Unit 9 Data Access with ADO.NET

Structure:

9.1 Introduction

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

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

Objectives:

After studying this unit, you will be able to:

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

ADO.NET:

ADO is called ADO.NET because it uses the .NET programming framework. ADO stands for ActiveX Data Object ,it is a model for gaining access to data that is widely implemented today. Using ADO.NET library, any windows or web based .NET application, can easily interact with the databases.

It transmit the data from the database and to the database with the help of XML format. Many classes, interfaces, and enumerated data types are provided for managing data access from a variety of locations. It includes a number of classes, interfaces, and enumerated data types that can be used to handle data access from various sources.

ADO.NET Data Access Mechanism:

The ADO.NET basically use for two purposes such as data access and storage purpose. For storage purpose it provides in-memory feature to store the data from the database tables. The interfaces are nothing but the concrete classes corresponds to a particular data access scheme. Data access and storage are the two main components of ADO.NET. While storing information, ADO.NET use the concept of classes. It's important to note that ADO.NET stores data via a variety of different mechanisms. Systems that allow users to read and write information make use of interfaces.

Types of Data Access Mechanism:

The data access mechanisms are further classified into five mechanisms which are provided.

1. System.Data.SqlClient : Represents a set of data commands and a database connection that are used to fill the dataset and update a SQL Server database.
2. System.Data.OleDb : It is the.NET Framework Data Provider for OLE DB. It describes a collection of classes used to access an OLE DB data source in the managed space
3. System.Data.Odbc: It is also the.NET Framework Data Provider for ODBC. It describes a collection of classes used to access an ODBC data source in the managed space. With the help of the OdbcDataAdapter class, you can fill a memory-resident dataset that you can use to query and update the data source
4. System.Data.SqlServerCE: Represents an open connection to a SQL Server Compact data source.
5. System.Data.OracleClient : It is also the.NET Framework Data Provider for oracle. The OracleDataAdapter class is used to fill a memory-resident dataset that you can use to query and update the data source.

Data Access Architecture:

Data access architecture is the part of an application's architecture that deals with storing and retrieving information. By acting as an abstraction layer between the business-logic layer and the data storage system, it enables the business-logic layer to communicate with the data storage system without needing to understand the details of how the data storage system is really implemented.

Data Access Architecture is further divided into disconnected architecture and connected architecture.
a. Disconnected Architecture
b.Connected Architecture

In disconnected architecture the features tells that It makes use of the data set component.

a. It is capable of simultaneously storing multiple tables.
b. It does not communicate with the data source it just stores the data.
c.It reads data from the data source using the DataAdapter object.

Were as in Connected Architecture it state that
a. The application establishes a link to the data store.
b. Afterward, carry on your conversations by sending and receiving SQL requests.
c. There is a delay between when the connection is established and when the programme finally breaks it.
d. The data is read using the object DataReader.

Let us discuss the various objects involved in Data Acees of ADO.NET

1. **Connection objects:**

Each interaction with the data source is represented by a separate Connection object. This could be analogous to a direct network connection in a client/server database architecture. Each given Connection object's collections, methods, or attributes could be unavailable, based on the features provided by the provider.. The connection object is the starting point for all data connections in ADO.NET. SqlConnection and OleDbConnection objects are included, among others. The SqlConnection object establishes a connection to a SQL server hosted by Microsoft, while the OleDbConnection object establishes a connection to non-SQL databases such as Microsoft Access and Oracle. After a connection is established, the application can easily exchange data with the database by using the credentials specified for the database server.

2. Command object:

Another object that is used in ADO.NET is command object which involves the use of various commands to perform various operations such as insert, delete and update. The Command Object is used to execute SQL commands like SELECT, INSERT, DELETE, and UPDATE against the database. With the command object, you may tell the database what kind of command to execute. Several execute methods are utilised by Command Object; for example, ExecuteScalar() returns the first column of the first row of a data set, ExecuteReader() displays tables on the client side, and ExecuteNonQuery() returns nothing from the database.

3. Data Reader Object:

Next object is DataReader object which is generally used to read the data from the databases during the process of data access. DataReader is an umbrella term for a class of objects in ADO.NET that share the common goal of progressively consuming data from a data source. DataReaders are a fast and effective alternative to server-side cursors, and can be compared to the Firehose cursor in ASP Classic. This data reader object is used to retrieve the outcome of a SELECT query included within a command object. There's no write support, and you can only read from one table at a time. Data can be read sequentially using the DataReadetr object.

4.Data Adapter object:

Let us understand the importance of Data Adapter Object. A dataset is populated with a sequence of data instructions and an associated database connection; this architecture style is an example of disconnected design. The method relies on the integration of various classes of data adapter with their corresponding data sources. For instance, you can use SqlDataAdapter in conjunction with SQL Server to access data. The Data adapter object has the fill() and update() methods. The rows needed for the Fill() method are retrieved from the data source using the SELECT command.

4. Data set Object:

If you need ADO.NET to work with data that is not centralised, the DataSet object is where to start. For uniformity in programming across different data sources, the DataSet stores data in memory and provides a relational representation of that data.Data set also provides Read and write access and supports the integration with XML. A data set is a typical building block in decentralised systems. The Dataset Object is comprised of the Data Table and the Data Relations. The components of a data table are the columns of data, the rows of data, and the constraints that govern the data. It's purpose is to hold many data tables from many sources simultaneously. The primary issue with using this object is that it is noticeably slower than the data reader object while reading data

5. Command Builder object:

The data adapter utilises the commands generated by the command builder to reflect any modifications made to a DataSet in the original data source. All updates in this class must be made to a single table using SQL statements. Using ADO.NET, the DataSet object is your first port of call for dealing with decentralised data. The DataSet keeps data in memory and offers a relational representation of that data so that programming can be consistent across many data sources.The syntax of Command builder object is bldr = new SqlCommandBuilder(da); In addition, it adjusts the information depending on just one table.

Features of ADO.NET
Let us discus about the features of ADO.NET

- To put it simply, ADO.NET is an ADO.NET extension model that operates largely as a decoupled architecture.
- ADO.NET uses the XML format to facilitate the transfer of data across systems. It outlines the steps necessary to construct dependable data access apps rapidly and with minimal effort.

•To share data through disparate environments, ADO.NET utilizes the XML format.

•It explains how to quickly and easily create reliable data access applications.

Operation of Database:

The purpose of operation Database. The term "operation databases" refers to a database's ability to carry out tasks like "select," "insert," "update," and "delete.“Data in operational databases can be updated instantly. When working with SqlDataAdapter, using SqlCommandBuilder eliminates the requirement to provide an appropriate SqlCommand. However, there are constraints on how SqlCommandBuilder can be used. If the tables within a DataSet are related to one another, it will not provide SqlCommands. In order to avoid corrupting the table, ensure that you are passing the correct SqlCommand to the SqlDataAdapter. If, for the sake of argument, you erase one row of data from a DataSet and then provide

The importance of Operation Databases: One of the primary uses for databases that track operational data is to supply decision-makers with up-to-the-moment insights. Operations databases are useful because they securely store data and keep transactions apart in an asynchronous states. One further perk is that you can work on multiple programmes simultaneously while being in separate locations. Operations databases are useful because they store data reliably and keep transactions in a parallel state.

Steps for Creating Operation Databases:
Step1: Data set Creation
Step2 Database Updating
step 3: Connection Establishment and final step is Filling the data , Let us see each steps in detail in the coming sessions

1. Data set creation : After this command, a dataset can be created and populated with information collected from various sources.The formula:
 objDataset = new DataSet("OrderId");

2. Database updating: Afterwards, use SqlCommand to modify the DataRow.A new version of the student's SET has been '" + strName + "'" + Fname + " FName in objRow = strName;

3. Connection establishment: [illegible]e follo[illegible] comm[illegible]s are u[illegible] a connect[illegible] [illegible]ver
[illegible]on= "[illegible]ser[illegible] [illegible]word = [illegible] objSq[illegible]nectio[illegible]
[illegible]ta[illegible] U[illegible]ng [illegible]ql Com[illegible]d and [illegible]ataAd[illegible]
[illegible]der = "Select [illegible]rderI[illegible]
[illegible] objcomm[illegible]d = [illegible]nd([illegible]r, ob[illegible]ne[illegible]on);

9.2 Record Navigation

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

(i) Move Back One Record at a Time

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

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

```
If inc > 0 Then
inc = inc - 1
NavigateRecords()
Else
```

```
MsgBox('First Record')
End If
```

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

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

```
If inc > 0 Then
inc = inc - 1
NavigateRecords()
ElseIf inc = -1 Then
MsgBox("No Records Yet")
ElseIf inc = 0 Then
MsgBox('First  Record')
End If
```

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

(ii) Moving to the Last Record in the DataSet

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

```
If inc <> MaxRows - 1 Then
inc = MaxRows - 1
NavigateRecords()
End If
```

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

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

(iii) Moving to the First Record in the DataSet

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

```
If inc <> 0 Then
inc = 0
NavigateRecords()
End If
```

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

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

Self Assessment Questions

1. The first record in the data set starts at_______________.
2. Process of browsing or moving through the data in the data set is called as___________________.

9.3 Add, Update and Delete Records

In the previous section, you learned how to move through the records in your DataSet, and how to display the records in Textboxes on your form. We also see how to add new records, how to delete them and how to Update a records.

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

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

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

Change the **Text** properties of the buttons to "**Add New Record** ", "**Commit Changes**", "**Update Record** ", "**Delete Record**", and "**Clear/Cancel**". Your form might look as depicted in figure 9.1.

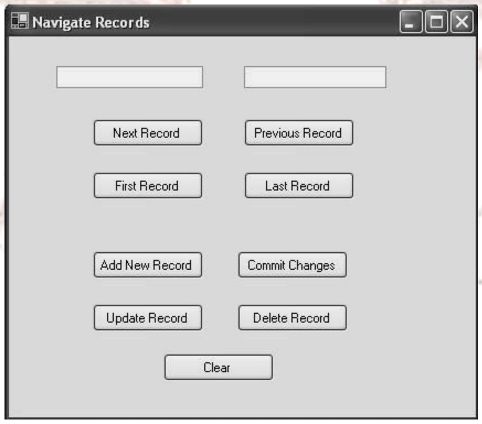


Fig. 9.1: Form design with updating controls

9.3.1 Updating record

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

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

This will retrieve the data available from Item 1 on Row 2.

You can also set a value with the following syntax

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

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

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

Execute your application, and click the **Next Record** button to move to the first record. "John" should be displayed in your first textbox, and "Smith" in the second textbox. Click inside the textboxes and change "John" to "Joan" and "Smith" to "Smithy". Now you can click your **Update Record** button. Move to the next record by clicking your **Next Record** button, and then move back to the first record. You should see that the first record is now "Joan Smithy".

Close the application and run it again. Click the **Next Record** button to move to the first record. It will still be "John Smith". The data you updated has been lost: it is because

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

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

```
Dim cb As New OleDb.OleDbCommandBuilder(da)
ds.Tables("AddressBook").Rows(inc).Item(1) =
txtFirstName.Text
ds.Tables("AddressBook").Rows(inc).Item(2) =
txtSurname.Text
da.Update(ds, "AddressBook")
MsgBox("Data updated")
```

The first new line is this:

Dim cb As New OleDb.OleDbCommandBuilder(da)

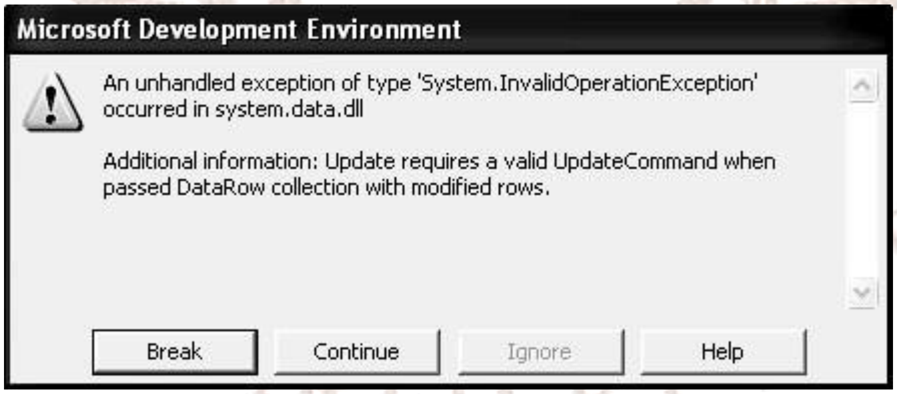
To update the database itself, you need something called a **Command Builder**. The Command Builder will build a SQL string for you. In between round brackets, you type the name of your Data Adapter, **da** in our case. The command builder is then stored in a variable, which we have called **cb**. The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

da.Update(ds, "AddressBook")

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

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

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


Microsoft Development Environment

An unhandled exception of type 'System.InvalidOperationException' occurred in system.data.dll

Additional information: Update requires a valid UpdateCommand when passed DataRow collection with modified rows.

Fig. 9.2: Error Message

The error is because you have not got a command builder - a Valid Update Command. Delete the comment from your Command Builder line and the error message goes away. You should now be able to make changes to the database itself (as long as the Access database isn't Read Only).

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

9.3.2 Adding a new record

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

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

```
btnCommit.Enabled = True
btnAddNew.Enabled = False
btnUpdate.Enabled = False
btnDelete.Enabled = False
txtFirstName.Clear()
txtSurname.Clear()
```

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

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

```
btnCommit.Enabled = False
btnAddNew.Enabled = True
btnUpdate.Enabled = True
btnDelete.Enabled = True
inc = 0
NavigateRecords()
```

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

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

```
If inc <> -1 Then
Dim cb As New OleDb.OleDbCommandBuilder(da)
Dim dsNewRow As DataRow
dsNewRow = ds.Tables("AddressBook").NewRow()
dsNewRow.Item("FirstName") = txtFirstName.Text
dsNewRow.Item("Surname") = txtSurname.Text
ds.Tables("AddressBook").Rows.Add(dsNewRow)
da.Update(ds, "AddressBook")
MsgBox("New Record added to the Database")
btnCommit.Enabled = False
btnAddNew.Enabled = True
btnUpdate.Enabled = True
btnDelete.Enabled = True
End If
```

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

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

Dim dsNewRow As DataRow

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

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

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

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

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

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

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

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

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

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

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

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

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

da.Update(ds, "AddressBook")

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

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

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

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

9.3.3 Deleting a record

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

```
Dim cb As New OleDb.OleDbCommandBuilder(da)
ds.Tables("AddressBook").Rows(inc).Delete()
MaxRows = MaxRows - 1
inc = 0
NavigateRecords()
da.Update(ds, "AddressBook")
```

Here you should first set up a Command Builder. Then we have this line:

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

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

```
Rows(inc)
```

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

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

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

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

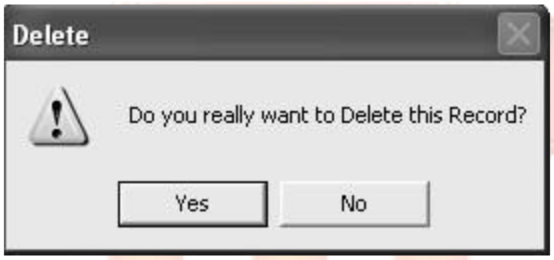
The **MaxRows** line in the code just deducts 1 from the variable. This just ensures that the number of rows in the DataSet matches the number we have in the MaxRows variable.

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

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

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

Another thing you can do is to display a message box asking users if they really want to delete this record as shown in figure 9.3.



Do you really want to Delete this Record?

Fig. 9.3: Delete confirmation window

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

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

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

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

MessageBox.Show()

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

MessageBoxButtons.YesNo

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

After typing a comma, we selected the **MessageBoxIcon**. Warning icon from the popup list. But you need to check which button the user clicked. This is done with this:

= DialogResult.No

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

The code for the If Statement itself is this:

MsgBox('Operation Cancelled')
Exit Sub

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

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

Data Grid Control:

Data grid control allows you to present data in a tabular format in an efficient and versatile way. The DataGridView control can be scaled to display editable views of very large collections of data, or it can be used to show read-only views of a limited amount of data.
Data grid control is a flexible and powerful tool for displaying data in a tabular style. The DataGridView control can be adjusted to show either a little preview of data or a huge editable view of a massive dataset. The DataGrid control, when bound to a data source, will populate itself with columns and rows in the appropriate format.
The Data Grid control renders an adaptable data table. The DataGrid class has properties such as DefaultCellStyle, ColumnHeadersDefaultCellStyle, CellBorderStyle, and GridColor that allow you to customise cells, rows, columns, and borders

Features of Data grid control:

DThe following are the different features of Data Grid Control.

- With the Rows and Columns attributes, you may add data-containing columns and rows to the DataGridView without referencing a data source.
- Both the DataGridViewRow and objects of type DataGridViewRow can be accessed through Rows. Use the Cells property to access and modify cell values without going through the DataTables property.
- Direct cell checks are also performed by the Item[] indexer.
- Use the Cells property to access and modify cell values directly. An additional function of the Item[] indexer is direct cell checking.

Ways to set the data grid control:

Data binding with data control comes under the complex data binding; it is an association between the control and one or more data elements of the data source. You can perform this type of binding with controls that allow more than one value to be displayed at one time, such as a data grid or a data listing this slide let us understand ways to Set Data Grid Control, there are two ways to set the data grid control, The first is to modify the control's "Columns" group in the properties and make the columns static. the second way , that is the alternative is to dynamically manipulate columns in code.

1. First way is to adjust static columns in the control's properties by filling the "Columns" group.
2. Second way is to write code that performs dynamic column manipulation.

Process to add a Grid control explained here. This data grid also works similar to the Text box control. Once you established a connection with the dataSource, you populate a dataset from the customers table using a data adapter. Next, you need to bind the DataGrid control to the newly populated dataset.

We are doing this with the following syntax:

DataGrid1.SetDataBinding(ds, “customers”)
Drag and drop a DataGridView control, as well as a button, into the windows form project form to open it.
Change the "Text" property for the button to "Refresh" by increasing the size of the DataGridView control and changing the "Text" property.
Change the button's "Name" property to "btnRefresh" and the DataGridView's name to "grdData" to make the code look cleaner and more adaptable.

Process of Retrieval of Data with the Data Grid Controls

DataGrid control to the newly populated dataset.

We are doing this with the following syntax:

DataGrid1.SetDataBinding(ds, "customers")
The SetDataBinding method of the DataGrid control accepts two parameters: the data course object and a string literal that describes a data member in the data source. Here, a data source can be any object that's capable of holding data, such us an array or table. The data member describes what element to bind to the control.

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

<u>Query Commands for Retrieving Sample Data</u>

<u>Running Query with Sample Database to Retrieve Data</u>

Create a SqlCommand object that will define the SQL query I will run against my SQL Server connection. No "old" data access classes are required. Alternatively, we might utilise the "SqlDataAdapter" and "Dataset" classes. My SqlDataAdapter object will run the database and populate a DataTable within a Dataset object. The final step of the code was to tell the DataGridView control to use the table in the Data Set. This final step generates the columns for the DataGridView control and fills it with the results of the query.

Customizing the data grid control steps shown here.

ol require

- Second, once you've selected the DataGridView control iBoth the background colour and grid colour can be modified.n the workspace, you may play about with its settings.

Virtual mode Property to display data

The DataGridView control's interaction with a user-defined data cache can be controlled and optimized with the help of virtual mode. Only when bound mode needs to be supplemented or replaced can virtual mode be used. When the DataSource property is set, the control switches to bound mode and retrieves data from the source you specify. Setting the VirtualMode property to true and responding to one or more of the events detailed above will enable virtual mode. The CellValueNeeded event, which triggers a lookup in the control's data cache, is the bare minimum you can expect to handle. While working with massive amounts of data, you can narrow your focus by using the Virtual Mode option. While working in virtual mode, the DataGridView control must be pre-populated from a data cache.

Self Assessment Questions

3. DataSet needs to be disconnected before you start coding for manipulation process. State [True/False].
4. Write the syntax to refer a data available in item 1 and row 2.
5. ____________is require to update the data base itself.
6. Before commit a new row into the database it is mandatory to add the new row in the data set. State [True/False].

9.4 SQL Server and ADO .NET

Now we are going to discuss the behavior and features that are related to ADO .NET data provider for SQL server. The basic root namespace for .NET is the System.Data. This supports the DataTable and DataSet objects to work with any data type. Apart from this .Net supports two more namespaces for retrieving or accessing the data.

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

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

SQL Connection class

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

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

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

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

SQLDataAdapter

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

The DataAdapter class is not inheritable Declaration is

*Punlic NotInheritable Class SqlDataAdapter Inherits DbDataAdapter_*

Implements IDbDataAdapter, IDataAdapter,ICloneable

DataSet class

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

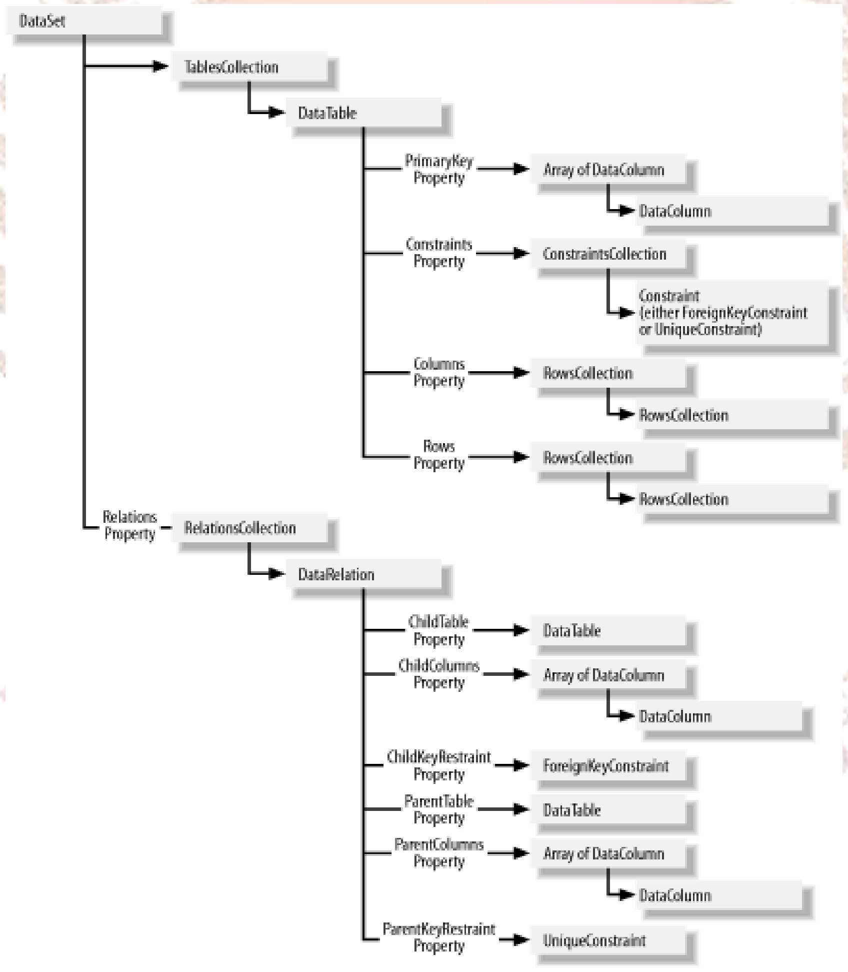


Fig. 9.4: Model diagram for DataSet and related classes.

Self Assessment Questions

7. ________________is the basic root namespace for .NET.
8. SqlConnection class can be inherited and establish an open connection. State [True/False].
9. DataAdapter class will be instantiated generally by the ________________________.

9.5 Data Binding

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

9.5.1 Data binding with TextBox

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

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

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

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

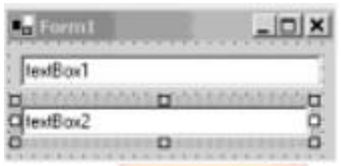


Fig. 9.5: TextBox Controls in form

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

```
Private Sub Form1_Load(ByVal sender As System.Object, - Byval e
As System.EventArgs) Handles MyBase.Load
Dim thisConnection As New SqlConnection_
(“Server=(local)\netsdk;” &_”integrated security=sspi;”&_
“database=northwind”)
‘ Sql Query
Dim sql As String = “SELECT * FROM Employees”
‘ Create Data Adapter
Dim da As new SqlDataAdapter(sql, thisconnection)
‘ Create and fill DataSet
Da.Fill(ds, “Employees”)
‘Bind to first name column of the employee table
TextBox1.DataBindings.Add(“text”, ds, “employees.firstname”)
‘Bind to lastname column of the employees table
TextBox2.DataBindings.Add(“text”, ds, “employees.lastname”)
End Sub
```

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

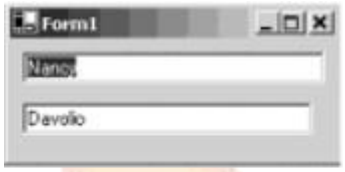


Fig. 9.6: DataBinding text boxes bound to data columns

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

9.5.2 Data binding with Data grid

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

Follow the steps below to develop complex data binding application

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

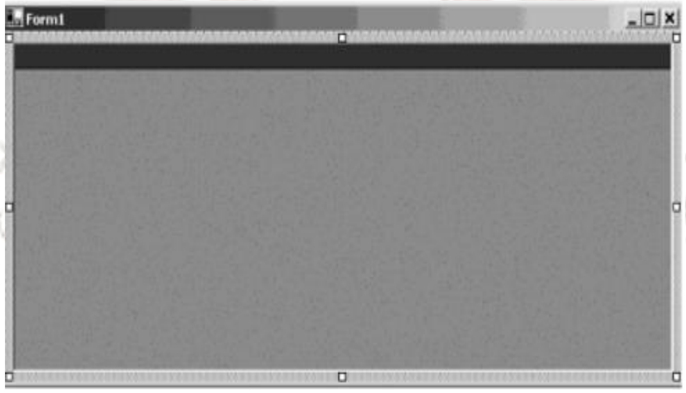


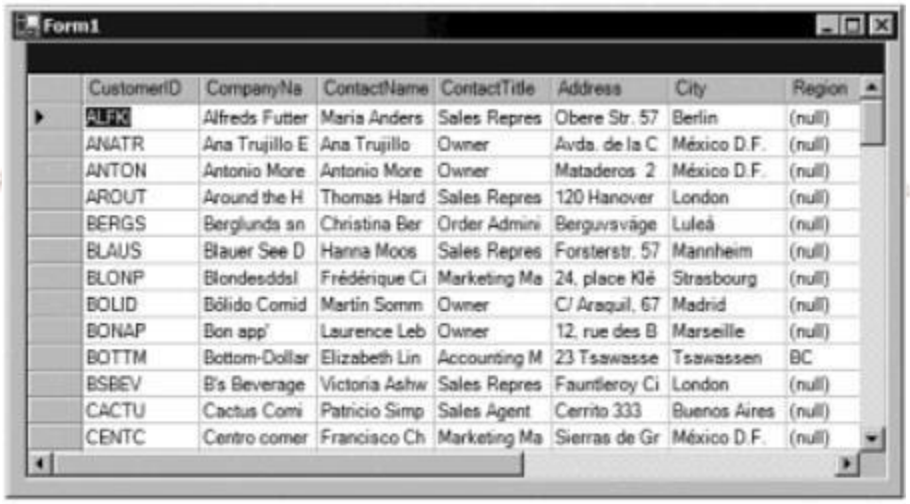
Fig. 9.7: Data grid control

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

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

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

‘ Sql Query
Dim sql As String = “SELECT * FROM Employees”

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

‘ Create and fill DataSet
Da.Fill(ds, “Employees”)

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

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



	CustomerID	CompanyNa	ContactName	ContactTitle	Address	City	Region
▶	ALFKI	Alfreds Futter	Maria Anders	Sales Repres	Obere Str. 57	Berlin	(null)
	ANATR	Ana Trujillo E	Ana Trujillo	Owner	Avda. de la C	México D.F.	(null)
	ANTON	Antonio More	Antonio More	Owner	Mataderos 2	México D.F.	(null)
	AROUT	Around the H	Thomas Hard	Sales Repres	120 Hanover	London	(null)
	BERGS	Berglunds sn	Christina Ber	Order Admini	Berguvsväge	Luleå	(null)
	BLAUS	Blauer See D	Hanna Moos	Sales Repres	Forsterstr. 57	Mannheim	(null)
	BLONP	Blondesddsl	Frédérique Ci	Marketing Ma	24, place Klé	Strasbourg	(null)
	BOLID	Bólido Comid	Martín Somm	Owner	C/ Araquil, 67	Madrid	(null)
	BONAP	Bon app'	Laurence Leb	Owner	12, rue des B	Marseille	(null)
	BOTTM	Bottom-Dollar	Elizabeth Lin	Accounting M	23 Tsawasse	Tsawassen	BC
	BSBEV	B's Beverage	Victoria Ashw	Sales Repres	Fauntleroy Ci	London	(null)
	CACTU	Cactus Comi	Patricio Simp	Sales Agent	Cerrito 333	Buenos Aires	(null)
	CENTC	Centro comer	Francisco Ch	Marketing Ma	Sierras de Gr	México D.F.	(null)

Fig. 9.8: ComplexBinding data grid bound to data table

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

DataGrid1.SetDataBinding(ds, "customers")

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

Self Assessment Questions

10. ______________________ binding is called the one-one binding happens with the individual controls.
11. Write the syntax to bind the column x from the connection string ds tot the control TextBox1.

9.6 Summary

- Navigation of records means browsing through the data set. It does four different operations like moving first, moving last, moving next and moving previous.
- Dataset are considered as the temporary buffer to hold the data with or without connection. So the changes whatever happened will not be reflected until you do explicitly.
- System.Data.OleDb and the System.Data.SqlClient are the two .NET namespaces that support the SQL Server data access.
- Simple and complex binding can be done to display the single or group of data in the VB .NET controls.
- Simple data binding can be done through the TextBox controls and the complex data binding can be achieved with grid controls.

9.7 Terminal Questions

1. List and explain the navigation techniques with an appropriate example.
2. Explain the various data manipulation operations in the dataset.
3. Discuss the following a. SQLDataAdapter b. DataSet class
4. Explain data binding with TextBox control with example.
5. Brief the complex data binding with the data grid control.

9.8 Answers

Self Assessment Questions

1. Zero
2. Navigation
3. True
4. ds.Tables("AddressBook").Rows(2).Item(1)
5. Command builder.
6. True.
7. System.Data
8. False.
9. Constructor
10. Simple
11. TextBox1.DataBindings.Add("text",ds,"x")

Terminal Questions

1. Move First, Move Last, Move Next and Move Previous are the four different operations can be done for record navigation. For more details refer section 9.2.
2. Addition, deletion and updation are the three important manipulations we can do with the dataset. For more details refer section 9.3.
3. The **DataAdapter** is an interface between the database and the dataset. **DataSe**t class consists of information about the set of tables and the relation among those set of tables. For more details refer section 9.4.
4. TextBox control deals with the simple data binding where it binds the data between data set and control on one-to-one basis. For more details refer section 9.5.
5. We call complex data binding where we have list of data needs to be bind with the control, this generally happen with grid controls. For more details refer section 9.5.

E-Reference:

- http://www.vkinfotek.com/dataset.html
- http://msdn.microsoft.com/en-us/library/system.data.sqlclient.qldataadapter. aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1
- http://www.codeproject.com/Articles/20326/ADO-NET-Data-Access-Component-for-SQL-Server-in-Cs
- http://my.safaribooksonline.com/book/web-development/microsoft-aspdotnet/0672323575/introduction-to-adodotnet/ch10lev1sec2