Interacting with a Database Using Visual Basic.NET

This exercise is designed to give students exposure to how computer programs are created and how they can be made to interact with a database. In this exercise, students will use Visual Basic.NET (VB.NET) to create a graphical user interface (GUI), along with the necessary programming code so that objects on the GUI can interact with a database and its contents (display, modify, and manipulate data). Students will accomplish these tasks as they work through a step-by-step tutorial that explains the Visual Basic.NET programming environment and the elements used to create a GUI capable of interacting with a database, along with features commonly found in computer programs such as IF statements, loops, variables, and other elements of computer programming. This activity must be completed in the CBA computer lab or on some other computer that already has Visual Studio/VB.NET installed.

A database management system (DBMS) is used to create and maintain a database. Some of the most popular DBMSs include Microsoft Access, Oracle, and SQL Server. After the database has been created, a number of different programs can be created to access and interact with the data contained in a database, and we are going to create a program capable of doing just that.

Database Terminology and Refresher

In this tutorial, you will learn how to access the data in a Microsoft Access database, and the first step in writing a program that is able to access and interact with a database is to understand how the database is arranged. Databases created by Microsoft Access are relational databases. In a relational database, information is stored in one or more tables, with each table containing a group of related records. For example, a fitness club’s database might contain an employees table, where information concerning all of the individuals employed by a fitness club would be stored. That same database might also contain a courses table, where information concerning the courses offered by a fitness club would also be stored. Rows within a table are records, which are the same as file folders in a manual storage system. All of Brian Strait’s information in the Employees table represents a record. Columns within a table represent attributes, or characteristics, of individual records. Each employee record is made up of the following attributes: EmployeeNumber, InstructorLast (name), InstructorFirst (name), HomePhone (number), SalaryAmount, and DateHired.

<table>
<thead>
<tr>
<th>EmployeeNumber</th>
<th>InstructorLast</th>
<th>InstructorFirst</th>
<th>HomePhone</th>
<th>SalaryAmount</th>
<th>DateHired</th>
</tr>
</thead>
<tbody>
<tr>
<td>654-96-3332</td>
<td>Strait</td>
<td>Brian</td>
<td>(812) 330-3750</td>
<td>$44,275.00</td>
<td>12/10/1997</td>
</tr>
<tr>
<td>655-91-1906</td>
<td>Nicklaus</td>
<td>Jack</td>
<td>(459) 146-6756</td>
<td>$40,000.00</td>
<td>9/21/1995</td>
</tr>
<tr>
<td>732-44-2975</td>
<td>Arston</td>
<td>Jean</td>
<td>(812) 335-5591</td>
<td>$39,000.00</td>
<td>12/6/1995</td>
</tr>
<tr>
<td>801-73-8812</td>
<td>Armstrong</td>
<td>Lance</td>
<td>(316) 532-8843</td>
<td>$41,500.00</td>
<td>1/8/2001</td>
</tr>
<tr>
<td>901-43-7611</td>
<td>Brewski</td>
<td>Randy</td>
<td>(612) 945-6987</td>
<td>$43,500.00</td>
<td>1/10/1996</td>
</tr>
</tbody>
</table>

An Introduction to ADO.NET

In order to read information from and write information to a database, Visual Basic.NET uses a Microsoft technology called ActiveX Data Objects.NET (ADO.NET). While prior versions of ADO and Visual Basic created a connection to a database that remained open the entire time the application was running, ADO.NET creates only a temporary connection that opens and closes as needed. Using ADO.NET, when an application first connects to a database, it makes a copy (called a dataset) of the records and fields that are going to be used by the application. The dataset is stored in the computer’s main memory, and both the connection to the database and the database itself are closed. The application will then make whatever changes are needed to the dataset. When changes to the dataset are to be saved, the application then reconnects to the database. For a VB.NET application to connect to a database, three ADO.NET objects (described in more detail later on) are needed (a Data Adapter object, a Connection object, and a Dataset), as well as a provider that is able to translate requests for data into a language that the database can understand.

In general, for a VB.NET application to be able to work with a database, three steps must be followed (see figure below):

1. Create a connection to a Data Source (Database)
2. A Data Adapter must also be created to facilitate the transfer of information back and forth from the data source to the computer application. All data transfer must occur through the data adapter.
3. A DataSet (internal memory data holding area) must be created to hold the data as the application is working with it.
These concepts also fit the idea of program and data independence. You should be able to change the program without impacting the database, and you should be able to change the data/database without impacting the program. Because our data files and programs are separate, different programs can interact with and share the same datasets.

Since your health club has just opened for business, and since you have been very busy hiring employees and working with members of your health club, you have not had much time to computerize your information and develop effective and efficient information systems. In fact, your current “database” is simply an Access file containing one table, and that table holds information about your employees. Since that is all that you have at the moment, we will begin with that one table and create a simple application using Visual Basic.NET and ADO.NET so that we can interact with the employee data.

Getting Started in Visual Basic.NET

VB.NET is one of the easier programming languages to learn, and VB.NET lets us create and code GUIs, or graphical user interfaces. In a graphical user interface environment (such as Windows and most of today’s other operating systems), we have windows, icons and other pictures/elements that we work with. We can click or double click on a picture or icon, and the computer does something. When we create a computer application, such as our program, there are two primary components that we create:

1. The graphical user interface (what the user sees and works with)
2. The actual program code, which is a set of instructions that tells the computer how to perform a given task. A computer does not know what to do unless there is some sort of program or code that tells it exactly what must be done and how it is to be done.

It is now time to open Visual Basic and begin creating our program. Prior to beginning this activity, you should have obtained several folders. Please open the folder called DataGrid Example. You should see four items in Windows Explorer (see picture)

HealthClub is the Microsoft Access database that we are going to make a connection to. Notice the file called DataGrid Example, Visual Studio Solution. A Visual Basic Solution file is simply a container file that stores all of the projects and files for an entire application. Double click the DataGrid Example solution file. If you ever close down the Visual Basic.NET program in the middle of a project, when you wish to resume your work, you will need to double click the solution file. It is always the file that you open, and opening that file gives you access to all of the other elements that make up your application.

Double clicking the/a solution file allows a programmer to enter the Visual Basic.NET programming environment, as seen below. Your screen should look similar to the picture below, although there is the possibility that items that you see in the picture might not be visible, or they may appear in other locations. (What you do or don’t see on your screen is largely dependent on the Visual Basic screen setup used by the last person using the computer that you now are using).
Notice the large 1 (one) on the right hand side of the picture above. The “1” represents the location of the Solution Explorer Window. (If you do not see a Solution Explorer Window on your screen, go to the View Menu, and select Solution Explorer to see the Solution Explorer Window.) The Solution Explorer Window displays a listing of all of the items contained within your current solution (DataGrid Example). References and AssemblyInfo are two files that VB.NET created automatically related to your present project. frmDataGrid.vb is a container control that holds other controls (Items found on the interface/GUI such as buttons, labels, textboxes, etc) and the event procedures (i.e. coding) that tells the application how to respond when certain events occur. Notice that frmDataGrid.vb is highlighted in the Solution Explorer window, with the design screen/GUI for the Data Grid Example appearing to the left of the Solution Explorer Window (See large “2” on picture). If you do not see the form labeled “2” on your screen, then double click frmDataGrid.vb in the Solution Explorer.

The large 3 (three) on lower right hand side of the picture identifies the Properties Window. (If you do not see a Properties Window on your screen, go to the View Menu, and select Properties Window to see the Properties Window.) Each object/control in Visual Basic.NET has a set of characteristics, called attributes or properties, which are assigned to it. The properties which determine an object’s appearance and behavior are listed in the Properties Window. The picture above shows the properties for the form called frmDataGrid. If another control were selected (rather than the actual form itself), then its properties would display instead.

The large 4 (four) appears to the right of the Toolbox. The Toolbox itself appears at the end of this paragraph. The Toolbox is typically found on the left-hand side of the screen, but it can be located elsewhere as well. (If you do not see a Toolbox on your screen, go to the View Menu, and select Toolbox to see the Toolbox. Clicking on the button (shown in the previous picture) that looks like a hammer and wrench also makes the Toolbox appear and disappear.) The Toolbox contains a collection of tools that you can use when designing the GUI/interface for an application. You use the Toolbox to place objects, or controls, on your form.

The tools/objects/controls that we will use in this exercise include:

- The label: used to simply display text.
• The textbox: used to both display and modify data.
• The button: used to initiate processing tasks (the user clicks on the button).
• The datagrid: used to display data in a series of rows and columns.

Notice the arrow leaving from the textbox where it says: “View Code: click to view the code on a form.” Click the button that the arrow points to, and you should be able to see the code that is currently on that form. At the moment, the only code on the form is for the Exit button.

Notice that the next button (on the right of the View Code button) is the View Designer button. You can click on that to see the interface/GUI of the form, which is what is currently displayed in the picture seen on the prior page. Click on both of those buttons to see what they do. When you are done clicking those buttons, you should see the GUI/interface, as seen in the large picture on the prior page.

Since you want to display the contents of a database in your Visual Basic.NET program, we will now start to make the necessary connections to access the data stored in a Microsoft Access database.

Building a Data Adapter
A Data Adapter facilitates the transfer of information back and forth from the data source to the computer application. All data transfer must occur through the data adapter. VB.NET provides a wizard to help programmers create a Data Adapter, which we will now begin using.

Using this tutorial’s figures, locate the Toolbox. Toward the top of the Toolbox, the word “Data” appears. Click this Data tab, and the various data-related tools/controls appear, as seen pictured.

Place your cursor over OleDbDataAdapter (you may need to widen out the data area of the toolbox). Hold down the left mouse button while clicking OleDbDataAdapter and then drag your mouse over to the form (usually appearing to your right). When you release the mouse (after having dragged the OleDbDataAdapter over to the form), VB.NET starts the Data Adapter Configuration Wizard. Please read the Welcome screen, and then click Next to continue.

After clicking Next, the “Choose your Data Connection” screen appears. Since we don’t have an existing data connection, click the New Connection button to create a new data connection. The Data Link Properties dialog box should now open. Click the Provider tab. The Provider tab is used to specify type of provider associated with the database that you wish to connect to. The provider for all Microsoft Access databases is named Microsoft Jet 4.0 OLE DB Provider. Select that option, and then click Next.

Clicking Next should result in the Connection tab being displayed. Click the … button (the 3 dots) next to the “Select or enter a database name” textbox. The Select Access Database dialog box appears. Find and select the database called HealthClub, and then click the Open button. You might now see a very extensive path appearing, such as C:\Documents and Settings\username\Desktop\DataGrid Example\HealthClub.mdb or something very similar (depending on where you have the files that you are working with stored). Please remove/delete the entire pathway, except for HealthClub.mdb, as seen in the next picture. If you leave the entire pathway there, as initially shown, and if you copy your files to a new
location, then you might find that a database connection that previously had been working is now not working, due to the files being copied to a new location. If the database file and the VB.NET files are all in the same storage location, as you will probably find when you are working on this tutorial, then you can copy the entire set of files over to a different location (on your hard drive or removable storage device) and the data connection will still work, as opposed to having the connection only work if your files remain in that exact storage location, but having it fail if you copy your files over to another folder (on your hard drive) or to a removable storage device.

The Connection tab also contains a Test Connection button, which you can use to test the connection to the database. If the connection is successful, then a message “Test connection succeeded” appears. If the connection is not successful, an error message appears, telling you that the test connection failed. A test connection will fail if the user did not enter the correct provider, or if the database name was not correct. If the test connection is successful, then click ok in response to the message box, and you should return back to the Connection tab. From there, click Ok, and you should return to the Choose Your Data Connection dialog box.

Make sure that ACCESS.HealthClub.mdb.Admin appears as the data connection. If it does not appear, click the drop-down option to specify that you want to use the ACCESS.HealthClub.mdb.Admin connection that you just made. Then, click Next.

The “Choose A Query Type” screen should now appear, which is used to specify how the data adapter should access the database. Select “Use SQL statements.”

SQL, pronounced sequel, is a set of commands used to access and manipulate the data stored in many database management systems. SQL commands can be used to store, retrieve, update, delete, and sort data. SQL syntax includes words such as SELECT, FROM, WHERE, and ORDER BY. SELECT allows the user to specify the fields that should be displayed. FROM is used to specify the table or tables that the selected fields will be taken from. WHERE is used to limit the records that are displayed if the user wants to display only the records that meet certain criteria. ORDER BY allows the user to control the order in which the selected records are displayed. Both WHERE and ORDER BY are optional SQL commands. The SQL syntax of a SELECT statement is:

```
SELECT fields FROM table [WHERE condition] [ORDER BY field]
```

After selecting “Use SQL statements”, click Next. The “Generate the SQL statements” screen should appear where the user is then asked “What data should the data adapter load into the dataset?” If the user knows SQL, the appropriate SELECT statement can be entered, or the user can use the Query Builder. Since you want to use the Query Builder, go ahead and click the Query Builder button, and the Add Table dialog box should appear, as seen in the next picture.

Since your health club database only contains one table, you should see just one table, tblEmployees. Click tblEmployees, click Add, and then click Close.

The fields found in the Employees table should now display. Notice that the field names are listed in alphabetical order, which is not the order that the fields are actually listed in the table. Notice that the Query Builder has already entered the FROM statement for you, and as you click the check box next to a field name, that particular field name
Click the check box next to the following fields (in this order): EmployeeNumber, InstructorLast, InstructorFirst, HomePhone, SalaryAmount, DateHired. Then, click Ok.

When a field’s check box is selected, the corresponding field name appears in the SELECT statement, as seen pictured.

Once the six required fields are selected, click ok, and the “Generate the SQL statements” screen should now appear, showing the SQL statement created by the Query Builder. Click Next, and the View Wizard Results screen should appear. If everything is working properly, then the Wizard should display the following sentence: “The data adapter “OleDbDataAdapter1” was configured successfully.”

Click the Finish button to close the Data Adapter Configuration Wizard dialog box. The final action of the Data Adapter Configuration Wizard is to add an OleDbDataAdapter1 object and an OleDbConnection1 object to the component tray, which is typically located at the bottom of the screen, as seen below.

One unique feature of VB.NET is that it has a Save All button, in addition to the traditional Save button. The Save All button saves all elements that make up your VB.NET project, as opposed to the traditional Save button that just saves the work on the form that the user is currently working on. Save All is usually located to the immediate right of the traditional Save button. Please click the SAVE ALL button now.

The Data Adapter has now been created and configured. The next step to accessing data stored in a Microsoft Access database is to create a dataset.

Creating and Filling a Dataset
A dataset is simply an object that represents the data that you want to access. Verify that OleDbDataAdapter1 is the currently selected object by clicking on it once. The Properties window, seen to the right in the prior picture, should display OleDbDataAdapter1. Immediately below the properties of OleDbDataAdapter1, the words “Generate Dataset” appear. Click Generate Dataset, and the Generate Dataset dialog box should appear. Click the New radio button, if it has not already been selected for you. In the textbox next to New, type in: dsEmployees as the name for this new dataset. The ds is a prefix that we will use to identify and represent a dataset in the coding that we will create. Click the Ok button, and a dataset object should appear in the component tray, found at the bottom of your screen. You should also find that the dataset object is highlighted in the component tray, with its properties also displaying in the Properties window to the right. At the moment, the dataset object probably does not have the name that you specified just a moment ago, or the same capitalization. Change the (Name) property to dsEmployees, press the Enter key, and the
(Name) property of the dataset should display **dsEmployees**. The dataset object, as seen in the component tray, should also read **dsEmployees**, as seen in the next picture. Notice that the Solution Explorer Window also displays a new file called **dsEmployees.xsd**. The .xsd extension of the file indicates that the file is an XML schema definition file, which defines tables and files that make up a dataset.

The next step is to use the data adapter’s Fill method so that when the application runs, the dataset will have data in it. Up to this point, we have seen that an object can have both properties and methods. Properties are the attributes or characteristics that describe something, and they can be found in the Properties Window. Just a short time ago, you changed the (Name) property of the dataset. Methods are actions that can be taken by an object. The Fill method is used to populate the dataset with actual data from the database.

VB.NET is an “event-driven” language because when the user does something or when some predetermined action occurs (such as clicking a button or some other type of activity), its associated event procedure (coding) begins to execute. When the Exit button is clicked, its coding (i.e. event procedure) tells the application what to do. While an event procedure can be written to respond to the actions of a user, an event procedure can also be written so that certain instructions (i.e. coding) will be executed when a form appears on the screen. This type of coding is referred to as a form’s load event procedure because these instructions will be executed when a form loads or appears, and the Fill method (used to populate a dataset) is usually found in a form’s load event procedure.

Click the form object (the dotted area somewhere above the exit button). This selects the form. Click the View Code button (please refer to first VB.NET labeled screenshot diagram for the location of the View Code button).

Click the Class Name list dropdown (circled in picture to the right) and then click (Base Class Events). The Method Name list (to the right of the Class Name list) should now switch to Declarations. Click the list dropdown. Scroll down until you see Load, and then click on Load.

A new event procedure, beginning with the words of “Private Sub frmDataGrid_Load” should appear, and the cursor should be located within that procedure.

Type in the following code:

```
Me.
```

As seen in the picture to the right, as soon as you type in the period (dot), the VB IntelliSense list appears; showing possibilities for what might come next in your coding. IntelliSense helps the program developer by checking spelling and providing suggestions on what to include in a statement.

Type in a capital **O** next, so that the line reads **Me.O**

As you type in the capital O, notice IntelliSense has highlighted OleDbConnection1. While that is not the next item that needs to be added to the coding, notice that the item right below it is what you wish to add to your code next (OleDbDataAdapter1). Rather than typing in additional letters, the programmer can simply use the down arrow or mouse to select that item, and then once the space bar is pressed,
IntelliSense will add that item (OleDbDataAdapter1) to the line of code, so that it now reads as:

\texttt{Me.OleDbDataAdapter1}

You should see a space between the 1 and the cursor. Use the backspace key to move the cursor so that it is right next to the 1. Press the period/dot and the IntelliSense listing appears once again. Type in a capital F. When IntelliSense highlights the word \texttt{Fill}, then press the spacebar to add Fill to the line of code, so that it now reads as:

\texttt{Me.OleDbDataAdapter1.Fill}

Continuing with that line of code, type in \texttt{(Me.dsEmployees)}, using the IntelliSense listing when applicable. \textbf{Make sure that \texttt{Me.dsEmployees} is surrounded by the brackets ( and )}. Once the line of code is completed, it should read as:

\texttt{Me.OleDbDataAdapter1.Fill(Me.dsEmployees)}

While typing lines of code in VB.NET, please remember that you can use IntelliSense and the spacebar to finish completing parts of your code. If you see a space appearing after a word that you added using IntelliSense, and if that space is not supposed to be there, simply use your backspace key to backup and remove that unwanted space, and then continue on with your typing. \textbf{You must be VERY EXACT with your typing. Typing errors (also known as coding errors) will cause your program not to work properly. IF YOU EVER SEE A LINE OF CODE UNDERLINED with a blue or green jagged line, THEN THAT MEANS THAT YOU HAVE MADE A TYPING/CODING/SYNTAX ERROR that you will need to fix before your program works correctly.}

Now, place your cursor in front of the line of code that you just typed. Press the Enter key to move your existing code down one line. After moving down one line, move the cursor back up to the prior line. Use the tab key to position the cursor so that it lines up with the first letter of the code that you already typed. Type the following (using IntelliSense where appropriate):

\texttt{dsEmployees.Clear()}

\texttt{dsEmployees.Clear()} clears the dataset (dsEmployees) of any existing data by removing all rows in all tables. Each time the form is loaded into memory, dsEmployees is cleared of all data that it might contain, and then the Fill method adds or refreshes rows in the dataset to match the rows in the associated data source (the original database that your application is interacting with).

Make sure that your code matches the code seen to the right. Once your code matches, \textbf{click the SAVE ALL button.}

To view the data contained in a dataset, the dataset must be connected to one or more controls on the interface. The process of making this connection is referred to as binding, and the connected controls are referred to as bound controls. Examples of bound controls include the DataGrid, label, textbox or listbox. A DataGrid displays data from a dataset as a series of rows and columns, much like the rows and columns in a spreadsheet. Let’s create a DataGrid to display our data.

\textbf{Adding a DataGrid and binding it to the dataset}

Make sure that the GUI/interface of the form is displayed by clicking on the View Designer button. Locate the Toolbox (typically located on the left-hand side of the screen). If the object seen pictured to the right are not visible, click \textbf{Windows Forms}. Click the DataGrid button. Move the cursor onto the form, and you should see the cursor changing into a plus sign and a DataGrid symbol. Move the cursor toward the top-left of the form. Hold down your left mouse button, drag the mouse down and to the right so that a square/DataGrid displays, covering approximately the top 2/3 of the form. The DataGrid should be the currently selected item (\texttt{Click the DataGrid to select it if it is currently not selected/highlighted}).

Locate the Properties Window (\texttt{typically located on the right-hand side of the screen, as seen in prior pictures}) The Properties Window is used to display the attributes, or properties, of a given object/control. Click the A/Z button found in the Properties Window to arrange the properties of the DataGrid in alphabetical order. To change a property for any object, simply type in a new entry for that particular property, and then press the Enter key. For example, scroll through the alphabetical order listing for properties for DataGrid1 (\texttt{the currently selected object}) until you reach the top of the alphabetical order listing. Above
the properties that begin with “A”, you will see the (Name) property, with the current property set to DataGrid1. Change the (Name) property to **dgDataGrid1**, and once you press the Enter key, the object is renamed dgDataGrid1. Go ahead and make that change now. Next, the Size property should be changed from its current value to **550, 184**.

To bind dataset dsEmployees to the datagrid dgDataGrid1, please change the following properties of dgDataGrid1:

- **DataSource**: change from (none) to **dsEmployees** (you can type in dsEmployees or you can use the drop-down option that appears when you click in the DataSource property box to make your selection). With this property, you are specifying which of your datasets should be used as the source to display your data from.

- **DataMember**: change to **tblEmployees** (instead of typing, you can also make this selection by using the drop-down option that appears in the DataMember property box once it is clicked.). With this property, you are specifying the table that the data will be taken from.

**Click the SaveAll button.** Now, it is time to run our application (as it currently exists) and see what it does.

First, make sure that you are viewing the design screen. To start your application while in the design screen, you need to click the Start button found on the toolbar (found to the left of Debug, as seen in the Visual Basic.NET programming environment pictured on page 3). **Go ahead and start/run the application.** Running the application changes your screen as VB.NET compiles the program code (checks for errors). If no compile/syntax errors are found, VB.NET enters runtime mode, where the application actually runs (but you also see the design screen).

Pictured to the right is the application in runtime mode. Everything else is part of the design screen. You might also see a new window opening toward the bottom of your screen (not pictured) called the Task List window, or possibly an Output window (also not pictured) could appear. The Task List window should also appear when you return to design mode (toward the bottom of the design screen). If you ever see “0 Build Error tasks shown” in the Task List window, then that means you did not make a typing error and your application will work as it is coded. If you ever see **Build 1 succeeded, 0 failed, 0 skipped** in a window labeled output, then you are also in good shape. If you ever make a syntax error or any other error, then the application will not run and you will see a message box indicating that you have build errors, which must be fixed before the application will run. An Output window (if it appears) would also indicate the type of error and/or location of the error. Two common errors can be seen below, labeled as Implementation Note #1 and Implementation Note #2. However, since we have only typed in two lines of code, there should hopefully not be any build errors.

**Implementation Note #1: See Picture to the Right**
One error that could be seen by a student completing this tutorial in the future would be an error message coming from the Microsoft Development Environment. When/if this error appears, the VB debugger highlights the line of code typed earlier (Me.OleDbDataAdapter1.Fill (Me.dsMembers)) and an error message box displays, saying “An unhandled exception of type ‘System.Data.OleDb.OleDbException’ occurred in system.data.dll”. If this happens, click the **Break button**; go to the Debug Menu, and the select **Stop Debugging** to return to the design screen. When
having a VB.NET program connect to and interact with a database, the database and the program’s executable file must be in the same storage area. When VB.NET compiles the program, the executable file is placed inside the **bin folder**, and the database being connected to must also be in the bin folder. While a student completing this activity won’t see the error described in this paragraph (since the downloaded files already contained the database inside the bin folder), this is something that an individual must be aware of when creating their own programs designed to interact with a database.

**Implementation Note #2: See next picture**

One error that could be seen by someone designing their own application from scratch, using the concepts discussed in this paper, is an error message coming from the Microsoft Development Environment, saying “There were build errors. Continue?” In the Output window, it could indicate that Sub Main was not found. This error is due to the form object being renamed, but then not updating the application to let it know which form is the Startup form (first form to be displayed once the project begins running). To fix this problem, make sure that you are at the design screen. Look in the Solution Explorer window. The solution (container that holds your entire application) appears at the topmost level of the hierarchy. On the next lower level is the project you are currently working on (DataGrid Example in the picture above). Right-click the project (DataGridExample) and select the Properties option. The StartUp object is probably Sub Main. Using the drop-down option that appears to the right of the StartUp object, change the StartUp object to the name of your current form (frmDataGrid, in this example). Click Ok, then SaveAll, and that error should now be fixed.

With no errors being displayed, the application should display, as seen earlier, with a datagrid containing selected fields from the Employees table. The vertical scroll bar can be used to scroll up and down through the data displayed. If the user scrolls to the bottom of the data being displayed, then new entries can be made by typing in the bottom row of the datagrid. The user can also change words/entries found in the datagrid, as well as delete words/entries. However, at the moment, any changes made to the datagrid will not be saved back into the original database. Our next task is to create the coding necessary to save changes back to the original database. Since it is time to make changes to the application, click the Exit button to exit the application, and you should return to the design screen.

**Updating a Database Using a Datagrid**

We will next modify our application so that the user can click on one of two buttons when it is time to exit the application. One button will allow the user to exit without saving changes, while the other button will allow the user to exit and save any changes that are made to the datagrid back to the original database. Since the current Exit button already has been coded to exit without saving changes, we will begin by making minor modifications to that button.

Change the following properties of the Exit button:

- **Size**: change to 168, 24
- **Location**: change to 96, 208
- **Text**: change to **E&xit without saving changes** (the & sign is used to underline the X, signaling the existence of a keyboard access key that the user can use to initiate processing instead of clicking on the button with the mouse. Pressing the Alt. key at the same time the underlined key is pressed (in this case, Alt. and X) will execute the event procedure associated with this button)

Now, it is time to create the command button used to save changes and exit. Click the button option in the Toolbox (pictured earlier). Then, move the cursor onto the form, and you should see your cursor changing into a plus sign and boxed ab. Create a rectangle to represent this button (don’t worry about where you place it, we will change that next), and assign it the following properties:

- **(Name)**: change from Button1 to **SaveChangesButton**
- **Size**: change to 168, 24
Location: change to 304, 208
Text: change to &Save changes and exit (which creates another keyboard access key. When the user presses the Alt. key and the S key at the same time, this key’s associated event procedure will execute)

In order to create the necessary code to save your changes and exit, double click the command button displaying the text of “Save changes and exit.” Double clicking that button should send you to the code window, where you can add new code to the application. In our case, double clicking the button created a new event procedure called Private Sub SaveChangesButton_Click, and the cursor should be indented inside the event procedure. Type the following code:

Me.OleDbDataAdapter1.Update(Me.dsEmployees)
MsgBox("Changes to the data have been saved", MsgBoxStyle.Information, "Update Data")
End

As the user changes an item displayed in a bound control, the in-memory copy of the dataset is changed as well. If this changes needs to be also made to the underlying database, the data adapter can change the original data source (the original database) by using its Update method. Each dataset has a property that stores a value, such as True or False, that indicates whether changes have been made to the dataset. If the dataset has values that have been changed and now differ from the original database, then the Update method can be executed. The Update method examines each item in a dataset, and if a change has been made, then it sends the appropriate Update, Insert, or Delete command back to the original database.

The Message Box statement causes a new window to open on the screen, displaying a message that stays on the screen until the user has acknowledged it by clicking on one of its buttons. The syntax of a message box statement is MsgBox ("The message to display", Buttons, "A title for the message box")

The message box syntax contains 2 commas. Before and after the commas are arguments. The first argument is the message to be displayed to the user, which is surrounded by quote marks. The second argument displays information regarding what types of icons and buttons that should be displayed within the message box. The third argument represents the text that will appear in the message box’s title bar.

Typing in the code of MsgBox("Changes to the data have been saved", MsgBoxStyle.Information, "Update Data") creates the message box that appears above.

Click the Save All button. Now, start your program and change the first name for one of the instructors. Then, click in the gray box to the left of one of your records to highlight the entire record, and then delete that information. Finally, click the asterisk/star icon on the bottom left of the datagrid. That is the new record button. After clicking that, each field displays the following: (null). Create a new record, supplying an appropriate value for each field. When you are finished, click the “Save changes and exit button.” Once you have exited the application, then restart the application, and when the form loads once again, your saved changes should now appear.

Implementation Note #3
The technique to update a database, as described above, works only with databases containing unrelated tables (tables that are not connected using relationships). Additional programming, beyond the scope of this paper, would be needed to update information stored in one table that is also related to another table.

Before ending this example, one additional programming technique should be examined, the use of comments (comment code). The programmer uses comments as program documentation since they are not considered “executable” and have no effect when the program runs. The purpose of comments is to make the program easier to read so you can tell what the programmer is trying to do with a block of code. Comments typically include information identifying the name of the programmer, the purpose of the program, or the purpose of a line of code. The apostrophe signals the beginning of a comment. Comments can be a separate line of code or they can be at the end of a line of code, as seen in the examples that follow:

' This is an example of a comment that is a separate line of code.
dsEmployees.Clear() ' This is an example of a comment at the end of a line of code.
Our final task with this example is to create a comment containing your name. Place your cursor in-between the Windows Form Designer generated code and the event procedure that you just created. Press the Enter key three time. Move the cursor up one line, and then type in the comment that follows. *(including the apostrophe.)*

*Programmed by (replace the parenthesis and these words by your own first name and last name)*

While still in the Code window view, **make a printout of the code that you have in this application.** Label the printout as **Printout #1: DataGridView Example.** Once the printout is made, click the Save All button, and close Visual Basic.NET.

**Displaying Data in Labels and Textboxes**

Your health club has continued to grow, and although you are very busy, you have decided to continue the development of an information system for your business. Lately, you have been suffering from headaches related to the fact that information about the members of your health clubs exists only on paper, so you have decided that add a Membership table to your database so you can electronically retrieve information much more efficiently than your current practice of shifting through file folders each time you wanted something. Each member record contains the following attributes: **MemberNumber, LastName, FirstName, Gender, Address, City, and State.**

Rather than writing a program that displays all of the member information in one location (as seen in a datagrid), you have decided to create an application where member information is displayed individually. Individual display of records is accomplished using labels and/or textboxes. Labels are VB.NET objects that serve the purpose of displaying information, giving the user no ability to edit or change the information being displayed. Textboxes, on the other hand, allow the user to change the information being displayed.

Prior to beginning this activity, you should have obtained several folders. Please open the folder called **LabelandTextbox Example.** You should now see four items in Windows Explorer (see picture)

HealthClub is your newly modified Microsoft Access database file, and we are going to connect to that file once again. The file **LabelandTextbox Example, Visual Studio Solution,** is the solution file that contains all of the projects and files for your new application. Go ahead and **double click the LabelandTextbox solution file** and you should once again return to the VB.NET development environment. Please make sure that **frmDisplay** appears in design mode (with the design screen displayed). *If you are in design mode, but frmDisplay does not appear on your screen, then double click frmDisplay in the Solution Explorer window.*

The graphical user interface (GUI) for...
frmDisplay already contains a number of VB.NET objects (primarily labels), and you will be added a few more to the GUI in order to display the membership data. Our next task is to add three labels and three textboxes to the form. When you are instructed to create a label or textbox, simply draw it on the form somewhere, without worrying about its size or position. We will set those, and other attributes, using the properties window.

Locate the Toolbox. Click the Label button. Move the cursor onto the form, and you should see the cursor changing into a plus sign and an “A.” Hold down your left mouse button, and drag the mouse down and to the right so that a label/shape appears. The label that you just made should still be highlighted. (_Click the label to select it if it is currently not selected/highlighted)._ Using its properties windows, assign the label the following properties.

- (Name) property: change to LastNameLabel
- BorderStyle property: change to FixedSingle
- Location property: change to 120, 88
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters so that the property value is empty/nothing

Create another label, and assign it the following properties:

- (Name) property: change to FirstNameLabel
- BorderStyle property: change to FixedSingle
- Location property: change to 120, 120
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters so that the property value is empty/nothing

Create a third label, and assign it the following properties:

- (Name) property: change to GenderLabel
- BorderStyle property: change to FixedSingle
- Location property: change to 120, 152
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters so that the property value is empty/nothing

Labels are used to display data that the user cannot change or edit. We will next create three textboxes. Unlike the label, a textbox can be used to both display and change data. Creating a label and a textbox uses the same basic process, with the exception being that a textbox is created by first clicking on the Textbox tool. As with the label, simply draw the textbox somewhere on the form, and we will set its attributes by using the Properties window.

Create your first textbox, and assign it the following properties:

- (Name) property: change to AddressTextBox
- Location property: change to 120, 182
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters so that the property value is empty/nothing

Create a second textbox, and assign it the following properties:

- (Name) property: change to CityTextBox
- Location property: change to 120, 214
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters

Create a third textbox, and assign it the following properties:

- (Name) property: change to StateTextBox
- Location property: change to 120, 246
- Size property: change to 152, 16
- Text property: eliminate all words and/or letters

Click the Save All button. Using the Start Button, run the application, just to see what it looks like. Since we have not yet coded anything, there should not be an error messages that are displayed. Try clicking inside a label. You will see that you cannot do that. Click inside a textbox and type something. When an application is running, a textbox and label serve different purposes. Use the exit button to stop the application, and you should return back to the design screen.
Next, a data adapter needs to be created so that a connection can be made between your VB.NET application and the database. Please refer to the tutorial steps seen earlier to create and configure the data adapter. Please use the following settings:

- Create a New Connection, with the provider Microsoft Jet 4.0 OLE DB Provider being specified.
- HealthClub.mdb is the Microsoft Access database that you want to access. Remember to remove/delete the entire pathway, except for HealthClub.mdb
- tblMembership is the table from HealthClub.mdb that contains the data that you wish to display. When the SQL statement is created, tblMembership appears in the FROM statement.
- The SELECT statement uses the following fields, in this exact order: MemberNumber, LastName, FirstName, Gender, Address, City, and State.

After the data adapter has been created, a new data set should be created. The new data set should be given the name dsMembers. Change the (Name) property of the data set to dsMembers.

After the dsMembers data set is created, use the earlier tutorial steps to create a form load event procedure for frmDisplay. One line of code in this event procedure should clear dsMembers, while the second line of code should use the data adapter’s Fill method to fill the dsMembers data set.

Just like the datagrid, in order to display information from a database in a label or textbox, the textbox or label control must be bound to the dataset (dsMembers, in this case). However, the process of binding a label or textbox is different than how a datagrid is bound. To bind a datagrid, the DataSource and DataMember properties of a datagrid must be set. To bind a label or textbox to a dataset, the control’s (DataBindings) property, subproperty Text must be specified (often referred in the coding as the DataBindings.Text property).

In the design window, select the MemberNumberLabel object. At the top of the alphabetical order listing of properties (above the (Name) property), the (DataBindings) property will be found. Click the plus sign to the left of (DataBindings), and the property will expand to display three subproperties (Advanced, Tag, and Text). Click the Text subproperty. Then, click the Text properties down arrow, and a drop-down box should display, showing the datasets found in this application. Click the plus sign next to dsMembers, and the tables from HealthClub included in the data adapter and dataset should appear. Click the plus sign next to tblMembership, and its fields should appear. Double click MemberNumber, and the Text property for MemberNumberLabel should display a value of dsMembers - tblMembership.MemberNumber (as seen in the pictures that follow)
Bind the remaining controls to the appropriate field in the dsMembers dataset in the following manner:

- Select LastNameLabel and set its DataBindings.Text property to LastName
- Select FirstNameLabel and set its DataBindings.Text property to FirstName
- Select GenderLabel and set its DataBindings.Text property to Gender
- Select AddressTextBox and set its DataBindings.Text property to Address
- Select CityTextBox and set its DataBindings.Text property to City
- Select StateTextBox and set its DataBindings.Text property to State

Once you have set the DataBindings property for all of the textboxes and labels, please click the Save All button. Then, run the application and one of the records from the dataset should appear on your screen, with each label or textbox displaying the appropriate field from the dataset. Click Exit to return back to the design window.

At the moment, the information for only one record displays in your application, and there is no way to see other records. The next modifications to our application will resolve those issues.

First, click on frmDisplay to select it (making sure that you are now clicking on one of the controls found on frmDisplay). Change the size property of frmDisplay to 400, 408. Next, click on the ExitButton command button. Change its location property to 232, 344. With the form now just a little bit larger, and with the Exit button moved toward the bottom of the form, we now have space to create buttons that can be used to navigate through the records.

Creating Navigation Buttons

The next step is to create three controls that will be used to navigate through the records (2 buttons and a label).

Create a button somewhere on your form, and assign it the following properties:

- (Name) property: change to PreviousButton
- Location property: change to 32, 280
- Size property: change to 104, 24
- Text property: change to Previous

Create a label somewhere on your form, and assign it the following properties:

- (Name) property: change to RecordStatusLabel
- BackColor property: change to Desktop (which will result in the label having a bluish background color.
- ForeColor property: change to one of the white color options that you see (resulting in the label’s text displaying with a white font color.
- Font property: click the three dots next to the font options that currently appear for that property. Keep the existing font style/type, but select the Bold and 11 point font options as well.
- Location property: change to 136, 280
- Size property: change to 120, 24
- TextAlign property: change to and/or select MiddleCenter
- Text property: eliminate all words and/or letters so that they property value is empty/nothing

Create another button somewhere on your form, and assign it the following properties:

- (Name) property: change to NextButton
- Location property: change to 256, 280
- Size property: change to 104, 24
- Text property: change to Next

Click the Save All button.

Sub Procedures, Variables and Selection Statements

Programmers create event procedures that respond to the actions of the user, such as clicking on a button. As the event procedure executes, certain activities happen. Suppose two or more event procedures have a common set of instructions that are executed, each event procedure follows the same instructions, at least for a brief period of time. Instead of placing that commonly used code in more than one event procedure (multiple times), a programmer can avoid duplication of effort by creating a general sub procedure, and then place that common code there. The reason this procedure is referred to as “general” is because it is not associated with any specific control. Each event procedure can use, or call, the
general sub procedure, and when it is “called”, then its instructions begin to execute, before jumping back to the event
to the event procedure that made the call in the first place. Therefore, one of our next tasks is to create a general sub procedure that
can be used, or called, by both of our buttons.

Please switch from the Design window to the Code window (where you can see all of the code contained on your form.
You should see two event procedures on the form (Private Sub ExitButton_Click and Private Sub frmDisplay_Load).
Locate the last End Sub command. Place the cursor right after the “b” of End Sub. Press the Enter key twice, and then
type in the following:

Private Sub DisplayStatus()

Press the Enter key, and VB.NET should complete the procedure ending of End Sub for you.

The purpose of this general sub procedure that you have started to create is to identify the current record that you are
displaying in your application, as well as display the total number of records found in the dataset. Remember that you
just created a label. As you navigate through the records using the Next or Previous buttons, the text property of the label
should change, displaying information such as “1 of 15”, “2 of 15”, etc. The general sub procedure will also introduce
two features commonly found in computer programs, variables and selection statements.

Variables are locations in the computer’s memory that are used to store data. They are called variables because the data
stored in these memory locations can change (or vary) as the program runs. Dim statements are used to create variables.
When a variable is created, its “type” must be declared (what type of data will be stored in it). The string data type is
usually assigned when you want to store a sequence of characters as text (letters and numbers not used in calculations). If
a variable is to be used in a calculation, one of the numeric data types (short, integer, decimal, single, double, long) should
be chosen. The general sub procedure that we are about to create will feature two variables that will be used in a
calculation, so we will declare the variable using the numeric data type of “short”, which takes up the least amount of
computer memory space.

When creating variables, a programmer must think about the scope of the variable. A variable’s scope is the set of all
code that can refer to, or use, the variable. A variable’s scope is determined by where and how the variable is declared.
In this project, we will examine two different levels of variable scope: procedure-level (local) scope and module-level
scope.

Any variable that is declared inside an event procedure has **procedure-level scope**. Variables that have procedure-level
scope are said to be **local to the procedure in which they are declared**. That means that a variable that is declared
within an event procedure is visible only to that event procedure. Other event procedures on that form, or within that
project, are unable to “see” or use a variable that has procedure-level variable. Because we can create variables (with
procedure-level/ local scope) that other event procedures are unable to “see” or use, that means that you can have a
variable called TotalPrice in event procedure A and another variable called TotalPrice in event procedure B. Even though
both of those variables have the same name, they can hold/contain entirely different values, simply because their scope is
different. Event procedure A can change its variable called TotalPrice, and event procedure B can change its variable
called TotalPrice, all without affecting the TotalPrice variable stored by the other event procedure. Although both
variables have the same name, they are actually two different variables, representing two different memory locations. A
variable declared within an event procedure is only visible within that event procedure. It is not visible outside of the
procedure where it is declared. Dim statements are used to create procedure-level variables. In fact, the VB keywords of
Private Sub are indicators of private subroutines used only by individual controls (such as a button). No other control or
element of a VB program can use a private subroutine or the variables and other elements that exist only in that subroutine.

Module-level variables are also declared using a Dim statement. However, instead of placing the Dim statement inside an event
procedure, the Dim statement is placed outside of an event procedure, in what is known as the general declarations section. The general
declarations section is the area between the Windows Form Designer generated code box and the first event procedure on the form. Any
**event procedure on the form can access module-level variables**.
The screen shot that appears next illustrates local procedure-level variables, as well as module-level variables which are placed in the general declarations section.

Selection statements are used by computer programs to make a decision, based on certain criteria found in the data or based upon input from the user. If one condition occurs, take one course of action. If another condition occurs, then do something else. If….Then….Else statements are one of the most common forms of selection statements.

Within your new procedure, type the following code:

```vbscript
Dim CurrentPosition As Short
```

Now, move your cursor to the general declarations section of the form (pictured earlier). Type in the following code:

```vbscript
Dim Total As Short
```

You have now created two variables: one called Total and one called CurrentPosition. The primary difference between these two variables is their scope. Total is a module-level variable that can be used by all event procedures on the form, and CurrentPosition is a procedure-level variable that can be used only by Private Sub DisplayStatus(). The numeric data type of “Short” will be the data type of these two variables. Upon their declaration/creation, both the Total variable and the CurrentPosition variable have a value of zero (0). An assignment statement is used to assign/give a variable a new value.

Move the cursor back into Private Sub DisplayStatus(). Type in the following assignment statement (which will be explained later):

```vbscript
Total = Me.BindingContext(Me.dsMembers, "tblMembership").Count
```

Me is used to refer to the current form. Each form has a BindingContext object, which keeps track of all of the data sources associated with a particular form since it is possible for a form to have more than one data source. dsMembers is the name of the dataset you have created for this application. tblMembership is the name of the table within dsMembers that contains the data that you wish to work with (notice that tblMembership is in parenthesis). Count is a property that indicates the exact number of records in the dataset. If there are five records in the dataset, then Count returns a value of five (5), which is stored in the variable called Total.

A Selection statement, containing two possibilities, comes next. Either there will be no records found in the dataset, or the dataset will contain records. If no records are found in the dataset, then “No records” should be displayed in the record status label. However, if the dataset contains records, then the CurrentPosition property will be used to assign a number (representing the position of the current record in the dataset) to the CurrentPosition variable.

When a VB.NET program works with records in a dataset or items in a list, it uses a somewhat strange (at least that is how students describe this numbering system) indexing system. For instance, say you have five records in a dataset (Count property = 5). Record #1 is indexed as 0, record #2 is indexed as 1, record #3 is indexed as 2, etc. Indexing begins with zero, and continues until it reaches (Count – 1), which is the last record or item in the list. Therefore, to have record #3 show up as #3 (instead of as #2, as signified by its index number,), the number one (1) must be added to the current position property value. Continue by typing the following selection statement:

```vbscript
If Total = 0 Then
    Me.RecordStatusLabel.Text = "(No Records)"
Else
    CurrentPosition = Me.BindingContext(Me.dsMembers, "tblMembership").Position + 1
    Me.RecordStatusLabel.Text = CurrentPosition & " of " & Total
```

17
If you ever see a word that appears to be the word “Position” with a blue underline, then that means that your current spelling of that word does not match the spelling that was used when the variable was declared, and you will want to fix that inconsistency. This last line of code uses a technique known as string concatenation. As seen earlier with the string data type, strings are text or numbers not used in calculations. One string can be joined together with another string to create an even longer string, which is one way that programmers create sentences or phrases. While the plus sign (+) is used to add two numbers together, the ampersand (&) is used to concatenate strings. CurrentPosition and Total are both variables. Suppose you are viewing record #3 in a dataset that contains five records. Using the concatenation sequence coded above results in the following text displayed in RecordStatusLabel: 3 of 5.

The keywords **End If** and **End Sub** should already be included in your coding. Make sure that your work matches the coding for **Private Sub DisplayStatus()**, as seen below. Then, click the Save All button. Once your work is saved, return to the form’s design screen. We will next create the code that will execute when the user clicks either the Previous or Next buttons.

```vbnet
Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitButton.Click
    ExitForm()
End Sub
```

**Coding the Previous Button**

From within the design screen, double click the Previous button, which returns you to the code window and you will see that the event procedure **Private Sub PreviousButton_Click** has already been started. Type in the code for that event procedure, as pictured above, and then read the remainder of this paragraph to find out what your code does. When the Previous button is clicked, the current binding context position is determined, and then reduced by one position, resulting in the prior record (going back one record) being displayed on the form. The DisplayStatus sub procedure then executes, resulting in the RecordStatusLabel displaying the current record position (in the data set) out of the total number of records (in the data set). After DisplayStatus executes, the program enters a selection statement. If the user tries to advance past the first record (which is something that can’t be done), then the application displays a message to the user indicating that the beginning of the file has been reached. Click Save All, and then return to the design screen.

```vbnet
Private Sub PreviousButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PreviousButton.Click
    If Me.BindingContext(Me.dmnMembers, "tblMembership").Count = 0 Then
        MessageBox("You have reached the beginning of the file.")
    End If
End Sub
```

**Coding the Next Button**

From within the design screen, double click the Next button, which returns you to the code window and you will see that the event procedure **Private Sub NextButton_Click** has already been started. Using the picture see above, type in the event procedure’s code. This button functions like the Previous button, except that the Next button advances forward to the next record (Position +1). If the user tries to advance past the last record, then its selection statement indicates that the end of the file has been reached. Click the Save All button.
Now, it is time to run the application. If a syntax error or other coding error is found, then fix the error, and then start the application again. When the application starts, the first record should be displayed. However, notice that the record status label is empty. This is because DisplayStatus(), which is used to determine the content of the label, has not yet run. Our next coding will fix that minor problem. Go ahead and use the Next button to advance through the records. Then, use the Previous button to view earlier records. If you reach the first or last record, the appropriate message box should display. Once you have finished testing the application, click the Exit button.

As noted previously, when the application displays the first record, the record status label is empty. We need to add the “call” statement to invoke DisplayStatus() to the form’s Load event procedure, so that when the first record is displayed, the record status label displays “1 of 22” (or however many records that you have). Modify your code for Private Sub frmDisplay_Load to match the code in the prior picture, adding the ending “call” statement of DisplayStatus(). Therefore, when the form is displayed to the user for the first time, the record status label will show information to the user. Finally, move your cursor to the General Declarations section of the form (immediately underneath the Windows Form Designer generated code) and create a comment that says:

‘Programmed by (replace the parenthesis and these words by your own first and last name)

Click the Save All button. Test the application to make sure everything is working properly. Once you are finished, exit the application and return to the design screen.

Adding a new form to a VB.NET application

One common use of database is to search for records that match certain criteria. Although our next example is simply a simple query, more advanced queries can also be built by simply extended the following concepts. Our search will simply show the records (one-by-one) that match the state abbreviation that is entered into the State Parameter textbox. Before coding, we need to add a new form to our application, and then place several objects on the form.

You should still be on the design screen. From your current position in VB.NET, go to the File Menu, and click Add New Item. Click once on Windows Form, and click Open. A new form (Form1.vb) should appear on your screen (in design view) and in the Solution Explorer window listing. Click once on Form1.vb in the Solution Explorer Window. Look at the Properties window. You should see five, and only five properties. For the File Name property, highlight Form1 (and not the .vb). Type in frmStateQuery so that the new File Name will display as frmStateQuery.vb. Press the Enter key, and you should see frmStateQuery.vb appearing in the Solution Explorer window (instead of the previous Form1.vb). You have just named the Form File, just like you would name a spreadsheet or word processing document. Next, click inside the form object itself (in the gray dotted area). Although we just named the file, we must also name the form control. When you click in the gray dotted area, you should see that the object in the Properties window changes to Form1. Assign the form control the following properties:

- (Name) property: change to frmStateQuery
- Size property: change to 400, 415
- Text property: change to Query Records by State

The next task is to add controls to create the form’s GUI. Rather than creating each control one-by-one like you have already done, you can also use the copy operation to quickly add the necessary controls to your new form. Double click frmDisplay and make sure that its design screen is visible. Click once on the label that says Member Information to select it. Press the Control key, and then while pressing the Control key, click on these additional objects:

- The label that says Member Number
- The MemberNumberLabel (appearing to the right of the prior label)
- The label that says Last Name
- The LastNameLabel (appearing to the right of the prior label)
- The label that says First Name
- The FirstNameLabel (appearing to the right of the prior label)
- The label that says Gender
- The GenderLabel (appearing to the right of the prior label)
- The label that says Address
- The label that says City
- The label that says State
- The button that says Previous
- The blue RecordStatusLabel
- The button that says Next
- The button that says Exit
After copying those objects, double click frmStateQuery.vb. The form’s design screen should appear. Click the **paste button** (or use a paste operation), and each of the objects that you just copied should now appear on frmStateQuery.vb (in the same spot as those objects appeared on frmDisplay), as seen pictured.

Change the label that says **Member Number** so that its new text property is **Query Records by State**.

Now, create a new label and assign it the following properties:
- (Name) property: change to **AddressLabel**
- Location property: change to 124, 186
- Size property: change to 152, 16
- Text property: **eliminate all words and/or letters**

Create a second label and assign it the following properties:
- (Name) property: change to **CityLabel**
- Location property: change to 124, 218
- Size property: change to 152, 16
- Text property: **eliminate all words and/or letters**

Create a third label and assign it the following properties:
- (Name) property: change to **StateLabel**
- Location property: change to 124, 250
- Size property: change to 152, 16
- Text property: **eliminate all words and/or letters**

Create a fourth label and assign it the following properties:
- Location property: change to 32, 312
- Size property: change to 72, 16
- Text property: change to **Enter State:**
- TextAlign property: change to **MiddleRight**

Next, create a textbox and assign it the following properties:
- (Name) property: change to **StateTextBox**
- Location property: change to 104, 312
- Size property: change to 40, 20
- Text property: **eliminate all words and/or letters**

Finally, create a button and assign it the following properties:
- (Name) property: change to **FindButton**
- Location property: change to 152, 312
- Size property: change to 48, 24
- Text property: change to **Find**

Next, a data adapter needs to be created so that a connection can be made between your new form and the database. Please refer to the tutorial steps seen earlier to create and configure the data adapter. Please use the following settings:
- Create a New Connection, with the provider Microsoft Jet 4.0 OLE DB Provider being specified.
- HealthClub.mdb is the Microsoft Access database that you want to access. Remember to remove/delete the entire pathway, except for HealthClub.mdb
- tblMembership is the table from HealthClub.mdb that contains the data that you wish to display. When the SQL statement is created, tblMembership appears in the FROM statement.
- The SELECT statement uses the following fields, in this exact order: MemberNumber, LastName, FirstName, Gender, Address, City, State.
- After the SELECT statement has been created in the Query Builder, place the cursor right after the “p” in the FROM tblMembership statement. Press the Enter key, and then type in:
  ```sql
  WHERE (State=?)
  ```
  While the application is running, the “?” is replaced by a value provided by the user. The WHERE statement is used to create a filter that selects a subset of records from the original data.
After the data adapter has been created, a new data set should be created. The new data set should be given the name `dsStateQuery`. Change the (Name) property of the data set to `dsStateQuery`.

Using the picture that follows, please create the following (in the order specified):

- Create the module-level variable called Total
- Create the general sub procedure called Private Sub DisplayStatus()
- Create the event procedure Private Sub PreviousButton_Click (executes when the user clicks Previous)
- Create the event procedure Private Sub NextButton_Click (executes when the user clicks Next)
- Notice that this form does not have a Load event procedure
- Create the event procedure Private Sub ExitButton_Click (executes when the user clicks Exit)

```vbnet
Private Sub DisplayStatus()
    Dim CurrentPosition As Short
    Total = Me.Recordset1.RecordCount
    If Total = 0 Then
        Me.RecordStatusLabel.Text = "(No Records)"
    Else
        CurrentPosition = Me.Recordset1.RecordNumber + 1
        Me.RecordStatusLabel.Text = "CurrentPosition 1 of " & Total
    End If
End Sub
```

```vbnet
Private Sub PreviousButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PreviousButton.Click
    Me.Recordset1.RecordNumber = Me.Recordset1.RecordNumber - 1
    Me.RecordStatusLabel.Text = "CurrentPosition 1 of " & Total
End Sub
```

```vbnet
Private Sub NextButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles NextButton.Click
    If Me.Recordset1.RecordNumber = Me.Recordset1.RecordCount Then
        MsgBox("You have reached the last record that meets your criteria.")
    Else
        Me.Recordset1.RecordNumber = Me.Recordset1.RecordNumber + 1
        Me.RecordStatusLabel.Text = "CurrentPosition 1 of " & Total
    End If
End Sub
```

```vbnet
Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitButton.Click
End Sub
```

```vbnet
Private Sub FindButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles FindButton.Click
    Dim State As String
    State = Me.StateTextBox.Text
    Me.OleDbDataAdapter1.SelectCommand.Parameters("State").Value = State
    Me.dsStateQuery.SelectCommand.Clear()
    Me.OleDbDataAdapter1.Fill(Me.dsStateQuery)
End Sub
```

Click the Save All button. Next, the labels used to display information from the dataset need to be bound to the `dsStateQuery` dataset. Using the procedures described earlier in this tutorial, please set the DataBindings.Text property of the following controls:

- Select the MemberNumberLabel and set its DataBindings.Text property to MemberNumber
- Select LastNameLabel and set its DataBindings.Text property to LastName
- Select FirstNameLabel and set its DataBindings.Text property to FirstName
- Select GenderLabel and set its DataBindings.Text property to Gender
- Select AddressLabel and set its DataBindings.Text property to Address
- Select CityLabel and set its DataBindings.Text property to City
- Select StateLabel and set its DataBindings.Text property to State

Once you have set the DataBindings property for all of the textboxes and labels, please click the Save All button.

Creating a Query using Search Parameters

One common use of database is to search for records that match certain criteria. Although our next example is simply a simple query, more advanced queries can also be built by simply extended the following concepts. Our search will simply show the records (one-by-one) that match the state abbreviation that is entered into the State textbox. Please use the prior picture to create the event procedure Private Sub FindButton_Click (which executes when the user clicks the Find button).
In order to create a query that searches the records to find a specific value, we first need to create a variable to hold the value that we wish to search for. The line “Dim State As String” creates a procedure-level variable called State, which will be used to store the state value that you wish to search for. The string data type indicates that the data stored in this variable will be composed of letters and/or numbers not used in a calculation. Textboxes can be used as a means to input data into a computer program as it is running, and the assignment statement of “State = StateTextBox.Text” says to store the current value of the textbox’s text property (what is contained in the textbox) in the State variable. The line “Me.OleDbDataAdapter1.SelectCommand.Parameters("State").Value = State” is used to search the parameters of the SQL Select Statement (created using the Query Builder when building the data adapter’s SQL statement). When the SQL statement was created for the data adapter, one of the lines was WHERE (State=?). A filter was created, but the State value was not specified. The information entered into the State textbox, which was assigned to the State variable, replaces the question mark (?) in the WHERE portion of the SQL Select statement.

Click the Save All button to save your work. Don’t run the application now, but if you did run the application, you would not be able to get from the first/Display form over to the StateQuery form. In order to navigate from one form to another, we need to add buttons to each form that makes this movement possible.

Go to the design screen for frmDisplay. Create a button somewhere, and assign it the following properties:

- (Name) property: change to GoToQueryButton
- Location property: change to 32, 320
- Size property: change to 152, 24
- Text property: change to Query Information by State

Next, create an event procedure for when the user clicks on the button that says “Query Information by State”. Use the following code to create the event procedure:

```vbnet
Private Sub GoToQueryButton_Click()
    Dim objFormStateQuery As New frmStateQuery()
    objFormStateQuery.Show()
    Me.Hide()
End Sub
```

The Dim statement creates an object variable (object variables take up the greatest amount of memory, but they can be used to store anything) called objFormStateQuery. The “As New” coding indicates we want to create a new object, and “As New frmStateQuery ()” indicates that we want to pattern this new object based upon our existing form file of frmStateQuery (basically, a copy of the form file frmStateQuery will be stored in that object variable). Therefore, the line “Dim objFormStateQuery As New frmStateQuery()” says to create a new object variable and store a copy of the form called frmStateQuery in that variable.

The line “objFormStateQuery.Show()” asks the new object variable (objFormStateQuery) to execute its Show method, which will result in the object variable (its current contents) being loaded into memory and then displayed. The line “Me.Hide ()” tells VB.NET to keep the current form (frmDisplay) active in memory, but it will be hidden (invisible) to the user. In essence, when the “Query Information by State” button is clicked, VB.NET creates a new object, representing frmStateQuery, displays the new form used to query by state, and then hides the prior form (frmDisplay) that the user was seeing.

Click the Save All button to save your work. Next, go to the design screen for frmStateQuery. Create a button somewhere, and assign it the following properties:

- (Name) property: change to GoToDisplayButton
- Location property: change to 40, 346
- Size property: change to 160, 22
- Text property: change to Return to the Original Display

Next, create an event procedure for when the user clicks on the button that says “Return to the Original Display”. Use the following code to create the event procedure:

```vbnet
Private Sub GoToDisplayButton_Click()
    Dim objFormDisplay As New frmDisplay()
    objFormDisplay.Show()
    Me.Hide()
End Sub
```

Finally, move your cursor to the General Declarations section of the form (immediately underneath the Windows Form Designer generated code) and create a comment that says:

‘Programmed by (replace the parenthesis and these words by your own first and last name)’

Once you have created the comment for this form, then click the Save All button.
Now, it is time to run your completed application. Click on the Start button. If you have not made any syntax or coding errors, then the application should begin running and the Display form should appear on your screen (as before). If an error occurs or the program crashes, then you will need to find and correct the source of the error.

Go ahead and test the Previous and Next buttons, just to make sure that they still work. Then, click on the “Query Information by State button”, and you should move over to frmStateQuery (and frmDisplay disappears). Enter each of the following values into the state textbox and then click the Find button to see your query results (Valid state abbreviations for this database include CA, AZ, KS, OR, Fl, and OK). After seeing your results with a valid state abbreviation, type in an invalid state abbreviation and the Record Status label should display (No Records). Click on the “Return to the Original Display” button, and you should be back on the Display form. Both forms have an Exit button which you can use to exit the application.

Within the folder called LabelandTextbox example is another folder called answerkey. That folder contains a file called Answer Key, which you can run as a point of comparison.

Once you have tested all aspects of your application, and once everything works, then the tutorial is complete.