properties window, until the Data section property come into sight.

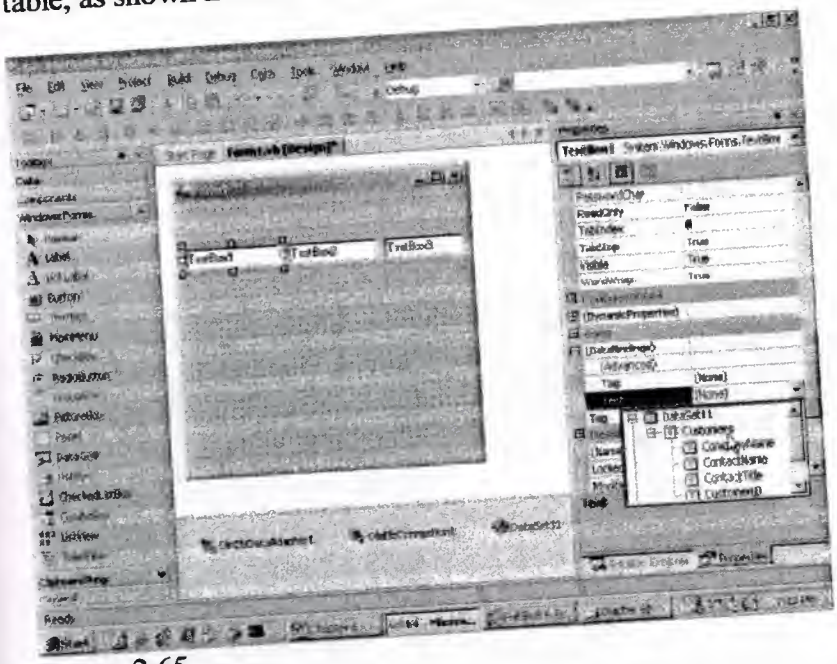
Then, click on the plus sign of DataBindings, as shown in screen 2.64. The main purpose of this action is to set the Text property to a field in the database. The Text property of the textbox has a drop-down listbox, which when clicked on, shows DataSet11, which is the lone dataset created in the project.



Screen 2.64

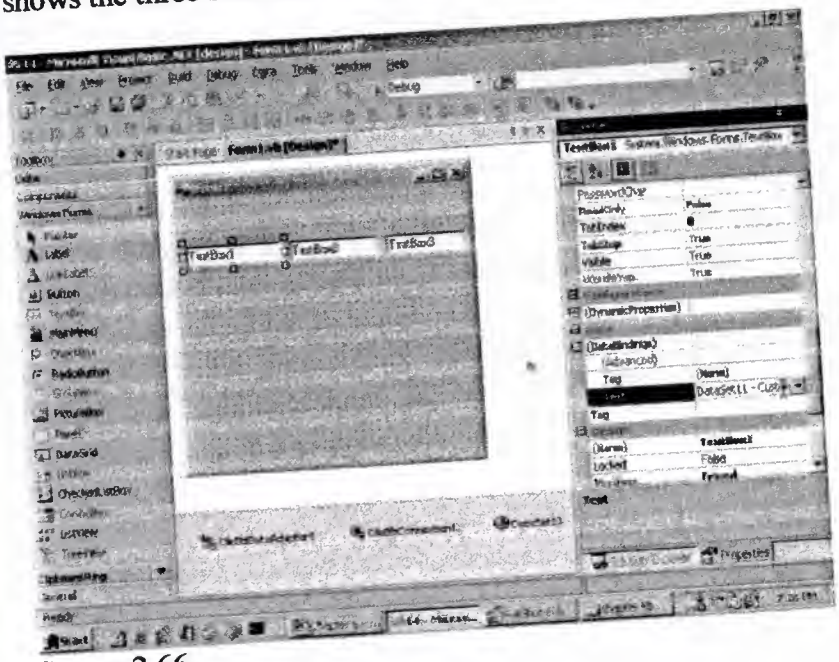
A point to be considered here is that, we have not added a DataSet to our textbox. In spite of this, the TextBox is astute enough to display all the DataSet objects that have been created in the project.

Clicking on the plus sign of the DataSet object, will bring up a list of tables that comprise the DataSet. Our listing reveals only one table named Customers. Then, we click on the plus sign in front of Customers, to arrive at a list of fields that make up the table, as shown in screen 2.65.



Screen 2.65

Not all the fields from the table are visible, because the SQL Select statement shows the three selected fields along with the Primary key.



Screen 2.66

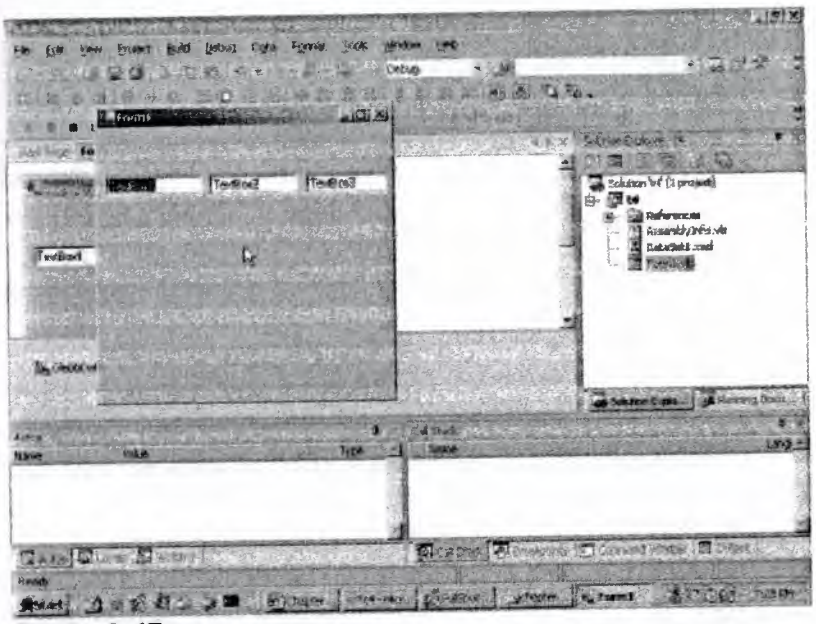
Select the field of Company name. Doing so will change the value of the Text Property to DataSet11 - Customers.CompanyName. The syntax for the value is as follows: The name of the DataSet, followed by table name, and finally, followed by the field name. Thus, a field from the table present in a DataSet is now bound to a textbox.

A similar approach is followed for the second textbox named TextBox2.

Here, we click on the textbox TextBox2 and scroll down to DataBindings. From the Drop down list, which is obtained after clicking on the plus sign of DataSet11 and the Customers table, we select the third field named ContactTitle.

For the third textbox, the above procedure is repeated, and the field named CustomerID is selected.

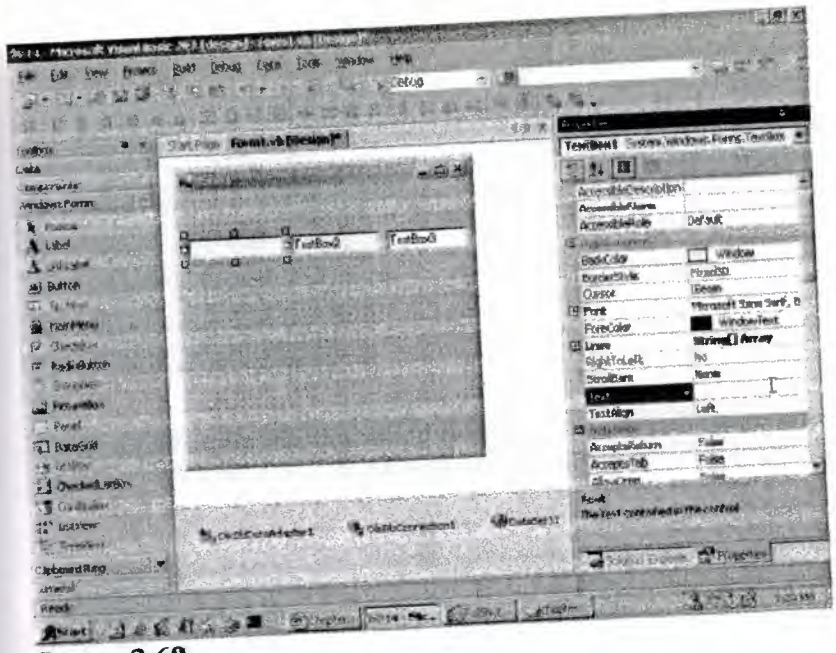
Press F5 to run the program. The result of our actions is shown in screen 2.67, where some default text has already been entered in the textboxes.



Screen 2.67

Since this is evidently distinct from what we had anticipated, we close the running application and effect the requisite amendments.

Select the first TextBox and scroll down to the Text property. Then, delete the value assigned to the Text property, as shown in screen 2.68.



Screen 2.68

Now, press Enter, and as an outcome of this, the textbox in the Form does not show any text at all. The same process is reiterated for the other two textboxes.

If you have been sufficiently attentive and assiduous, you would have realized that earlier, the Text property did not contain a database icon, but the moment the Data Binding Text property is set to a field, the Text property obtains an icon, indicating the fact that the data property will overwrite this Text property.

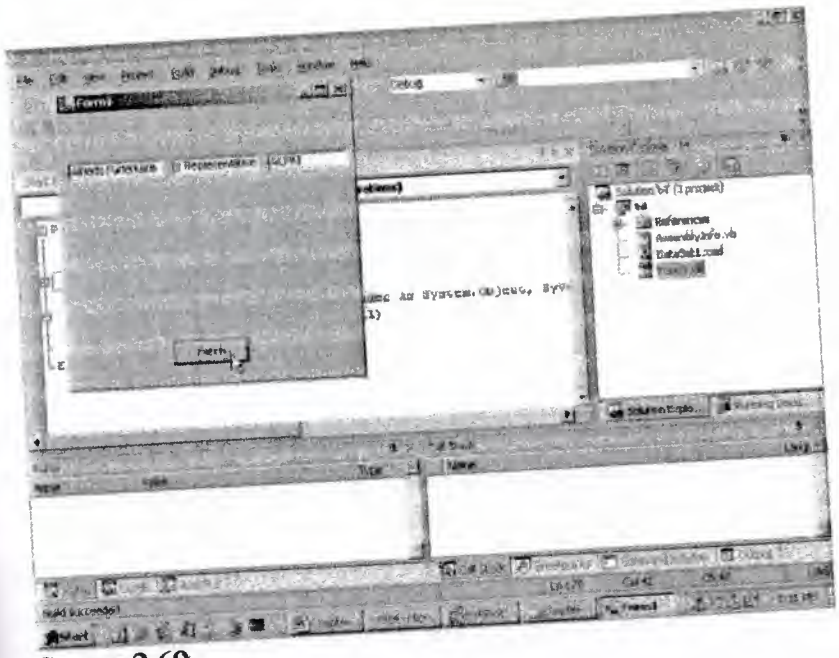
The designers of the TextBox control should have added one more feature in that, they should have ensured that the Text property blanks out as soon as the field is set to a Data Text property. The main reason that the textboxes do not display any data from the table is that, the Customers table in the DataSet11 is empty. The Fill function has not been introduced so far, and hence, it has not been executed.

Now, we introduce a Button by selecting it in the toolbox, and changing the Text to 'Fetch'. Then, after double clicking on the button, the following code is introduced in the mouse event handler, using the code painter:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
OleDbDataAdapter1.Fill(DataSet11)
End Sub
```

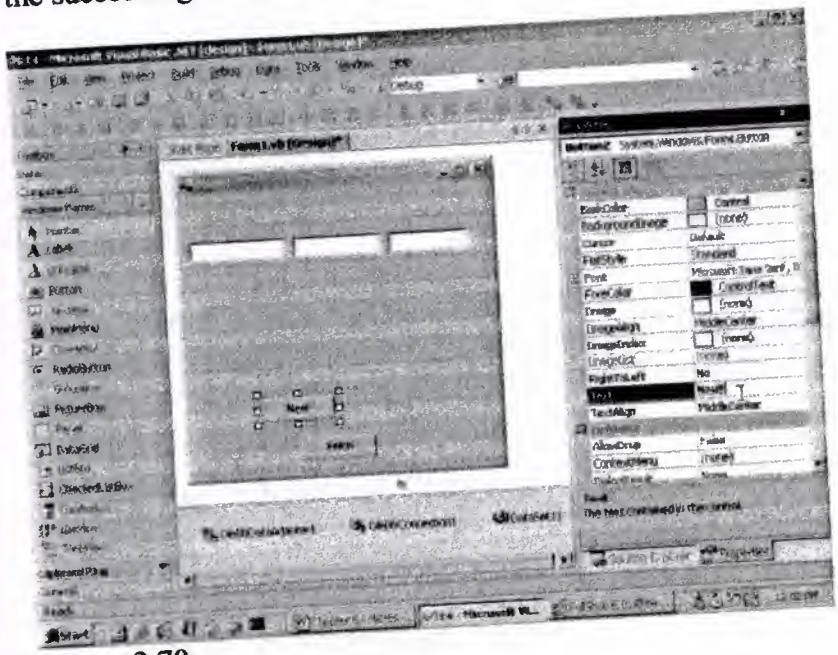
Now, when the above program is executed by pressing the F5 key, the screen does not display any text in the textboxes.

When the button is clicked, the window shows the first record, as seen in screen 2.69.



Screen 2.69

It gladdens the heart to see the record values in the textboxes. However, this is not enough, since we intend to view the subsequent records also. So, close the windows of the running application and add one more button, which shall facilitate scrolling to the succeeding records. Change its label to 'Next', as shown in screen 2.70.



Screen 2.70

Double click on this button to add the following code to the event-handling sub of Button2_Click.

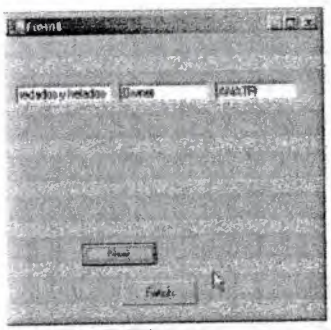
```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
Dim b As BindingContext
b = BindingContext
Dim bb As BindingManagerBase
bb = b(DataSet11, "customers")
```

```
bb.Position += 1
End Sub
```

The class Form has a property called BindingContext, which returns a BindingContext object. This object is stored in b. The task of a BindingContext object is to attend to all the BindingManagerBase objects. So, by using b, which is a BindingContext containing the default property or indexer of the DataSet, and using the table, the BindingManagerBase object can easily be retrieved.

Every DataSet and table in the DataSet combination has its own BindingManagerBase object. This BindingManagerBase object takes care of all the Binding objects. The BindingManagerBase object has a property called Position, which is of type read-write. It reveals the current active record in the Data Table. A new value can be assigned to it, in order to make the record active.

Now, Run the project by pressing the F5 key. Then, click on the Fetch button. The first record will get displayed. On clicking the Next button, the second record will get displayed, as shown in screen 2.71. Thus, it is the Position property that establishes the active record.

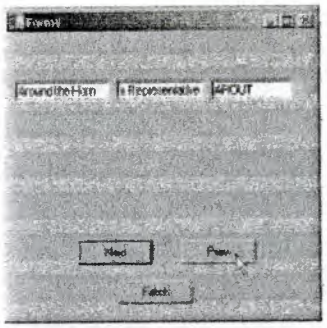


Screen 2.71

Next, we intend to move backwards amidst the records. So, we repeat the same procedure, wherein a button is brought into the Form from the toolbox and its label is changed to 'Prev'. Then, the following lines of code are entered in the event handler function:

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
Dim b As BindingContext
b = BindingContext
Dim bb As BindingManagerBase
bb = b(DataSet11, "customers")
bb.Position = bb.Position - 1
End Sub
```

After pressing the F5 key, we first click on Fetch, followed by the Next button, which takes us to the next record. Then, we click on the Prev button, and we are transported back to the previous record. This is shown in screen 2.72. Thus, we have been able to scroll through a series of records.



Screen 2.72

We would now like to implement the following:

- Move to the first or last record, depending on the button that is clicked.
- Display the current record number and the total number of records in the table.

To achieve this, we do the following: One more button is introduced in the Form, and its label is changed to 'First'. This button is internally assigned the name of 'button4', and the following code is inserted in it:

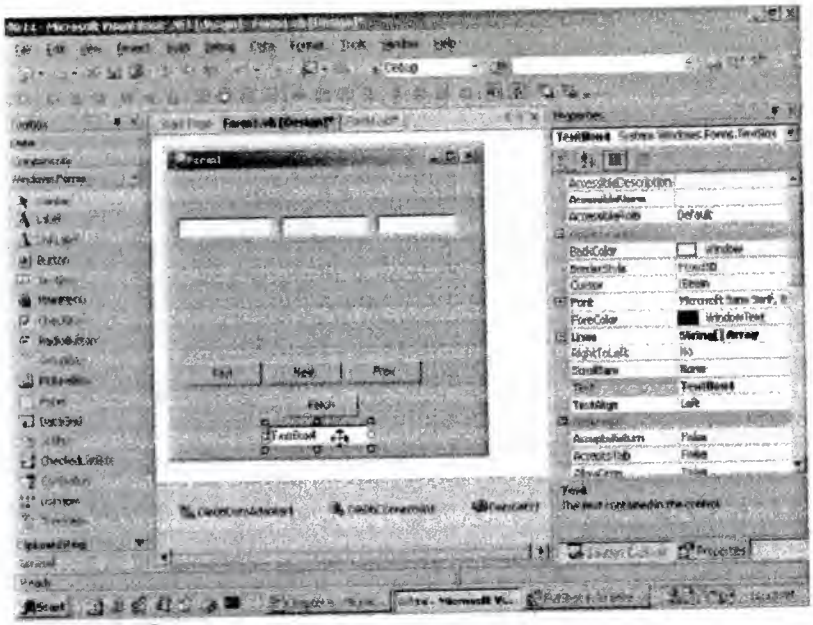
```
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
Dim b As BindingContext
b = BindingContext
Dim bb As BindingManagerBase
bb = b(DataSet11, "customers")
bb.Position = 0
End Sub
```

The only statement that has been modified in the code above is that of the Position property, Which is set to 0, and not to 1, since the counting starts from 0.

Press F5 to run the application. First, click on the 'Fetch' button. Then, click on the 'Next' button twice, to move to the third record. Now, click on the button labeled 'First'. This brings us back to the first record.

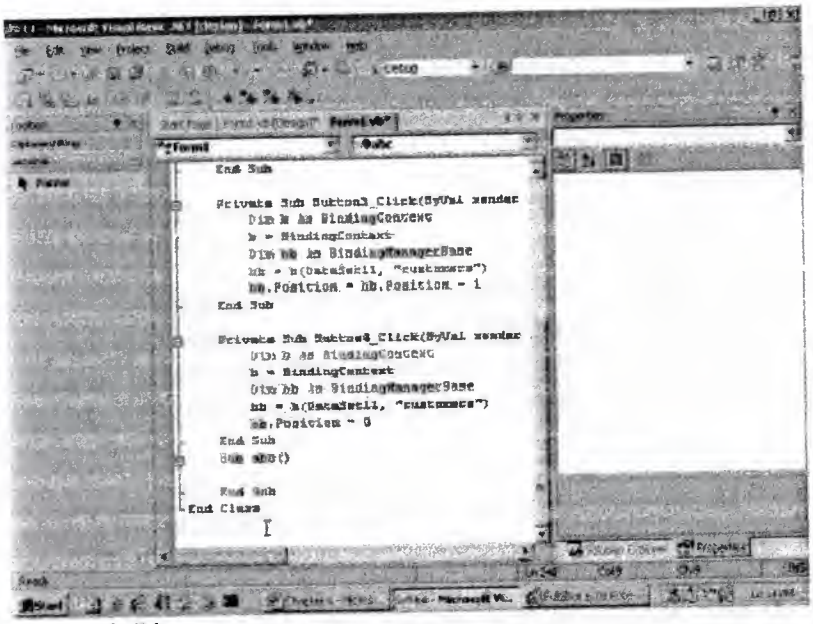
Close the Form and revert back to Visual Studio.Net. Now, we shall display the current active record number and the total number of records present in the customer table.

Select the TextBox control and drag and drop it onto the Form. This TextBox is named textBox4, as shown in screen 2.73.



Screen 2.73

Delete the default value assigned to the Text property of the TextBox. Then, select the tab Form1.vb, and at the very bottom, just prior to 'end class', enter the following lines of code in Sub abc. The moment we press the Enter key for the Sub abc, the words End Sub automatically get added. This is evident from screen 2.74.



Screen 2.74

```
Sub abc()
Dim c As Integer
Dim p As Integer
c = BindingContext(DataSet11, "customers").Count
p = BindingContext(DataSet11, "Customers").Position + 1
TextBox4.Text = "Record No " & p.ToString & " of " & c.ToString
End Sub
```

We start by creating two variables named c and p, both of type integer. The variable c will store the number of records in the table, and p will store the current record position.

The BindingManagerBase object possesses a Position property for the current record number, and a Count property for the total number of records that the table contains. The variables c and p are initialized to the values returned by these properties, respectively.

An alternative approach would be to split up the above statement.

Firstly, the BindingContext property, which returns a BindingContext object, can be stored in a variable. Then, an indexer that takes two strings, can be used to return a BindingManagerBase object. Finally, the Count property of this object is furnished to determine the total number of records in the table. We leave it to you to determine as to which of these two approaches is simpler to comprehend.

We finally initialize the Text property to the string that results from displaying the variables c and p. To write it all on a single line, the & sign is used, which acts like a delimiter while concatenating strings. Before we can see the output, there is one more task to be performed. The last objective is to introduce a button and label it as 'Last'. When this button is clicked on, the last record should become visible. So, double click on it and enter the following code:

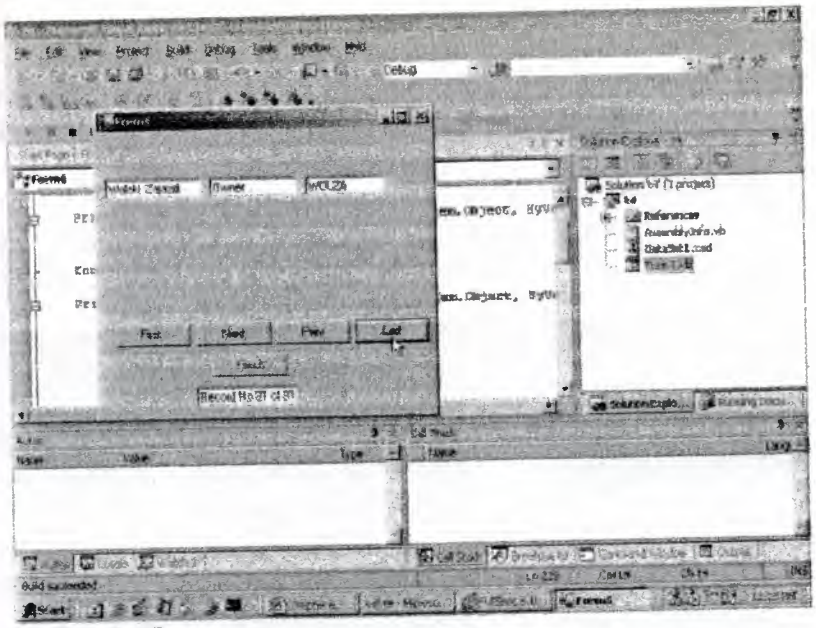
```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
BindingContext(DataSet11, "Customers").Position = 10000
abc()
End Sub
```

In this sub, the Position property is assigned a value of 10000. We allocated such a mammoth value, despite being cognizant of the fact that such a large number of records do not exist. This is because, whenever a very large number is specified, which goes beyond the record count, the active record becomes the last record.

The sub abc must be called in each of the four button events, in the manner shown above, otherwise, the textbox will not be updated.

Another course of action to achieve the above would be to use the Count property, to ascertain the total number of records, and then, to initialize the Position property to this value. In effect, the accurate value is the count value minus 1, because the value of record position for the first record is 0.

Now, when we click on Fetch, followed by the button labeled as Last, the last record becomes visible, as shown in screen 2.75.



Screen 2.75

The textbox gets updated, informing us about the record that is currently active. Again, since the Position property starts from 0 and not from 1, the value to be displayed first needs to be incremented by 1.

This is the last application in the chapter, before we call it a day. What we intend to achieve through this application is to display those records of customers in the datagrid, which meet a certain criteria.

Create a new project by choosing the menu option File-New-Project. Then, in the New Project dialog box, as usual, select Visual Basic projects in the first pane and Windows Application in the second pane. The name assigned to this project is t5.

Since the application revolves around fetching data from the Customers table in the database, the OleDbDataAdapter control is brought in from the toolbox. By this time, you must have committed all the steps to memory, since this process has been harped upon throughout this chapter. Everything remains the same, as has been explained earlier, except for the SQL Select statement, which is as follows:

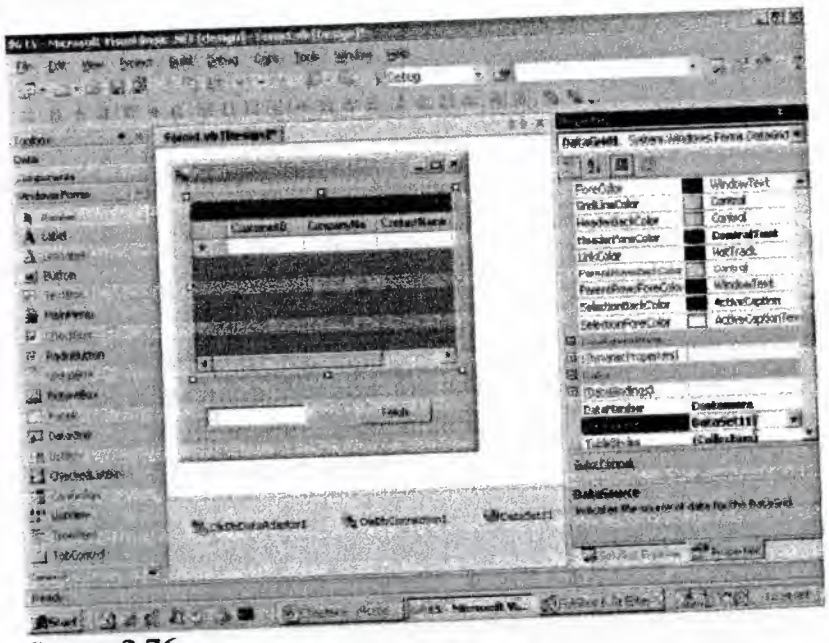
```
SELECT CustomerID, CompanyName, ContactName, ContactTitle FROM
Customers where (Country = ?)
```

The above SQL Select statement will retrieve records from the Customers table, where the field Country matches a specified value. At this instant, the value is not known, hence the ? symbol is used. The control ignores the new amendment in the syntax, and in its usual manner, it generates the OleDbDataAdapter object and the OleDbConnection Object.

Now, click on the menu Data, followed by the Generate DataSet. All the values are acceptable to us. Therefore, we leave everything untouched, and click on OK.

Subsequent to this, initiate a DataGrid control from the Windows Forms category in the ToolBox into the Form. The DataSource property of this Form is set to DataSet11 and the DataMember is set to Customers.

Then, bring in a button on the Form, and change its label to 'Fetch'. Then usher in a textbox named TextBox1. Blank out the value assigned to the text property. The Form should look like what is shown in screen 2.76.



Screen 2.76

Double click on the button, and enter the following code:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim c As System.Data.OleDb.OleDbCommand
c = OleDbDataAdapter1.SelectCommand
Dim p As System.Data.OleDb.OleDbParameterCollection
p = c.Parameters
Dim o As System.Data.OleDb.OleDbParameter
o = p("Country")
o.Value = TextBox1.Text
DataSet11.Clear()
OleDbDataAdapter1.Fill(DataSet11) End Sub
```

In the above code, the property SelectCommand of the OleDbDataAdapter1 object is used to retrieve an OleDbCommand object.

This OleDbCommand object, which is present in the System.Data.OleDb namespace, represents the Select statement entered in the OleDbDataAdapter wizard. It has a property called 'parameters', which stores all the parameters in the SQL Select statement.

A parameter is a field in the 'where' clause, which is assigned a value of a ?. The parameters property returns an OleDbParameterCollection object. Then, using an

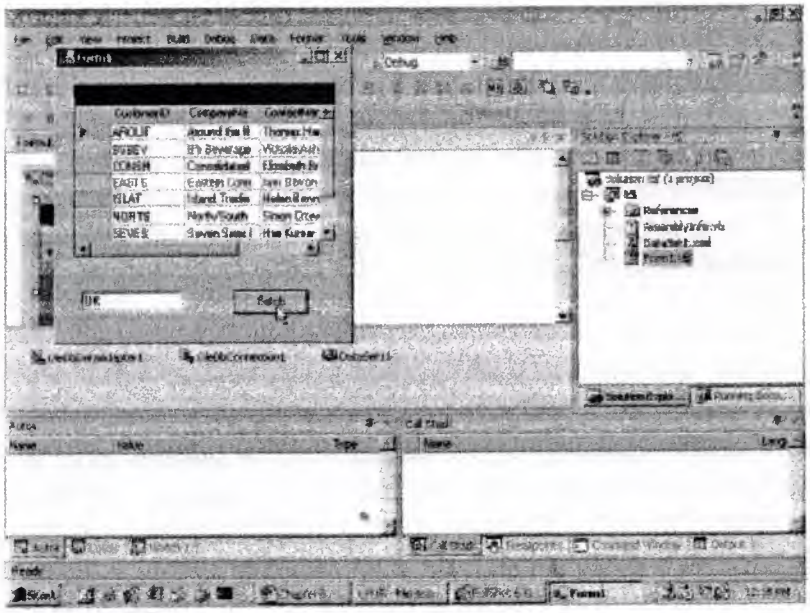
indexer, the individual parameters can then be accessed. Since it is a collection object, multiple parameters can be added to it.

The indexer takes the name of the field as a string parameter, in order to access the parameters. It returns an OleDbParameter object, which has a member of Value. This member is set to a value that relates to the field Country. This is what the user enters in the Textbox. Thus, the value of the field country can be set to whatever the user enters in the textbox.

At times, the DataGrid may contain certain values from a previous query. Hence, it is a good idea to Clear the contents of the table Customers. If the Clear function is not used, at times, the residual data from some earlier query may linger on, thereby cluttering the existing data.

As usual, the Fill function fills up the dataset with records that match the value of the Country that the user enters in the textbox.

Press F5 to run the program, and enter the name of the country as "UK". When the Fetch button is clicked, the output displayed will be the records of customers who reside in UK, thereby filtering the required records from amongst all the records in the table. This is shown in screen 2.77.



Screen 2.77

This is an example of a dynamic query, wherein the user decides the criteria, based on which, the data is dynamically retrieved and displayed.

Chapter three

I have used Microsoft Access 2002 database aplication in my project. So, I am going to give you basic information about Microsoft Access.

3. What is Microsoft Access?

Microsoft access is a database program.It is not complex. It is genaraly used by small and middle enterprices. It is one of the Microsoft Ofice program. So, it is easy to find and setup it.

3.1 Get Started with Access 2002

There are different ways you can get started with Access 2002, depending on your experience level.

Working with databases and database objects can be a daunting task when you first get started. The following information should help you become more familiar with the components that make up a Microsoft Access database.

3.1.1 Databases: What they are and how they work?

A database is a collection of information that's related to a particular subject or purpose, such as tracking customer orders or maintaining a music collection. If your database isn't stored on a computer, or only parts of it are, you may be tracking information from a variety of sources that you're having to coordinate and organize yourself.

For example, suppose the phone numbers of your suppliers are stored in various locations: in a card file containing supplier phone numbers, in product information files in a file cabinet, and in a spreadsheet containing order information. If a supplier's phone number changes, you might have to update that information in all three places. In a database, however, you only have to update that information in one place the supplier's phone number is automatically updated wherever you use it in the database.

3.1.1.1 Access Database Files

Using Microsoft Access, you can manage all your information from a single database file. Within the file, you can use:

- Tables to store your data.
- Queries to find and retrieve just the data you want.
- Forms to view, add, and update data in tables.
- Reports to analyze or print data in a specific layout.
- Data access pages to view, update, or analyze the database's data from the Internet or an intranet.

Store data once in one table, but view it from multiple locations. When you update the data, it's automatically updated everywhere it appears.

3.1.1.2 Tables and Relationships

To store your data, create one table for each type of information that you track. To bring the data from multiple tables together in a query, form, report, or data access page, define relationships between the tables.

A unique ID, such as a Customer ID, distinguishes one record from another within a table. By adding one table's unique ID field to another table and defining a relationship, Microsoft Access can match related records from both tables so that you can bring them together in a form, report, or query.

3.1.1.3 Queries

To find and retrieve just the data that meets conditions that you specify, including data from multiple tables, create a query. A query can also update or delete multiple records at the same time, and perform predefined or custom calculations on your data.

3.1.1.4 Forms

To easily view, enter, and change data directly in a table, create a form. When you open a form, Microsoft Access retrieves the data from one or more tables, and displays it on the screen with the layout you choose in the Form Wizard or with the layout that you created on your own in Design view.

A table displays many records at the same time, but you might have to scroll to see all of the data in a single record. Also, when viewing a table, you can't update data from more than one table at the same time.

A form focuses on one record at a time, and it can display fields from more than one table. It can also display pictures and other objects.

A form can contain a button that prints, opens other objects, or otherwise automates tasks.

3.1.1.5 Reports

To analyze your data or present it a certain way in print, create a report. For example, you might print one report that groups data and calculates totals, and another report with different data formatted for printing mailing labels.

3.1.1.6 Data Access Pages

To make data available on the Internet or an intranet for interactive reporting, data entry, or data analysis, use a data access page. Microsoft Access retrieves the data

from one or more tables and displays it on the screen with the layout that you created on your own in Design view, or with the layout you chose in the Page Wizard.

3.1.2 Table Design View

In table Design view, you can create an entire table from scratch, or add, delete, or customize the fields in an existing table.

If you want to track additional data in a table, add more fields. If an existing field name isn't descriptive enough, you can rename the field.

Setting a field's data type defines what kind of values you can enter in a field. For example, if you want a field to store numerical values that you can use in calculations, set its data type to **Number** or **Currency**.

You use a unique tag, called a primary key, to identify each record in your table. A table's primary key is used to refer to related records in other tables.

Field properties are a set of characteristics that provide additional control over how the data in a field is stored, entered, or displayed. Which properties are available depends on a field's data type.

3.1.2.1 How to Relate Two Tables

A common field relates two tables so that Microsoft Access can bring together the data from the two tables for viewing, editing, or printing. In one table, the field is a primary key that you set in table Design view. That same field also exists in the related table as a foreign key.

In the Suppliers table, you enter a supplier ID, company name, and so on, for each supplier. SupplierID is the primary key that you set in table Design view.

In the Products table, you include the SupplierID field, so that when you enter a new product, you can identify its supplier by entering that supplier's unique ID number. SupplierID is the foreign key in the Products table.

3.1.2.2 Table Datasheet View

In a table or query, Datasheet view provides the tools you need to work with data.

3.1.2.3 Using the Table Datasheet and Query Datasheet Toolbars

The Table Datasheet and Query Datasheet toolbars provide many of the tools you need to find, edit, and print records.

3.1.2.4 Working with columns, rows, and subdatasheets

You can find tools for working with columns, rows, and subdatasheets in the datasheet itself, or by right-clicking a column selector.

3.1.2.5 Moving Through Records

You can use the navigation toolbar to move through the records in a datasheet.

3.1.3 Queries

You use queries to view, change, and analyze data in different ways. You can also use them as a source of records for forms, reports, and data access pages. There are several types of queries in Microsoft Access.

3.1.3.1 Select Queries

A select query is the most common type of query. It retrieves data from one or more tables and displays the results in a datasheet where you can update the records (with some restrictions). You can also use a select query to group records and calculate sums, counts, averages, and other types of totals.

3.1.3.2 Parameter Queries

A parameter query is a query that when run displays its own dialog box prompting you for information, such as criteria for retrieving records or a value you want to insert in a field. You can design the query to prompt you for more than one piece of information; for example, you can design it to prompt you for two dates. Access can then retrieve all records that fall between those two dates.

Parameter queries are also handy when used as the basis for forms, reports, and data access pages. For example, you can create a monthly earnings report based on a parameter query. When you print the report, Access displays a dialog box asking for the month that you want the report to cover. You enter a month and Access prints the appropriate report.

3.1.3.3 Crosstab Queries

You use crosstab queries to calculate and restructure data for easier analysis of your data. Crosstab queries calculate a sum, average, count, or other type of total for data that is grouped by two types of information one down the left side of the datasheet and another across the top.

3.1.3.4 Action Queries

An action query is a query that makes changes to or moves many records in just one operation. There are four types of action queries:

- **Delete Queries** A delete query deletes a group of records from one or more tables. For example, you could use a delete query to remove products that are discontinued or for which there are no orders. With delete queries, you always delete entire records, not just selected fields within records.
- **Update Queries** An update query makes global changes to a group of records in one or more tables. For example, you can raise prices by 10 percent for all dairy products, or you can raise salaries by 5 percent for the people within a

certain job category. With an update query, you can change data in existing tables.

- **Append Queries** An append query adds a group of records from one or more tables to the end of one or more tables. For example, suppose that you acquire some new customers and a database containing a table of information on those customers. To avoid typing all this information into your own database, you'd like to append it to your Customers table.
- **Make-Table Queries** A make-table query creates a new table from all or part of the data in one or more tables. Make-table queries are helpful for creating a table to export to other Microsoft Access databases or a history table that contains old records.

3.1.3.5 SQL Queries

An SQL query is a query you create by using an SQL statement. You can use Structured Query Language (SQL) to query, update, and manage relational databases such as Access.

When you create a query in query Design view, Access constructs the equivalent SQL statements behind the scenes for you. In fact, most query properties in the property sheet in query Design view have equivalent clauses and options available in SQL view. If you want, you can view or edit the SQL statement in SQL view. However, after you make changes to a query in SQL view, the query might not be displayed the way it was previously in Design view.

Some SQL queries, called SQL-specific queries, can't be created in the design grid. For pass-through, data-definition, and union queries, you must create the SQL statements directly in SQL view. For subqueries, you enter the SQL in the Field row or the Criteria row of the query design grid.

3.1.4 Relationships in a Database

After you've set up different tables for each subject in your Microsoft Access database, you need a way of telling Microsoft Access how to bring that information back together again. The first step in this process is to define relationships between your tables. After you've done that, you can create queries, forms, and reports to display information from several tables at once

3.1.4.1 How Relationships Work

In the previous example, the fields in four tables must be coordinated so that they show information about the same order. This coordination is accomplished with relationships between tables. A relationship works by matching data in key fields usually a field with the same name in both tables. In most cases, these matching fields are the primary key from one table, which provides a unique identifier for each record, and a foreign key in the other table. For example, employees can be associated with orders they're responsible for by creating a relationship between the EmployeeID fields.

A one-to-many relationship is the most common type of relationship. In a one-to-many relationship, a record in Table A can have many matching records in Table B, but a record in Table B has only one matching record in Table A.

3.1.4.2 A many-to-many Relationship

In a many-to-many relationship, a record in Table A can have many matching records in Table B, and a record in Table B can have many matching records in Table A. This type of relationship is only possible by defining a third table (called a junction table) whose primary key consists of two fields — the foreign keys from both Tables A and B. A many-to-many relationship is really two one-to-many relationships with a third table. For example, the Orders table and the Products table have a many-to-many relationship that's defined by creating two one-to-many relationships to the Order Details table. One order can have many products, and each product can appear on many orders.

3.1.4.3 A one-to-one Relationship

In a one-to-one relationship, each record in Table A can have only one matching record in Table B, and each record in Table B can have only one matching record in Table A. This type of relationship is not common, because most information related in this way would be in one table. You might use a one-to-one relationship to divide a table with many fields, to isolate part of a table for security reasons, or to store information that applies only to a subset of the main table. For example, you might want to create a table to track employees participating in a fundraising soccer game. Each soccer player in the Soccer Players table has one matching record in the Employees table.

3.1.4.4 Defining Relationships

The kind of relationship that Microsoft Access creates depends on how the related fields are defined:

- A one-to-many relationship is created if only one of the related fields is a primary key or has a unique index.
- A one-to-one relationship is created if both of the related fields are primary keys or have unique indexes.
- A many-to-many relationship is really two one-to-many relationships with a third table whose primary key consists of two fields — the foreign keys from the two other tables.

You can also create a relationship between a table and itself. This is useful in situations where you need to perform a Lookup within the same table. In the Employees table, for example, you can define a relationship between the EmployeeID and ReportsTo fields, so that the ReportsTo field can display employee data from a matching EmployeeID.

Note If you drag a field that isn't a primary key and doesn't have a unique index to another field that isn't a primary key and doesn't have a unique index, an indeterminate relationship is created. In queries containing tables with an indeterminate

relationship, Microsoft Access displays a default join line between the tables, but referential integrity won't be enforced, and there's no guarantee that records are unique in either table.

3.1.4.5 Referential Integrity

Referential integrity is a system of rules that Microsoft Access uses to ensure that relationships between records in related tables are valid and that you don't accidentally delete or change related data. You can set referential integrity when all of the following conditions are met:

- The matching field from the primary table is a primary key or has a unique index.
- The related fields have the same data type. There are two exceptions. An AutoNumber field can be related to a Number field with a FieldSize property setting of Long Integer, and an AutoNumber field with a FieldSize property setting of Replication ID can be related to a Number field with a FieldSize property setting of Replication ID.
- Both tables belong to the same Microsoft Access database. If the tables are linked tables, they must be tables in Microsoft Access format, and you must open the database in which they are stored to set referential integrity. Referential integrity can't be enforced for linked tables from databases in other formats.

The following rules apply when you use referential integrity:

- You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table. However, you can enter a Null value in the foreign key, specifying that the records are unrelated. For example, you can't have an order that is assigned to a customer that doesn't exist, but you can have an order that is assigned to no one by entering a Null value in the CustomerID field.
- You can't delete a record from a primary table if matching records exist in a related table. For example, you can't delete an employee record from the Employees table if there are orders assigned to the employee in the Orders table.
- You can't change a primary key value in the primary table, if that record has related records. For example, you can't change an employee's ID in the Employees table if there are orders assigned to that employee in the Orders table.

3.1.4.6 Cascading Updates and Deletes

For relationships in which referential integrity is enforced, you can specify whether you want Microsoft Access to automatically cascade update and cascade delete related records. If you set these options, delete and update operations that would normally be prevented by referential integrity rules are allowed. When you delete records or change primary key values in a primary table, Microsoft Access makes necessary changes to related tables to preserve referential integrity.

If you select the **Cascade Update Related Fields** check box when defining a relationship, any time you change the primary key of a record in the primary table,

Microsoft Access automatically updates the primary key to the new value in all related records. For example, if you change a customer's ID in the Customers table, the CustomerID field in the Orders table is automatically updated for every one of that customer's orders so that the relationship isn't broken. Microsoft Access cascades updates without displaying any message.

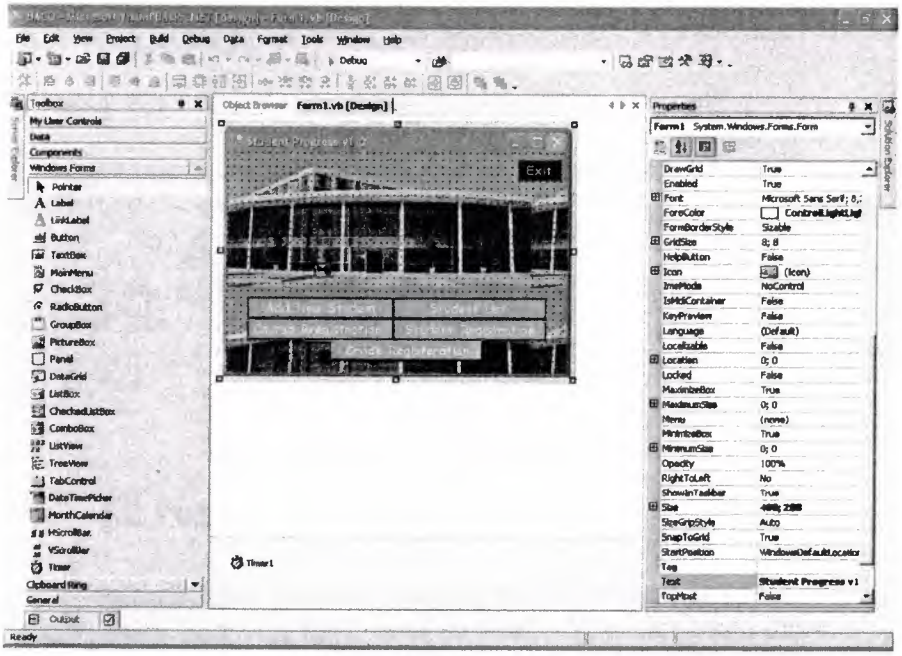
Note If the primary key in the primary table is an AutoNumber field, setting the **Cascade Update Related Fields** check box will have no effect, because you can't change the value in an AutoNumber field.

If you select the **Cascade Delete Related Records** check box when defining a relationship, any time you delete records in the primary table, Microsoft Access automatically deletes related records in the related table. For example, if you delete a customer record from the Customers table, all the customer's orders are automatically deleted from the Orders table (this includes records in the Order Details table related to the Orders records). When you delete records from a form or datasheet with the **Cascade Delete Related Records** check box selected, Microsoft Access warns you that related records may also be deleted. However, when you delete records using a delete query, Microsoft Access automatically deletes the records in related tables without displaying a warning.

Chapter four

4.1 Form1 (Main Menu);

This form (screen 4.1) is the main form of my program. There are five buttons. When you press 'add new student' button, you will access 'add new student' form.



SCREEN 4.1

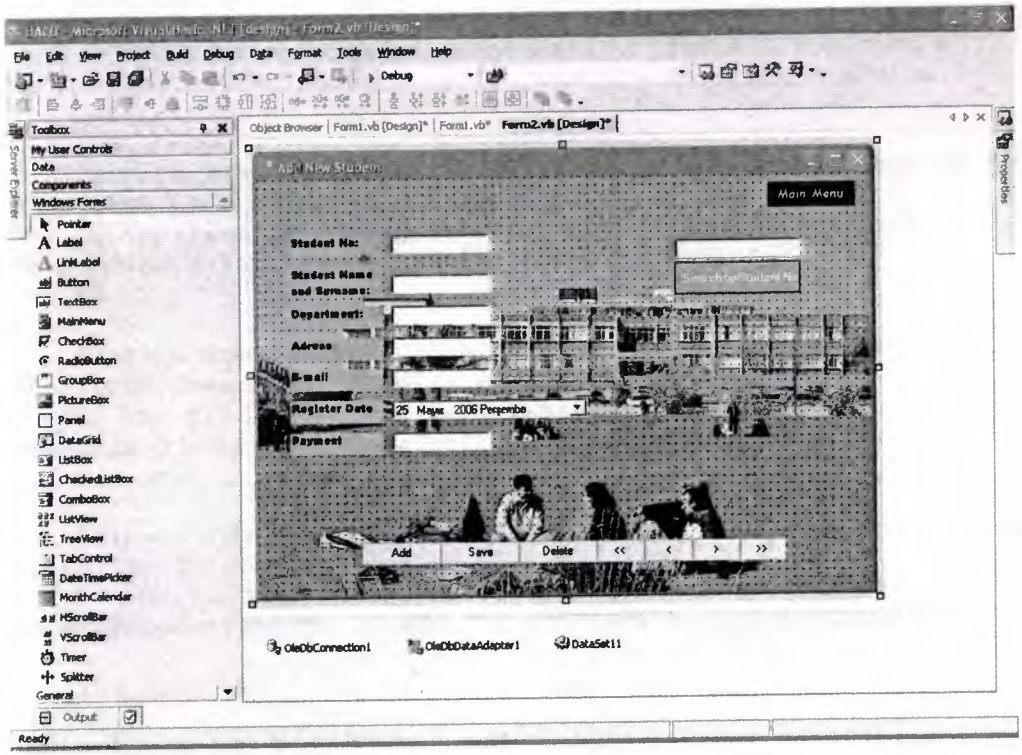
Source code for student stock program form1 giving below;

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim a As New Form3

        Me.Hide()
        a.ShowDialog()
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Dim a As New Form2
        Me.Hide()
        a.ShowDialog()



    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button3.Click
        Application.Exit()


    End Sub
```

```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button4.Click
        Dim a As New Form5
        Me.Hide()
        a.ShowDialog()

    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button5.Click
        Dim a As New Form4
        Me.Hide()
        a.ShowDialog()
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button6.Click
        Dim a As New Form6
        a.Show()
        Me.Hide()
    End Sub

End Class
```

4.2 Form2 (Add New Student);

Suppose that you use this program in collage, small university or one department of university and you have new students. You can store datas of new student in this form. His name, number, e_mail, payment and registration date.



SCREEN 4.2

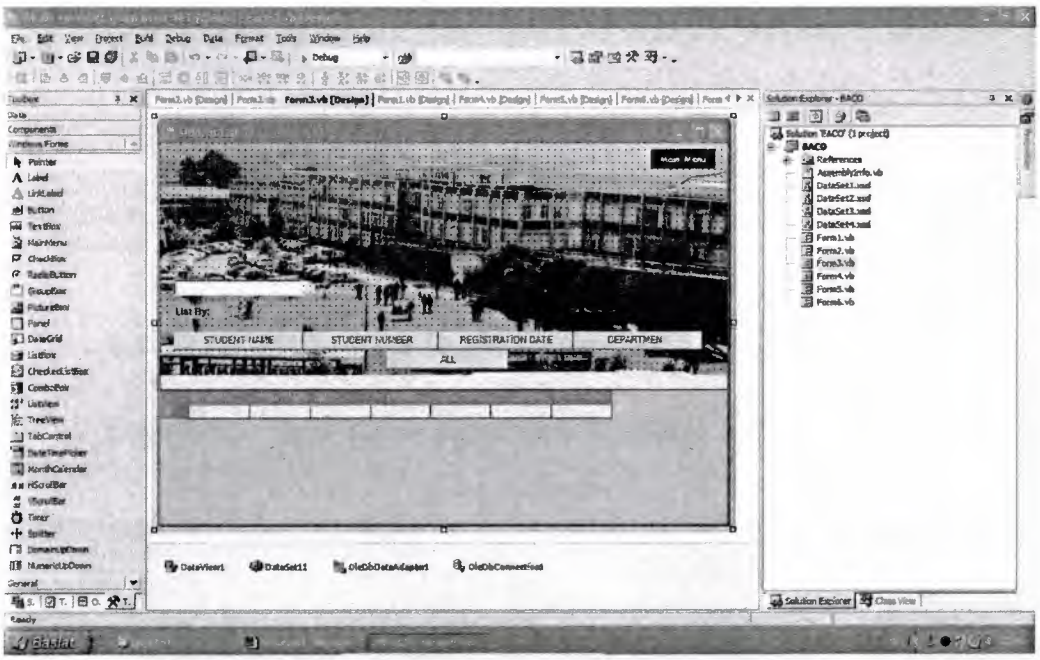
When you click 'add new student' button in form1 (screen4.2), you will access form2. Form2's code has given below;

```
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.OleDbDataAdapter1.Fill(Me.DataSet11, "TABLE")

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Me.BindingContext(Me.DataSet11, "TABLE").EndCurrentEdit()
        Me.BindingContext(Me.DataSet11, "TABLE").AddNew()
        Me.OleDbDataAdapter1.Update(Me.DataSet11, "TABLE")

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        Me.BindingContext(Me.DataSet11, "TABLE").EndCurrentEdit()
        Me.OleDbDataAdapter1.Update(Me.DataSet11, "TABLE")

    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button3.Click
        Dim sil As DataSet1.TABLERow
        sil = Me.DataSet11.TABLE.FindBySTUDENTNUMBER(TextBox2.Text)
        sil.Delete()
        Me.OleDbDataAdapter1.Update(Me.DataSet11, "TABLE")

    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button4.Click
        Me.BindingContext(Me.DataSet11, "TABLE").Position =
Me.BindingContext(Me.DataSet11, "TABLE").Position.MinValue
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button5.Click
        Me.BindingContext(Me.DataSet11, "TABLE").Position =
Me.BindingContext(Me.DataSet11, "TABLE").Position - 1
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button7.Click
        Me.BindingContext(Me.DataSet11, "TABLE").Position =
Me.BindingContext(Me.DataSet11, "TABLE").Position + 1
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button6.Click
        Me.BindingContext(Me.DataSet11, "TABLE").Position =
Me.BindingContext(Me.DataSet11, "TABLE").Position.MaxValue

    End Sub

    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button9.Click
        Dim row As DataSet1.TABLERow
```

```
        row = Me.DataSet11.TABLE.FindBySTUDENTNUMBER(TextBox7.Text)

        Try
            TextBox1.Text = row.STUDENTNUMBER
        Catch ex As Exception

        End Try

        Try
            TextBox2.Text = row.STUDENTNAME
        Catch
        End Try
        Try
            TextBox3.Text = row.DEPARTMENT
        Catch
        End Try
        Try
            TextBox4.Text = row.ADDRESS
        Catch
        End Try

        Try
            TextBox5.Text = row.EMAIL
        Catch
        End Try

        Try
            Me.DateTimePicker1.Text = row.REGISTERDATE
        Catch
        End Try

        Try
            TextBox6.Text = row.PAYMENT
        Catch
        End Try

    End Sub


    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button8.Click
        Dim a As New Form1
        a.Show()
        Me.Hide()
    End Sub
End Class
```

4.3 Form3 (Student List);

In this form (screen 4.3), you can see studens that you registered in form2 (screen 4.2) and you can list student by their name, number, registration date or department.



SCREEN 4.3

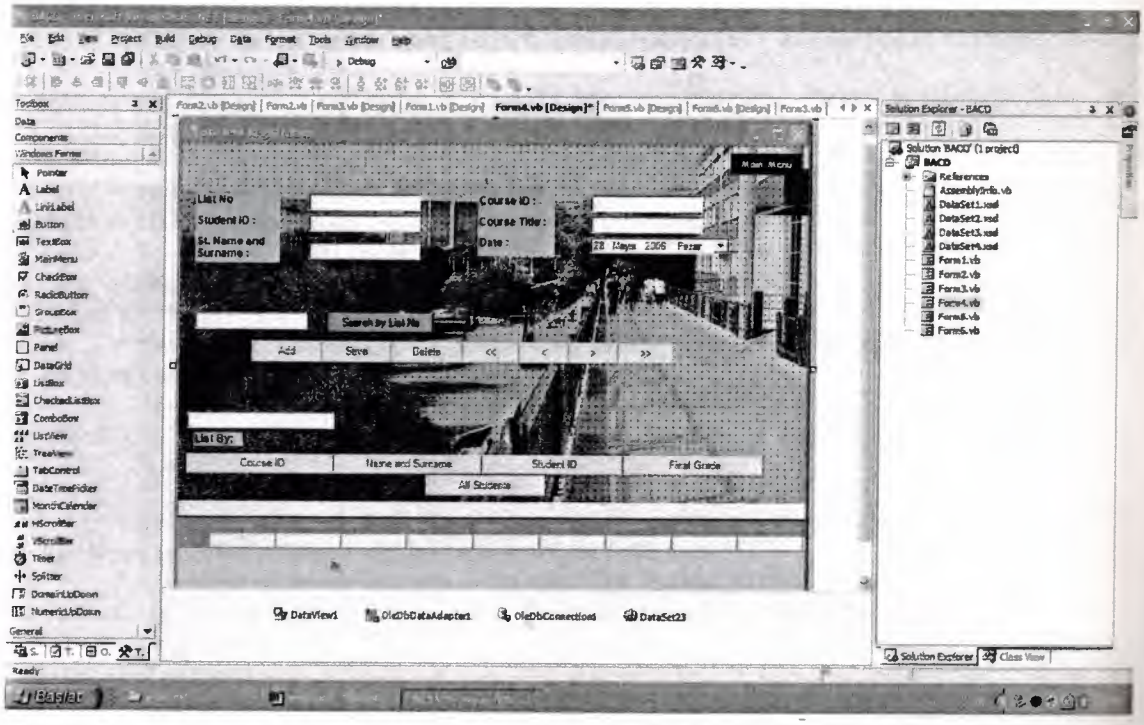
If you click 'student list' in main form, you will see the form3. Form3's code has given below;

```
Public cm As CurrencyManager
    Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        cm = CType(Me.BindingContext(DataView1), CurrencyManager)
        OleDbDataAdapter1.Fill(DataSet11.TABLE)
    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button8.Click
        Dim a As New Form1
        a.Show()
        Me.Hide()

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        DataView1.RowFilter = "STUDENTNUMBER='" & TextBox1.Text & "'"
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        DataView1.RowFilter = "STUDENTNAME='" & TextBox1.Text & "'"
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button4.Click
```

```
        DataView1.RowFilter = "REGISTARDATE='" & TextBox1.Text & "'"
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button3.Click
        DataView1.RowFilter = "DEPARTMENT='" & TextBox1.Text & "'"
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button5.Click
        DataView1.RowFilter = ""

    End Sub

End Class
```

4.4 Form4 (Student Registration);

Registration form makes registration to course. Process is 'who is taking which course?'



SCREEN 4.4

Source code for student stock program form4 giving below;

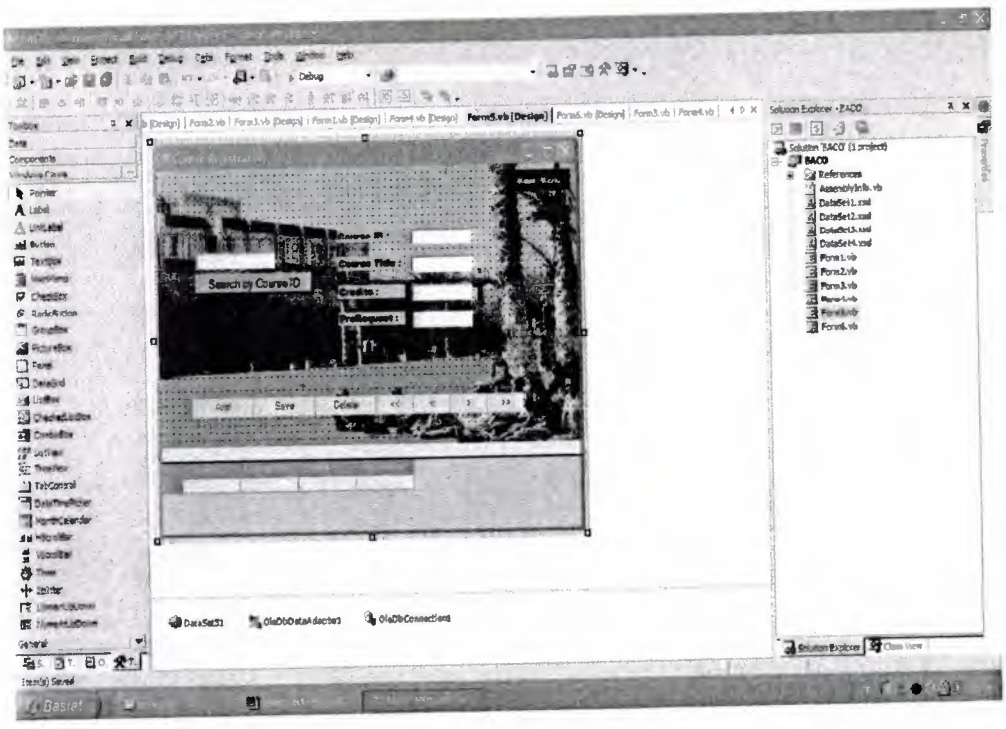
```
Public fin, mid, att, tot As Integer
    Public cm As CurrencyManager

        Private Sub Form4_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
            cm = CType(Me.BindingContext(DataView1), CurrencyManager)
        OleDbDataAdapter1.Fill(DataSet23.registration)
       End Sub
```

```
    Private Sub Button9_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button9.Click
        Dim a As New Form1
        a.Show()
        Me.Hide()
    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button8.Click
        Me.BindingContext(Me.DataSet23,
"registration").EndCurrentEdit()
        Me.BindingContext(Me.DataSet23, "registration").AddNew()
        Me.OleDbDataAdapter1.Update(Me.DataSet23, "registration")
        End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button2.Click
        Me.BindingContext(Me.DataSet23,
"registration").EndCurrentEdit()
        Me.OleDbDataAdapter1.Update(Me.DataSet23, "registration")
        End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button3.Click
        Dim sil As DataSet2.registrationRow
        sil = DataSet23.registration.FindBylist_no(TextBox3.Text)
        sil.Delete()
        Me.OleDbDataAdapter1.Update(Me.DataSet23, "registration")
        End Sub

    Private Sub Button10_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button10.Click
        DataView1.RowFilter = "courseId='" & TextBox2.Text & "'"
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button5.Click
        Me.BindingContext(Me.DataSet22, "registration").Position =
Me.BindingContext(Me.DataSet23, "registration").Position - 1
        End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button7.Click
        Me.BindingContext(Me.DataSet22, "registration").Position =
Me.BindingContext(Me.DataSet23, "registration").Position + 1
        End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button4.Click
        Me.BindingContext(Me.DataSet22, "registration").Position =
Me.BindingContext(Me.DataSet23, "registration").Position.MinValue
        End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button6.Click
        Me.BindingContext(Me.DataSet22, "registration").Position =
Me.BindingContext(Me.DataSet23, "registration").Position.MaxValue
        End Sub
```

```
    Private Sub Button14_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button14.Click
        DataView1.RowFilter = ""
    End Sub

    Private Sub Button12_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button12.Click
        DataView1.RowFilter = "studentnumber='" & TextBox2.Text & "'"
    End Sub

    Private Sub Button11_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button11.Click
        DataView1.RowFilter = "studentname='" & TextBox2.Text & "'"
    End Sub

    Private Sub Button13_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button13.Click
        DataView1.RowFilter = "grade='" & TextBox2.Text & "'"
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Dim row As DataSet2.registrationRow
        row = Me.DataSet23.registration.FindBylist_no(TextBox3.Text)

        Try
            TextBox5.Text = row.list_no

        Catch ex As Exception

        End Try

        Try
            TextBox6.Text = row.courseId

        Catch
        End Try
        Try
            TextBox7.Text = row.coursename

        Catch
        End Try
        Try
            TextBox1.Text = row.studentnumber
        Catch
        End Try

        Try
            TextBox4.Text = row.studentname
        Catch
        End Try

        Try
            Me.DateTimePicker1.Text = row._date
        Catch
        End Try

    End Sub

    End Class
```

4.5 Form5 (Couse Registration);

It makes course registration. You should add how many courses are there.



SCREEN 4.5

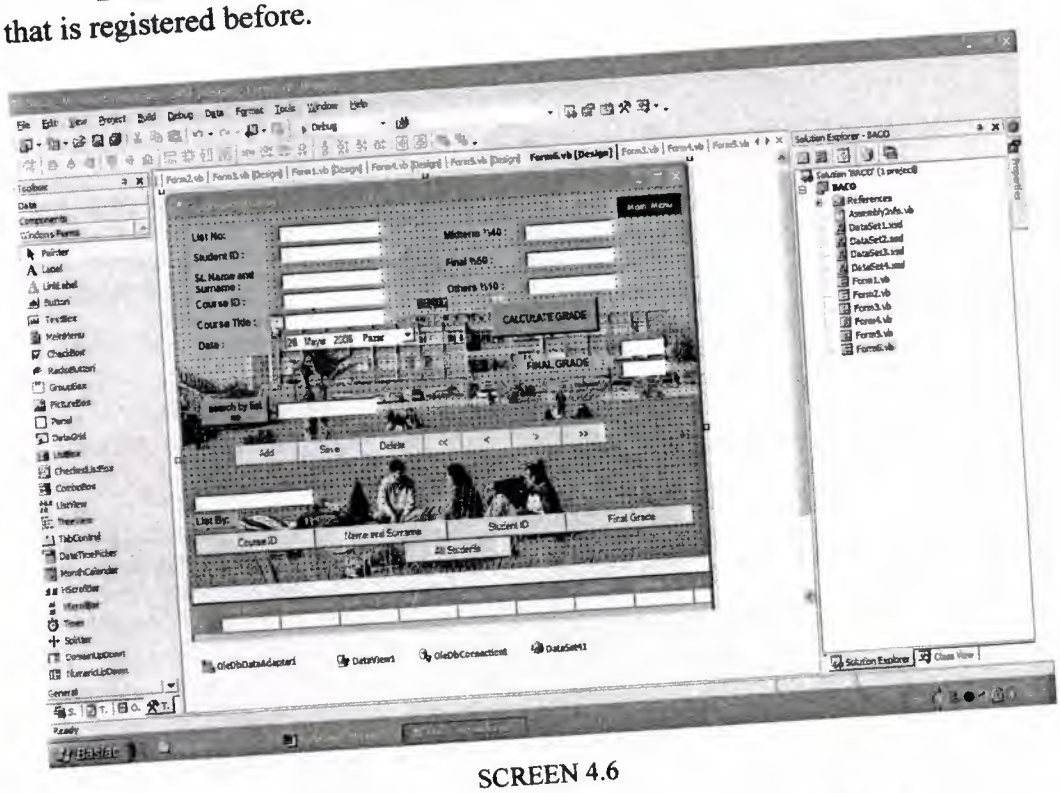
Source code for student stock program form5 giving below;

```
Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button8.Click
        Dim a As New Form1
        a.Show()
        Me.Hide()
    End Sub


Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Me.BindingContext(Me.DataSet31, "course").EndCurrentEdit()
        Me.BindingContext(Me.DataSet31, "course").AddNew()
        Me.OleDbDataAdapter1.Update(Me.DataSet31, "course")


    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click
        Me.BindingContext(Me.DataSet31, "course").EndCurrentEdit()
        Me.OleDbDataAdapter1.Update(Me.DataSet31, "course")

    End Sub


    Private Sub Form5_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.OleDbDataAdapter1.Fill(Me.DataSet31, "course")

    End Sub
```

```
    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button9.Click
        Dim row As DataSet3.courseRow
        row = Me.DataSet31.course.FindByCourseId(TextBox1.Text)

        Try
            TextBox5.Text = row.CourseId
        Catch ex As Exception

        End Try

        Try
            TextBox4.Text = row.Credits

        Catch
        End Try
        Try
            TextBox3.Text = row.PreRequest

        Catch
        End Try

    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button6.Click
        Me.BindingContext(Me.DataSet31, "course").Position =
Me.BindingContext(Me.DataSet31, "course").Position.MaxValue
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button7.Click
        Me.BindingContext(Me.DataSet31, "course").Position =
Me.BindingContext(Me.DataSet31, "course").Position + 1
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button5.Click
        Me.BindingContext(Me.DataSet31, "course").Position =
Me.BindingContext(Me.DataSet31, "course").Position + 1
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button4.Click
        Me.BindingContext(Me.DataSet31, "course").Position =
Me.BindingContext(Me.DataSet31, "course").Position.MinValue
    End Sub
End Class
```

4.6 Form6 (Grade Registration);

In this form stores exam result and calculate final grade of student for the course that is registered before.



SCREEN 4.6

Source code for student stock program form6 giving below;

```
Public fin, mid, att, tot As Integer
    Public cm As CurrencyManager

    Private Sub Form4_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        cm = CType(Me.BindingContext(DataView1), CurrencyManager)
        OleDbDataAdapter1.Fill(DataSet41.registration)


    End Sub

    Private Sub Button9_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button9.Click
        Dim a As New Form1
        a.Show()
        Me.Hide()
    End Sub

    Private Sub Button8_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button8.Click
        Me.BindingContext(Me.DataSet41,
"registration").EndCurrentEdit()
        Me.BindingContext(Me.DataSet41, "registration").AddNew()
        Me.OleDbDataAdapter1.Update(Me.DataSet41, "registration")
    End Sub
```

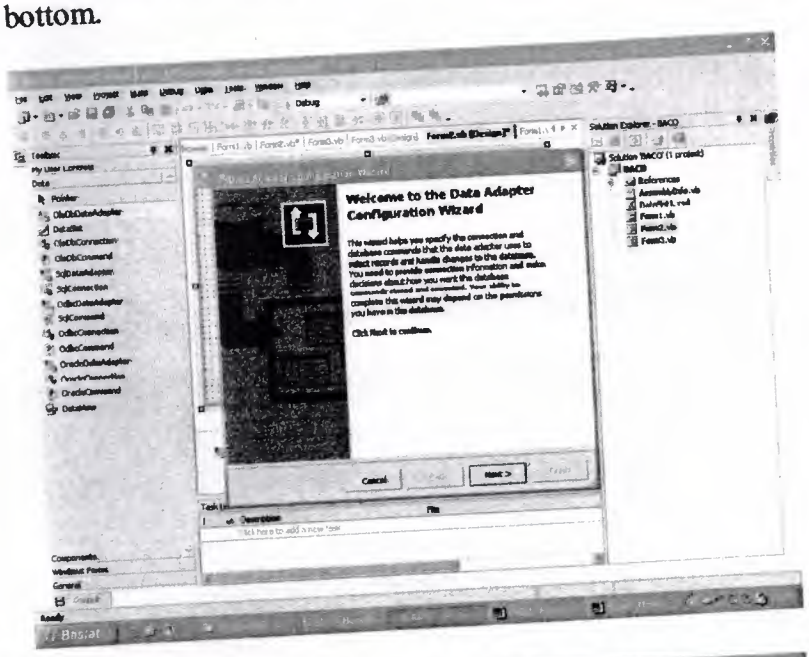
```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click

        MessageBox.Show(TextBox1.Text & " added your database
successfully", "Added successfully", MessageBoxButtons.OK)


        Me.BindingContext(Me.DataSet41,
"registration").EndCurrentEdit()
        Me.OleDbDataAdapter1.Update(Me.DataSet41, "registration")
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button3.Click
        Dim sil As DataSet4.registrationRow
        sil = DataSet41.registration.FindBylist_no(TextBox9.Text)
        sil.Delete()
        Me.OleDbDataAdapter1.Update(Me.DataSet41, "registration")
    End Sub


    Private Sub Button10_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button10.Click
        DataView1.RowFilter = "courseId='" & TextBox2.Text & "'"
    End Sub



    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button5.Click
        Me.BindingContext(Me.DataSet41, "registration").Position =
Me.BindingContext(Me.DataSet41, "registration").Position - 1
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button7.Click
        Me.BindingContext(Me.DataSet41, "registration").Position =
Me.BindingContext(Me.DataSet41, "registration").Position + 1
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button4.Click
        Me.BindingContext(Me.DataSet41, "registration").Position =
Me.BindingContext(Me.DataSet41, "registration").Position.MinValue
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button6.Click
        Me.BindingContext(Me.DataSet41, "registration").Position =
Me.BindingContext(Me.DataSet41, "registration").Position.MaxValue
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        mid = TextBox15.Text
        mid = mid * 0.4
        fin = TextBox16.Text
        fin = fin * 0.5
        att = TextBox5.Text
        att = att * 0.1
        tot = fin + mid + att
```

```
        TextBox3.Text = "AA"
    ElseIf tot >= 85 And tot <= 90 Then
        TextBox3.Text = "BA"
    ElseIf tot >= 80 And tot < 85 Then
        TextBox3.Text = "BB"
    ElseIf tot >= 75 And tot < 80 Then
        TextBox3.Text = "CB"
    ElseIf tot >= 70 And tot < 75 Then
        TextBox3.Text = "CC"
    ElseIf tot >= 65 And tot < 70 Then
        TextBox3.Text = "DC"
    ElseIf tot >= 60 And tot < 65 Then
        TextBox3.Text = "DD"
    ElseIf tot >= 55 And tot < 60 Then
        TextBox3.Text = "FD"
    ElseIf tot < 55 Then
        TextBox3.Text = "FF"
    End If


End Sub

Private Sub Button14_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button14.Click
    DataView1.RowFilter = ""

End Sub

Private Sub Button12_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button12.Click
    DataView1.RowFilter = "studentnumber='" & TextBox2.Text & "'"

End Sub

Private Sub Button11_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button11.Click
    DataView1.RowFilter = "studentname='" & TextBox2.Text & "'"

End Sub

Private Sub Button13_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button13.Click
    DataView1.RowFilter = "grade='" & TextBox2.Text & "'"

End Sub

Private Sub Button15_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button15.Click
    Dim row As DataSet4.registrationRow


    row = Me.DataSet41.registration.FindBylist_no(TextBox9.Text)

    Try
        TextBox10.Text = row.list_no

    Catch ex As Exception
```

```
        End Try

        Try
            TextBox6.Text = row.courseId

        Catch
        End Try
        Try
            TextBox7.Text = row.coursename

        Catch
        End Try
        Try
            TextBox1.Text = row.studentnumber
        Catch
        End Try

        Try
            TextBox4.Text = row.studentname
        Catch
        End Try

        Try
            Me.DateTimePicker1.Text = row._date

        Catch
        End Try

        Try
            Me.TextBox15.Text = row.midterm

        Catch
        End Try
        Try

            Me.TextBox16.Text = row.final

        Catch
        End Try
        Try
            Me.TextBox5.Text = row.others
        Catch
        End Try
        Try
            Me.TextBox3.Text = row.grade
        Catch
        End Try


    End Sub
End Class
```

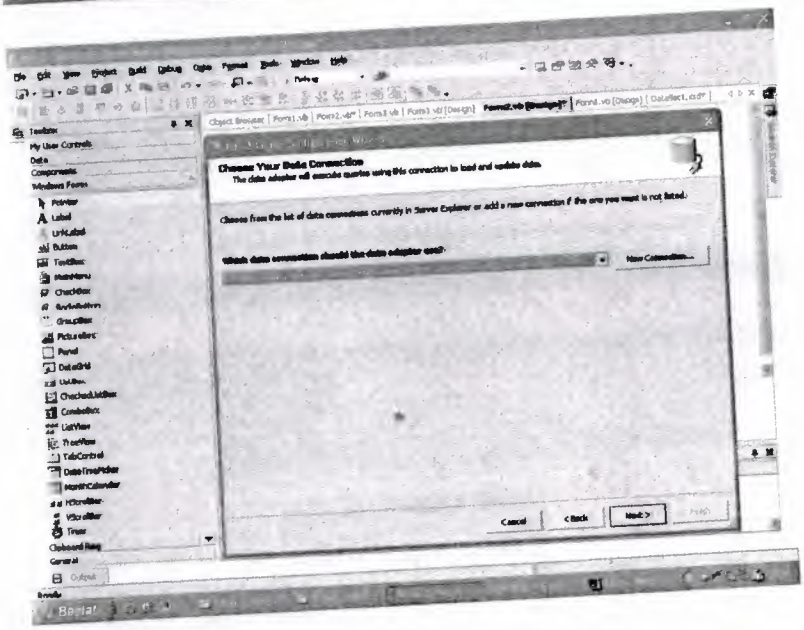
4.7 Data Adapter Configuration

Take one data adapterfrom toolbax in data components and put it into form. You will access data adapter configuration wizart. Follow settings which are shown in bottom.



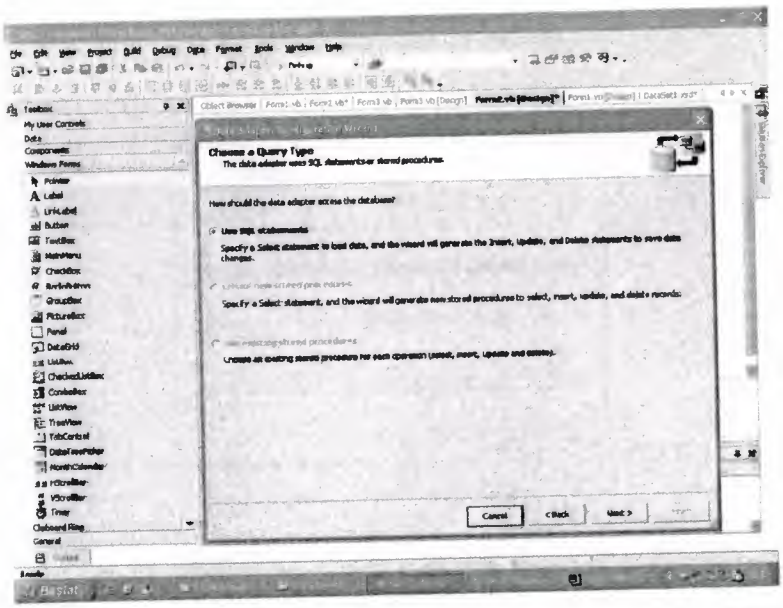
SCREEN 4.7

Press next botton



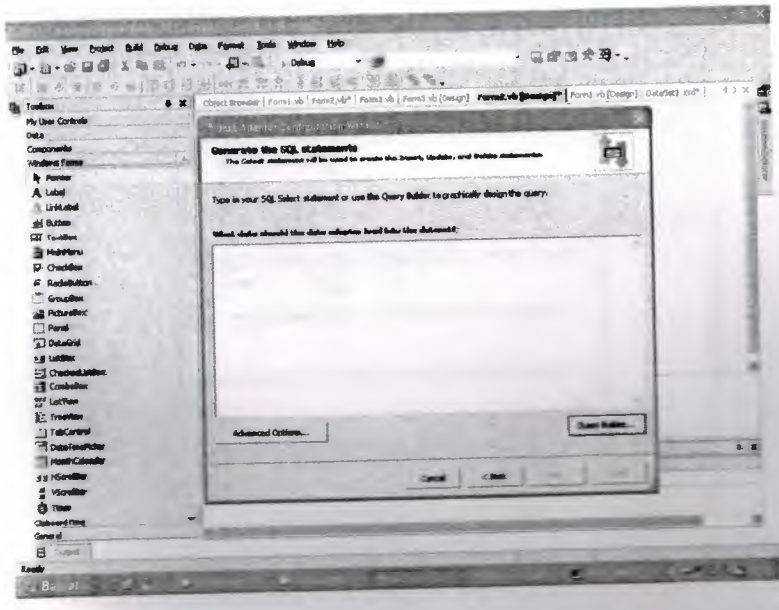
SCREEN 4.8

If you made you database connection before by using server explorer bar, you can see you connection in that combobox.Click it and press next.



SCREEN 4.9

Press next button.

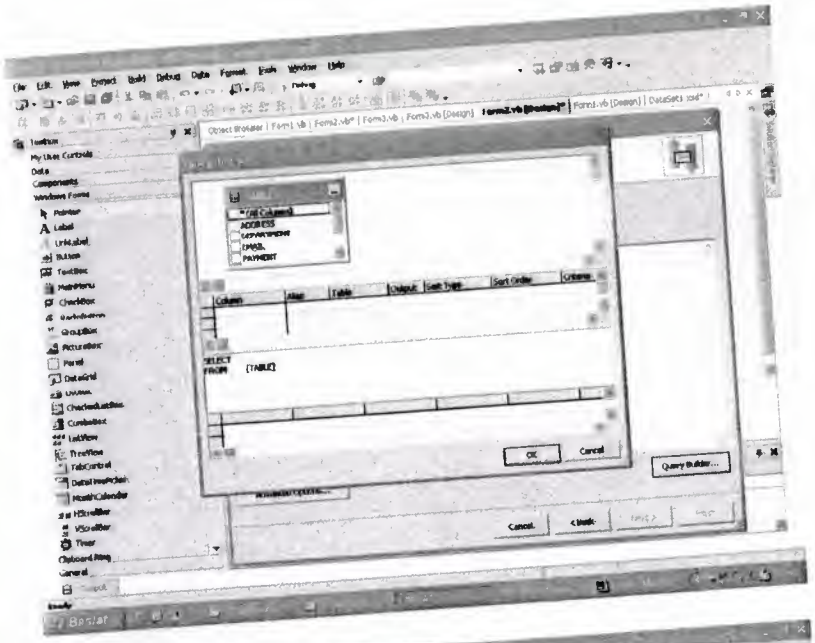


SCREEN 4.11

Press query builder.

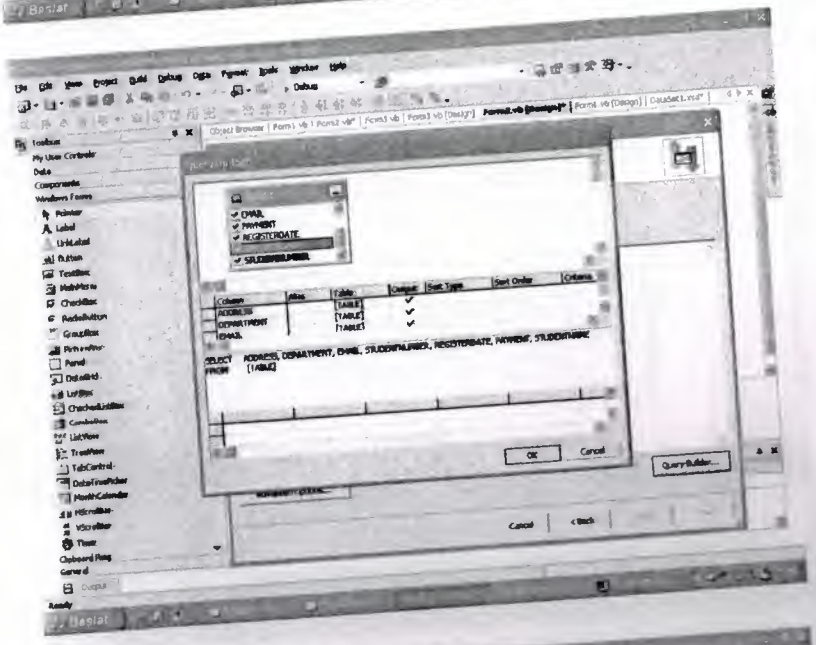
SCREEN 4.12

Add your table to query



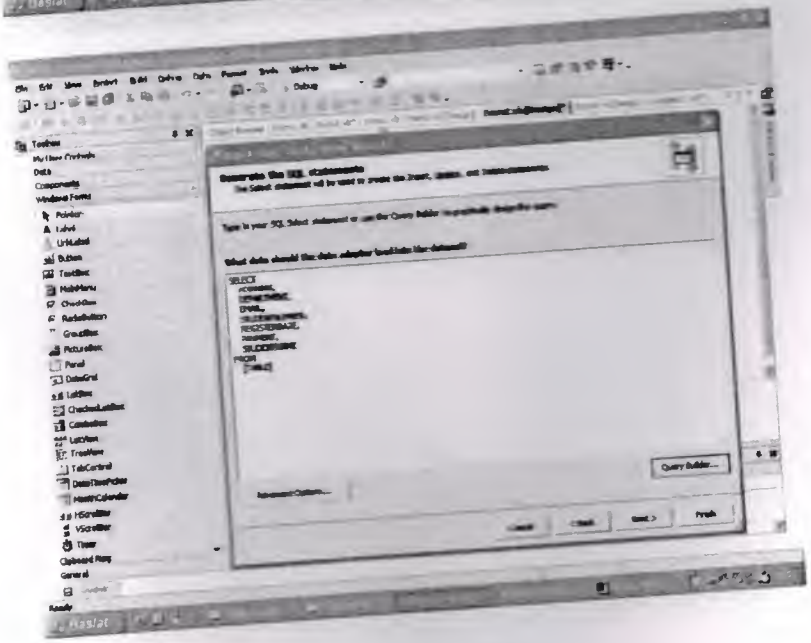
SCREEN 4.13

Click which colomn you want to add in you table.



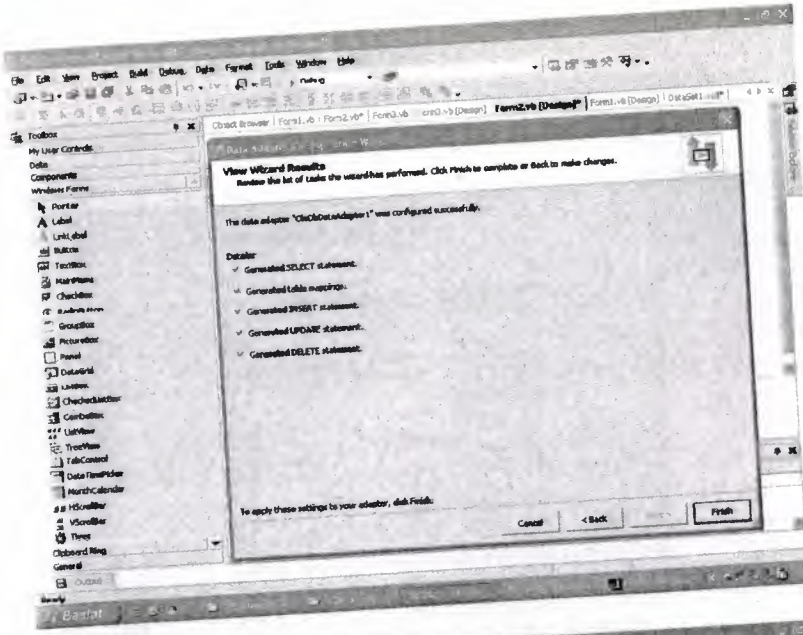
SCREEN 4.14

Press ok.



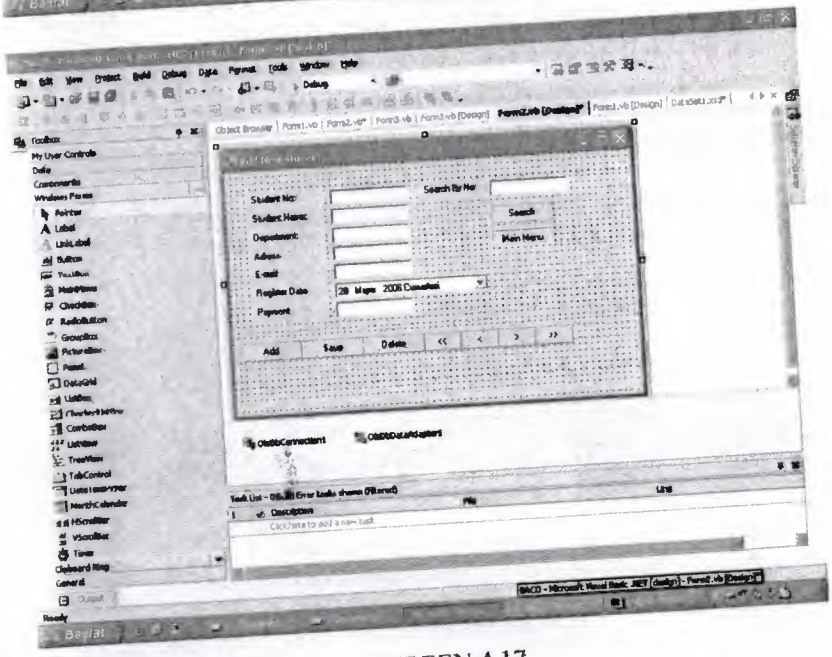
SCREEN 4.15

See what did you add in your dataset



SCREEN 4.16

Click finish, and then it is configurated

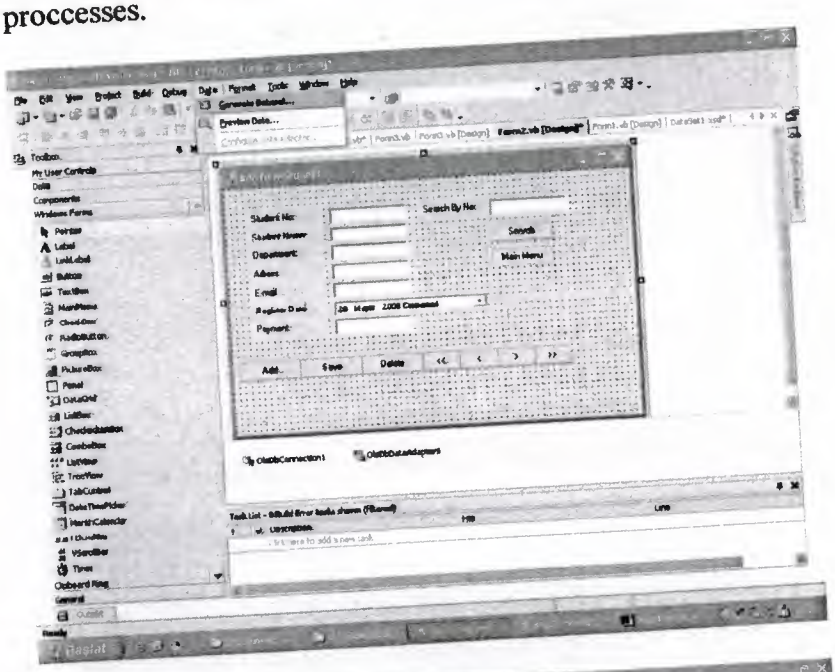


SCREEN 4.17

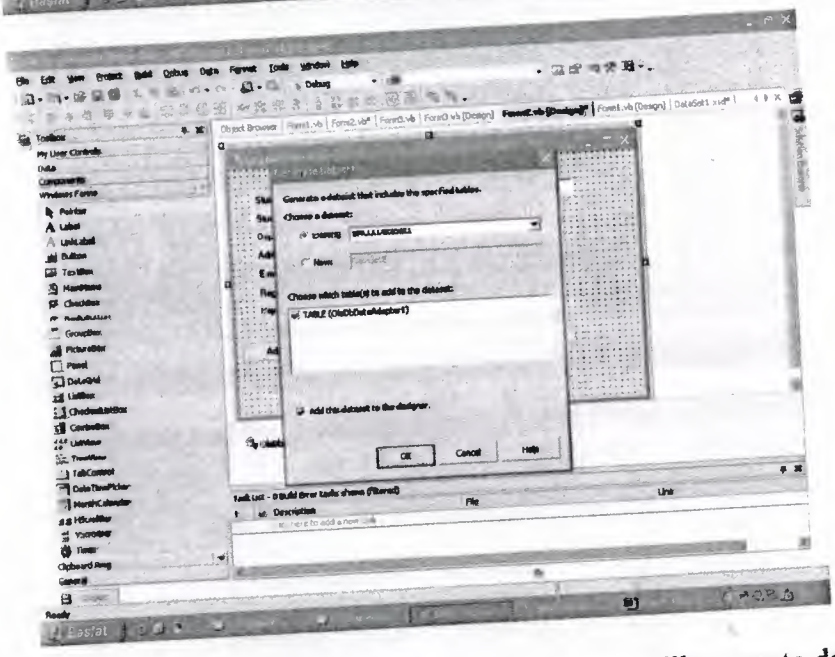
OleDbconnection1 is added outomaticly

4.8 Generate Dataset

Press data button on main menu and press generate dataset. Apply comming proccesses.



SCREEN 4.18

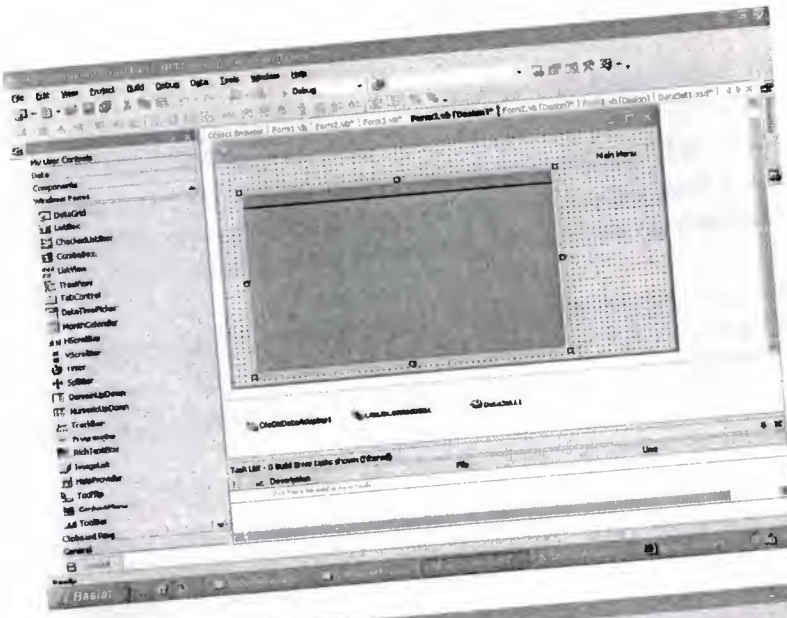


SCREEN 4.19

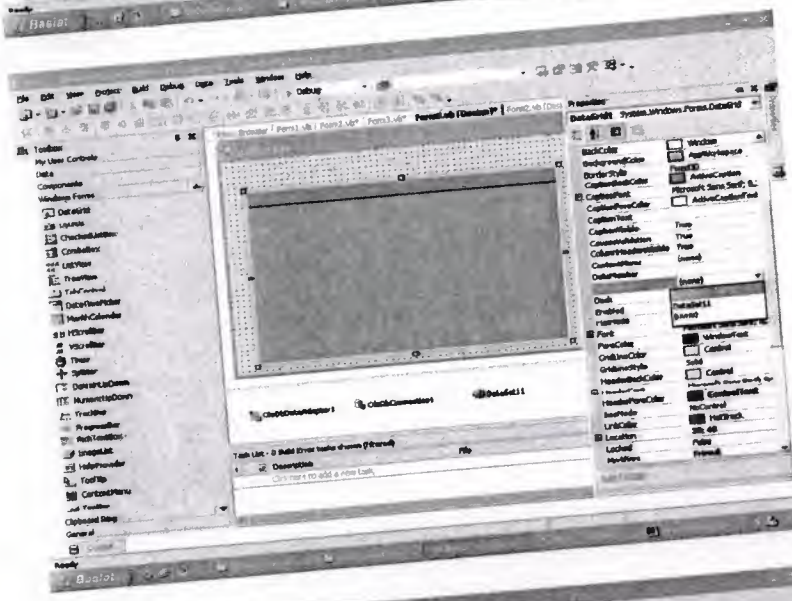
When you press ok on screen3.19, you will generate dataset.

4.9 How to Make Relations between Datagrid and Dataset

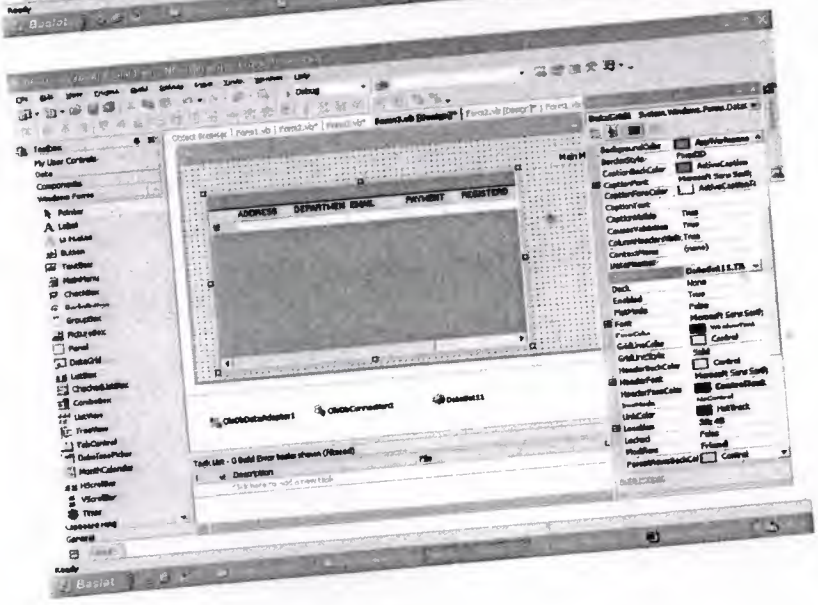
After adding datagrid, enter proporties of datagrid and find datasource setting. When you click front space of datasource, you will see dataset and table name together. Click it. You will see selected table in datagrid



SCREEN 4.21



SCREEN 4.22



SCREEN 4.23

CONCLUSION

This project obtains me to increase my knowledge about Visual Basic.NET and Object Oriented Database and also finding solutions to some specific problems. Before we learned programming in traditional languages. This kind of programming language obligates the programmer to write lots of codes send procedures to develop similar program that includes, data functions, class, librarys, and declarations.

Nowadays programming languages are going to easier then before for programmer. Programmer needs intelligent opinions, creative idea. They don't need to memorize understand lots of codes, functions, procedures.

In Vb.NET these is allowed by program maintenance and improvement of program is very easy to rearrangement. Vb.NET programming language gives me optimum about to make best situation for designing program in very sufficient conditions.

REFERENCE

1. **Pelme Zirvedeki Beyinler 1: Visual Basic.NET**
YÜKSEL İLHAN AND NIHAT DEMIRLI

2. **Programming VB.NET:**
A Guide for Experienced
Programmers
GARY CORNELL AND JONATHAN MORRISON

3. **A Programmer's Guide to Visual Basic.NET**
Copyright © 2001 by Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

4. **Learning Visual Basic .NET**
Jesse Liberty
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

5. **www.codepower.com**

6. **www.programlama.com**

7. **Microsoft Office User Guide**