Chapter 15 – Working with Databases

## Objectives

* Understand Microsoft’s database models
* Creating SQL Server LocalDB Database Files through Visual Studio
* Working with databases through the Wizards
* Working with databases through VB Code
* Review of ADO.NET Related Classes
* Using SQL
* Notes on SQL Server LocalDB

It's taken us a while to get here, but we are finally ready to look at working with databases from inside of VB. We will begin with a brief introduction to the various database models that Microsoft has created over time.

We will then learn how to display the contents of a database through various controls that we can place on our application's forms. We will also learn how to directly drive everything natively through VB code: from the creation of the database through SQL (Structured Query Language) queries.

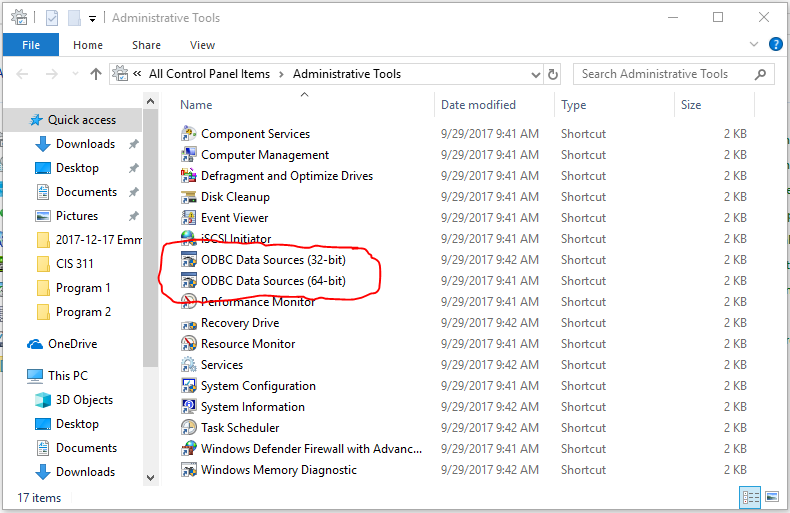
The chapter will end with a concise discussion and description of the SQL language, including how to return values from the database, how to insert values into the database, how to change values in the database and how to remove values from the database.

# Understanding Microsoft's Database Models

One of the earliest strengths that VB provided to developers was an integrated approach to working with database management systems (DBMS). Microsoft Access, of course, was typically the database of choice since it was Microsoft's desktop database solution at the time. Since at least VB 3, VB programmers have had the ability to read, write and use Access files natively from within VB. Microsoft called this Jet technology and the Jet Engine was built right into Visual Basic. In fact, many programmers think of Access as being little more than a pretty wrapper around the Jet Engine.

Microsoft came out with a plethora of different database technologies over the years. The primary reason why there are so many different options are that each option was an evolutionary step forward, bringing more power and productivity to developers. Let's pause for a second and review the major database processing pieces that Microsoft has introduced.

* ***ODBC*** (Open DataBase Connectivity) – ODBC is essentially a piece of middleware that allows a Microsoft Windows user to set up a connection to a 3rd party database such as Oracle, Informix and so forth. The idea is that as new database systems came to the market, only a new ODBC driver would need to be supplied to allow existing applications to work with that new database. This technology has been quite successful and emulated in the Java world, where it is called JDBC (Java DataBase Connectivity). ODBC is still on your system today, tucked under the Control Panel🡪Administrative Tools:



* ***OLE DB*** (Object Linking and Embedding for Databases) – OLE DB was another database attempt by Microsoft that was originally intended as a higher-level replacement for [ODBC](http://en.wikipedia.org/wiki/ODBC), extending its feature set to support a wider variety of non-[relational databases](http://en.wikipedia.org/wiki/Database#Relational), such as [object-oriented databases](http://en.wikipedia.org/wiki/Object_database) and [spreadsheets](http://en.wikipedia.org/wiki/Spreadsheet) that do not necessarily implement [SQL](http://en.wikipedia.org/wiki/SQL).
* ***DAO*** (Data Access Objects) – This was Microsoft’s first high-level programmatic database access system. DAO could talk natively to Jet (Access) databases or utilize ODBC to talk to third party databases. DAO would allow programmers to use standard SQL commands to manipulate the data stored in these databases. Think of it as the first real database API that was built in.
* ***RDO*** (Remote Data Objects) – RDO was a step forward from DAO. Essentially RDO allowed VB programmers to work with databases that might be residing on remote (networked) systems. RDO was only used for a brief amount of time due to ADO quickly following on RDO's heels.
* ***ADO*** (ActiveX Data Objects) – This was Microsoft’s last pre-.NET technology that created lightweight ActiveX (COM) controls that permitted web pages to utilize database technologies. Remember that at one point in time, Microsoft didn’t “see” the internet as anything to worry about and most of its technologies were not web-aware. Once it became apparent that the world wide web wasn’t going anywhere and customers expected applications to be network-smart, the company retooled everything. ADO was their solution for database connectivity. Since Microsoft wanted ADO to be able to work with web technology, the controls tended to be small, fast and efficient in their database processing. With the appearance of Visual Studio.NET, Microsoft modified ADO and released ADO.NET. This is currently *the* way to do databases in VB.

No matter which middleware technology you choose to employ, the lingua franca behind almost all of them is SQL (Structured Query Language). If you know it and have a connection to a database, you can pretty much do whatever you want with the data that's contained inside the database.

Remember that SQL is divided into two halves: DDL and DML. The Data Definition Language (DDL) allows you to manipulate the schema and metadata that make up databases. This is where you can create databases, tables, fields, keys, indexes and so forth. The DML (Data Manipulation Language), on the other hand, allows you to – well – manipulate data using the four basic SQL DML statements of SELECT, INSERT, UPDATE and DELETE.

The last section in this chapter provides a quick summary of using SQL – if you're not familiar with using SQL, you might want to skip ahead and look at it before continuing with the examples in this chapter. This chapter will not be examining the ideas of normalization, proper database design, or any of the mathematical underpinnings of databases, e.g. select, project & join. No discussions of CRUD or ACID tests are in here. Those are topics that you will see and need to learn from CIS 386/CS 411.

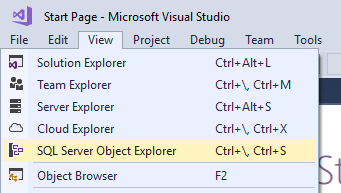
# Creating SQL Server LocalDB Database Files through Visual Studio

As you might expect, VB makes it almost ridiculously easy to work with databases. If you have a database lying around somewhere, it’s almost a moot point to use sequential or direct access files anymore. After all, why bother since databases were designed to quickly allow access to as much data as a user would need?

All of our database discussions are going to involve SQL Server, which is Microsoft’s premier enterprise database solution. You may be wondering why? Well quite simply, it’s built right into Visual Studio. That’s right, there is a file based version that will allow you to create local databases on your hard drive and access them through classes that are built right into VB. You don’t need to install anything new, and when you want to transport the database from one machine to another, it’s pretty much simply copying the database file.

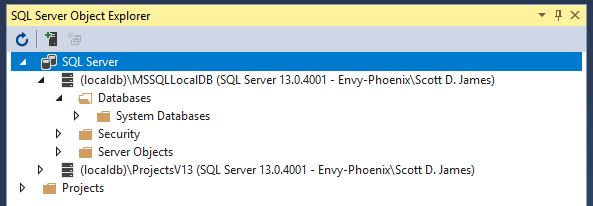
The other thing that’s important to note is that Visual Studio has a tremendous amount of database tools built right into it. While it’s still not as complete as installing Microsoft SQL Server Developer Edition, it’s not too shabby, and it’s enough for our class purpose. So, sit back, strap in and let’s build a SQL Server database file.

Fire up Visual Studio but don’t start a new project. We are going to open a new tool window that can be used at any time, even before project creation. Go to View🡪SQL Server Object Explorer and click on that menu choice:



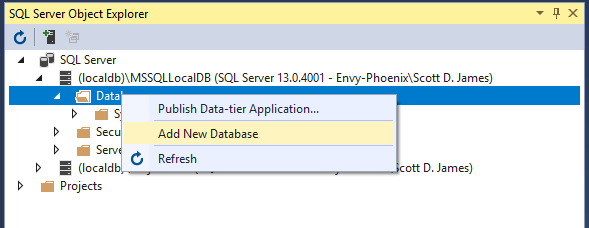
This will launch Visual Studio’s interface to the SQL Server installations that you have on your computer. At a minimum, Visual Studio installed the newest copy of its file based LocalDB on your system. If you have had older versions of Visual Studio on your computer, you may have several older versions of LocalDB as well. Finally, if you’ve ever installed versions of the full SQL Server product, you’ll see all of those as well. This is the management front end that VS provides for you to work with databases.

Here’s what the SQL Server Object Explorer looks like on my PC. You may have to click on the little arrows out in front of SQL Server to get your system to look like mine. You should have one instance that is named (localdb)\MSSQLLocalDB. That will be the branch that you want to expand; you can leave any other instances minimized (not expanded). Note that you may have a different version number for your LocalDB – that really isn’t all that important right now. Also, don’t freak out if you don’t have something called MSSQLLocalDB – if you have a different LocalDB instance, you can expand that one for the time being. In any case the next screenshot shows the SQL Server Object Explorer (SSOE) pane:

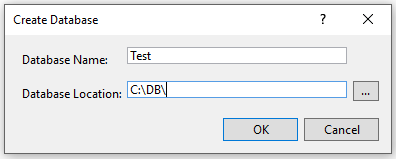


You’ll notice that under Databases, there is something called System Databases – do not mess with anything in there! That serves as a template for creating new databases and if you change things in there, every database you create will be altered as a result. You’ve been warned!!!

The first task that we want to do is create a brand new database. To do this, right click on the Databases folder under the instance that you want to create the new database in. From the context menu that appears, click on Add New Database:

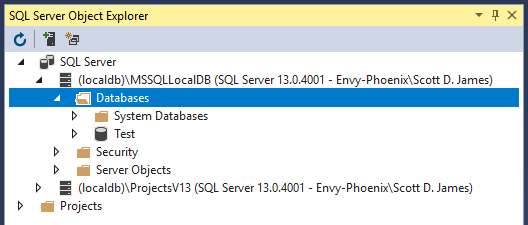


You’re then asked about what you want the database to be named and where you want it to live:

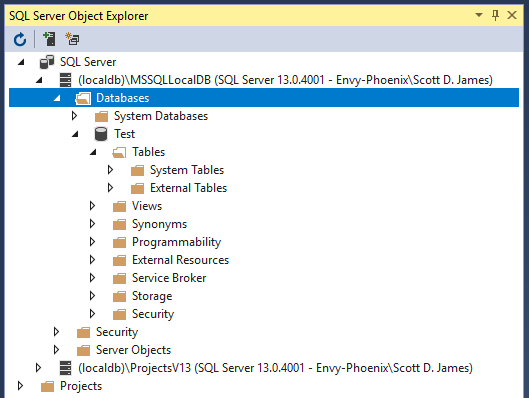


I called my database Test. I also placed the database in a location that is convenient to find, right off the root of the C: drive in a folder called DB. You should make sure that the path you specify exists before pressing the OK button. Make sure you know where you’ve put your database, or you’ll be doing some digging later on.

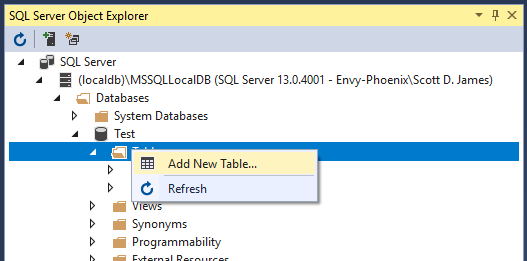
Once you’re ready, press the OK button. After a few seconds, the Create Database dialog will disappear and you should see your new database in your SSOE pane:



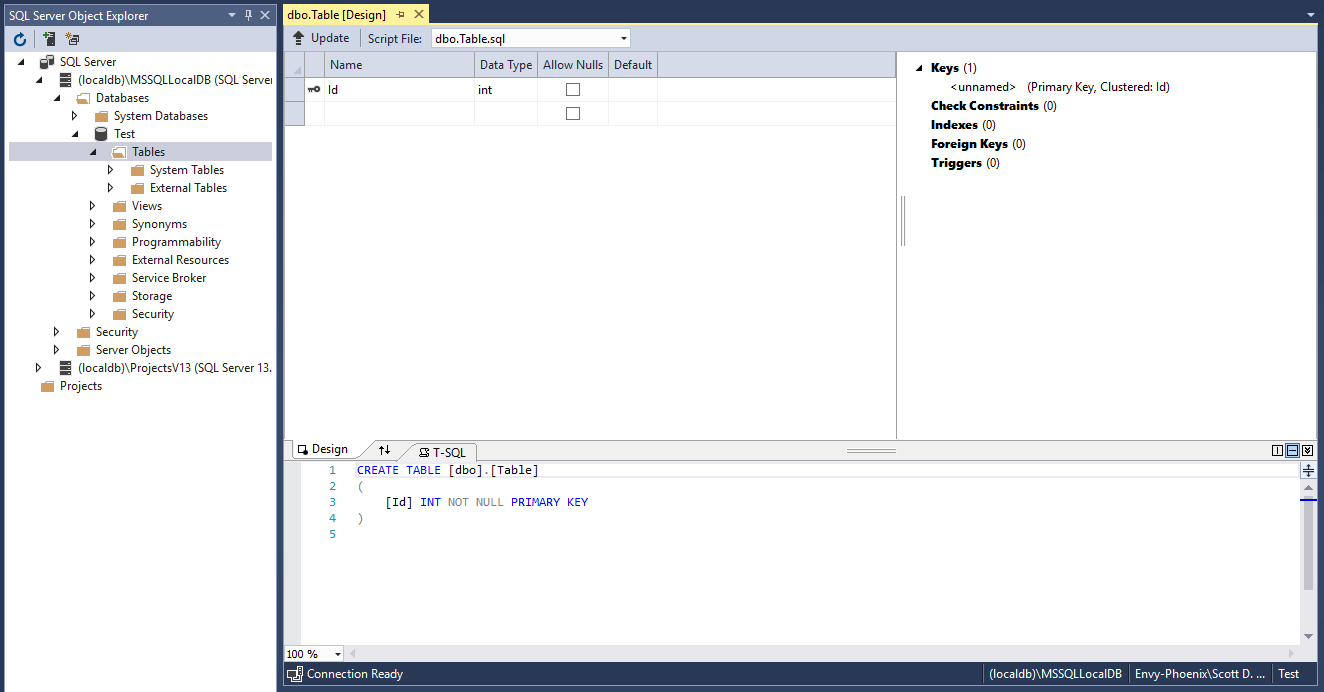
Now that you have a database, you can start creating tables in it! Go ahead and expand the arrow in front of the database and also in front of the Tables section that appears when you expand the database. When you’re done, you should see something like this in the SSOE pane:



There you have it – a new database with no tables. Let’s fix this… Right click on the Tables folder icon (not any of the icons below Tables). Click on the Add New Table menu option:

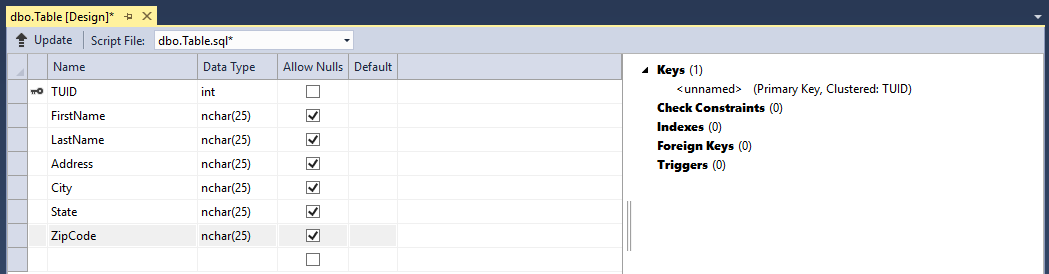


Visual Studio’s center window area changes to show the table building tools:



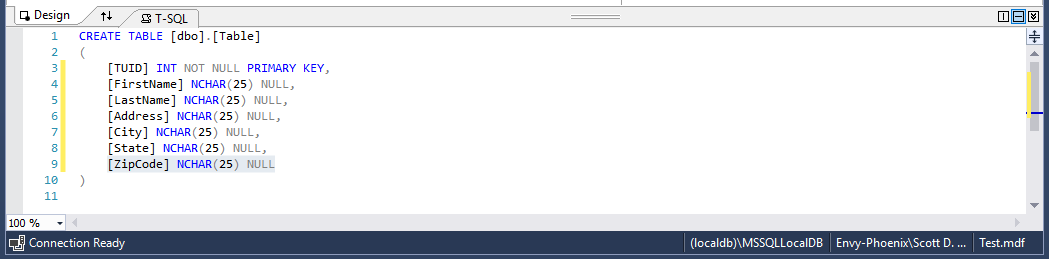
Notice that we still have the SSOE pane to the left. In the center, we have a table builder where we can add fields that looks a lot like Microsoft Access. As we change things up in the field builder, you’ll notice that the T-SQL script window is also changing to support whatever you’re doing in the upper pane. This is real cool stuff, because if you know SQL’s DDL, you can just jump into the lower pane and write SQL. If you don’t know SQL, use the top pane with its “guides” and it will write the SQL for you!

We are going to name this table Addresses. It will have a TUID, which I renamed from the default id, of type int and it serves as the primary key. We will then add FirstName, LastName, Address, City, State and ZipCode fields, which will all be of type nchar(25). Go ahead and start entering this information into the upper window. It should look something like this once you’ve entered all of that information:

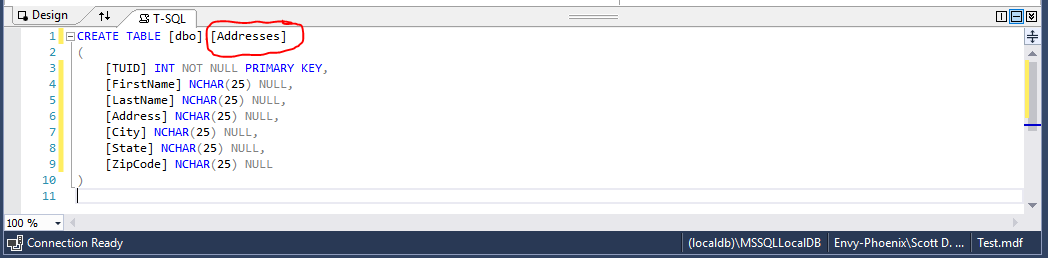


That wasn’t too bad to build our table. Even if you haven’t had a database class yet, what we did there for the most part ought to make sense to you. We created the necessary fields to store a person’s name and address. We also made sure that each address line stored in the database is unique by requiring each to have a TUID primary key.

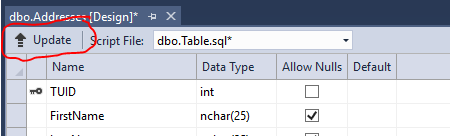
We are very close to having the table built, but there’s still one thing that we need to change before we commit to the build: the table’s name. Right now, it’s set up to be called Table, which is, well, a terrible name. So, to fix this, go into the lower T-SQL script pane and change the script from:



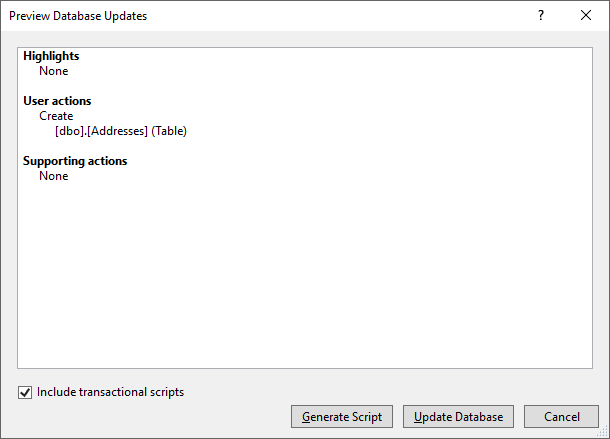
to:



You won’t see that anything else has changed, but we now have a nice name for our table (c’mon it’s at least better than Table). Now that we have all of our fields defined and we have a good table name, it’s time to have VS build our table for us. To do this, click on the Update button in the upper left corner of the field building pane:

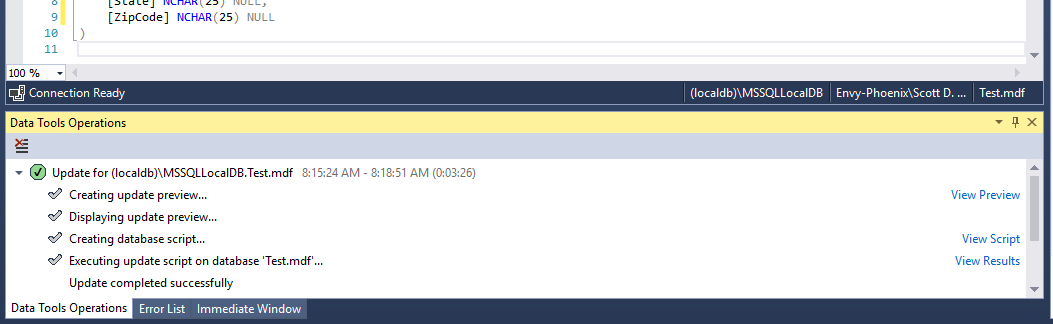


You’re then taken to another dialog that shows you a summary of what’s about to happen, which in our case is to create a table called Addresses:

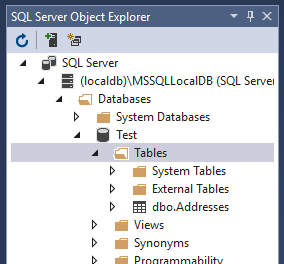


You’ll notice that at the bottom you have the choices of Generate Script (which would write the T-SQL CREATE TABLE code to a file), Update Database (which will run the T-SQL CREATE TABLE script right now) or Cancel (which would not do anything with the CREATE TABLE script and return you back to the previous screen where you could make more changes to the fields and/or table name). We want to create the table, so go ahead and click on the Update Database button.

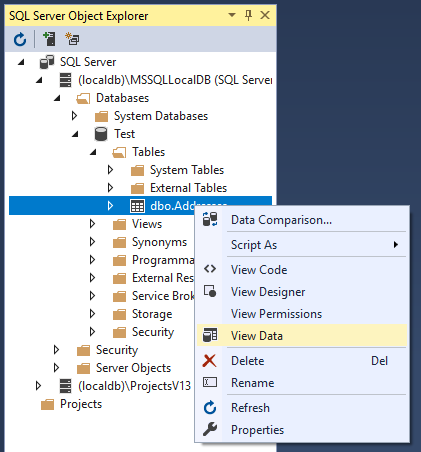
Down at the bottom of the screen, in the same area where our Error List and Immediate Window live, a new pane has appeared called Data Tools Operations:



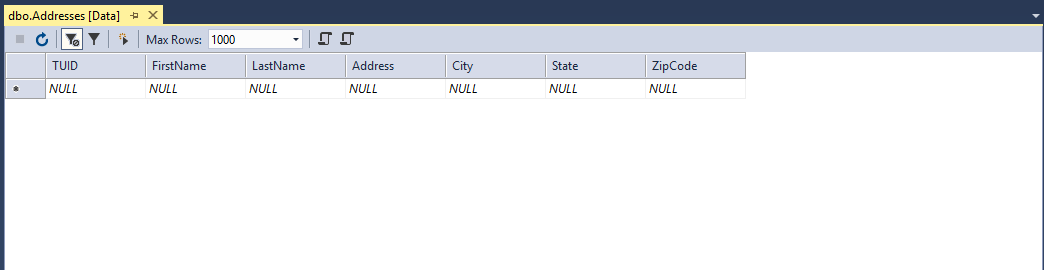
The green checkmark indicates that our T-SQL script ran fine, e.g. the database table was successfully created. We can view the actual script that was generated and executed. We can also view the results from SQL Server when it ran the script. These items are probably only of use to you if you’ve had a database class. Furthermore, there’s way more code happening in that script than the little bit we saw in the T-SQL pane. Sometimes knowing less is better…and that’s the way we will leave this. All I care about is that the update was successful (the green checkmark again shows that) and I can confirm that my table exists by looking over in the SSOE pane:



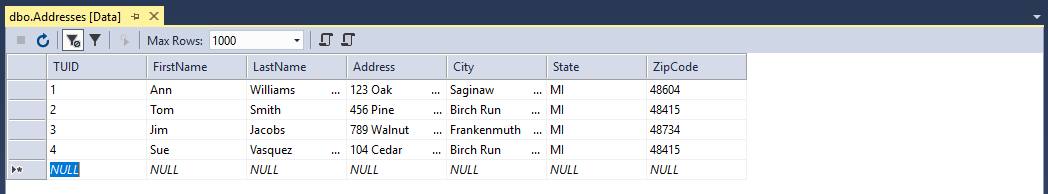
Alright! Good old Addresses exists. I can now close the center table building pane and the Data Tools Operations tab since I am done with them. You can reopen them if you need them… Our next step is to get some data into the database. This is pretty easy too, as VS provides you with a data editing window. Go ahead and right click on the Addresses table:



From the context menu, choose the View Data option. This will launch a new pane in the center of Visual Studio where the table builder used to be:



This is a live look at the data in your database. There’s nothing in there right now, so let’s add a couple of records by pointing, clicking and typing. Don’t worry about the red arrows that you see as you add records; they will disappear once you’ve entered values for the whole row. Here’s what my finished set of records look like:



That’s it! Since this is a live connection between Visual Studio and your SQL Server database, there’s nothing else to do or see here. As you typed each record, VS inserted the record into your database table. There’s no save button or anything else to do. You can close the data editor window now.

This concludes the SQL Server Object Explorer piece. We used it to create a new database and from there we added a table and data to the tables. I have shown you about 5% of what’s available in SSOE and the VS pieces that we looked at. This is a very comprehensive, powerful front end for working with SQL Server. However, the focus of this class is VB and so all of this was just a means to an end – getting a database built and populated with data so that we can learn how to access the database from VB.

Once you’ve taken a database course, you really owe it to yourself to go back to the SSOE, the table builder, the scripts generator and the data editor and familiarize yourself with what’s there. I hope that one day Microsoft will automatically bundle the full SQL Server Management Studio with Visual Studio, but I don’t think that will happen. Even when you install SQL Server today, you are often pointed out to the web to download the Management Studio as a separate piece of software. Microsoft has made this choice of unbundling the front end so that they can continue to update it as a separate product from the SQL Server database engine itself.

While what we have in VS isn’t as complete as the full SQL Server Management Studio, it’s still quite powerful. Each generation of Visual Studio adds more and more capabilities to it, so we are getting there!

# Working it with the Wizards

Now that we know how to create a SQL Server database, we want to turn our attention to how do we get VB to “talk” with the database. This section is going to look at the simplest way of making this happen with draggy-droppy, pointy-clicky.

You may recall that we started this chapter out by discussing Microsoft’s various database technologies. We concluded by saying that ADO.NET is the choice to be using today. Let’s start this out with a little "behind the scenes" on how ADO.NET works. There are two namespaces that we can use to interact with databases. You will import one or the other depending on which type of database you are working with. The first, System.Data.OleDb, is used for working with non-SQL Server databases such as Oracle, mySQL and Microsoft Access. The second namespace is System.Data.SqlClient, which is used exclusively for Microsoft SQL Server databases. Both namespaces contain classes to connect to and modify data sources.

Why do we have two different namespaces? Well, the SqlClient version is optimized to write as high performance SQL Server applications as possible. Microsoft owns both Visual Studio and SQL Server, so it shouldn’t surprise you that they have a specialized interface to allow the fastest database throughput code possible. Now that’s not to say there’s anything wrong with OleDB. Everything through OleDb works just fine, but it isn’t going to be as optimized for efficiency since it’s written for accessing *any* database. It is completely possible to drive SQL Server through the System.Data.OleDb namespace too (but unfortunately not the SQL Server File Databases that we will be using)! Since there’s a decent possibility that you may be going to work for a company that doesn’t use SQL Server, I simply want you to know that you can access those non-SQL Server databases by picking System.Data.OleDb and changing the prefixes from Sql to OleDb on the classes within the namespace as I’ll show in a minute. So, if you swap out Sql for OleDb, you’re pretty much golden!

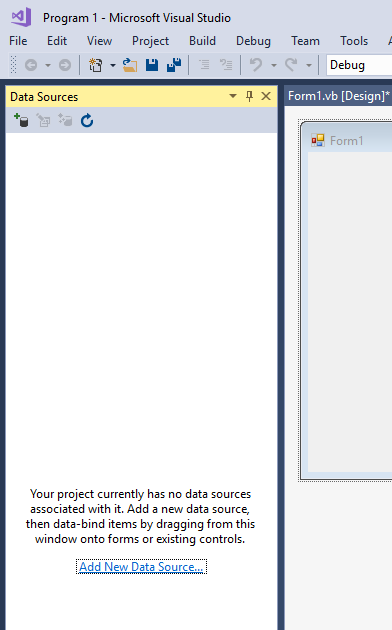
Some of the other classes that you will work with when using databases include System.Data.DataSet. This class consists of a set of data returned from a data source which are a database's tables and its relationships. The class OleDb/SqlConnection represents a communication connection to a data source. The third class, OleDb/SqlCommand, issues SQL commands that are to be executed on a data source.

Here's an overview of how the database connectivity works. You will populate a dataset through an instance of OleDb/SqlConnection. Once the dataset has data in it, VB will automatically disconnect from the data source. Only when changes are to be written out to the data source will the connection be re-established. This will be automatically handled by VB for you. You might wonder why we simply wouldn't keep the initial connection we made open to the database. By closing the connection and only reopening it when needed, we can increase the throughput capabilities of our system and reduce wasted resources.

The default connection in ADO.NET is disconnected, as we mentioned above. It is also possible to create a connection that stays open from the time we create it until our application closes it. These issues are beyond the time we have to discuss connections – so we will use the default connection-as-needed ADO.NET mode.

Now that we have a database built, it’s time to access it from Visual Basic. Go ahead and start up a new VB Windows Application project type. You need to carefully walk through this example to try to understand everything that is taking place. I will provide screenshots detailing each step that I ask you to perform.

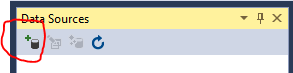
The first thing that you need to do is to create a connection to the database. VS provides you with a Data Sources tab to do just that. The following screenshot shows the Data Sources pane. Visual Studio turns this pane on by default at installation, but if you don’t see it, you can reactivate it through the View🡪Other Windows🡪Data Sources menu option:



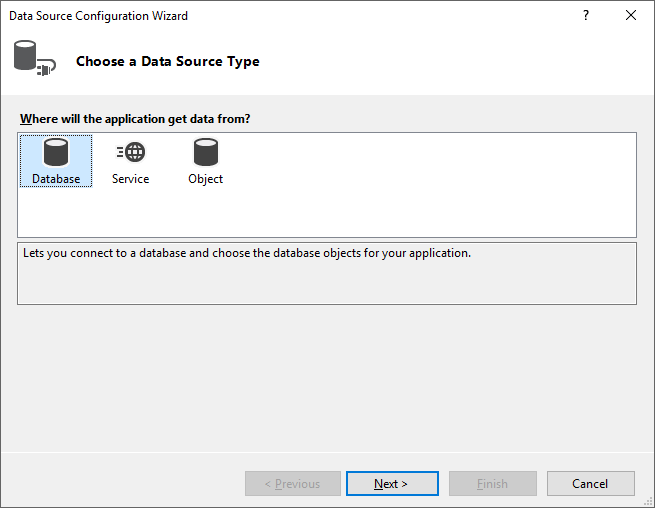
The Data Sources pane provides a quick way for you to wire up your application to a database. From the above screenshot you can see that we do not have any data sources attached to our current project. We need to remedy that problem since we want to pull in data from the Test database that we built in the previous section.

**Important Note:** If you happen to run into an error at any point that says something about the database file “being in use by another program,” all you need to do is go back under SQL Server Object Explorer, right click on the database filename and select Detach.

To form a data connection from within .NET to a database, you can either click the “Add New Data Source…” link in the middle of the Data Sources pane, or you can click on the first little plus sign icon in the Data Sources pane:

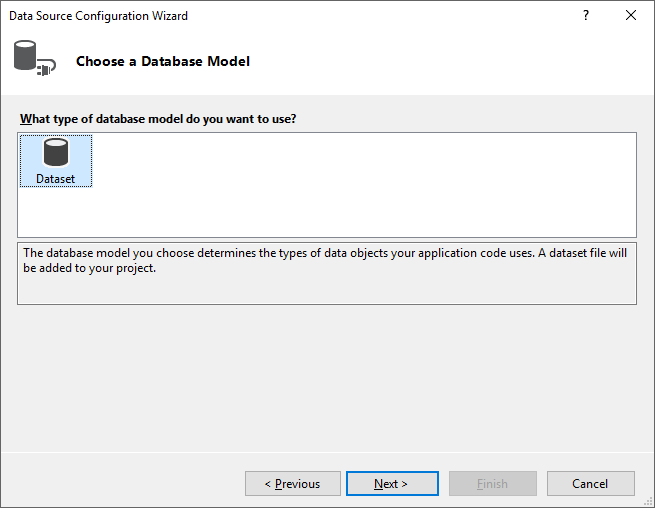


At this point, a wizard starts up that will walk you through the process of attaching a database to your project. The first screen that shows up is where you choose the Data Source Type:



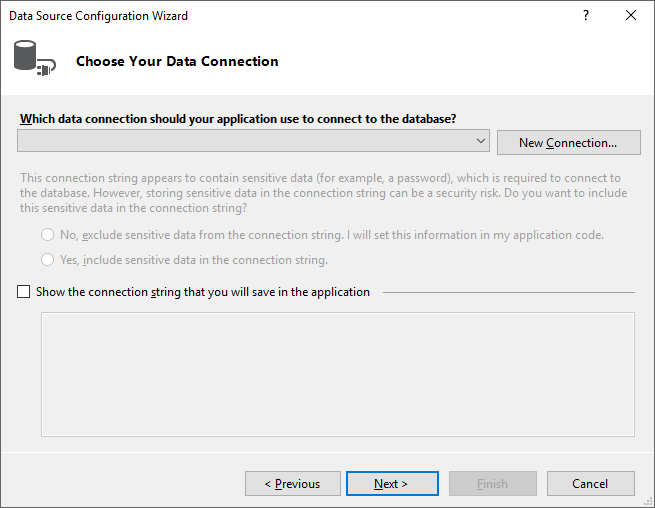
While we have three different choices available, we know that we want to work with our SQL Server LocalDB database file. We will make sure that Database is selected and then click on the Next button.

You’re now asked about what kind of a database model you want. You aren’t really given any choices other than DataSet:

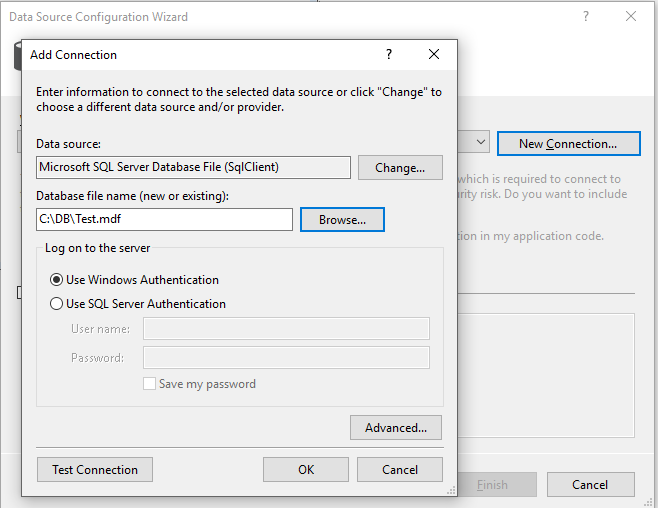


Click the Next button.

Here’s where we are asked about our Data Connection. Even if anything shows up in the dropdown box, you should press the New Connection button to configure a new data connection source:

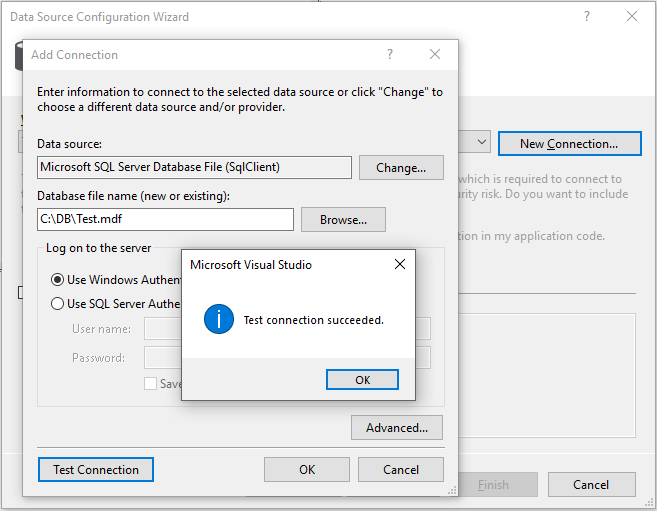


Once you’ve clicked the New Connection button, another screen pops up that you need to fill out:



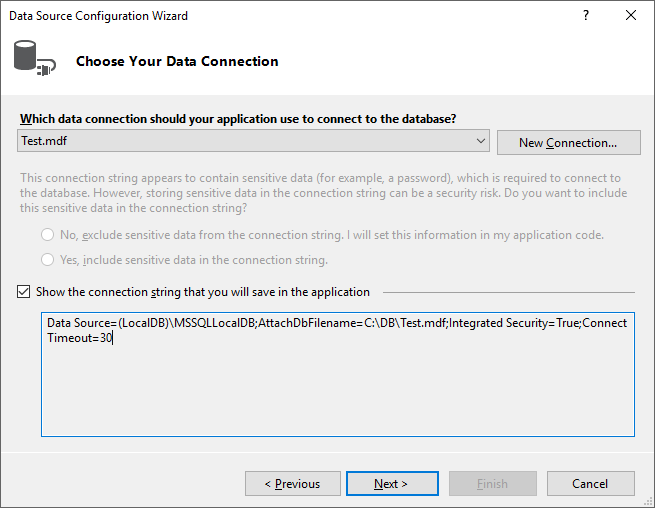
You can see that the Data Source needs to be set to a Microsoft SQL Server Database File (SqlClient). You use the Change button to modify the database source type. Once you’ve set the type correctly, you need to browse to where your database file is located. Remember that I stored mine in C:\DB and it is called Test.mdf. Before you get excited and press the OK button, we want to press the Test Connection button to make sure that our VB program can really talk to the SQL Server database.

I pressed the Test Connection button and after a few seconds, the following screen appears:



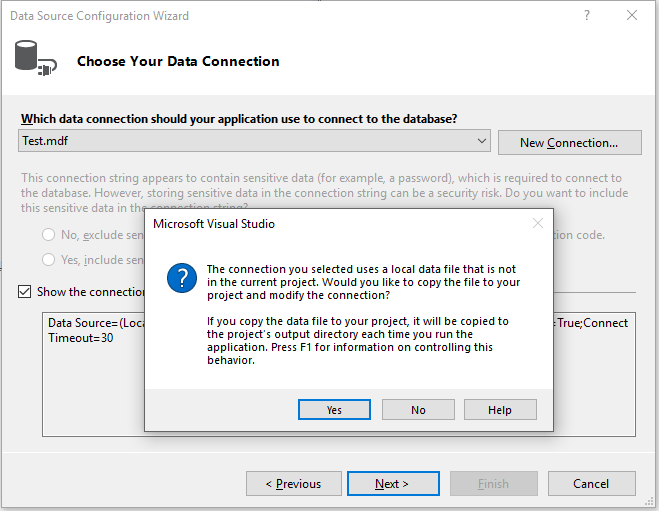
If you do not get the “Test connection succeeded.” message you are stuck until you get this problem resolved. Essentially not passing the test means that VB could not talk with your database server or that the database file could not be found. In any case, you won’t be going any farther until you figure out what’s wrong. Assuming the test passed, press the OK button to go back to the Add Connection screen. From the Add Connection screen, press the OK button on that screen as well.

You’re now back on the Data Source Configuration Wizard screen. Check the box at the bottom of the screen that says “Show the connection string that you will save in the application.” Your screen should look similar to this:



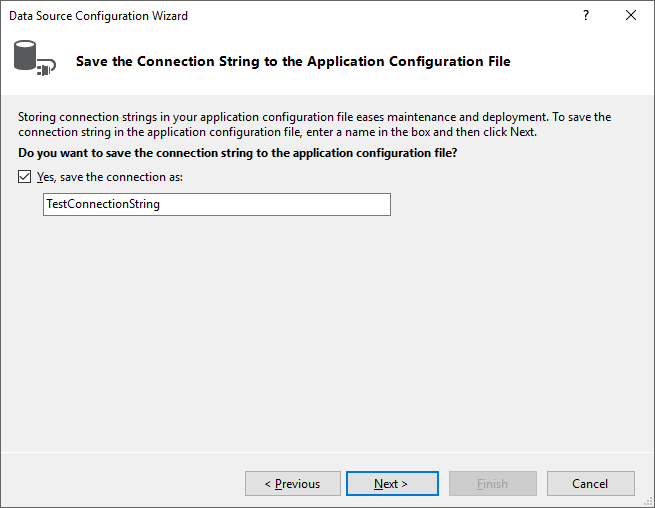
I am going to recommend that you jot down the connection string information that appears in the box under the checkmark – I have a strong feeling that you will need that later. Once you’ve got the connection string written down, press the Next button.

Another question appears for you:

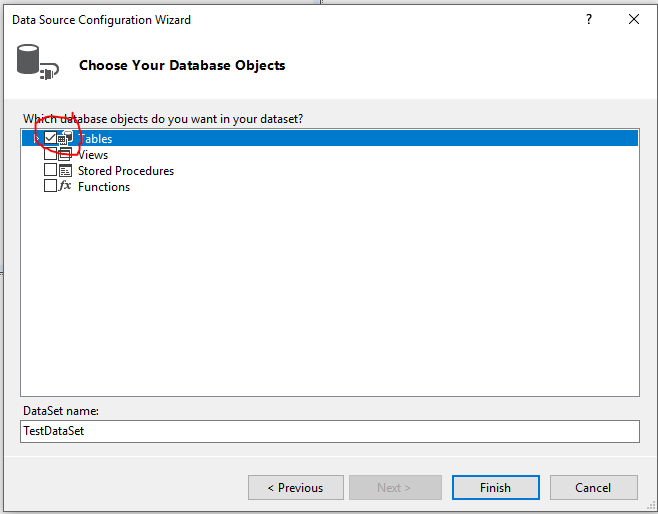


This is basically asking if you want a copy of the database file made every time you start your application. You do need to be somewhat careful in how you answer this question. Think about if you are pointing at a large production database. Do you think it’s going to be a good idea to have a project-based copy made every time you run your application? No, it’s not. But you may also be thinking, “Geez, I really don’t want to screw up the production database while I’m getting my code to work.” Good idea…so what do you do? Externally make a copy of the production database once, somewhere where only you and your program can access it and point to that copy. Friends don’t let VS clone databases for them. So, answer “No” to the question on the dialog box.

Finally, you are prompted about saving the connection string that was built into the project’s settings. By default, the Yes checkbox is marked. This is fine, so go ahead and press Next:

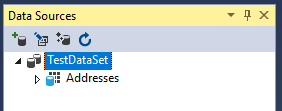


VS will now run out and talk to your database server to figure out what’s in the database. You’ll eventually be shown the following dialog that asks about what database items you want to have returned in your dataset. Check the Tables box and press Finish:



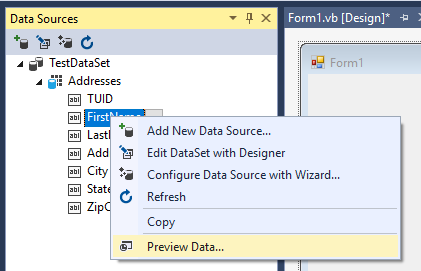
After a few seconds of watching the busy cursor, the wizard disappears. How rude!

If we take a careful look over at our Data Connections pane though, there is work that has been completed:

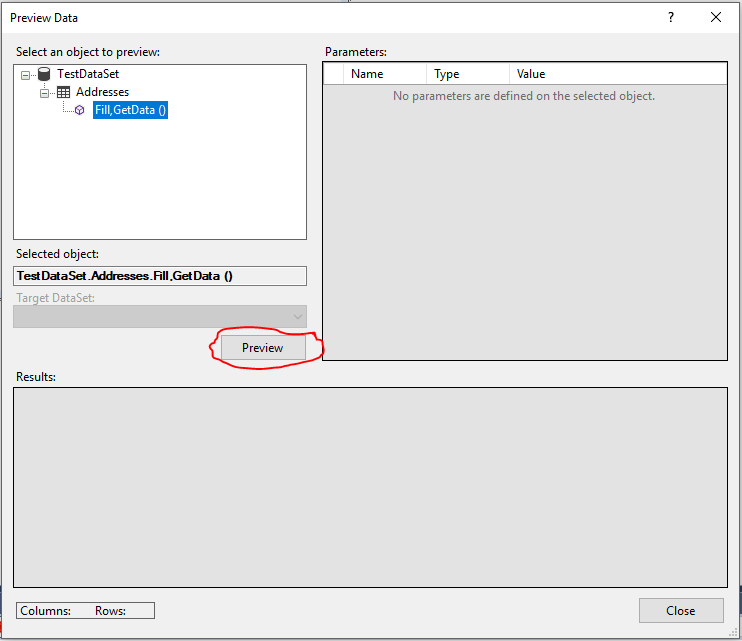


We see that we now have a Data Source that is pointing to our Addresses table in the database and that the data will be transferred back and forth between the application and our database using the TestDataSet dataset.

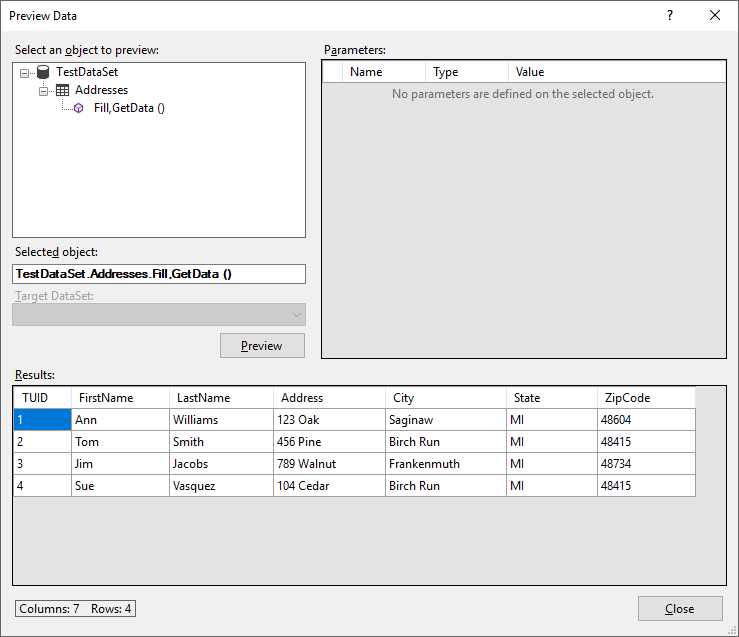
If we expand the Addresses arrow, we are shown all the fields that make up the table. If we right click on any of the fields, we get a context menu where we can Preview Data from the table:



Clicking the Preview Data menu option takes us to another screen where we can configure our preview. Just press the Preview button in the middle of the screen:

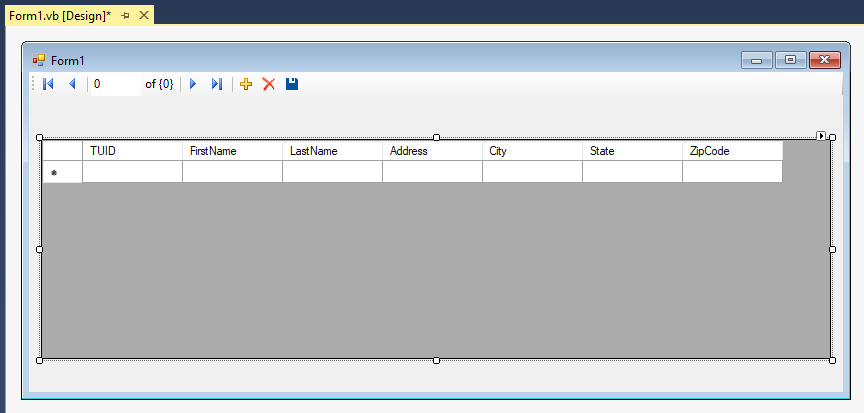


The preview results are shown at the bottom of the screen:

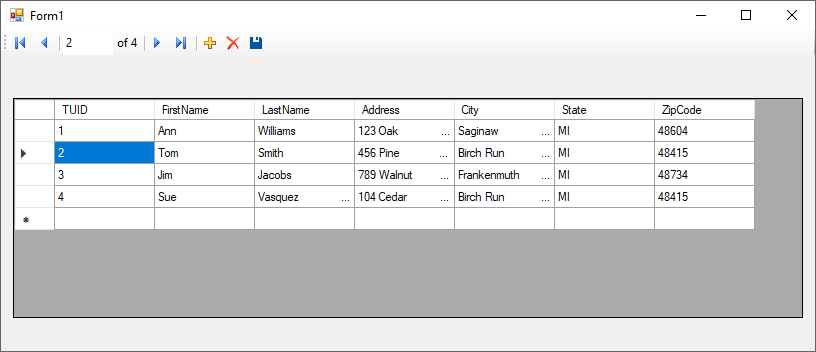


While it’s not nearly as nice as what we see in the SSOE, it at least lets us verify that we are bringing back the right data. You can press Close when you’re done looking at this screen. Also remember that if you want to edit the data, there is nothing stopping you from relaunching the SQL Server Object Explorer and going into the database and physically changing the data there. VS really is a one-stop shop for handling a lot of your database needs.

OK, deep breath: we created a data connection that essentially describes what kind of database we are going to talk to and where the database lives. The data connection ended up generating a connection string for us which it added into our project. Over in the Data Connection pane, we see that we have a connection to the database table we are interested in. So what? Big deal… Here’s comes the boom: drag and drop a copy of the Addresses data connection from the Data Connection pane and let go of it over your form. Wait a few seconds and after a bit of form resizing, you should have something that looks like this:

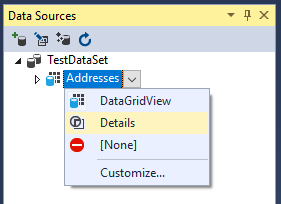


Run that application!

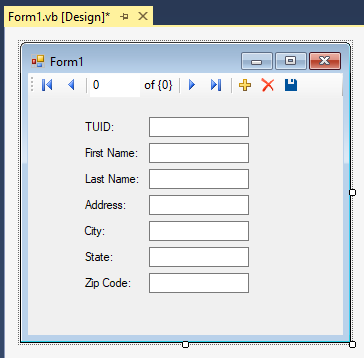


You just wrote a full-fledged database app without touching a single line of code! It uses a DataGridView control to show the addresses from the table. There’s even a button bar at the top that allows record navigation (including showing a record count), an add new record button (the +), delete record button (the x) and a save data button (the floppy). Too cool!

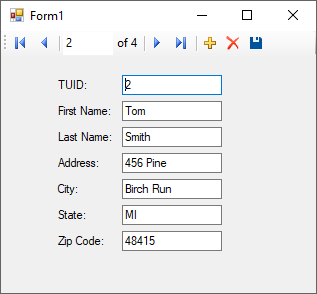
Want more? OK – try this. Delete the current form from your project. Add a new empty form back in. Go over to your Data Connections pane and click on the drop down chevron after the Addresses table:



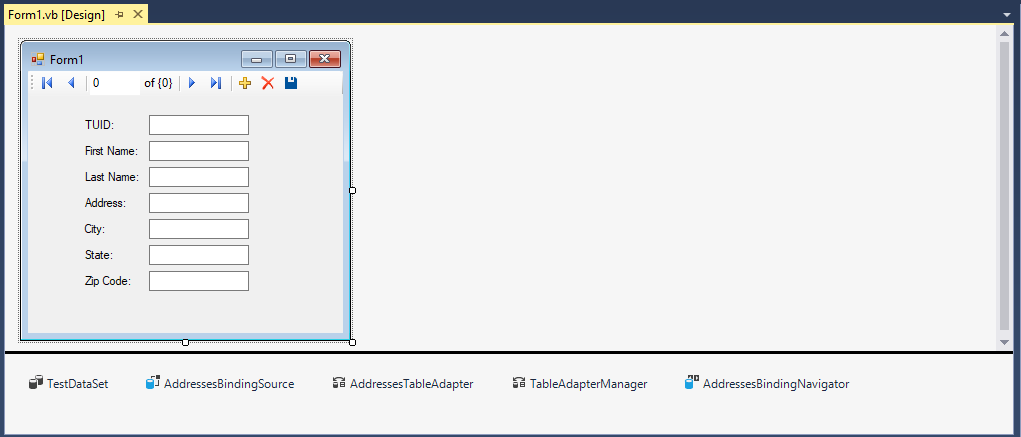
Click on Details. Then drag and drop the Addresses data connection over to your new empty form and let go. With a bit of resizing, we get something that looks like this:



Run this version of the application and we have the same functionality, but now instead of showing a DataGridView, only one record appears at a time with each field getting its own TextBox:



Fancy! If we go back over to the form designer in VS, we notice that several invisible controls were added to our project as well:



Let’s start out discussing these items by level of importance:

* AddressesTableAdapter is what allows communication between the Data Connection (remember all that wizard stuff we had to do?) and the DataSet.
* TestDataSet is a container where the contents of the database are placed for our program to work with them. Remember in our case, it is a disconnected dataset that is only updated when necessary.
* TableAdapterManager is another container object that ensures that table behavior is preserved. This means that we have something watching our inserts, updates and deletes to make sure they don’t violate anything, and in the case of tables that are joined, the TableAdapterManager will make sure that referential integrity is preserved.
* AddressesBindingSource is what ties the dataset’s data into the GUI representation on the form.
* AddressesBindingNavigator is what updates the GUI view whenever we move through the BindingSource. If you click on the next, previous, first or last buttons, notice the highlighted row in the DataGridView changes. This is due to the BindingNavigator. Think of it is a window, or pointer, that holds the current record we are examining in the database table.

So what about the code was written for us? Here’s the code behind the form (this is the version with the individual TextBoxes):

'Chapter 15 - Program 2

Public Class Form1

Private Sub AddressesBindingNavigatorSaveItem\_Click(sender As Object,

e As EventArgs)

Handles AddressesBindingNavigatorSaveItem.Click

Me.Validate()

Me.AddressesBindingSource.EndEdit()

Me.TableAdapterManager.UpdateAll(Me.TestDataSet)

End Sub

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

'TODO: This line of code loads data into the TestDataSet.Addresses

' table. You can move, or remove it, as needed.

Me.AddressesTableAdapter.Fill(Me.TestDataSet.Addresses)

End Sub

End Class

Wow! There’s not much too that – in fact it looks like we fill things up in Load and then everything else just deals with changes in the BindingNavigator. How hard can it be to learn to do things like this? Well, let’s look at the form’s Designer code (remember to see this, we have to click the Show All Files icon):

<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> \_

Partial Class Form1

Inherits System.Windows.Forms.Form

'Form overrides dispose to clean up the component list.

<System.Diagnostics.DebuggerNonUserCode()> \_

Protected Overrides Sub Dispose(ByVal disposing As Boolean)

Try

If disposing AndAlso components IsNot Nothing Then

components.Dispose()

End If

Finally

MyBase.Dispose(disposing)

End Try

End Sub

'Required by the Windows Form Designer

Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer

'It can be modified using the Windows Form Designer.

'Do not modify it using the code editor.

<System.Diagnostics.DebuggerStepThrough()> \_

Private Sub InitializeComponent()

Me.components = New System.ComponentModel.Container()

Dim resources As System.ComponentModel.ComponentResourceManager =

New System.ComponentModel.ComponentResourceManager(GetType(Form1))

Dim TUIDLabel As System.Windows.Forms.Label

Dim FirstNameLabel As System.Windows.Forms.Label

Dim LastNameLabel As System.Windows.Forms.Label

Dim AddressLabel As System.Windows.Forms.Label

Dim CityLabel As System.Windows.Forms.Label

Dim StateLabel As System.Windows.Forms.Label

Dim ZipCodeLabel As System.Windows.Forms.Label

Me.TestDataSet = New Program\_1.TestDataSet()

Me.AddressesBindingSource =

New System.Windows.Forms.BindingSource(Me.components)

Me.AddressesTableAdapter =

New Program\_1.TestDataSetTableAdapters.AddressesTableAdapter()

Me.TableAdapterManager =

New Program\_1.TestDataSetTableAdapters.TableAdapterManager()

Me.AddressesBindingNavigator =

New System.Windows.Forms.BindingNavigator(Me.components)

Me.BindingNavigatorMoveFirstItem =

New System.Windows.Forms.ToolStripButton()

Me.BindingNavigatorMovePreviousItem =

New System.Windows.Forms.ToolStripButton()

Me.BindingNavigatorSeparator =

New System.Windows.Forms.ToolStripSeparator()

Me.BindingNavigatorPositionItem =

New System.Windows.Forms.ToolStripTextBox()

Me.BindingNavigatorCountItem = New System.Windows.Forms.ToolStripLabel()

Me.BindingNavigatorSeparator1 =

New System.Windows.Forms.ToolStripSeparator()

Me.BindingNavigatorMoveNextItem =

New System.Windows.Forms.ToolStripButton()

Me.BindingNavigatorMoveLastItem =

New System.Windows.Forms.ToolStripButton()

Me.BindingNavigatorSeparator2 =

New System.Windows.Forms.ToolStripSeparator()

Me.BindingNavigatorAddNewItem = New System.Windows.Forms.ToolStripButton()

Me.BindingNavigatorDeleteItem = New System.Windows.Forms.ToolStripButton()

Me.AddressesBindingNavigatorSaveItem =

New System.Windows.Forms.ToolStripButton()

Me.TUIDTextBox = New System.Windows.Forms.TextBox()

Me.FirstNameTextBox = New System.Windows.Forms.TextBox()

Me.LastNameTextBox = New System.Windows.Forms.TextBox()

Me.AddressTextBox = New System.Windows.Forms.TextBox()

Me.CityTextBox = New System.Windows.Forms.TextBox()

Me.StateTextBox = New System.Windows.Forms.TextBox()

Me.ZipCodeTextBox = New System.Windows.Forms.TextBox()

TUIDLabel = New System.Windows.Forms.Label()

FirstNameLabel = New System.Windows.Forms.Label()

LastNameLabel = New System.Windows.Forms.Label()

AddressLabel = New System.Windows.Forms.Label()

CityLabel = New System.Windows.Forms.Label()

StateLabel = New System.Windows.Forms.Label()

ZipCodeLabel = New System.Windows.Forms.Label()

CType(Me.TestDataSet,

System.ComponentModel.ISupportInitialize).BeginInit()

CType(Me.AddressesBindingSource,

System.ComponentModel.ISupportInitialize).BeginInit()

CType(Me.AddressesBindingNavigator,

System.ComponentModel.ISupportInitialize).BeginInit()

Me.AddressesBindingNavigator.SuspendLayout()

Me.SuspendLayout()

'

'TestDataSet

'

Me.TestDataSet.DataSetName = "TestDataSet"

Me.TestDataSet.SchemaSerializationMode =

System.Data.SchemaSerializationMode.IncludeSchema

'

'AddressesBindingSource

'

Me.AddressesBindingSource.DataMember = "Addresses"

Me.AddressesBindingSource.DataSource = Me.TestDataSet

'

'AddressesTableAdapter

'

Me.AddressesTableAdapter.ClearBeforeFill = True

'

'TableAdapterManager

'

Me.TableAdapterManager.AddressesTableAdapter = Me.AddressesTableAdapter

Me.TableAdapterManager.BackupDataSetBeforeUpdate = False

Me.TableAdapterManager.UpdateOrder =

Program\_1.TestDataSetTableAdapters.TableAdapterManager

.UpdateOrderOption.InsertUpdateDelete

'

'AddressesBindingNavigator

'

Me.AddressesBindingNavigator.AddNewItem = Me.BindingNavigatorAddNewItem

Me.AddressesBindingNavigator.BindingSource = Me.AddressesBindingSource

Me.AddressesBindingNavigator.CountItem = Me.BindingNavigatorCountItem

Me.AddressesBindingNavigator.DeleteItem = Me.BindingNavigatorDeleteItem

Me.AddressesBindingNavigator.Items.AddRange(New

System.Windows.Forms.ToolStripItem()

{Me.BindingNavigatorMoveFirstItem,

Me.BindingNavigatorMovePreviousItem,

Me.BindingNavigatorSeparator, Me.BindingNavigatorPositionItem,

Me.BindingNavigatorCountItem, Me.BindingNavigatorSeparator1,

Me.BindingNavigatorMoveNextItem, Me.BindingNavigatorMoveLastItem,

Me.BindingNavigatorSeparator2, Me.BindingNavigatorAddNewItem,

Me.BindingNavigatorDeleteItem,

Me.AddressesBindingNavigatorSaveItem})

Me.AddressesBindingNavigator.Location = New System.Drawing.Point(0, 0)

Me.AddressesBindingNavigator.MoveFirstItem =

Me.BindingNavigatorMoveFirstItem

Me.AddressesBindingNavigator.MoveLastItem =

Me.BindingNavigatorMoveLastItem

Me.AddressesBindingNavigator.MoveNextItem =

Me.BindingNavigatorMoveNextItem

Me.AddressesBindingNavigator.MovePreviousItem =

Me.BindingNavigatorMovePreviousItem

Me.AddressesBindingNavigator.Name = "AddressesBindingNavigator"

Me.AddressesBindingNavigator.PositionItem =

Me.BindingNavigatorPositionItem

Me.AddressesBindingNavigator.Size = New System.Drawing.Size(315, 25)

Me.AddressesBindingNavigator.TabIndex = 0

Me.AddressesBindingNavigator.Text = "BindingNavigator1"

'

'BindingNavigatorMoveFirstItem

'

Me.BindingNavigatorMoveFirstItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorMoveFirstItem.Image =

CType(resources.GetObject("BindingNavigatorMoveFirstItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorMoveFirstItem.Name = "BindingNavigatorMoveFirstItem"

Me.BindingNavigatorMoveFirstItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorMoveFirstItem.Size = New System.Drawing.Size(23, 22)

Me.BindingNavigatorMoveFirstItem.Text = "Move first"

'

'BindingNavigatorMovePreviousItem

'

Me.BindingNavigatorMovePreviousItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorMovePreviousItem.Image =

CType(resources.GetObject("BindingNavigatorMovePreviousItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorMovePreviousItem.Name =

"BindingNavigatorMovePreviousItem"

Me.BindingNavigatorMovePreviousItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorMovePreviousItem.Size = New System.Drawing.Size(23, 22)

Me.BindingNavigatorMovePreviousItem.Text = "Move previous"

'

'BindingNavigatorSeparator

'

Me.BindingNavigatorSeparator.Name = "BindingNavigatorSeparator"

Me.BindingNavigatorSeparator.Size = New System.Drawing.Size(6, 25)

'

'BindingNavigatorPositionItem

'

Me.BindingNavigatorPositionItem.AccessibleName = "Position"

Me.BindingNavigatorPositionItem.AutoSize = False

Me.BindingNavigatorPositionItem.Name = "BindingNavigatorPositionItem"

Me.BindingNavigatorPositionItem.Size = New System.Drawing.Size(50, 23)

Me.BindingNavigatorPositionItem.Text = "0"

Me.BindingNavigatorPositionItem.ToolTipText = "Current position"

'

'BindingNavigatorCountItem

'

Me.BindingNavigatorCountItem.Name = "BindingNavigatorCountItem"

Me.BindingNavigatorCountItem.Size = New System.Drawing.Size(35, 15)

Me.BindingNavigatorCountItem.Text = "of {0}"

Me.BindingNavigatorCountItem.ToolTipText = "Total number of items"

'

'BindingNavigatorSeparator1

'

Me.BindingNavigatorSeparator1.Name = "BindingNavigatorSeparator"

Me.BindingNavigatorSeparator1.Size = New System.Drawing.Size(6, 6)

'

'BindingNavigatorMoveNextItem

'

Me.BindingNavigatorMoveNextItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorMoveNextItem.Image =

CType(resources.GetObject("BindingNavigatorMoveNextItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorMoveNextItem.Name = "BindingNavigatorMoveNextItem"

Me.BindingNavigatorMoveNextItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorMoveNextItem.Size = New System.Drawing.Size(23, 20)

Me.BindingNavigatorMoveNextItem.Text = "Move next"

'

'BindingNavigatorMoveLastItem

'

Me.BindingNavigatorMoveLastItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorMoveLastItem.Image =

CType(resources.GetObject("BindingNavigatorMoveLastItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorMoveLastItem.Name = "BindingNavigatorMoveLastItem"

Me.BindingNavigatorMoveLastItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorMoveLastItem.Size = New System.Drawing.Size(23, 20)

Me.BindingNavigatorMoveLastItem.Text = "Move last"

'

'BindingNavigatorSeparator2

'

Me.BindingNavigatorSeparator2.Name = "BindingNavigatorSeparator"

Me.BindingNavigatorSeparator2.Size = New System.Drawing.Size(6, 6)

'

'BindingNavigatorAddNewItem

'

Me.BindingNavigatorAddNewItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorAddNewItem.Image =

CType(resources.GetObject("BindingNavigatorAddNewItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorAddNewItem.Name = "BindingNavigatorAddNewItem"

Me.BindingNavigatorAddNewItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorAddNewItem.Size = New System.Drawing.Size(23, 22)

Me.BindingNavigatorAddNewItem.Text = "Add new"

'

'BindingNavigatorDeleteItem

'

Me.BindingNavigatorDeleteItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorDeleteItem.Image =

CType(resources.GetObject("BindingNavigatorDeleteItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorDeleteItem.Name = "BindingNavigatorDeleteItem"

Me.BindingNavigatorDeleteItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorDeleteItem.Size = New System.Drawing.Size(23, 20)

Me.BindingNavigatorDeleteItem.Text = "Delete"

'

'AddressesBindingNavigatorSaveItem

'

Me.AddressesBindingNavigatorSaveItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.AddressesBindingNavigatorSaveItem.Image =

CType(resources.GetObject("AddressesBindingNavigatorSaveItem.Image"),

System.Drawing.Image)

Me.AddressesBindingNavigatorSaveItem.Name =

"AddressesBindingNavigatorSaveItem"

Me.AddressesBindingNavigatorSaveItem.Size =

New System.Drawing.Size(23, 23)

Me.AddressesBindingNavigatorSaveItem.Text = "Save Data"

'

'TUIDLabel

'

TUIDLabel.AutoSize = True

TUIDLabel.Location = New System.Drawing.Point(54, 47)

TUIDLabel.Name = "TUIDLabel"

TUIDLabel.Size = New System.Drawing.Size(36, 13)

TUIDLabel.TabIndex = 1

TUIDLabel.Text = "TUID:"

'

'TUIDTextBox

'

Me.TUIDTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "TUID", True))

Me.TUIDTextBox.Location = New System.Drawing.Point(121, 44)

Me.TUIDTextBox.Name = "TUIDTextBox"

Me.TUIDTextBox.Size = New System.Drawing.Size(100, 20)

Me.TUIDTextBox.TabIndex = 2

'

'FirstNameLabel

'

FirstNameLabel.AutoSize = True

FirstNameLabel.Location = New System.Drawing.Point(54, 73)

FirstNameLabel.Name = "FirstNameLabel"

FirstNameLabel.Size = New System.Drawing.Size(60, 13)

FirstNameLabel.TabIndex = 3

FirstNameLabel.Text = "First Name:"

'

'FirstNameTextBox

'

Me.FirstNameTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"FirstName", True))

Me.FirstNameTextBox.Location = New System.Drawing.Point(121, 70)

Me.FirstNameTextBox.Name = "FirstNameTextBox"

Me.FirstNameTextBox.Size = New System.Drawing.Size(100, 20)

Me.FirstNameTextBox.TabIndex = 4

'

'LastNameLabel

'

LastNameLabel.AutoSize = True

LastNameLabel.Location = New System.Drawing.Point(54, 99)

LastNameLabel.Name = "LastNameLabel"

LastNameLabel.Size = New System.Drawing.Size(61, 13)

LastNameLabel.TabIndex = 5

LastNameLabel.Text = "Last Name:"

'

'LastNameTextBox

'

Me.LastNameTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"LastName", True))

Me.LastNameTextBox.Location = New System.Drawing.Point(121, 96)

Me.LastNameTextBox.Name = "LastNameTextBox"

Me.LastNameTextBox.Size = New System.Drawing.Size(100, 20)

Me.LastNameTextBox.TabIndex = 6

'

'AddressLabel

'

AddressLabel.AutoSize = True

AddressLabel.Location = New System.Drawing.Point(54, 125)

AddressLabel.Name = "AddressLabel"

AddressLabel.Size = New System.Drawing.Size(48, 13)

AddressLabel.TabIndex = 7

AddressLabel.Text = "Address:"

'

'AddressTextBox

'

Me.AddressTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"Address", True))

Me.AddressTextBox.Location = New System.Drawing.Point(121, 122)

Me.AddressTextBox.Name = "AddressTextBox"

Me.AddressTextBox.Size = New System.Drawing.Size(100, 20)

Me.AddressTextBox.TabIndex = 8

'

'CityLabel

'

CityLabel.AutoSize = True

CityLabel.Location = New System.Drawing.Point(54, 151)

CityLabel.Name = "CityLabel"

CityLabel.Size = New System.Drawing.Size(27, 13)

CityLabel.TabIndex = 9

CityLabel.Text = "City:"

'

'CityTextBox

'

Me.CityTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "City", True))

Me.CityTextBox.Location = New System.Drawing.Point(121, 148)

Me.CityTextBox.Name = "CityTextBox"

Me.CityTextBox.Size = New System.Drawing.Size(100, 20)

Me.CityTextBox.TabIndex = 10

'

'StateLabel

'

StateLabel.AutoSize = True

StateLabel.Location = New System.Drawing.Point(54, 177)

StateLabel.Name = "StateLabel"

StateLabel.Size = New System.Drawing.Size(35, 13)

StateLabel.TabIndex = 11

StateLabel.Text = "State:"

'

'StateTextBox

'

Me.StateTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "State", True))

Me.StateTextBox.Location = New System.Drawing.Point(121, 174)

Me.StateTextBox.Name = "StateTextBox"

Me.StateTextBox.Size = New System.Drawing.Size(100, 20)

Me.StateTextBox.TabIndex = 12

'

'ZipCodeLabel

'

ZipCodeLabel.AutoSize = True

ZipCodeLabel.Location = New System.Drawing.Point(54, 203)

ZipCodeLabel.Name = "ZipCodeLabel"

ZipCodeLabel.Size = New System.Drawing.Size(53, 13)

ZipCodeLabel.TabIndex = 13

ZipCodeLabel.Text = "Zip Code:"

'

'ZipCodeTextBox

'

Me.ZipCodeTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"ZipCode", True))

Me.ZipCodeTextBox.Location = New System.Drawing.Point(121, 200)

Me.ZipCodeTextBox.Name = "ZipCodeTextBox"

Me.ZipCodeTextBox.Size = New System.Drawing.Size(100, 20)

Me.ZipCodeTextBox.TabIndex = 14

'

'BindingNavigatorSeparator2

'

Me.BindingNavigatorSeparator2.Name = "BindingNavigatorSeparator"

Me.BindingNavigatorSeparator2.Size = New System.Drawing.Size(6, 6)

'

'BindingNavigatorAddNewItem

'

Me.BindingNavigatorAddNewItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorAddNewItem.Image =

CType(resources.GetObject("BindingNavigatorAddNewItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorAddNewItem.Name = "BindingNavigatorAddNewItem"

Me.BindingNavigatorAddNewItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorAddNewItem.Size = New System.Drawing.Size(23, 22)

Me.BindingNavigatorAddNewItem.Text = "Add new"

'

'BindingNavigatorDeleteItem

'

Me.BindingNavigatorDeleteItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.BindingNavigatorDeleteItem.Image =

CType(resources.GetObject("BindingNavigatorDeleteItem.Image"),

System.Drawing.Image)

Me.BindingNavigatorDeleteItem.Name = "BindingNavigatorDeleteItem"

Me.BindingNavigatorDeleteItem.RightToLeftAutoMirrorImage = True

Me.BindingNavigatorDeleteItem.Size = New System.Drawing.Size(23, 20)

Me.BindingNavigatorDeleteItem.Text = "Delete"

'

'AddressesBindingNavigatorSaveItem

'

Me.AddressesBindingNavigatorSaveItem.DisplayStyle =

System.Windows.Forms.ToolStripItemDisplayStyle.Image

Me.AddressesBindingNavigatorSaveItem.Image =

CType(resources.GetObject("AddressesBindingNavigatorSaveItem.Image"),

System.Drawing.Image)

Me.AddressesBindingNavigatorSaveItem.Name =

"AddressesBindingNavigatorSaveItem"

Me.AddressesBindingNavigatorSaveItem.Size =

New System.Drawing.Size(23, 23)

Me.AddressesBindingNavigatorSaveItem.Text = "Save Data"

'

'TUIDLabel

'

TUIDLabel.AutoSize = True

TUIDLabel.Location = New System.Drawing.Point(54, 47)

TUIDLabel.Name = "TUIDLabel"

TUIDLabel.Size = New System.Drawing.Size(36, 13)

TUIDLabel.TabIndex = 1

TUIDLabel.Text = "TUID:"

'

'TUIDTextBox

'

Me.TUIDTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "TUID", True))

Me.TUIDTextBox.Location = New System.Drawing.Point(121, 44)

Me.TUIDTextBox.Name = "TUIDTextBox"

Me.TUIDTextBox.Size = New System.Drawing.Size(100, 20)

Me.TUIDTextBox.TabIndex = 2

'

'FirstNameLabel

'

FirstNameLabel.AutoSize = True

FirstNameLabel.Location = New System.Drawing.Point(54, 73)

FirstNameLabel.Name = "FirstNameLabel"

FirstNameLabel.Size = New System.Drawing.Size(60, 13)

FirstNameLabel.TabIndex = 3

FirstNameLabel.Text = "First Name:"

'

'FirstNameTextBox

'

Me.FirstNameTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"FirstName", True))

Me.FirstNameTextBox.Location = New System.Drawing.Point(121, 70)

Me.FirstNameTextBox.Name = "FirstNameTextBox"

Me.FirstNameTextBox.Size = New System.Drawing.Size(100, 20)

Me.FirstNameTextBox.TabIndex = 4

'

'LastNameLabel

'

LastNameLabel.AutoSize = True

LastNameLabel.Location = New System.Drawing.Point(54, 99)

LastNameLabel.Name = "LastNameLabel"

LastNameLabel.Size = New System.Drawing.Size(61, 13)

LastNameLabel.TabIndex = 5

LastNameLabel.Text = "Last Name:"

'

'LastNameTextBox

'

Me.LastNameTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"LastName", True))

Me.LastNameTextBox.Location = New System.Drawing.Point(121, 96)

Me.LastNameTextBox.Name = "LastNameTextBox"

Me.LastNameTextBox.Size = New System.Drawing.Size(100, 20)

Me.LastNameTextBox.TabIndex = 6

'

'AddressLabel

'

AddressLabel.AutoSize = True

AddressLabel.Location = New System.Drawing.Point(54, 125)

AddressLabel.Name = "AddressLabel"

AddressLabel.Size = New System.Drawing.Size(48, 13)

AddressLabel.TabIndex = 7

AddressLabel.Text = "Address:"

'

'AddressTextBox

'

Me.AddressTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"Address", True))

Me.AddressTextBox.Location = New System.Drawing.Point(121, 122)

Me.AddressTextBox.Name = "AddressTextBox"

Me.AddressTextBox.Size = New System.Drawing.Size(100, 20)

Me.AddressTextBox.TabIndex = 8

'

'CityLabel

'

CityLabel.AutoSize = True

CityLabel.Location = New System.Drawing.Point(54, 151)

CityLabel.Name = "CityLabel"

CityLabel.Size = New System.Drawing.Size(27, 13)

CityLabel.TabIndex = 9

CityLabel.Text = "City:"

'

'CityTextBox

'

Me.CityTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "City", True))

Me.CityTextBox.Location = New System.Drawing.Point(121, 148)

Me.CityTextBox.Name = "CityTextBox"

Me.CityTextBox.Size = New System.Drawing.Size(100, 20)

Me.CityTextBox.TabIndex = 10

'

'StateLabel

'

StateLabel.AutoSize = True

StateLabel.Location = New System.Drawing.Point(54, 177)

StateLabel.Name = "StateLabel"

StateLabel.Size = New System.Drawing.Size(35, 13)

StateLabel.TabIndex = 11

StateLabel.Text = "State:"

'

'StateTextBox

'

Me.StateTextBox.DataBindings.Add(New System.Windows.Forms.Binding("Text",

Me.AddressesBindingSource, "State", True))

Me.StateTextBox.Location = New System.Drawing.Point(121, 174)

Me.StateTextBox.Name = "StateTextBox"

Me.StateTextBox.Size = New System.Drawing.Size(100, 20)

Me.StateTextBox.TabIndex = 12

'

'ZipCodeLabel

'

ZipCodeLabel.AutoSize = True

ZipCodeLabel.Location = New System.Drawing.Point(54, 203)

ZipCodeLabel.Name = "ZipCodeLabel"

ZipCodeLabel.Size = New System.Drawing.Size(53, 13)

ZipCodeLabel.TabIndex = 13

ZipCodeLabel.Text = "Zip Code:"

'

'ZipCodeTextBox

'

Me.ZipCodeTextBox.DataBindings.Add(New

System.Windows.Forms.Binding("Text", Me.AddressesBindingSource,

"ZipCode", True))

Me.ZipCodeTextBox.Location = New System.Drawing.Point(121, 200)

Me.ZipCodeTextBox.Name = "ZipCodeTextBox"

Me.ZipCodeTextBox.Size = New System.Drawing.Size(100, 20)

Me.ZipCodeTextBox.TabIndex = 14

'

'Form1

'

Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)

Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font

Me.ClientSize = New System.Drawing.Size(315, 262)

Me.Controls.Add(TUIDLabel)

Me.Controls.Add(Me.TUIDTextBox)

Me.Controls.Add(FirstNameLabel)

Me.Controls.Add(Me.FirstNameTextBox)

Me.Controls.Add(LastNameLabel)

Me.Controls.Add(Me.LastNameTextBox)

Me.Controls.Add(AddressLabel)

Me.Controls.Add(Me.AddressTextBox)

Me.Controls.Add(CityLabel)

Me.Controls.Add(Me.CityTextBox)

Me.Controls.Add(StateLabel)

Me.Controls.Add(Me.StateTextBox)

Me.Controls.Add(ZipCodeLabel)

Me.Controls.Add(Me.ZipCodeTextBox)

Me.Controls.Add(Me.AddressesBindingNavigator)

Me.Name = "Form1"

Me.Text = "Form1"

CType(Me.TestDataSet, System.ComponentModel.ISupportInitialize).EndInit()

CType(Me.AddressesBindingSource,

System.ComponentModel.ISupportInitialize).EndInit()

CType(Me.AddressesBindingNavigator,

System.ComponentModel.ISupportInitialize).EndInit()

Me.AddressesBindingNavigator.ResumeLayout(False)

Me.AddressesBindingNavigator.PerformLayout()

Me.ResumeLayout(False)

Me.PerformLayout()

End Sub

Friend WithEvents TestDataSet As TestDataSet

Friend WithEvents AddressesBindingSource As BindingSource

Friend WithEvents AddressesTableAdapter As

TestDataSetTableAdapters.AddressesTableAdapter

Friend WithEvents TableAdapterManager As

TestDataSetTableAdapters.TableAdapterManager

Friend WithEvents AddressesBindingNavigator As BindingNavigator

Friend WithEvents BindingNavigatorAddNewItem As ToolStripButton

Friend WithEvents BindingNavigatorCountItem As ToolStripLabel

Friend WithEvents BindingNavigatorDeleteItem As ToolStripButton

Friend WithEvents BindingNavigatorMoveFirstItem As ToolStripButton

Friend WithEvents BindingNavigatorMovePreviousItem As ToolStripButton

Friend WithEvents BindingNavigatorSeparator As ToolStripSeparator

Friend WithEvents BindingNavigatorPositionItem As ToolStripTextBox

Friend WithEvents BindingNavigatorSeparator1 As ToolStripSeparator

Friend WithEvents BindingNavigatorMoveNextItem As ToolStripButton

Friend WithEvents BindingNavigatorMoveLastItem As ToolStripButton

Friend WithEvents BindingNavigatorSeparator2 As ToolStripSeparator

Friend WithEvents AddressesBindingNavigatorSaveItem As ToolStripButton

Friend WithEvents TUIDTextBox As TextBox

Friend WithEvents FirstNameTextBox As TextBox

Friend WithEvents LastNameTextBox As TextBox

Friend WithEvents AddressTextBox As TextBox

Friend WithEvents CityTextBox As TextBox

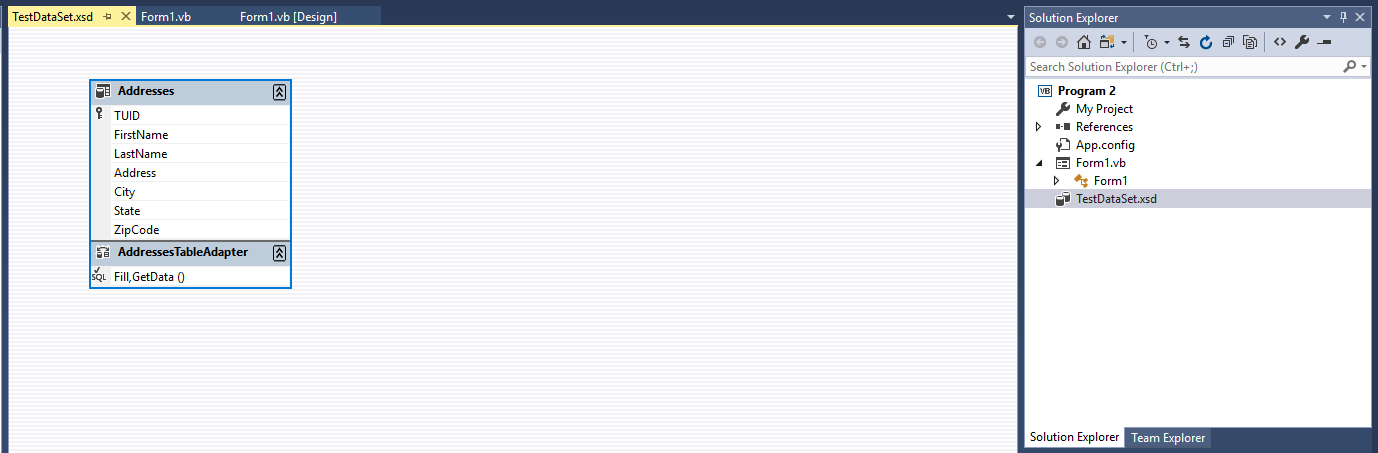
Friend WithEvents StateTextBox As TextBox

Friend WithEvents ZipCodeTextBox As TextBox

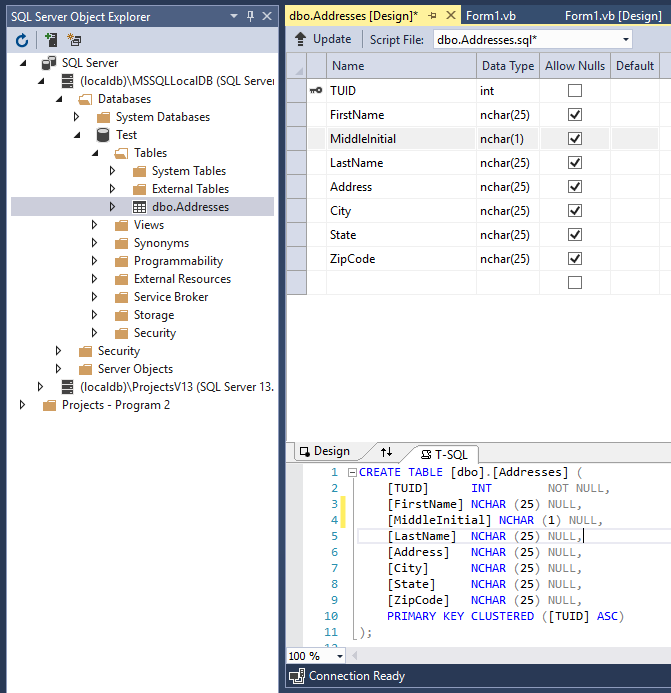
End Class

Yikes! We found where a whole bunch of nastiness was hidden. But, if we wade through it, we still see that it’s not horribly bad to duplicate. Not that we’d want to, mind you…

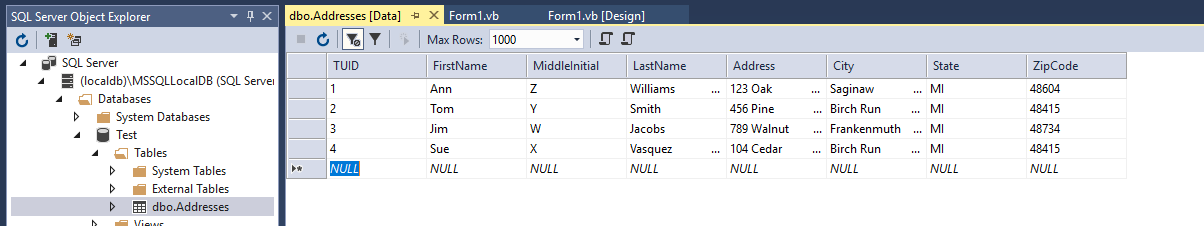
Want another piece to the mystery of what’s happening behind the scenes? Did you notice that a new file called TestDataSet.xsd shows up in the project’s Solution Explorer? Click on it to open the file up, and voila, Visual Basic shows an XML Schema Definition window (called XSD, hence the .xsd file extension) showing the makeup of the data source and how the adapter fills the dataset from it:



Go ahead and close the XSD designer window. So, now the question you should ask is, “Well all of this stuff is fine and good, but what happens if I need to change the underlying database?” Let’s find out by adding a Middle Initial column to our database using SSOE:

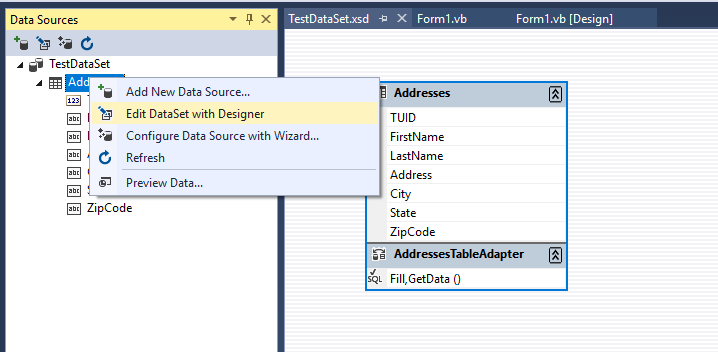


After I’ve updated the database to save the change to the database schema, I’ll view the data under SSOE and add middle initials to each record:



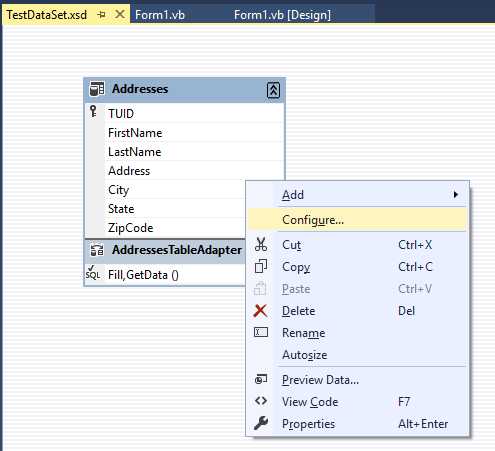
If I run my application at this point, I don’t see any of the new middle initial information and that’s because all of the backing information is still looking at the “old” XSD view of the database. Basically, we’re hosed…we will need to do whatever manipulation is necessary to bring the new field in – that’s problem number one with using magic: it works great on new stuff initially, but make a change and you get a headache.

So, if I do want to continue down this path, I would need to go back to my Data Sources window and click on the Edit DataSet with Designer:

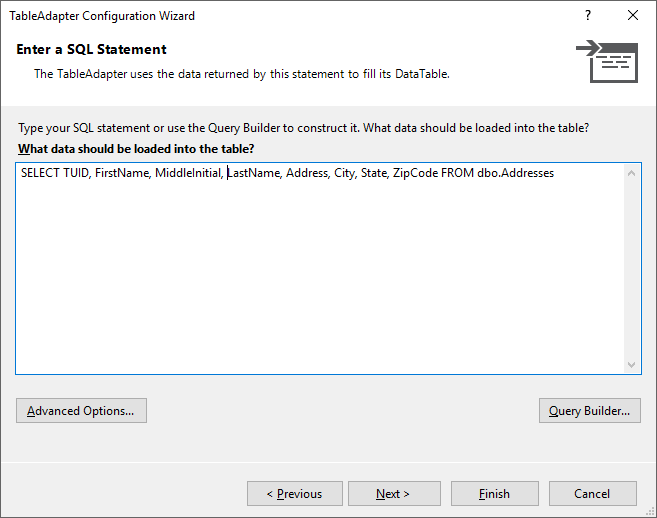


This brings up the same XSD designer page that we previously saw in the VS center pane.

If I right-click on the XSD diagram and select Configure from the context menu, I can modify the SQL to bring back the new field. Of course this presumes that you know SQL DML:

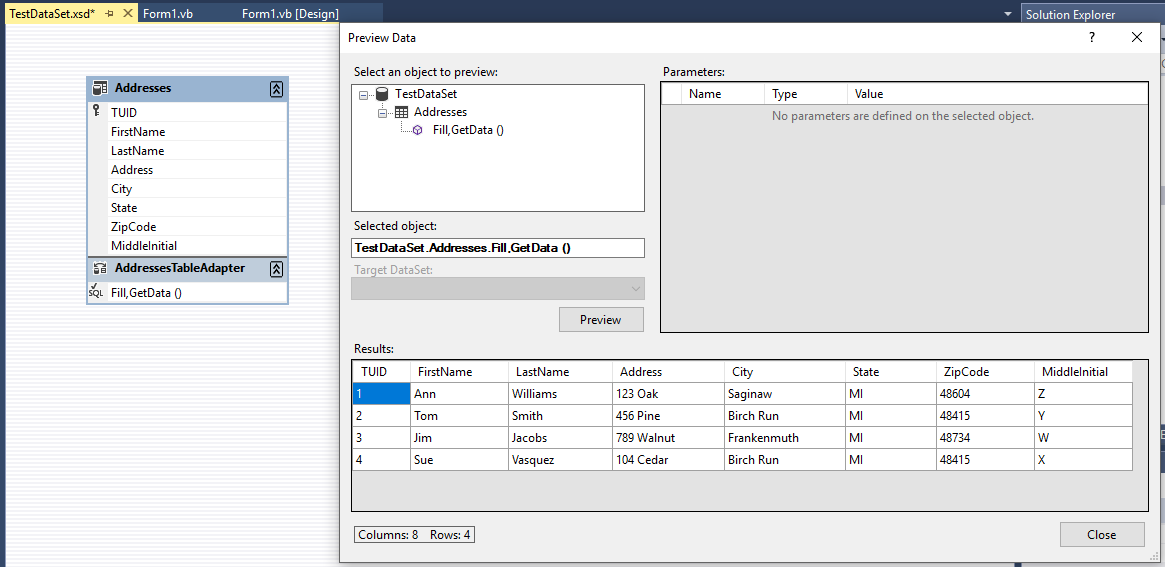


I need to modify the SQL query to pull the Middle Initial field when data comes back from the database. I typed the field name directly into the query, but you could use the Query Builder facility too:



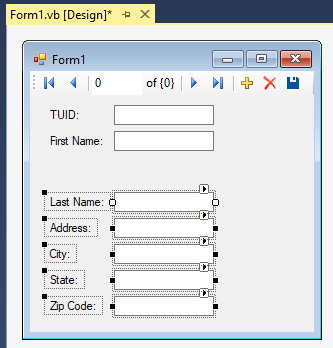
I pressed the Finish button to add the new field.

After a few seconds, the XSD diagram updates to show the new field is now being pulled. I then right clicked on the XSD diagram and chose Preview Data to make sure that the data is indeed coming back, notice it is the last column – not that it really matters:

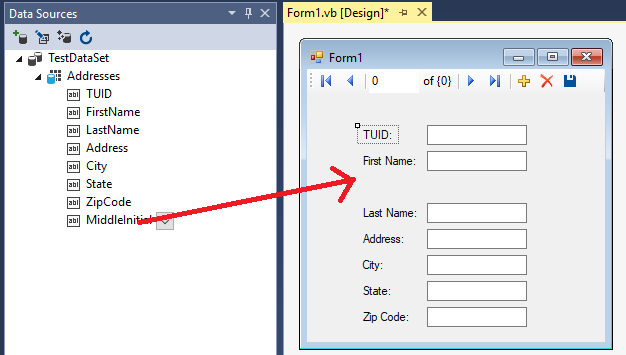


What’s important here is that we are pulling the new data from the database.

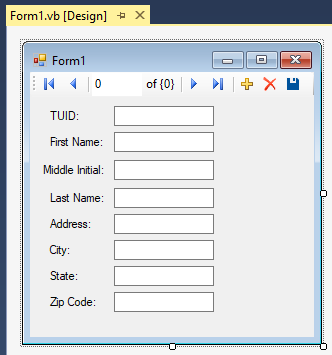
Now, I would have to go in and add the new field to the form and worry about getting it wired up to the BindingNavigator – Yuck! In our case, since it is one field and a very simple text field, we can just create room where we want the field and its label to be placed. I shifted everything below First Name downward to make space to accommodate the new field:



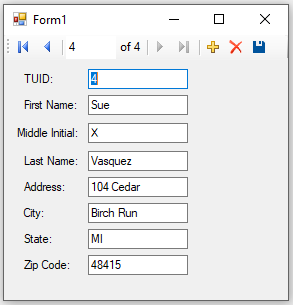
Next, we would go back over to the Data Connection pane and expand the Addresses dataset so that we can see all the individual fields. We would then drag an drop the Middle Initial field onto our form:



Once we let go, the textbox and its label can be repositioned wherever we want on the form so that everything looks right:



If we run the application now, we can see that we are able to display the middle initial field successfully:



Therein lies potential problem number two with this method: you’ve got no real understanding of where and what VB has done for you/to you. We ran a gamble that VB would wire the middle initial field up for us when we added it to our XSD. Let’s face it, VB isn’t really that smart or dependable because while it wasn’t bad too add the field to the database, we had to reconfigure the XSD to see the new field. VB didn’t help us out one bit there!

In this case, when we wanted to add Middle Initial from the Data Connection pane to our form it was pretty simple, mostly because the field was just a simple text field. However, if this had been something more complicated, like part of a join, we may not have been so lucky. It’s always better to be in the driver’s seat knowing what’s going on rather than hoping for the best!

The only other issue that you need to be aware of is regarding making changes to the data displayed in the DataGridView. You need to be careful in thinking that the data that you are seeing is a "live" copy because it is not. You do have the ability to make changes to the data that is in the DataGridView, but these changes are not automatically written out to the database. In other words, if you make some changes, exit the application and then start the application back up, you will see that all your changes disappeared and the database looks exactly like it did before you made them. If you want to save the changes out to the database, then there is additional work that you must do.

I have shown you the wizards that can help build the simplest of database applications. DO NOT USE THEM – I WILL GIVE YOU 0 POINTS IF YOU DO! Pointy-clicky draggy-droppy is extremely dangerous when you don’t know how things are being done. You need to take responsibility and build database applications where you are in control from the get go. The next section specifically addresses this!

# Working with Databases Through VB Code

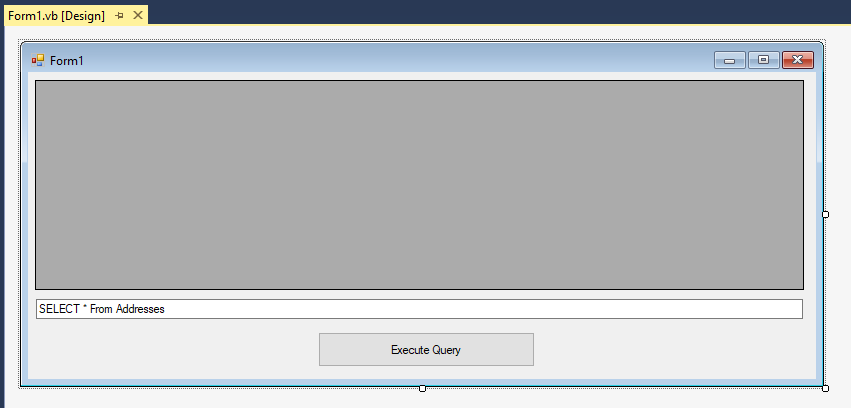
So, we’ve covered a lot of territory, although in the long run, not all of it will get us to where we want to be. Hopefully, with the negative aspects I pointed out above, I’ve now convinced you to not do databases through draggy-droppy since you really can’t easily see what VB’s done under the hood. Every time we make a change to our underlying data source, we’ve got rework ranging from easy (a DataGridView refresh of the data source) to hard (controls that are bound) that we have to add ourselves. While it might be more work, it’s better in my opinion to avoid magic and understand how things operate so that we can be in control of any changes that need to be made there.

Let’s look at the important ADO.NET objects we need to be able to work with. Remember there are versions of these objects under both the System.Data.SqlClient and System.Data.OleDb namespaces:

* Connection objects – again, this directs the communication between your program and the data source.
* DataAdapter objects – this is what makes the connection between the data source and your program’s DataSets. Think of this as the communication wire between the database and your program.
* DataSet objects – like mini databases, made up of DataTables and DataRelations – we can fill them from a database or completely build them up in memory.
* Command objects – this is how we execute our various SQL statements using the connection object.
* CommandBuilder objects – if our database tables are set up correctly, e.g. unique primary keys and what not, the CommandBuilder can write the SELECT, UPDATE, INSERT and DELETE SQL statements for us.
* DataReader objects – sometimes we just care about retrieving results quickly and efficiently – no changing, no backward reading; just pass through and print – that’s the time when you want to use a DataReader.

So now that we know who the team players are, let’s build up an application from scratch that connects to our database, fills a DataSet, hooks the DataSet to a DataGridView and displays the results. Oh yeah, let’s also throw in a TextBox that allows us to selectively filter records. The DataGridView is called dgvResults, the TextBox is called txtQuery and the Command button to run the query is called cmdExecuteQuery.

Here’s the form with our three controls on it:



Here’s the code for the application:

'Chapter 15 - Program 3

'Make sure that we bring in the SqlClient namespace to be able to

'access the SQL Server database file

Imports System.Data.SqlClient

Public Class Form1

'Here's where we will define the database; where it lives and how to access

'it. I am going to build this program up in as neutral a way as possible so

'that if you have to use OleDb instead of SqlClient, you know what the changes

'are

'Let's create a constant that points at the location of our database file

Public Const gstrDBName As String = "C:\DB\Test.mdf"

'We need to create a connection string that talks about where and how to

'access the database. Since we are using SqlClient, we don't need to

'indicate a provider -- basically the driver used to talk to the database.

'SqlClient only knows how to talk to one kind of database, so it's not

'necessary

'If you were using OleDb, you would have to include a Provider in the string

Public Const gstrConnString As String = "Server=(localdb)\MSSQLLocalDB;" &

"Database=Addresses;Integrated Security=SSPI;AttachDbFileName=" &

gstrDBName

'We list the following pieces:

' 1) The location of the server (you may need to get this from the connection

' string that you wrote down when we first created our Data Connection

' 2) What database we want to use on the server

' 3) We are using Security Support Provider Interface (SSPI) for our

' identity, e.g. the credentials we used to log onto Windows

' 4) For SQL Server Database files, we need to provide the actual path

' to the database file.

Private Sub cmdExecuteQuery\_Click(sender As Object, e As EventArgs)

Handles cmdExecuteQuery.Click

'Here's the connection to the database object

Dim DBConn As SqlConnection 'This would be OleDbConnection if using OleDb

'We'll build up our query in SQL as a string

Dim strSQLCmd As String

'We will place the SQL query in a command object

Dim DBCommand As New SqlCommand 'OleDbCommand under OleDb

'This is where our results will land

Dim myDataset As New DataSet

'We will use this to fill our Dataset with the DB's content

Dim DBAdapter As New SqlDataAdapter 'OleDbDataAdapter under OleDb

'Create a new connection based on the connection string and open it

DBConn = New SqlConnection(gstrConnString) 'OleDbCommand under OleDb

DBConn.Open()

'Get the current text from the textbox and that'll be our SQL query

strSQLCmd = txtQuery.Text

'Load the SQL query into the command object

DBCommand.CommandText = strSQLCmd

'Hook the data adapter up to run the SQL command on our connection

DBAdapter = New SqlDataAdapter(strSQLCmd, DBConn) 'OleDbDataAdapter

'Fill the dataset

DBAdapter.Fill(myDataset, "Addresses")

'Now hook the DataGridView up to the Addresses table in the Dataset

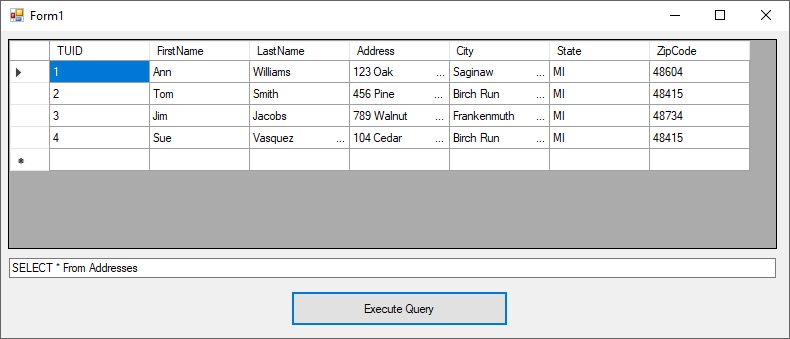
'to show what came back from the database

dgvResults.DataSource = myDataset.Tables("Addresses")

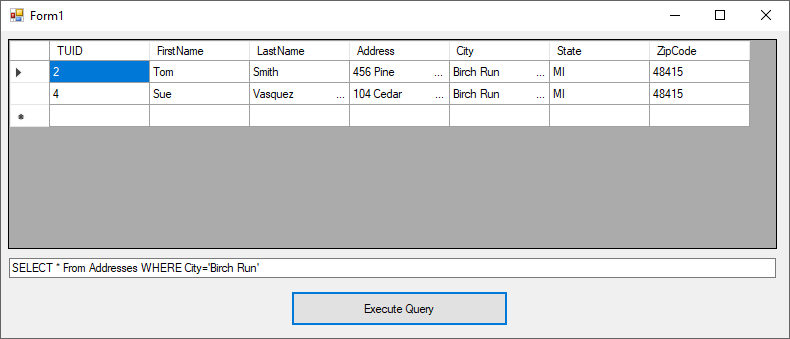
End Sub

End Class

Now let’s look at our application in action:



We can even type a live SQL query in the textbox and execute it. We just basically wrote a SQL front end allowing us to play around with our data:



Do you know what the best part of this application is? There was no magic in it! We were responsible for every piece – from defining how to get to the database, to building/executing our queries through displaying the records that came back. We were in control the entire time and so if any changes need to be made, we should be pretty confident that we can make them!

## Using the DataReader

The next example is going to show you how to work with the DataReader. Now even though we didn’t do anything other than display the contents in the database, because we used DataSets and what not, we had the ability to change things if we chose. That’s costly – what if all we really wanted to do was to iterate over each record for printing? Here’s where the data reader comes in.

'Chapter 15 - Program 4

Imports System.Data.SqlClient

Module Module1

Const gstrDBName As String = "C:\DB\Test.mdf" 'SQL Server database file

Const gstrConnString As String = "Server=(localdb)\MSSQLLocalDB;" &

"Database=Addresses;Integrated Security=SSPI;AttachDbFileName=" &

gstrDBName

'Databases do not have to be forms based!

Sub Main()

'These variables serve the same purpose as in the preceding example

Dim DBConn As SqlConnection

Dim strSQLCmd As String

Dim DBCommand As New SqlCommand

'Create a data reader object

Dim myDataReader As SqlDataReader 'OleDbReader under OleDb

'Create a new connection based on the connection string and open it

DBConn = New SqlConnection(gstrConnString)

DBConn.Open()

'Set the command up with its command text and connection

strSQLCmd = "Select \* From Addresses"

DBCommand.CommandText = strSQLCmd

DBCommand.Connection = DBConn

'Execute the reader

myDataReader = DBCommand.ExecuteReader

'While there's data, read it

While myDataReader.Read()

'and print it -- notice the use of numeric indexes or field names –

'either is okay

Debug.WriteLine("TUID: {0} Name: {1} {2} Address: {3} {4}," &

" {5} {6}", Trim(myDataReader(0)),

myDataReader("FirstName"),

myDataReader("LastName"), myDataReader(3),

Trim(myDataReader("City")),

Trim(myDataReader("State")),

Trim(myDataReader("ZipCode")))

End While

'Close the reader up

myDataReader.Close()

End Sub

End Module

Here’s the execution:

TUID: 1 Name: Ann Williams Address: 123 Oak Saginaw, MI 48604

TUID: 2 Name: Tom Smith Address: 456 Pine Birch Run, MI 48415

TUID: 3 Name: Jim Jacobs Address: 789 Walnut Frankenmuth, MI 48734

TUID: 4 Name: Sue Vasquez Address: 104 Cedar Birch Run, MI 48415

## Driving Databases Purely Through VB Code

I want you to realize that you can handle everything database-wise through code in .NET. This is the way that you ought to be developing your software solutions. We’ve already seen what the wizards produce and I’d be afraid to stake my career on a solution using them. Oh sure, they’re great for a database with a table or two, but what are you going to do when you need to interact with a commercial production database system that has hundreds of tables? What about when the system requirements change? Don’t think it’ll happen? Talk to somebody at Dynatrace, Ford, Dow, Nexteer or Auto-Owners and see. Commercial systems have complicated joins, stored procedures and triggers. You want the wizard generating your code for you? I didn’t think so.

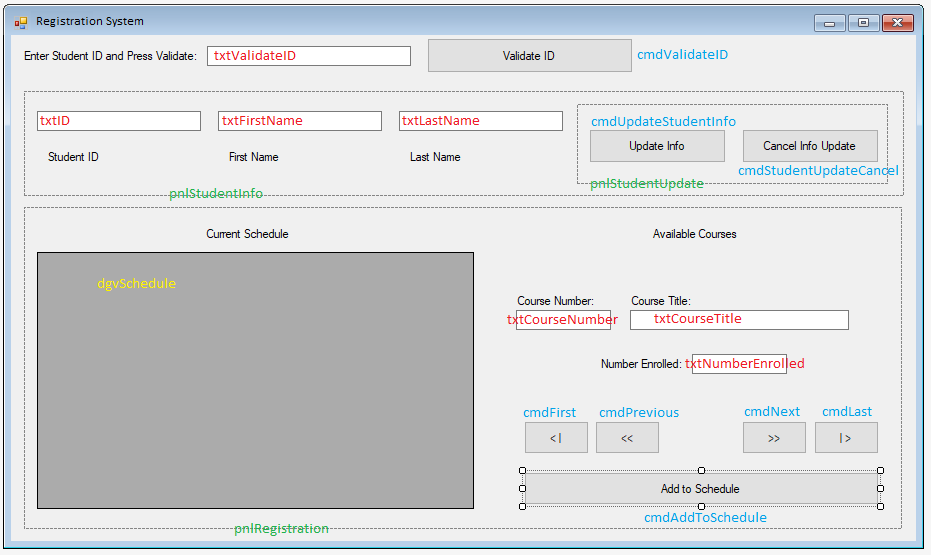
This example is the last formal one that we will examine at in this chapter with regard to building a robust database application. It rolls everything that we've talked about up into one application. The purpose of this application is to simulate a college registration system. Three separate tables will be used in this application: Students, Courses and Registration. All three tables reside in the same database.

The first table, Students, is responsible for holding student information, which in this application is the Student's ID Number (in the table this field is known as SID), First Name (FirstName) and Last Name (LastName). The second table, Courses is a listing of all courses that the college offers. There is a Course ID Number (CID), the Course Title (CourseTitle) and the number of students enrolled in the course (NumberEnrolled). The last table, Registration, simply contains the SID of students that are signed up for CID courses.

Let's spend a moment and discuss how the registration process should work. The first thing that should happen is a student should type in his or her Student ID number. If the ID number is found in the Students table, the application can proceed. If the number is not located, the student should be given an opportunity to enter his or her personal information into the Students database. Assuming the student types in his or her data, the application will proceed exactly in the same manner as if the student's ID number had immediately been located.

Once the Student ID number has been verified, the application system should then provide the student with the ability to add courses to his or her schedule. In addition, the application needs to display the student's current course schedule if the student has any previously registered courses. The application doesn't perform any error trapping such as if the student attempts to register for a course that he or she has already registered for. It would not be difficult to add this component, but the example is already complex enough. You can look at the walkthrough of the application which is located just after the source code to learn how it operates.

The next screenshot shows the main application form. Notice that there are three panels in place. There is a panel in the middle that contains the student's personal information and a panel nested inside of that panel which contains Update and Cancel buttons. The third panel is located on the lower portion of the application and holds the student's current schedule in a DataGridView control as well as the course listing information:



Red controls are TextBoxes, blue controls are Command Buttons, green controls are Panels and the yellow control is a DataGridView. The following table lists the names of the various controls on the form from top to bottom and left to right:

|  |  |
| --- | --- |
| Control Name | Description |
| txtStudentID | Textbox where the student enters his or her ID number |
| cmdValidateID | Command button used to indicate that the student has entered his or her ID and wants the application to attempt to validate it |
| pnlStudentInfo | Panel containing the student information as well as the second panel containing the Update and Cancel command buttons |
| txtID | Textbox containing the student's identification number |
| txtFirstName | Textbox containing the student's first name |
| txtLastName | Textbox containing the student's last name |
| pnlStudentUpdate | Panel containing the Update and Cancel command buttons |
| cmdUpdateStudentInfo | Command button used to add a new student's information to the Students table |
| cmdStudentUpdateCancel | Command button used to cancel adding a new student's information to the Students table |
| pnlRegistration | Panel containing the current student schedule as well as all the course related controls |
| dgvSchedule | DataGridView containing the student's current schedule |
| txtCourseNumber | Textbox containing the current Courses record's course number |
| txtCourseTitle | Textbox containing the current Courses record's course title |
| txtNumberEnrolled | Textbox containing the current number of students enrolled in the current Courses record's course |
| cmdFirst | The "<|" command button used to move to the first record in the Courses table |
| cmdPrevious | The "<" command button used to move to the previous record in the Courses table |
| cmdNext | The ">" command button used to move to the next record in the Courses table |
| cmdLast | The "|>" command button used to move to the last record in the Courses table |
| cmdAddToSchedule | The command button used to add the current course to the student's schedule |

The real work is all in the application's code, which consists of a code module and the code behind the application's form. If you spend time combing over this code example you will get an idea of what is involved in building a robust database application. Here's the code from the Module1.vb module:

Chapter 15 - Program 5

'Database Example - this is a forms application but

'it uses Sub Main, so make sure you uncheck Application

'Events and tell it the startup form is Sub Main...

'We will be working with SQL Server Database files

Imports System.Data.SqlClient

Module Module1

Sub Main()

Const strDBNAME As String = "Registration" 'Name of database

'Name of the database server

Const strSERVERNAME As String = "(localdb)\MSSQLLocalDB"

'Path to database in executable

Dim strDBPATH As String = My.Application.Info.DirectoryPath &

"\" & strDBNAME & ".mdf"

'This is the full connection string

Dim strCONNECTION As String = "SERVER=" & strSERVERNAME & ";DATABASE=" &

strDBNAME & ";Integrated Security=SSPI;AttachDbFileName=" &

strDBPATH

'If the database doesn't exist, create it

If Not (System.IO.File.Exists(strDBPATH)) Then

CreateDatabase(strSERVERNAME, strDBNAME, strDBPATH, strCONNECTION)

End If

'Make sure all tables are cleaned out each time we run this

CleanOutStudentsTable(strCONNECTION)

CleanOutCoursesTable(strCONNECTION)

CleanOutRegistrationTable(strCONNECTION)

'Put some data into the tables

PopulateStudentsTable(strCONNECTION) '1 student record

PopulateCoursesTable(strCONNECTION) '6 courses

PopulateRegistrationTable(strCONNECTION) 'signed up for 2 courses

'Show the main dialog

frmRegistration.ShowDialog()

End Sub

Sub CreateDatabase(ByVal strSERVERNAME As String, ByVal strDBNAME As String,

ByVal strDBPATH As String, ByVal strCONNECTION As String)

'Let's build a SQL Server database from scratch

Dim DBConn As SqlConnection

Dim strSQLCmd As String

Dim DBCmd As SqlCommand

'All we need to do initially is just point at the server

DBConn = New SqlConnection("Server=" & strSERVERNAME)

'Let's write a SQL DDL Command to build the database

'There are a lot of other parameters but we can let them default

'All we need are these three

strSQLCmd = "CREATE DATABASE " & strDBNAME & " On " &

"(NAME = '" & strDBNAME & "', " &

"FILENAME = '" & strDBPATH & "')"

DBCmd = New SqlCommand(strSQLCmd, DBConn)

Try

'Open the connection and try running the command

DBConn.Open()

DBCmd.ExecuteNonQuery()

MessageBox.Show("Database was successfully created", "",

MessageBoxButtons.OK, MessageBoxIcon.Information)

Catch ex As Exception

'If we can't build the database, we are dead in the water so bail...

MessageBox.Show(ex.ToString())

MessageBox.Show("Cannot build database! Closing program down...")

End

End Try

'We are currently pointing at the [MASTER] database, so we

'need to close the connection and reopen it pointing at the

'Registration database...

If (DBConn.State = ConnectionState.Open) Then

DBConn.Close()

End If

'Now we need to use the full connection string with the Integrated

'Security line, et cetera

DBConn = New SqlConnection(strCONNECTION)

DBConn.Open()

'Build the Tables one at a time

'Build the Student Table by writing the SQL DDL Command

DBCmd.CommandText = "CREATE TABLE Students (" &

"SID varchar(6), " &

"FirstName varchar(50), " &

"LastName varchar(50))"

DBCmd.Connection = DBConn

Try

DBCmd.ExecuteNonQuery()

MessageBox.Show("Created Students Table")

Catch Ex As Exception

MessageBox.Show("Clients Table Already Exists")

End Try

'Build the Courses Table

DBCmd.CommandText = "CREATE TABLE Courses (" &

"CID varchar(6), " &

"CourseTitle varchar(50), " &

"NumberEnrolled varchar(50))"

DBCmd.Connection = DBConn

Try

DBCmd.ExecuteNonQuery()

MessageBox.Show("Created Courses Table")

Catch Ex As Exception

MessageBox.Show("Courses Table Already Exists")

End Try

'Build the Registration Table

DBCmd.CommandText = "CREATE TABLE Registration (" &

"SID varchar(6), " &

"CID varchar(6))"

DBCmd.Connection = DBConn

Try

DBCmd.ExecuteNonQuery()

MessageBox.Show("Created Registration Table")

Catch Ex As Exception

MessageBox.Show("Registration Table Already Exists")

End Try

'We can check to see if we're open before trying to

'issue a connection close

If (DBConn.State = ConnectionState.Open) Then

DBConn.Close()

End If

End Sub

Sub CleanOutStudentsTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Use SQL DML to zap the contents of the table

DBCmd.CommandText = "DELETE FROM Students"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

MessageBox.Show("Deleted Everything In Students")

DBConn.Close()

End Sub

Sub CleanOutCoursesTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Use SQL DML to zap the contents of the table

DBCmd.CommandText = "DELETE FROM Courses"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

MessageBox.Show("Deleted Everything In Courses")

DBConn.Close()

End Sub

Sub CleanOutRegistrationTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Use SQL DML to zap the contents of the table

DBCmd.CommandText = "DELETE FROM Registration"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

MessageBox.Show("Deleted Everything In Registration")

DBConn.Close()

End Sub

Sub PopulateStudentsTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Add a student using SQL DML

DBCmd.CommandText = "INSERT INTO Students (SID, FirstName, LastName) " &

"VALUES ('123','Scott','James')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBConn.Close()

MessageBox.Show("Added a Student To Students Table")

End Sub

Sub PopulateCoursesTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Add courses using SQL DML

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CS 116','Computer Programming I','0')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CS 216','Computer Programming II','0')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CIS311','Visual Basic Programming','0')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CIS422','Systems Analysis & Design','1')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CIS424','Systems Design & Implementation','0')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Courses (CID, CourseTitle,

NumberEnrolled) " &

"VALUES ('CIS486','Enterprise Database Systems','1')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBConn.Close()

MessageBox.Show("Added Courses To Courses Table")

End Sub

Sub PopulateRegistrationTable(ByVal strConn As String)

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

'Now try to open up a connection to the database

DBConn = New SqlConnection(strConn)

DBConn.Open()

'Add student registration using SQL DML

DBCmd.CommandText = "INSERT INTO Registration (SID, CID) " &

"VALUES ('123','CIS422')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBCmd.CommandText = "INSERT INTO Registration (SID, CID) " &

"VALUES ('123','CIS486')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery()

DBConn.Close()

MessageBox.Show("Added Registrations to Registration Table")

End Sub

Sub DeleteDatabase(ByVal strSERVERNAME As String, ByVal strDBNAME As String)

'This routine shows how to delete a database completely

'from code. It does not consider deleting the data from

'the tables nor dropping the tables -- it just zaps the

'database completely

Dim DBConn As SqlConnection

Dim strSQLCmd As String

Dim DBCommand As SqlCommand

'We need to point back at the [Master] database itself

DBConn = New SqlConnection("Server=" & strSERVERNAME)

'Try to force single ownership of the database so that we have the

'permissions to delete it

strSQLCmd = "ALTER DATABASE [" & strDBNAME & "] SET " &

"SINGLE\_USER WITH ROLLBACK IMMEDIATE"

DBCommand = New SqlCommand(strSQLCmd, DBConn)

Try

DBConn.Open()

DBCommand.ExecuteNonQuery()

MessageBox.Show("Database set for exclusive use", "",

MessageBoxButtons.OK, MessageBoxIcon.Information)

Catch ex As Exception

MessageBox.Show(ex.ToString())

End Try

If (DBConn.State = ConnectionState.Open) Then

DBConn.Close()

End If

'Now drop the database

strSQLCmd = "DROP DATABASE " & strDBNAME

DBCommand = New SqlCommand(strSQLCmd, DBConn)

Try

DBConn.Open()

DBCommand.ExecuteNonQuery()

MessageBox.Show("Database has been deleted", "", MessageBoxButtons.OK,

MessageBoxIcon.Information)

Catch ex As Exception

MessageBox.Show(ex.ToString())

End Try

If (DBConn.State = ConnectionState.Open) Then

DBConn.Close()

End If

End Sub

End Module

Now we need to look at the code behind the frmRegistration form:

'Chapter 15 - Program 5

'We need to bring in the SqlClient namespace

Imports System.Data.SqlClient

Imports System.ComponentModel

Public Class frmRegistration

'Create a dataset to point to each table -- do it here so that we don't

'have to keep passing things around

Dim dsStudents As New DataSet

Dim dsCourses As New DataSet

Dim dsRegistration As New DataSet

'Here's the connection string related pieces - the same as in the module

'A smarter way would be to declare them once, but I wanted the code to

'be as simple as possible

Const strDBNAME As String = "Registration" 'Name of database

'Name of the database server

Const strSERVERNAME As String = "(localdb)\MSSQLLocalDB"

'Path to database in executable

Dim strDBPATH As String = My.Application.Info.DirectoryPath &

"\" & strDBNAME & ".mdf"

'This is the full connection string

Dim strCONNECTION As String = "SERVER=" & strSERVERNAME & ";DATABASE=" &

strDBNAME & ";Integrated Security=SSPI;AttachDbFileName=" &

strDBPATH

'We'll also create a SqlConnection object since we will execute some

'straight SQL rather than relying on the DBAdapters...

Dim myConn As New SqlConnection(strCONNECTION)

'Likewise create three data adapters so we don't mess stuff up

'trying to be cute with one adapter

Dim DBAdaptStudents As SqlDataAdapter

Dim DBAdaptCourses As SqlDataAdapter

Dim DBAdaptRegistration As SqlDataAdapter

Private Sub frmRegistration\_Load(sender As Object, e As EventArgs)

Handles MyBase.Load

Dim strSQLCmd As String

'All panels should start out hidden

pnlRegistration.Visible = False

pnlStudentInfo.Visible = False

pnlStudentUpdate.Visible = False

'The user will never be able to type anything into the txtID textbox

'We will pull the value from the txtStudentID textbox

txtID.Enabled = False

'Load up all courses since they will never change while the program runs

strSQLCmd = "Select \* From Courses"

DBAdaptCourses = New SqlDataAdapter(strSQLCmd, strCONNECTION)

DBAdaptCourses.Fill(dsCourses, "Courses")

'Set bindings on the course related fields since we now have some data in

'context and can understand the schema of the table

txtCourseNumber.DataBindings.Add(New Binding("Text", dsCourses,

"Courses.CID"))

txtCourseTitle.DataBindings.Add(New Binding("Text", dsCourses,

"Courses.CourseTitle"))

txtNumberEnrolled.DataBindings.Add(New Binding("Text", dsCourses,

"Courses.NumberEnrolled"))

End Sub

Private Sub cmdValidateID\_Click(sender As Object, e As EventArgs)

Handles cmdValidateID.Click

'This is called when we put in a student number

Dim strSQLCmd As String

Dim cmdBuilder As SqlCommandBuilder 'This is cool -- look below...

'Make sure other panels aren't showing yet

pnlRegistration.Visible = False

pnlStudentUpdate.Visible = False

pnlStudentInfo.Visible = False

'Set up the data adapter for Students...

strSQLCmd = "Select \* From Students Where SID = '" &

Trim(txtStudentID.Text) & "'"

DBAdaptStudents = New SqlDataAdapter(strSQLCmd, strCONNECTION)

'We must include how an "AddNew" to the Bound Controls will handle an

'Insert. Forunately, VB provides a cmdBuilder object that will create

'the right statements...you can also get the DeleteCommand and

'UpdateCommands since we don't allow that functionality

cmdBuilder = New SqlCommandBuilder(DBAdaptStudents)

DBAdaptStudents.InsertCommand = cmdBuilder.GetInsertCommand

'Fill the dataset

dsStudents.Clear()

DBAdaptStudents.Fill(dsStudents, "Students")

'See if we've already set bindings on the student related controls...

If txtID.DataBindings.Count = 0 Then

'We need to set the bindings for the student information

txtID.DataBindings.Add(New Binding("Text", dsStudents,

"Students.SID"))

txtFirstName.DataBindings.Add(New Binding("Text", dsStudents,

"Students.FirstName"))

txtLastName.DataBindings.Add(New Binding("Text", dsStudents,

"Students.LastName"))

End If

'Attempt to get this student's Schedule

strSQLCmd = "Select \* From Registration Where SID = '" &

Trim(txtStudentID.Text) & "'"

DBAdaptRegistration = New SqlDataAdapter(strSQLCmd, strCONNECTION)

dsRegistration.Clear()

DBAdaptRegistration.Fill(dsRegistration, "Registration")

'We couldn't bind the datagridview until we had a dataset, so do it now

dgvSchedule.DataSource = dsRegistration.Tables("Registration")

'Show student information panel

pnlStudentInfo.Visible = True

'Check to see if student exists

If dsStudents.Tables("Students").Rows.Count <= 0 Then

'Student doesn't exist...

MessageBox.Show("Enter info into the boxes and press Update to " &

"add a new student or Cancel to go back to Validation")

'Clear out the current edits

BindingContext(dsStudents, "Students").EndCurrentEdit()

'Add a new record to the recordset

BindingContext(dsStudents, "Students").AddNew()

'Show panel with push buttons and enable textboxes that

'we want user to enter information into

pnlStudentUpdate.Visible = True

txtFirstName.Enabled = True

txtLastName.Enabled = True

txtID.Text = txtStudentID.Text

Else

'Student already exists so show registration panel

pnlRegistration.Visible = True

End If

End Sub

Private Sub cmdUpdateStudentInfo\_Click(sender As Object, e As EventArgs)

Handles cmdUpdateStudentInfo.Click

'If we got here, then a new student number was entered and the user

'clicked on the Update button to indicate that they wanted the

'student information they entered to be placed into the database.

'Notice that this routine is using the AddNew methods of the

'dataset and not any direct SQL manipulation...this is one way

'that we can modify what's in the database

'Stop any current edits.

BindingContext(dsStudents, "Students").EndCurrentEdit()

'Update the database

myConn.Open()

DBAdaptStudents.Update(dsStudents, "Students")

myConn.Close()

'Update the dataset to correspond with database.

dsStudents.AcceptChanges()

'Hide the panel and reset the textboxes

txtFirstName.Enabled = False

txtLastName.Enabled = False

pnlStudentUpdate.Visible = False

'Show the registration panel since the student is now valid

pnlRegistration.Visible = True

End Sub

Private Sub cmdStudentUpdateCancel\_Click(sender As Object, e As EventArgs)

Handles cmdStudentUpdateCancel.Click

'If we got here, the user typed in a new student number and then

'decided that they did not want it entered and hit the Cancel button

'Throw away the current edit

BindingContext(dsStudents, "Students").CancelCurrentEdit()

'Hide the panel and reset the textboxes

txtFirstName.Enabled = False

txtLastName.Enabled = False

pnlStudentInfo.Visible = False

pnlStudentUpdate.Visible = False

End Sub

Private Sub cmdFirst\_Click(sender As Object, e As EventArgs)

Handles cmdFirst.Click

'Called when we click the move to first button <| in Course listing

BindingContext(dsCourses, "Courses").Position = 0

End Sub

Private Sub cmdPrevious\_Click(sender As Object, e As EventArgs)

Handles cmdPrevious.Click

'Called when we click the move to previous button << in Course listing

BindingContext(dsCourses, "Courses").Position = (BindingContext(dsCourses,

"Courses").Position - 1)

End Sub

Private Sub cmdNext\_Click(sender As Object, e As EventArgs)

Handles cmdNext.Click

'Called when we click the move to next button >> in Course listing

BindingContext(dsCourses, "Courses").Position = (BindingContext(dsCourses,

"Courses").Position + 1)

End Sub

Private Sub cmdLast\_Click(sender As Object, e As EventArgs)

Handles cmdLast.Click

'Called when we click the move to last button |> in Course listing

BindingContext(dsCourses, "Courses").Position =

(dsCourses.Tables("Courses").Rows.Count - 1)

End Sub

Private Sub cmdAddToSchedule\_Click(sender As Object, e As EventArgs)

Handles cmdAddToSchedule.Click

Dim DBConn As SqlConnection

Dim DBCmd As SqlCommand = New SqlCommand()

Dim strSQLCmd As String

Dim dsTemp As New DataSet()

Dim intCurrentRecord As Integer

'This technique is going to illustrate the second way that we can

'update a database...that is, through direct SQL manipulation.

'Now try to open up a connection to the database

DBConn = New SqlConnection(strCONNECTION)

DBConn.Open()

'Use SQL to insert a new row into Registration

DBCmd.CommandText = "INSERT INTO Registration (SID, CID) VALUES ('" &

txtID.Text & "','" & txtCourseNumber.Text & "')"

DBCmd.Connection = DBConn

DBCmd.ExecuteNonQuery() 'Since it's a non-SELECT statement

'Now use another SQL statement to update the number of students that

'are enrolled in the class...

DBCmd.CommandText = "UPDATE Courses Set NumberEnrolled = '" &

Trim(CStr((CInt(txtNumberEnrolled.Text) + 1))) &

"' WHERE CID = '" & txtCourseNumber.Text & "'"

DBCmd.ExecuteNonQuery()

DBConn.Close()

'Refresh the Course database to reflect the new student counts

'and then move back to current record

'Store current record number that we are on

intCurrentRecord = BindingContext(dsCourses, "Courses").Position

'Refresh dataset

strSQLCmd = "Select \* From Courses"

DBAdaptCourses = New SqlDataAdapter(strSQLCmd, strCONNECTION)

dsCourses.Clear()

DBAdaptCourses.Fill(dsCourses, "Courses")

'Move back to same record we were on before refresh

BindingContext(dsCourses, "Courses").Position = intCurrentRecord

'Now we need to refresh this student's Schedule as well, so...

strSQLCmd = "Select \* From Registration Where SID = '" &

Trim(txtStudentID.Text) & "'"

DBAdaptRegistration = New SqlDataAdapter(strSQLCmd, strCONNECTION)

dsRegistration.Clear()

DBAdaptRegistration.Fill(dsRegistration, "Registration")

'Refresh the DataGridView showing the schedule so that it's accurate

dgvSchedule.Refresh()

End Sub

Private Sub frmRegistration\_Closing(sender As Object, e As CancelEventArgs)

Handles Me.Closing

If MessageBox.Show("Do you want to physically delete the database?", "",

MessageBoxButtons.YesNo) = DialogResult.Yes Then

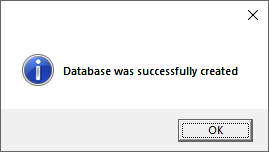
DeleteDatabase(strSERVERNAME, strDBNAME)

End If

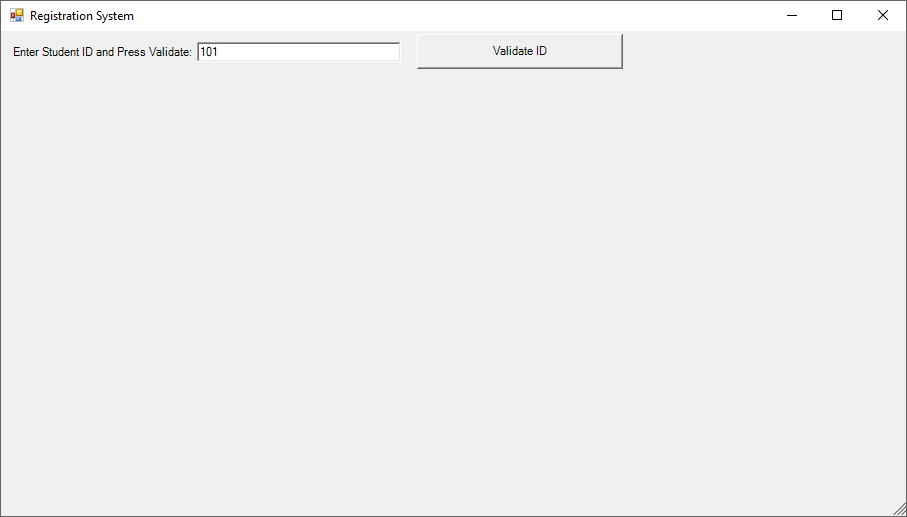
End Sub

End Class

Now that we've seen the source code, let's watch the application when it's running. I skipped over the several MessageBoxes that simply inform the user that the database was created, the tables were created and some data was placed in the tables. As an example, here’s the very first notification when the database is created:

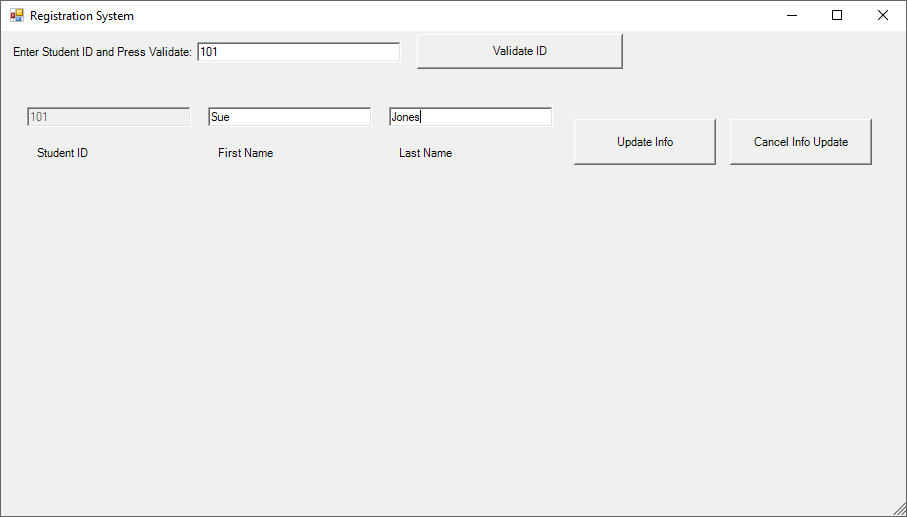


The next screenshot shows the main application form as it first appears to the user. The only thing that a user can do at this point is enter his or her student ID number and press the Validate button to attempt to get access to the system.

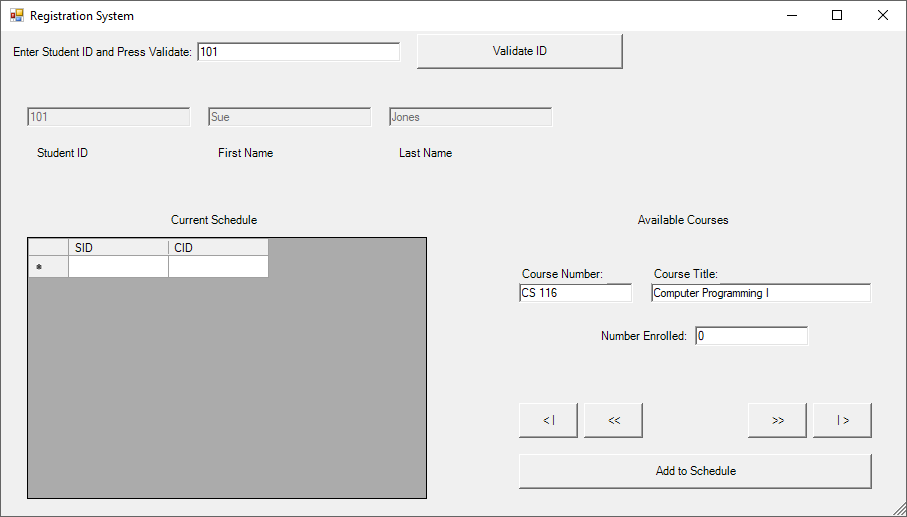


If the student was successfully validated by the system, i.e. the student's ID number was located in the Students table, then the student will then be shown his or her current schedule along with the ability to move through the courses and add new ones to his/her schedule. If the student couldn't be validated, then he or she is given the opportunity to add his or her personal information and press the Update button which will write the data out to the Students table.

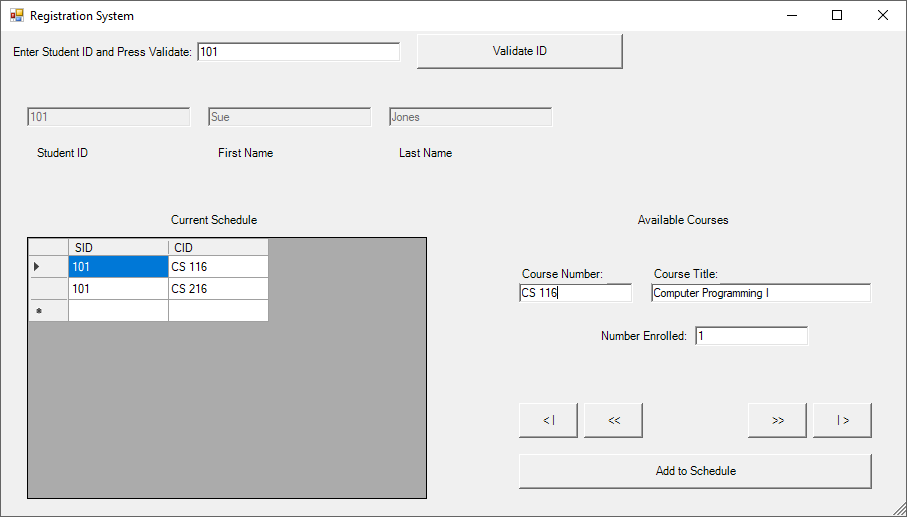
The following screenshot shows a student entering in her personal information. The student can press the Cancel button at any time to abort the data entry and return back to the initial ID and Validate piece. Notice the appearance of the Update/Cancel panel:



Once the student has been validated (or entered and updated new information), the system will then show the student's current course schedule in the lower left hand DataGridView. The lower right hand group of controls will allow the student to move through and select courses that he or she wants to add to his or her schedule. The following screenshot shows an example of this:



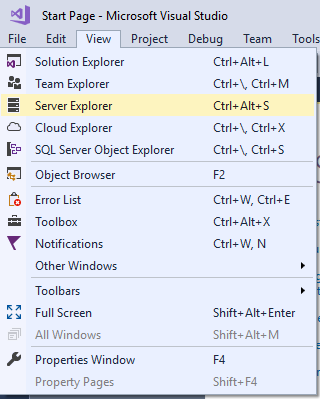
The student can add classes to his or her schedule. If you watch the Number Enrolled textbox, you will see that it automatically is incremented by 1 each time the student adds that particular course. The application as it currently exists does not provide any deletion options to remove courses nor does it contain any course conflict logic.



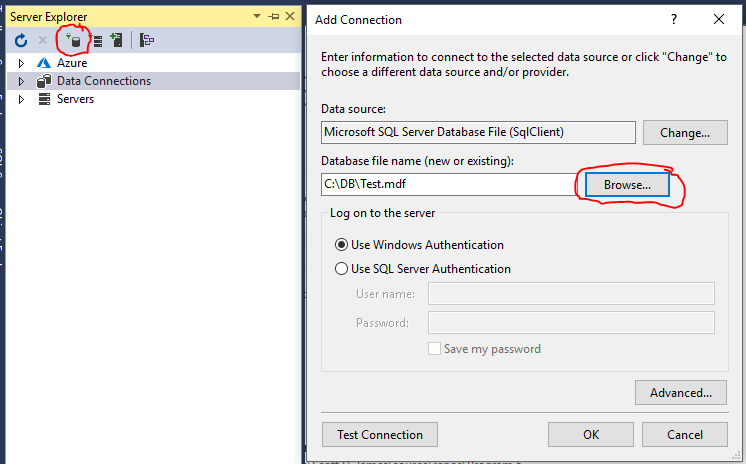
When the application ends, the user is prompted if he/she wants to delete the database. While we wouldn’t show that in a production environment, it is here simply for instructional purposes on how to remove a database through code. As you can see, working with databases and building up elaborate forms isn’t really all that tough – and the good news is that by handling everything ourselves, we are always in control!

## Reattaching a SQL Server Database File

To attach a SQL Server database file (.mdf), start off in the Server Explorer window. Note that this is not the SQL Server Object Explorer window! The Server Explorer allows us to see our various servers, data connections and Azure cloud connections and manage them. Access the Server Explorer through View🡪Server Explorer:

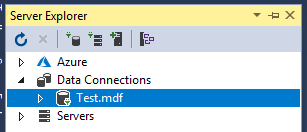


Click on the Data Connections line in the Server Explorer pane and then click the plus icon at the top of the Server Explorer. The Add Connection dialog box appears. We need to make sure that the Data Source type is set to Microsoft SQL Server Database File (SqlClient) and then press the browse button to locate the .mdf file that you want to reattach:

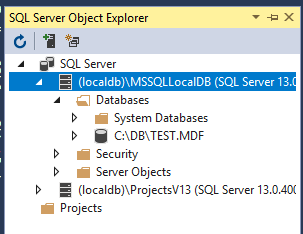


Once you’ve selected the database file name, you should press the Test Connection button to ensure that the file can be accessed. If the test runs fine, then go ahead and press the OK button to reattach the database file.

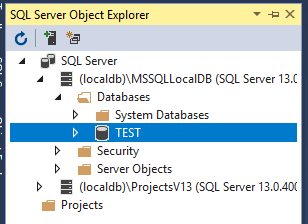
After a few seconds, you should see the file appear in the Server Explorer:



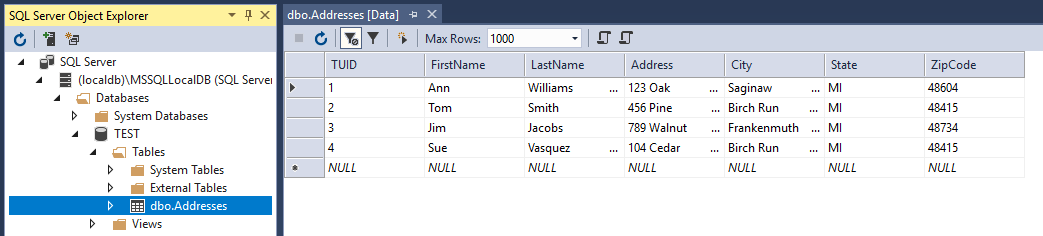
You can now access the database again through the SQL Server Object Explorer too:



You may want to right click on the database file and rename it so that it doesn’t have all the path and file extension stuff on it just to make it a bit cleaner:



That’s it – the database will now remain connected to SQL Server until you explicitly detach or delete it. Any programs that originally referred to it should have no problems accessing it as long as: (1) they were coded against using the default (localdb)\MSSQLLocalDB server and (2) you renamed the database back to its original name as used by the program.



# Review of the Database Related Classes

As I previously mentioned, we didn't hardly touch any of the classes, methods or properties that ADO.NET has for you to use. It is quite impressive that we can write applications as complicated as those that we looked at in this chapter with the few pieces that were used in the example programs.

This section is simply going to serve as a summary of the various classes that you encountered through the database examples. We will note the methods or properties that were used in the code examples. This is by no means an exhaustive list! You should plan on spending time reading up on ADO.NET to familiarize yourself with all of the facilities that it can provide for you.

### BindingContext

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| AddNew | Adds a new record to the DataSet that the item is bound to |
| CancelCurrentEdit | Tells the bound DataSet to cancel any changes made to any records currently marked with changes |
| EndCurrentEdit | Tells the bound DataSet to end any records that may currently be being edited |
| Position | Returns the current record location within a DataSet |
| RemoveAt | Deletes a record from the DataSet at a given position |
| ResumeBinding | Resumes a data binding |
| SuspendBinding | Suspends a data binding |
| UpdateIsBinding | Updates a data binding |

### Controls

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| DataBindings.Add | Adds a new data binding to a control from a field in a DataSet |
| SetDataBinding | Specifies the field to bind to in a DataSet |

### DataSet

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| AcceptChanges | Accepts changes to the DataSet |
| CaseSensitive | Gets or sets a value indicating whether string comparisons within a DataTable are case sensitive |
| Clear | Empties the DataSet |
| CreateDataReader | Returns a DataReader |
| DataSetName | Gets or sets the name of the current DataSet |
| DefaultViewManager | Gets a custom view of the data contained in the DataSet for filtering, searching and navigating using a custom DataViewManager |
| EnforceConstraints | Gets or sets a value indicating whether constraint rules are followed when attempting an update operation |
| GetChanges | Gets a copy of the DataSet containing all changes made to it since it was loaded or since AcceptChanges was called |
| HasChanges | Gets a value indicating whether the DataSet has new, deleted or modified rows |
| HasErrors | Indicates whether there are errors in any of the DataTable objects in the DataSet |
| Load | Fills a DataSet with values from a data source using a DataReader |
| Merge | Merges an entity with the current DataSet |
| Relations | The collection of relations that link tables and allow navigation from parent to child |
| RejectChanges | Rolls back all the changes made to the DataSet since it was created or since the last time AcceptChanges was called |
| Reset | Clears all tables, relations, and so forth from the DataSet |
| Tables | The collection of tables contained in the DataSet |

### OleDb/SqlCommand

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| CommandText | A string containing the text of a SQL command |
| CommandTimeout | How long the command waits before terminating an attempt to execute a command and generating an error |
| Connection | The connection over which the OleDb/SqlCommand object is to be executed |
| ExecuteReader | Executes commands that return rows which cannot be changed |
| ExecuteNonQuery | Used to execute any OleDb/SqlCommand containing any SQL statement other than a SELECT statement, in other words, INSERT, DELETE and UPDATE statements |
| ExecuteScalar | Retrieves a single value, like an aggregate (MIN, MAX, COUNT) from a database |
| Parameters | Gets the OleDb/SqlParameter collection |
| Transaction | Gets or sets the OleDb/SqlTransaction within which the OleDb/SqlCommand executes |
| UpdatedRowSource | Gets or sets how command results are applied to the DataRow when used by the Update method |

### OleDb/SqlConnection

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| ChangeDatabase | Changes the current database for an open OleDb/SqlConnection |
| Close | Closes a connection to a database |
| ConnectionString | Gets the string used to open a database |
| ConnectionTimeout | Specifies how long to wait trying to establish a connection before terminating and generating and error |
| CreateCommand | Creates and returns an OleDb/SqlCommand object associated with the OleDb/SqlConnection |
| Database | Gets the name of the current database or the database to be used after a connection is opened |
| DataSource | The server name or file name of the data source |
| Open | Opens a connection to a database |
| Provider | Gets the name of the OLE DB provider specified in the connection string (this is only used on OleDbConnection) |
| ResetState | Updates the state property of the OleDb/SqlConnection |
| ServerVersion | Returns the server version that the client is connection to |
| State | Gets the current state of the connection |

### OleDb/SqlDataAdapter

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| AcceptChangesDuringFill | Indicates whether AcceptChanges is called on a DataRow after it is added to the DataTable during a Fill |
| AccceptChangesDuringUpdate | Indicates whether AcceptChanges is called during an Update |
| ContinueUpdateOnError | Specifies whether to generate an exception when an error is encountered during a row update |
| DeleteCommand.CommandText | DeleteCommand is an OleDb/SqlCommand object. In particular, this object holds the text string designating how to SQL DML DELETE information from a database |
| Fill | Loads data into a DataSet |
| FillLoadOption | Determines how the adapter fills the DataTable from a DataReader |
| FillSchema | Adds a DataTable to a DataSet and configures the schema to match that in the data source |
| GetFillParameters | Gets the parameters set by the user when executing a SQL SELECT DML statement |
| InsertCommand.CommandText | InsertCommand is an OleDb/SqlCommand object. In particular, this object holds the text string designating how to SQL DML INSERT information into a database |
| MissingMappingAction | Determines the action to take when incoming data does not have a matching table or column |
| MissingSchemaAction | Determines the action to take when an existing DataSet schema does not match incoming data |
| ResetFillLoadOption | Resets FillLoadOption back to its default state and causes Fill to again AcceptChangesDuringFill |
| SelectCommand.CommandText | SelectCommand is an OleDb/SqlCommand object. In particular, this object holds the text string designating how to SQL DML SELECT information from a database |
| TableMappings | The collection that provides the master mapping between a source table and a DataTable |
| Update | Sends information containing in a DataSet back to its underlying database |
| UpdateBatchSize | The value that enables or disables batch processing support and specifies the number of commands that can be executed in a batch |
| UpdateCommand.CommandText | UpdateCommand is an OleDb/SqlCommand object. In particular, this object holds the text string designating how to SQL DML UPDATE information in a database |

### DataReader

|  |  |
| --- | --- |
| *Method or Property* | *Purpose* |
| Close | Closes the OleDb/SqlDataReader |
| FieldCount | The number of columns in the current row |
| GetXXX | Gets the value of the specified column as type XXX (for example GetBoolean, GetBytes …) |
| GetDataTypeName | Gets the name of the source data type |
| GetFieldType | Gets the type that is in the data type of the object |
| GetName | Gets the name of the specified column |
| GetOrdinal | Gets the column ordinal given a column name |
| GetSchemaTable | Returns a DataTable that describes the column metadata of the OleDb/SqlDataReader |
| HasRows | Indicates whether the OleDb/SqlDataReader contains one or more rows |
| IsClosed | Indicates whether the data reader is closed |
| Item | Gets the value of a column in its native format |
| RecordsAffected | Gets the number of rows changes, inserted or deleted by the execution of the SQL statement |
| VisibleFieldCount | Gets the number of fields in the OleDb/SqlDataReader that are not hidden |

# Using SQL

This section of the chapter is going to provide with a very brief introduction to using the Structured Query Language (SQL). I strongly recommend that you take some time and learn this language in depth since it will open all sorts of database avenues for you. You’ll be amazed at how a little bit of SQL code with some functions and advanced keywords can replace hundreds of lines of procedural logic!

We will work through the following examples using an Addresses table that has the following fields: FirstName, LastName, Address, City, State and ZipCode. Five records have been added to the table, which are displayed below for your information:

| **FirstName** | **LastName** | **Address** | **City** | **State** | **ZipCode** |
| --- | --- | --- | --- | --- | --- |
| Bill | Jones | 1523 Maple | Birch Run | MI | 48415 |
| Sue | Franklin | 100 South Huron | Burt | MI | 48417 |
| Tim | McMann | 4876 Beyer Road | Birch Run | MI | 48415 |
| Dawn | Davis | P.O. Box 174 | University Center | MI | 48734 |
| Ron | Williams | 15463 Elms Road | Montrose | MI | 48457 |

The main SQL statements that you need to know for performing queries that return information include SELECT, WHERE, FROM, GROUP BY and ORDER BY. SQL is not case-sensitive. I tend to put all SQL statements in uppercase simply to differentiate between what is a keyword and what isn't.

The purpose of the SELECT statement is to return rows of information from a database table. Let's look at the purpose of each of the keywords related to the SELECT DML.

### Select Statement

SQL SELECT related keywords:

|  |  |
| --- | --- |
| *Keyword* | *Purpose* |
| SELECT | Retrieves specified fields from a table |
| WHERE | Provides criteria for selecting records |
| FROM | Used to name the table from where to retrieve records |
| GROUP BY | Specifies how to group records |
| ORDER BY | Specifies how to sort records |
| DISTINCT | Limits values returned to be unique |

Here are a series of queries which return information from the Addresses table. Following each query is a summary of what the query does and a copy of the information that would be returned from the Addresses table.

SELECT \* FROM Addresses

This query would bring back every field and row from the Addresses table. The purpose of the \* is to tell SQL to bring back all fields contained in the named table. Since there is no WHERE statement present in this query, there is no filter criteria applied to the database rows. Therefore, every row would be returned. Here’s the results of the query:

| **FirstName** | **LastName** | **Address** | **City** | **State** | **ZipCode** |
| --- | --- | --- | --- | --- | --- |
| Bill | Jones | 1523 Maple | Birch Run | MI | 48415 |
| Sue | Franklin | 100 South Huron | Burt | MI | 48417 |
| Tim | McMann | 4876 Beyer Road | Birch Run | MI | 48415 |
| Dawn | Davis | P.O. Box 174 | University Center | MI | 48734 |
| Ron | Williams | 15463 Elms Road | Montrose | MI | 48457 |

SELECT LastName, City, State FROM Addresses

This query would bring back all rows since there is again no WHERE statement present. Notice that the \* was not provided after the SELECT statement; instead three fields were specifically named. This query, then, will only retrieve the LastName, City and State fields for each record from the table. The results look like this:

| **LastName** | **City** | **State** |
| --- | --- | --- |
| Jones | Birch Run | MI |
| Franklin | Burt | MI |
| McMann | Birch Run | MI |
| Davis | University Center | MI |
| Williams | Montrose | MI |

SELECT \* FROM Addresses WHERE City = "Birch Run"

This \* specifies that all fields will be returned. This query, however, has a filter clause. Only the rows whose City field contains the value Birch Run will be returned. The results:

| **FirstName** | **LastName** | **Address** | **City** | **State** | **ZipCode** |
| --- | --- | --- | --- | --- | --- |
| Bill | Jones | 1523 Maple | Birch Run | MI | 48415 |
| Tim | McMann | 4876 Beyer Road | Birch Run | MI | 48415 |

SELECT FirstName FROM Addresses WHERE FirstName > "Jim"

This query will print the FirstName for all records in the Addresses table where the FirstName is greater than “Jim.” Greater, in this case, means alphabetically past “J”. Three results are returned:

| **FirstName** |
| --- |
| Sue |
| Tim |
| Ron |

SELECT City FROM Addresses WHERE City LIKE "B\*"

This will select only the City field from the table for rows whose City starts with the letter B. Notice that both instances of "Birch Run," as well as the single instance of "Burt" are returned:

| **City** |
| --- |
| Birch Run |
| Burt |
| Birch Run |

SELECT DISTINCT City FROM Addresses WHERE City LIKE "B\*"

This query is identical in nature to the one above, with the exception that the DISTINCT keyword was added after the SELECT statement. Only one instance of each City will be returned now. Notice that "Birch Run" only appears once in this result set:

| **City** |
| --- |
| Birch Run |
| Burt |

SELECT ZipCode FROM Addresses WHERE ZipCode LIKE "4841?"

This will select all ZipCodes from the table which start with 4841 and have only one more character after the 1. We don’t care what that character is. Here’s the result set:

| **ZipCode** |
| --- |
| 48415 |
| 48417 |
| 48415 |

SELECT \* FROM Addresses ORDER BY LastName ASC

This will return all records but they will be sorted in ascending order (ASC) by the person's LastName. This is what is returned:

| **FirstName** | **LastName** | **Address** | **City** | **State** | **ZipCode** |
| --- | --- | --- | --- | --- | --- |
| Dawn | Davis | P.O. Box 174 | University Center | MI | 48734 |
| Sue | Franklin | 100 South Huron | Burt | MI | 48417 |
| Bill | Jones | 1523 Maple | Birch Run | MI | 48415 |
| Tim | McMann | 4876 Beyer Road | Birch Run | MI | 48415 |
| Ron | Williams | 15463 Elms Road | Montrose | MI | 48457 |

SELECT \* FROM Addresses ORDER BY City, Lastname ASC

Again, all records are returned but they will be sorted by City and then by LastName within City in ascending order. Here’s the results – pay attention to the last names ordering within “Birch Run”:

| **FirstName** | **LastName** | **Address** | **City** | **State** | **ZipCode** |
| --- | --- | --- | --- | --- | --- |
| Bill | Jones | 1523 Maple | Birch Run | MI | 48415 |
| Tim | McMann | 4876 Beyer Road | Birch Run | MI | 48415 |
| Sue | Franklin | 100 South Huron | Burt | MI | 48417 |
| Ron | Williams | 15463 Elms Road | Montrose | MI | 48457 |
| Dawn | Davis | P.O. Box 174 | University Center | MI | 48734 |

Here are some other examples of the SELECT statement using additional keywords that we didn't cover in detail. I also didn't print the details of the results since you can probably figure them out. If you’re stumped, create the Addresses database, put the sample records in and try things out!

SELECT \* FROM Addresses WHERE City BETWEEN "A\*" and "C\*"

This would return all fields from the Addresses table where the City name starts with an A, B or C.

SELECT \* FROM Addresses WHERE City IN ("Birch Run", "Burt", "Escanaba")

This would return all fields from the Addresses table where the City name is in the set of Birch Run, Burt or Escanaba. That means that if a record has a City with any of those three values, it will be returned.

SELECT COUNT(EmployeeID) AS NumberEmployees, AVG(Age) AS AverageAge, MIN(Salary) AS SmallestSalary, MAX(Salary) AS LargestSalary FROM Employees

This query will count the number of Employee IDs (which would tell us the number of employees that the company has) and print the results under the name NumberEmployees. The query will also use the SQL aggregate AVG function to calculate the average age of all employees, which is printed under the name AverageAge. Likewise, the aggregate MIN and MAX functions are used to calculate the SmallestSalary and LargestSalary that any employee earns. Note that EmployeeID, Age and Salary don’t exist in the database, but it wouldn’t be too hard to add the columns and populate them with some data! Hint, hint: try things out if you haven’t worked with SQL before…

While the SELECT statement is probably the most common that you will use in your database processing, there are three other SQL commands that you should be aware of. Again, we will look at some simple examples of using each command. You should be able to figure out what each does from the previous discussion on the SELECT statement.

### The Insert Statement

The purpose of the SQL INSERT statement is to place new records into a table. The syntax for the Insert statement has the following form:

INSERT INTO tableName (fieldName1, fieldName2, … fieldNameN) VALUES (value1, value2, … valueN)

INSERT INTO Employees (EmployeeID, Age, Salary) VALUES (105, 22, 15.37)

This statement inserts a new record into the Employees table. The EmployeeID field receives the value 105, the Age field receives the value 22 and the Salary field receives the value 15.37.

INSERT INTO BREmployees (FirstName, LastName) SELECT FirstName, LastName From Employees WHERE City = "Birch Run"

This statement inserts anywhere between zero and an infinite number of records. Assume that we have a new table named BREmployees and we want to populate it with a copy of the employees from our Employees table who live in the City of “Birch Run.” Notice that where the VALUES clause would normally be in the INSERT statement is a SELECT statement which returns the records matching our filtering criteria on the City field. Regardless of however many records come back, we pull the FirstName and LastName from each of those matching records. What do we do with them? Turn around and insert them into the BREmployees table. Pretty sweet, huh?

### The Update Statement

The purpose of the SQL Update statement is to change values for records that already exist in a database. The syntax for the Update statement has the following form:

UPDATE tableName SET fieldName1 = value1, fieldName2 = value2, fieldNameN = valueN WHERE criteria

UPDATE Addresses SET City = "Marquette" WHERE ZipCode = 49855

This statement would set the City name to Marquette for all records in the Addresses table where the ZipCode is 49855.

### The Delete Statement

The DELETE statement is used to remove records from a table. The syntax for the DELETE statement is:

DELETE FROM tableName WHERE criteria

Note that the WHERE criteria is optional

DELETE FROM Addresses

This would DELETE all records from the Addresses table.

DELETE FROM Addresses WHERE City NOT IN ("Birch Run", "University Center")

This statement would DELETE all records in the Addresses table that have cities other than “Birch Run” or “University Center”.

This concluded our quick introduction to the SQL language. It is by no means even close to scratching the surface of the language. There are many great books on both databases and SQL available. If you haven't worked much with a database, it is really in your best interest to learn SQL since just about every database that's used today allows you to manipulate its data via the SQL language. Learn it once and use it everywhere…

## Notes on SQL Server LocalDB

Since I don’t really like having to rediscover what I’ve done to get something working, I wanted to jot down a few notes on how LocalDB works behind the scenes. First, there can be one installed instance of LocalDB for each released version of SQL Server. For example, LocalDB version 13 relates to SQL Server 2016, version 12 to SQL Server 2014 and so on.

Typically, there is a SQLLocalDB.MSI installer program that can be downloaded from Microsoft to install the LocalDB functionality onto a system that lacks it. You used to be able to directly download these files from Microsoft, but that has kind of shifted over the past few SQL Server versions. You now may need to download SQL Server Express (for whatever version you are interested in). You may then be able to just download the media for the SQLLocalDB installer from the SQL Server Express installer or you may be able to do a minimal install of SQL Server Express that just adds the LocalDB to your system. Visual Studio usually loads the most recent version of SQLLocalDB as part of its install process, so you usually don’t have to worry about this on your development machines.

Once you have installed LocalDB, you will be able to locate the instance installation/configuration utility called SqlLocalDb.exe in C:\Program Files\Microsoft SQL Server\xx0\Tools\Binn\, where you’ll replace the xx with whatever version of LocalDB you have installed. In Visual Studio 2017, I have version 13, so it would be …Server\130\Tools…

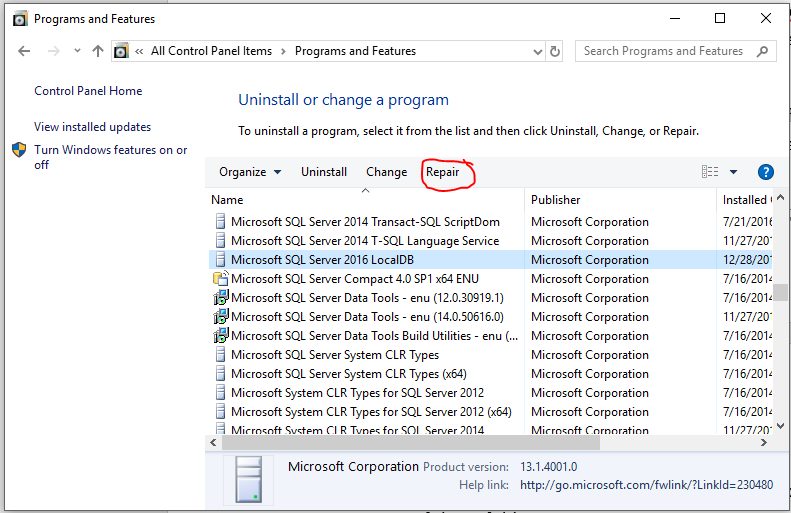
When you have located the SqlLocalDb executable, which is a command-line program, you can use it for the following tasks:

* info – lists all SQLLocalDB instances on the computer
* version – lists the various LocalDB versions that are available on the computer
* stop – will shut down a particular instance
* start – will start up a particular instance
* delete – will remove a particular instance
* create – will create an instance and you can optionally specify the version number
* If you decide you want to upgrade the installed MSSQLLocalDB instance to the newest version, you can stop and delete the current MSSQLLocalDB and then create a new one. If a version number is not specified, the newest LocalDB instance version will be used by default.

The actual files that make up the SQL Server LocalDB engine are located in C:\Program Files\Microsoft SQL Server\xx0\LocalDB\Binn. Again replace the xx with your version number. You should not mess with these files…

Finally, the working location for all instances of LocalDB, by default, should be found in the following path starting with %AppData%. From there, move back up to the \Local\Microsoft\Microsoft SQL Server LocalDB\Instances\ folder. Again, do not mess with the contents of these folders.

This information could become valuable to you if you need to troubleshoot a LocalDB installation. If you do run into any errors, the quickest way to try to resolve them is by finding the installed LocalDB version under Control Panel🡪Program and Features and then running a Repair on it:



The repair only takes a few seconds and it will fix most issues with LocalDB!