Using VBA to Calling a Sub Procedure inside Excel - Featherman©

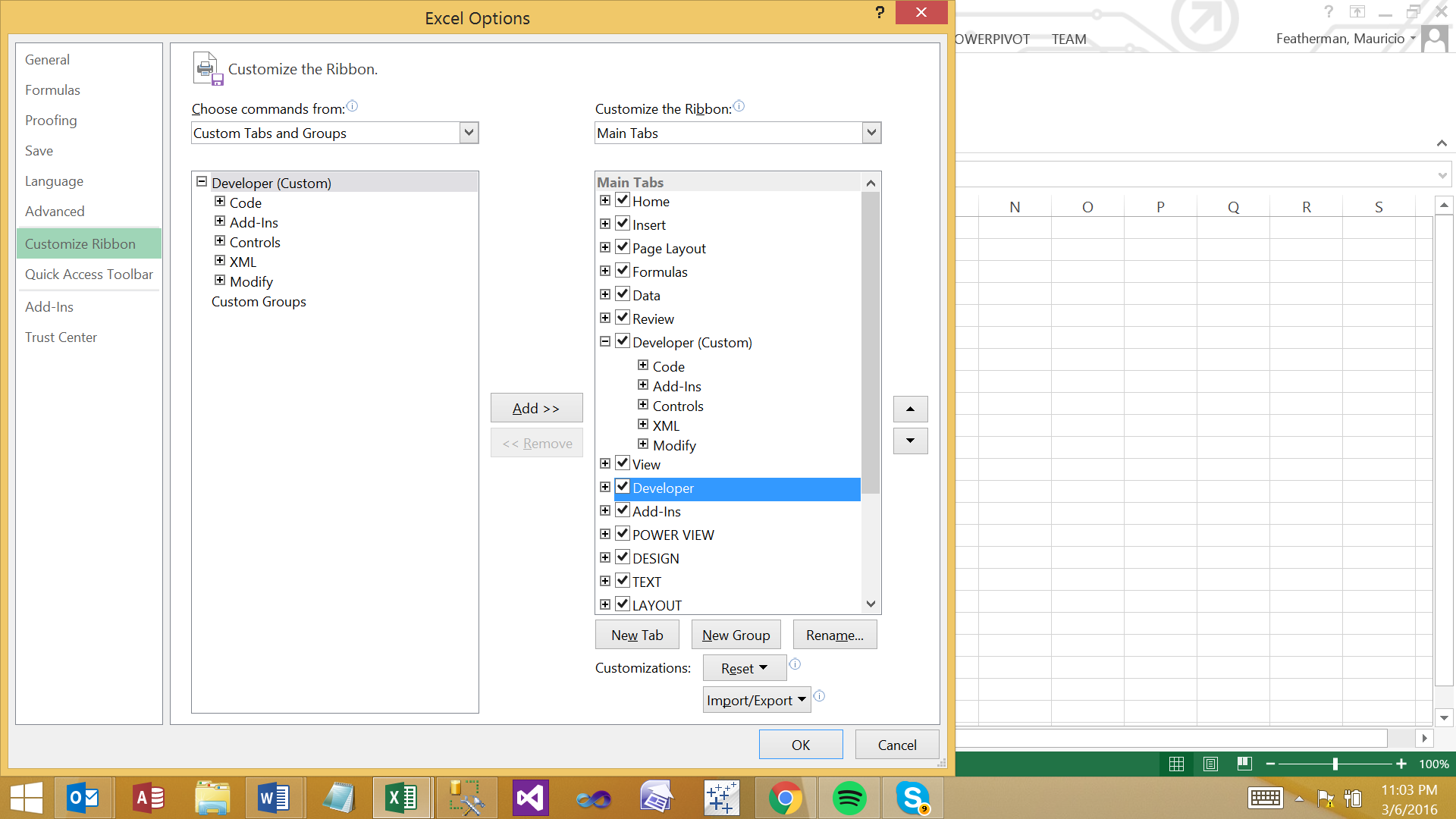
There are lots of ways to connect to the data you need, some more laborious than others. To date we have connected Excel, Tableau & SSRS to datasets, or created the dataset inside Tableau and SSRS with a SQL Query or the query builder. The SSIS trick of having an empty table filled with compiled data (provided by queries), then using the table as the datasource for reports and dashboards is also good trick. Our original methodology was to use a SQL query to retrieve compiled data into memory and then copy/paste into Excel files. But how else can we get data into Excel for further processing or to serve as the datasource for the workbook?   
  
Soon we will foray into PowerBI which uses PowerPivot to build a data model inside Excel (in essence bloating your Excel files to megabyte size and having yet another copy of data to keep track of and ensure the data inside is refreshed). If you are the lucky owner however of a SQL stored procedure that uses the inner joins, etc. that builds the dataset that retrieves and formats the data you need to analyze (with your pivot tables, pivot charts or Powerview reports) you can access the dataset that the stored procedure builds in a couple of ways;  
  
1) turn the stored procedure into a view and connect your Excel spreadsheet to the view.

2) use VBA Excel’s programming language VBA to run the stored procedure inside an Excel module.

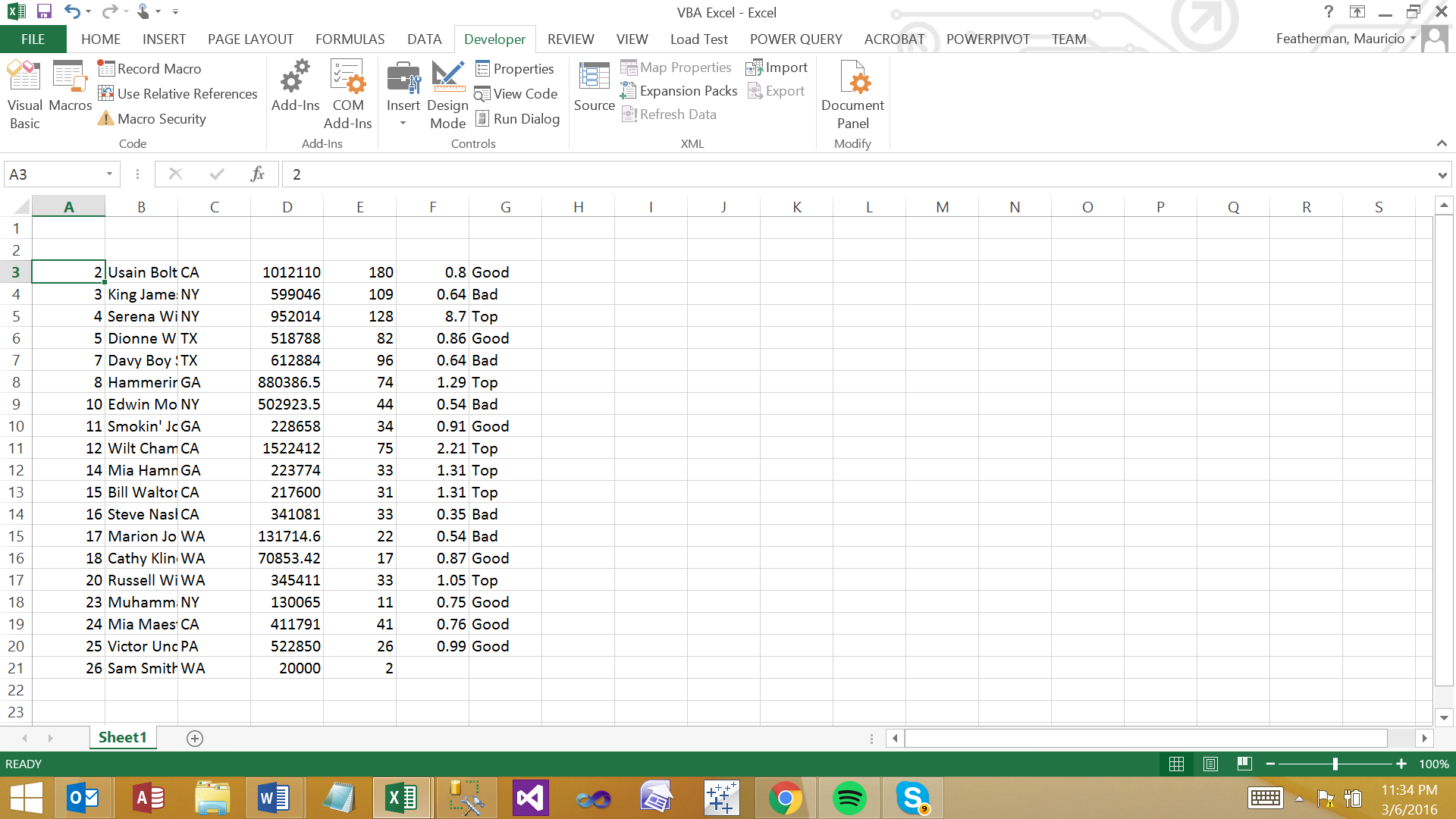
You can also run a query from Excel, you don’t need to have the query saved into a stored procedure.

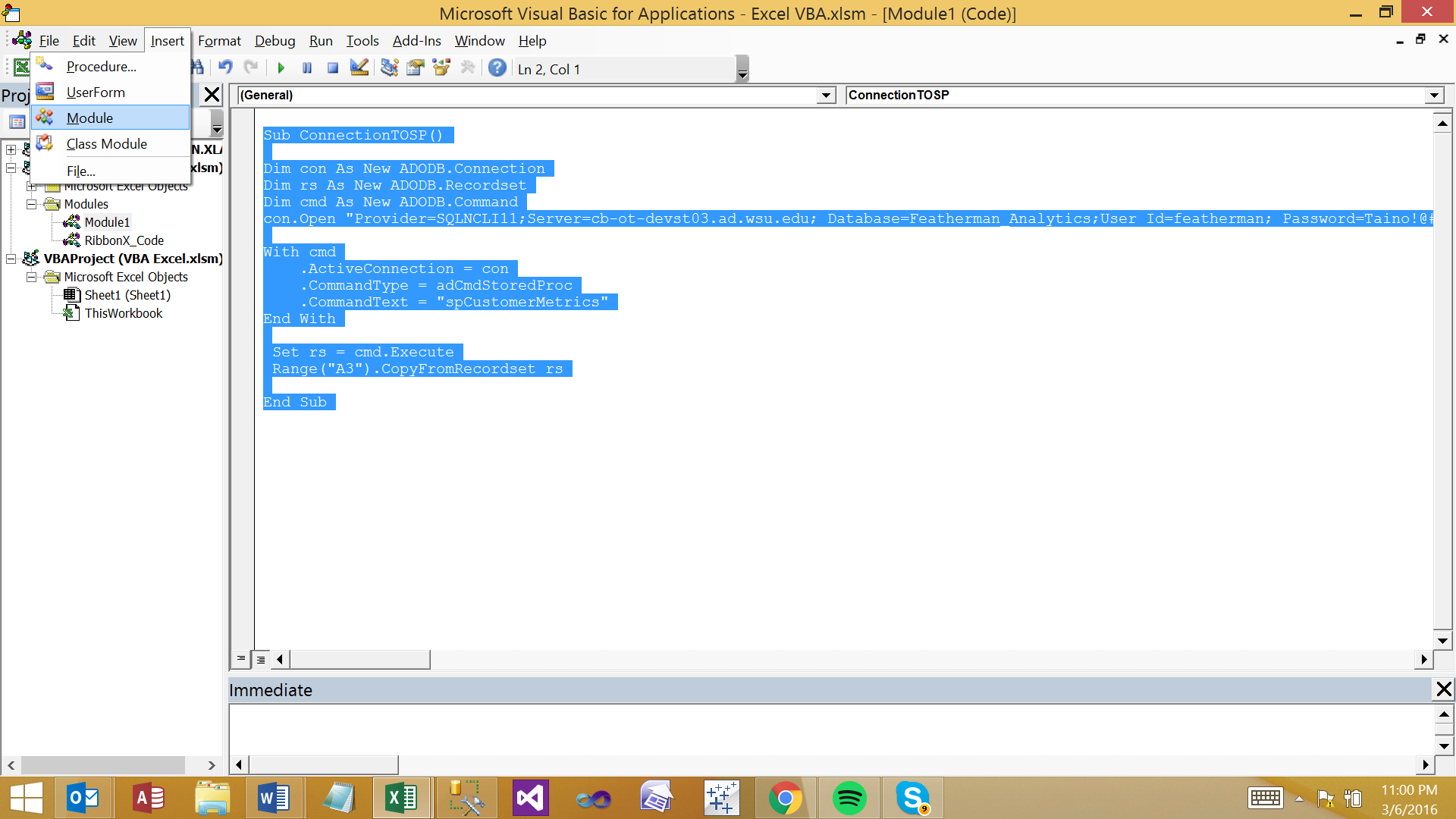
Each method above is superior in that DBA’s or at least well-trained database professionals create the SQL queries, typically checking and testing results and treating the stored procedure or query as an asset in the library of ETL queries. This document shows how to use VBA to run the stored procedure.

The first thing you need to do is make sure you can see the developer tools in Excel, so click on File |Options | Customize Ribbon to activate the developer tools tab.

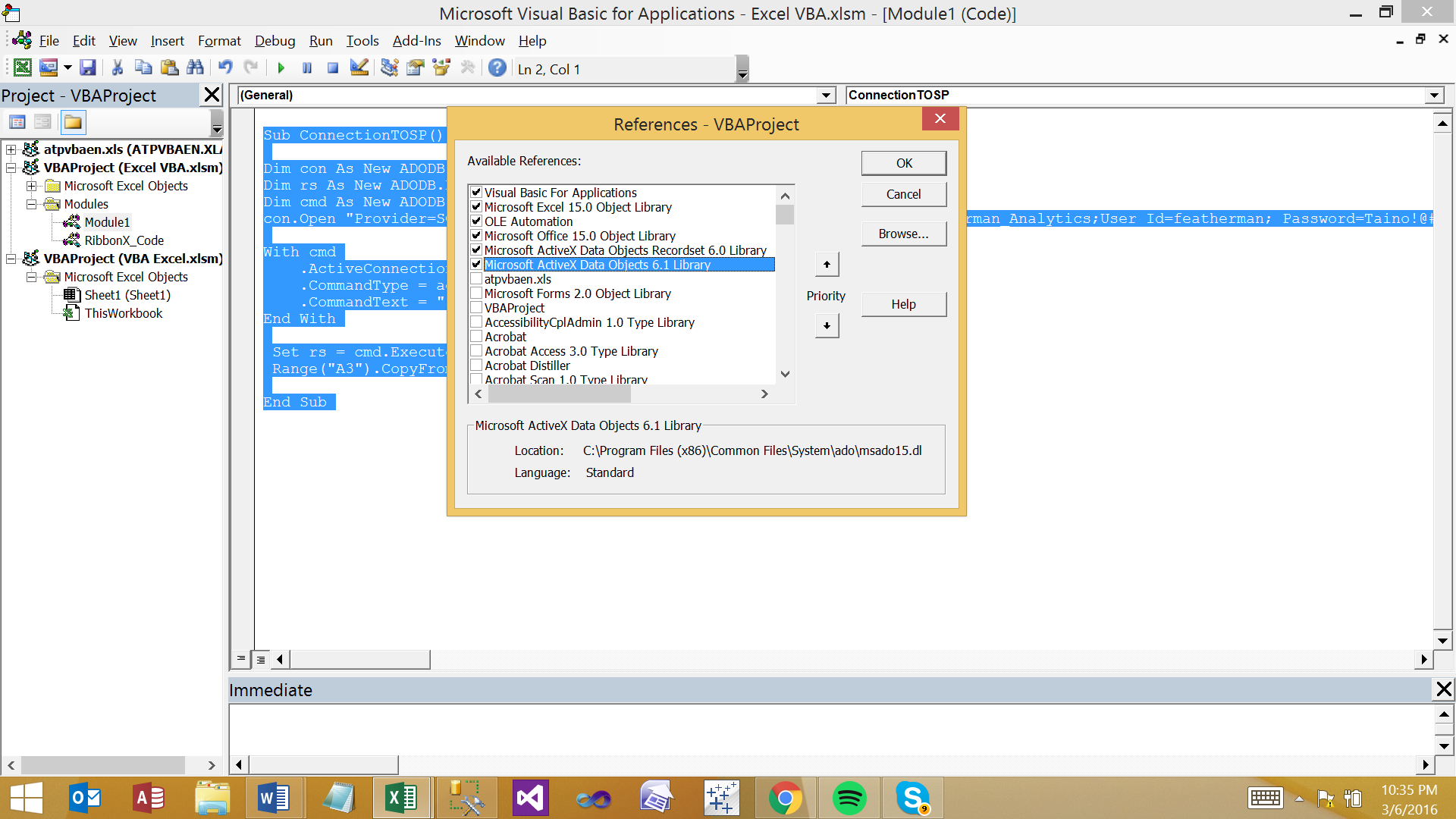


Next click on the developer tab and then on the first icon, for Visual Basic, next click on Insert | Module. Finally copy and paste the code below into the module, modifying the code to specify your database, user ID, password, and stored procedure name.

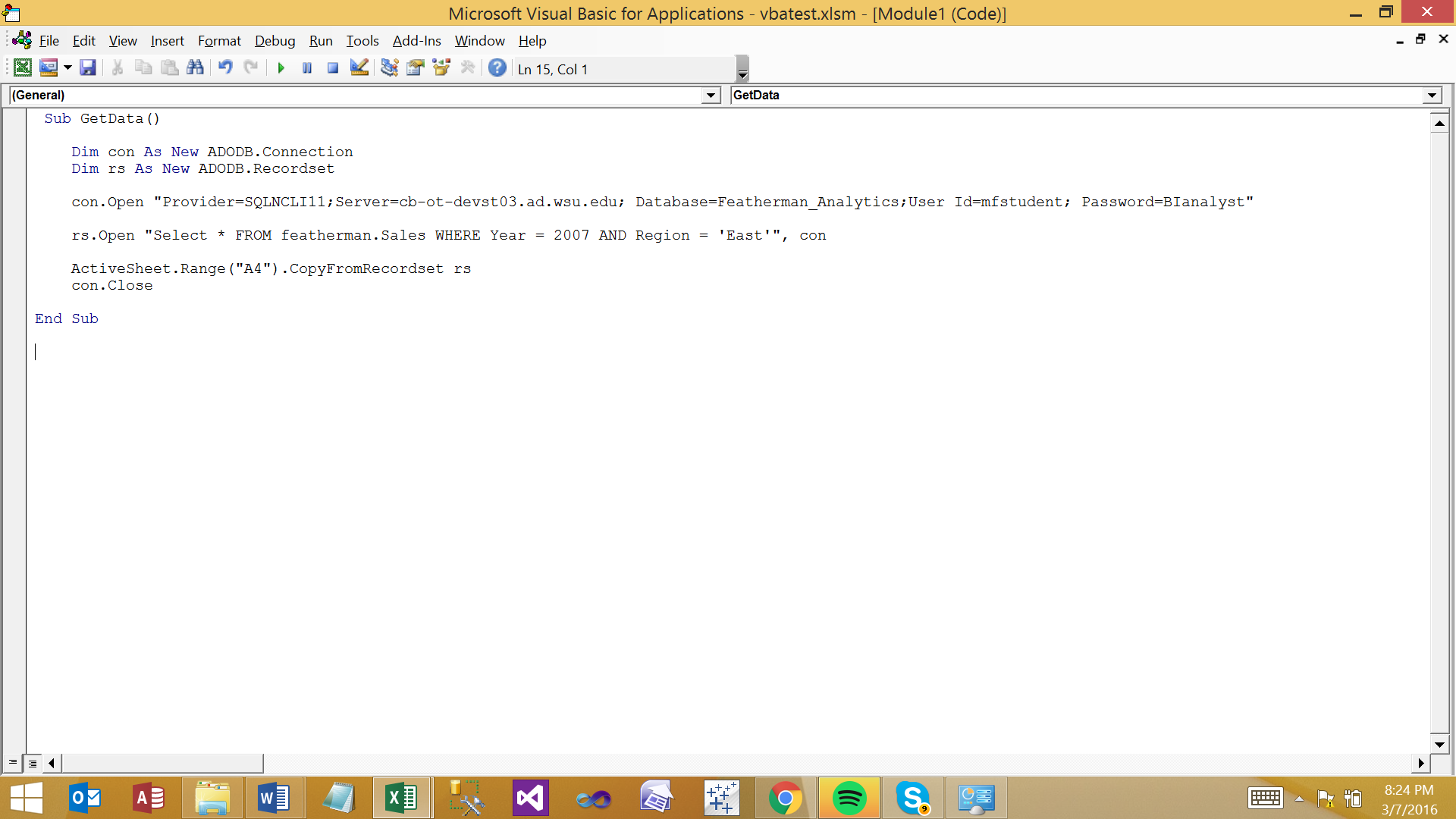




Next select Tool | References and select the newest version of Microsoft ActiveX Data Objects Library. We will use ADODB object in the code, so this set of classes needs to be activated. (Presumably a lot happens in the background so that we have very few lines of code to run.

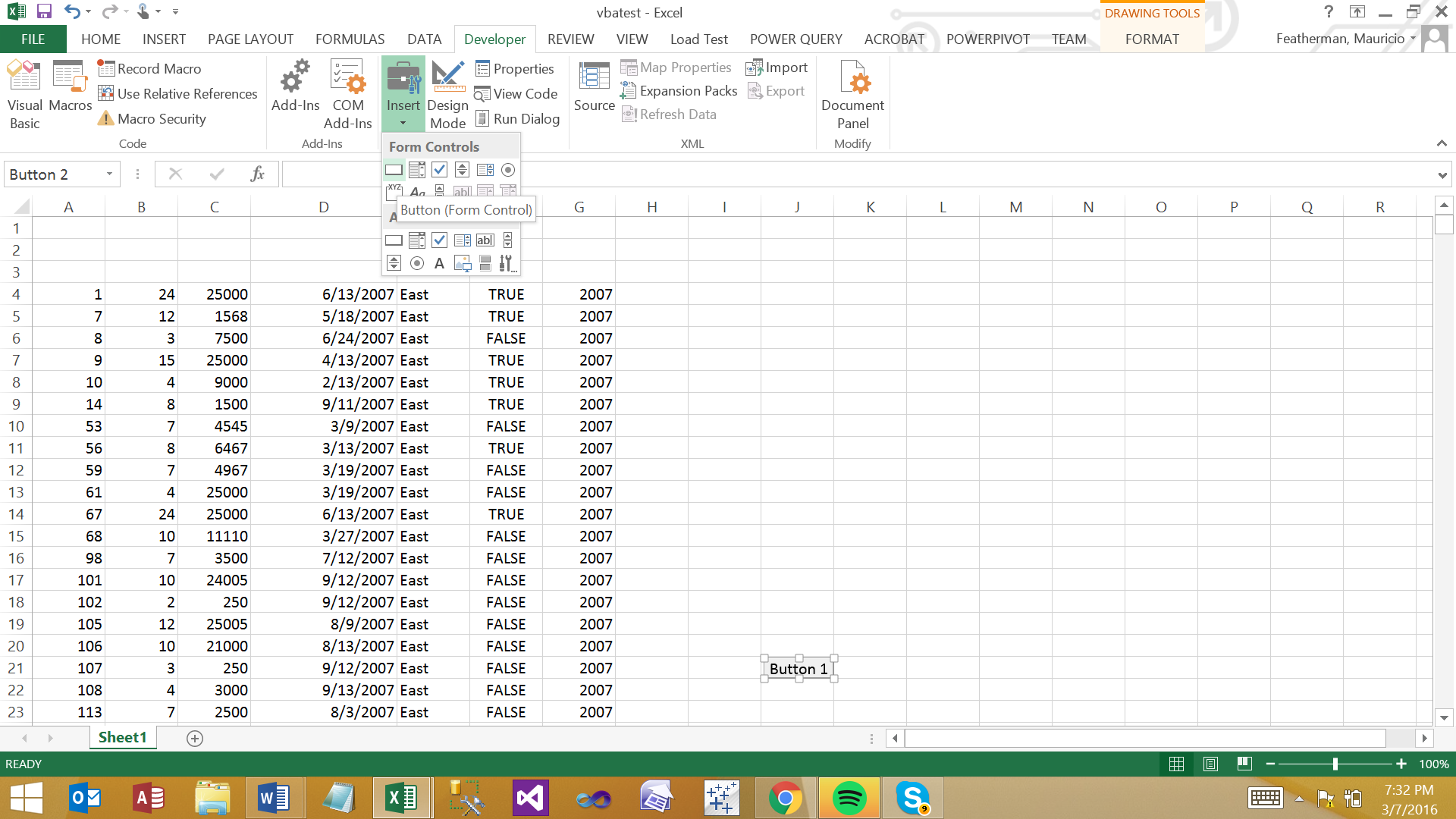


**Demonstration #1** – Here we will run a simple SQL query from an Excel VBA procedure. Select Insert | module and copy and paste the code from the next page into your new module and press the green triangle run button, next press the Excel icon to return to the spreadsheet you should have data! After you know the procedure works, you can add a button to the form and connect the code to the button. This is what is what your module should look like.

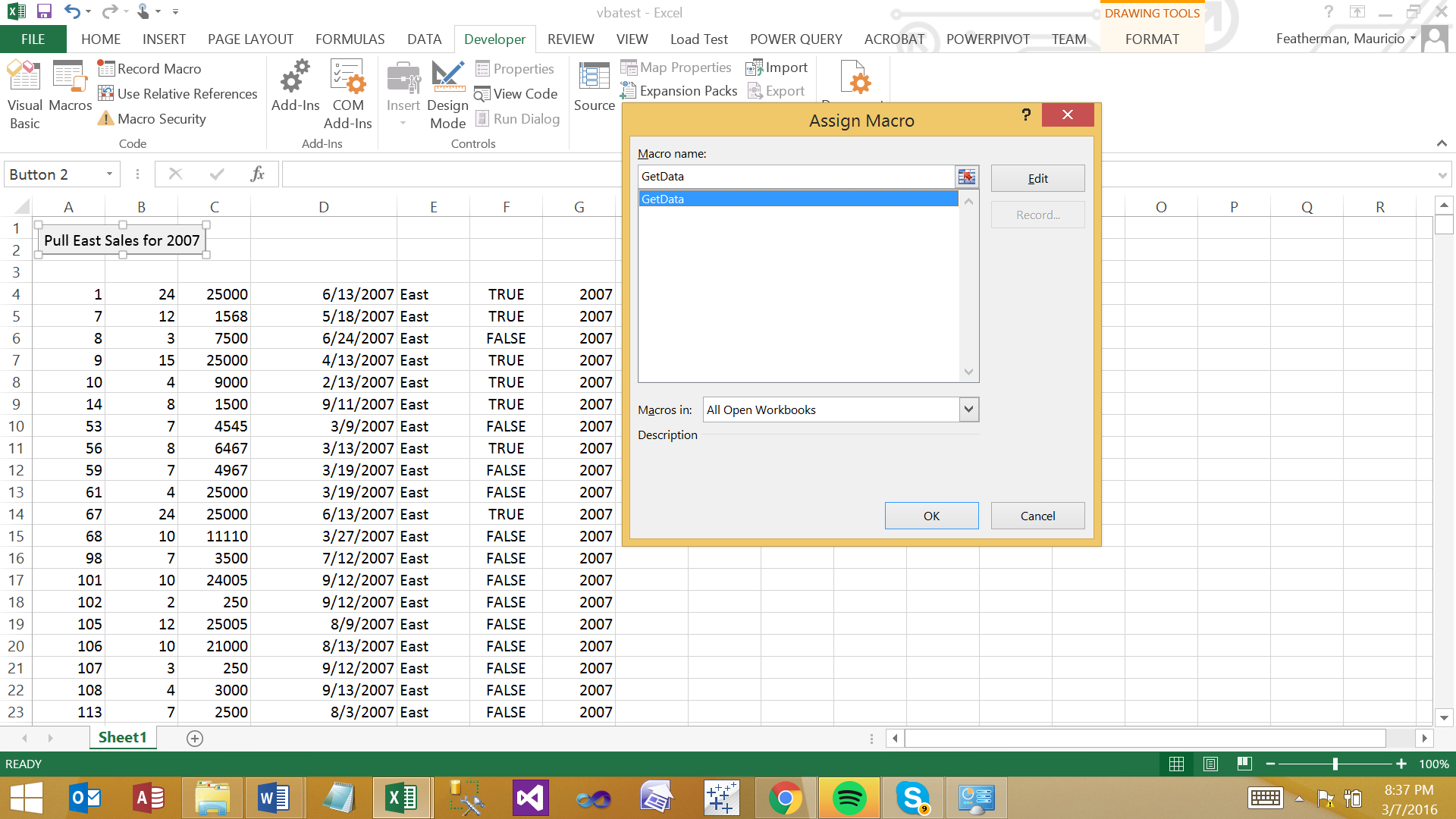


Explanation of code: This is the simplest demonstration of bringing data into Excel. The only simpler method to pull data is to connect to a view (which is a stored procedure).

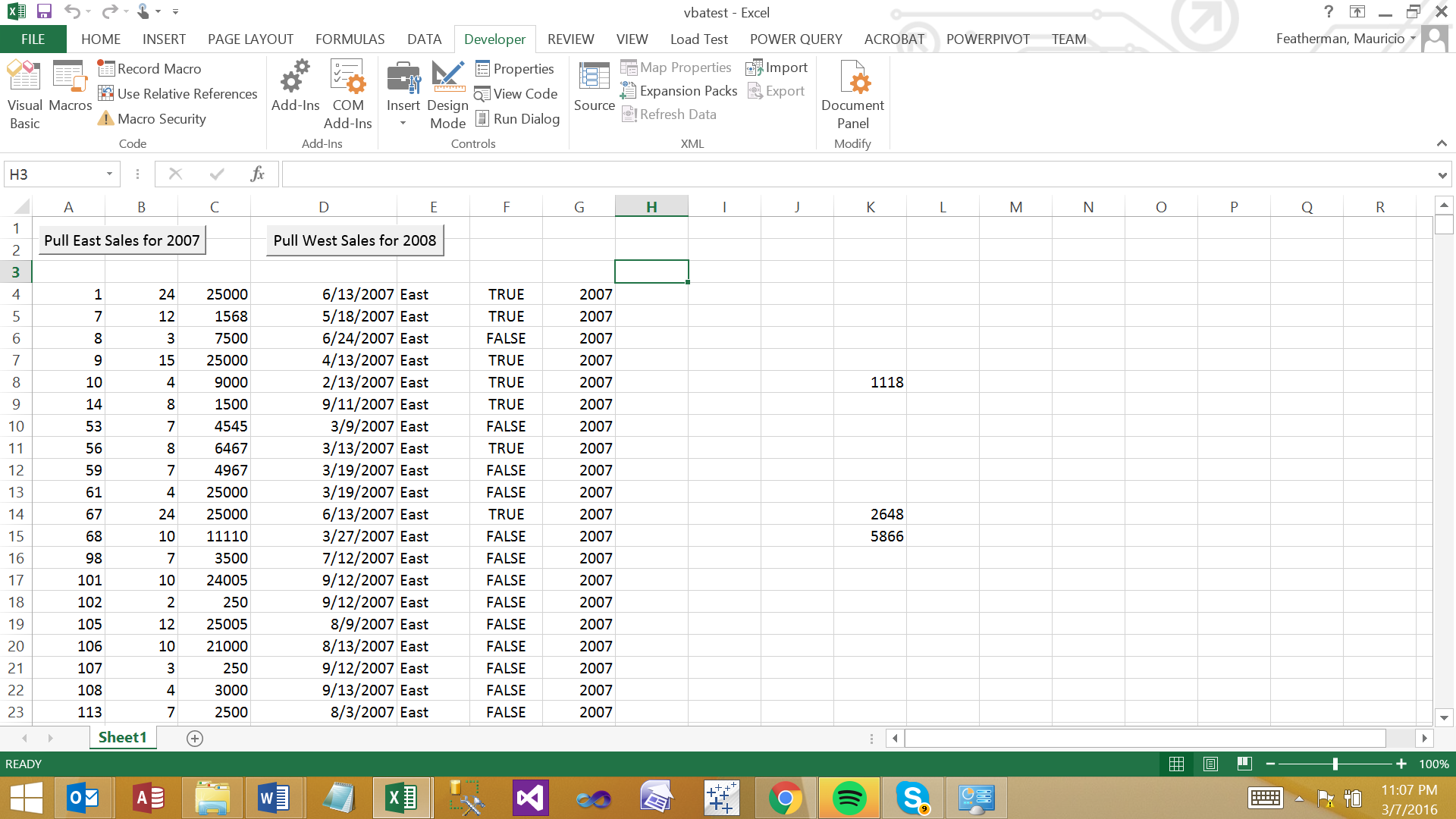
|  |  |
| --- | --- |
| Sub GetData()  Dim con As New ADODB.Connection  Dim rs As New ADODB.Recordset  con.Open "Provider=SQLNCLI11;Server=cb-ot-devst03.ad.wsu.edu; Database=Featherman\_Analytics;User Id=mfstudent; Password=BIanalyst"    rs.Open "Select \* FROM featherman.Sales", con    ActiveSheet.Range("A4").CopyFromRecordset rs  con.Close  End Sub | A sub procedure is a code block. The code between the sub and end sub is run whenever the GetData procedure in invoked.  First we create a connection to the database, and a recordset which will hold the data retrieved from the database.  Next we open the connection supplying the database connectivity information which includes the server name, the database, and the userID and password.  Next we open the recordset and run the SQL statement over the specified connection, in effect stuffing the retrieved data into the recordset.  Next we copy the recodset into the active spreadsheet tab starting at cell A4, and finish by closing the connection. |



After you run the procedure by clicking the green button, you can insert a button onto the form then you can assign the procedure to the button. If the assign macro screen below does not display, then right click the button and select Assign macro (also change the text of the button)

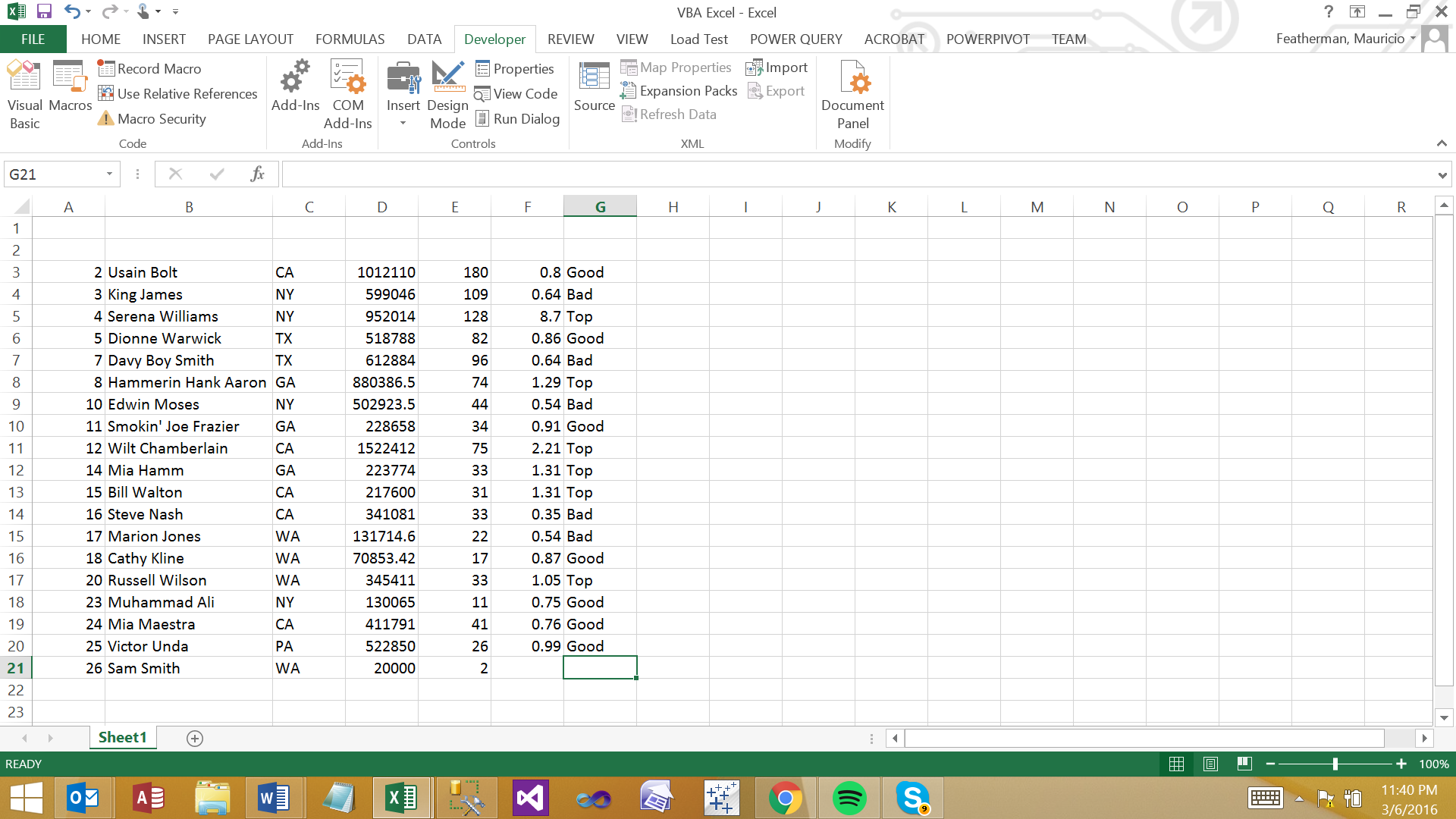


A downfall of the technique (so far) is that the column headings are not automagically copied to excel when the procedure is run.

**Demonstration #2** – To recap, if you have a DBA that makes a great SQL query for you and sends a .sql file to you or just the SQL text, you can drop the query it into an excel VBA procedure (if Excel is your tool of choice). You can perhaps sense that it is not very far of a stretch for an analyst to build a file with many buttons on it, each with a special purpose of bringing in a dataset for further analysis. Below is such an example, two buttons each hard coded to bring a special dataset into the form for further analysis and charting. Lets look at a coding trick though to shorten up the lines of code.  
  


|  |  |
| --- | --- |
| Public con As New ADODB.Connection  Public rs As New ADODB.Recordset  Public Const str As String = "Provider=SQLNCLI11;Server=cb-ot-devst03.ad.wsu.edu; Database=Featherman\_Analytics;User Id=mfstudent; Password=BIanalyst"    Sub GetData()  con.Open str  rs.Open "Select \* FROM featherman.Sales WHERE Year = 2007   AND Region = 'East'", con    ActiveSheet.Range("A4").CopyFromRecordset rs  con.Close  End Sub  Sub GetData2()  con.Open str  rs.Open "Select \* FROM featherman.Sales WHERE Year = 2008   AND Region = 'West'", con    ActiveSheet.Range("A4").CopyFromRecordset rs  con.Close End Sub | Just like in C# and VB.Net programming in visual Studio, you have the concept of Public objects that many sub procedures can each use.  Here we really do not need to create the same connection and recordset and connection string over and over for each procedure.  So if you use the constructor Public rather than dim, then the object can be used in all of the sub procedures. So the objects are global to the module, rather than local to the procedure.  Other processing must be local to the procedure as shown |

**Demonstration #3** – this procedure uses a VBA procedure in a module to call a simple SQL stored procedure (not parameterized). This is similar to connecting to a SQL view which is a saved query. Demonstration #3 uses VBA to send parameter values to a parameterized stored procedure. Here is what the output looks like. The explanation of the code is below. So far the code is run from the module page, not from a button click on the form.

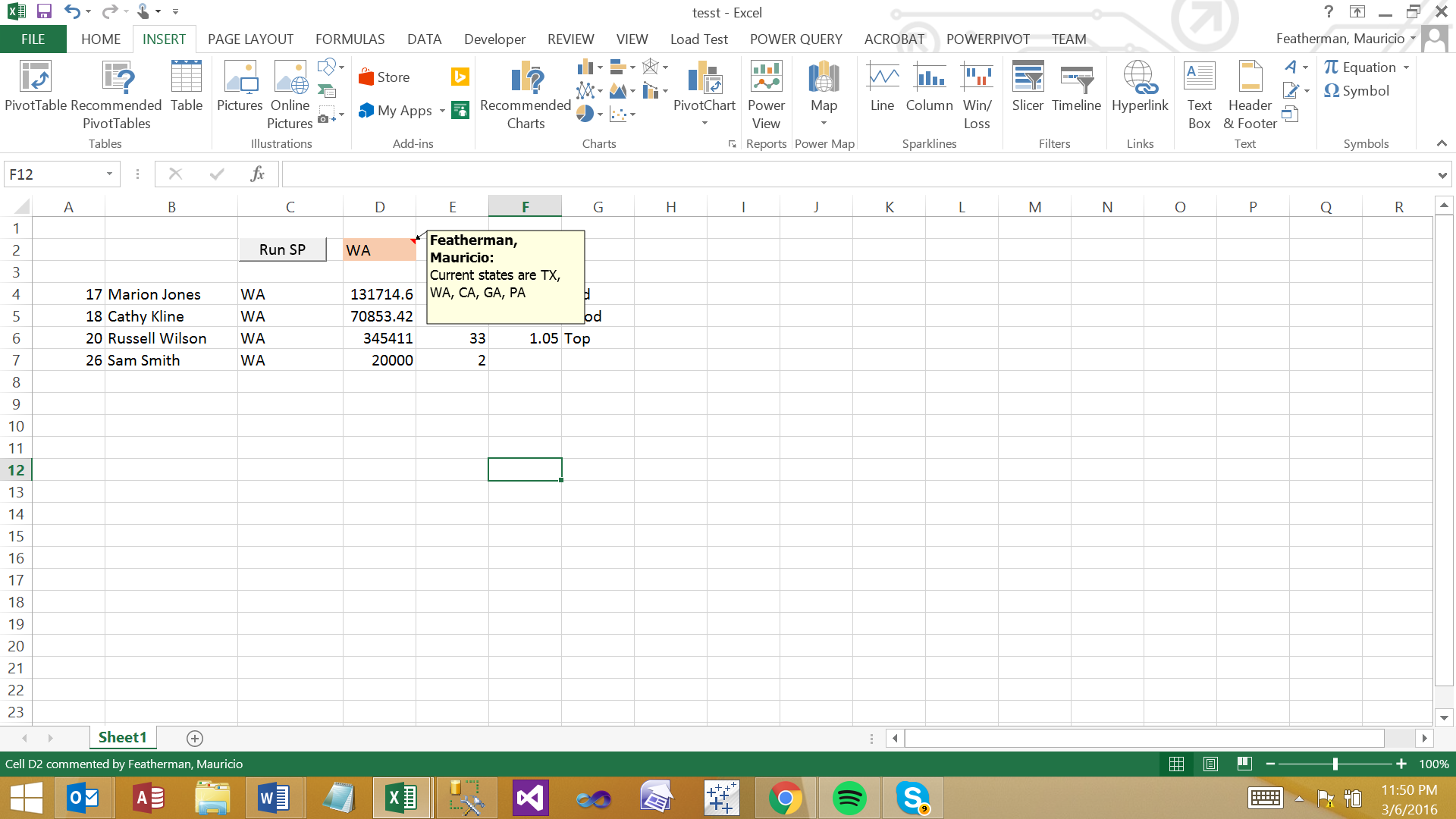


|  |  |
| --- | --- |
| Sub ConnectionTOSP()  Dim con As New ADODB.Connection  Dim rs As New ADODB.Recordset  Dim cmd As New ADODB.Command  con.Open "Provider=SQLNCLI11;Server=cb-ot-devst03.ad.wsu.edu; Database=yourdatabase ; User Id=yourUserID; Password= your password”  With cmd  .ActiveConnection = con  .CommandType = adCmdStoredProc  .CommandText = "spCustomerMetrics"  End With  Set rs = cmd.Execute  Range("A3").CopyFromRecordset rs  con.Close  Set con = Nothing  Set cmd = Nothing  End Sub | The entire code is here and is placed in this table for easy copying into a module that you create in your Excel file. Be sure to save your excel file as a macro-enabled Excel file.  ADO is a set of classes called Active Data Objects that are used to connect to databases and fire off SQL commands (yes this means you can use Excel for insert, update and delete records transactions – which is probably better performed in a webpage, but many people wish for this functionality).  The Visual Studio version of these data connection objects is called ADO.NET  The typical objects are connections, commands, and recordsets or tables. Here commands are new. Commands can run SQL text typed into the module or they can refer to a stored procedure. Notice the commandType is stored procedure and the command text is the name of the stored procedure.  This is a procedure that retrieves compiled data into an excel range starting at cell A3 |

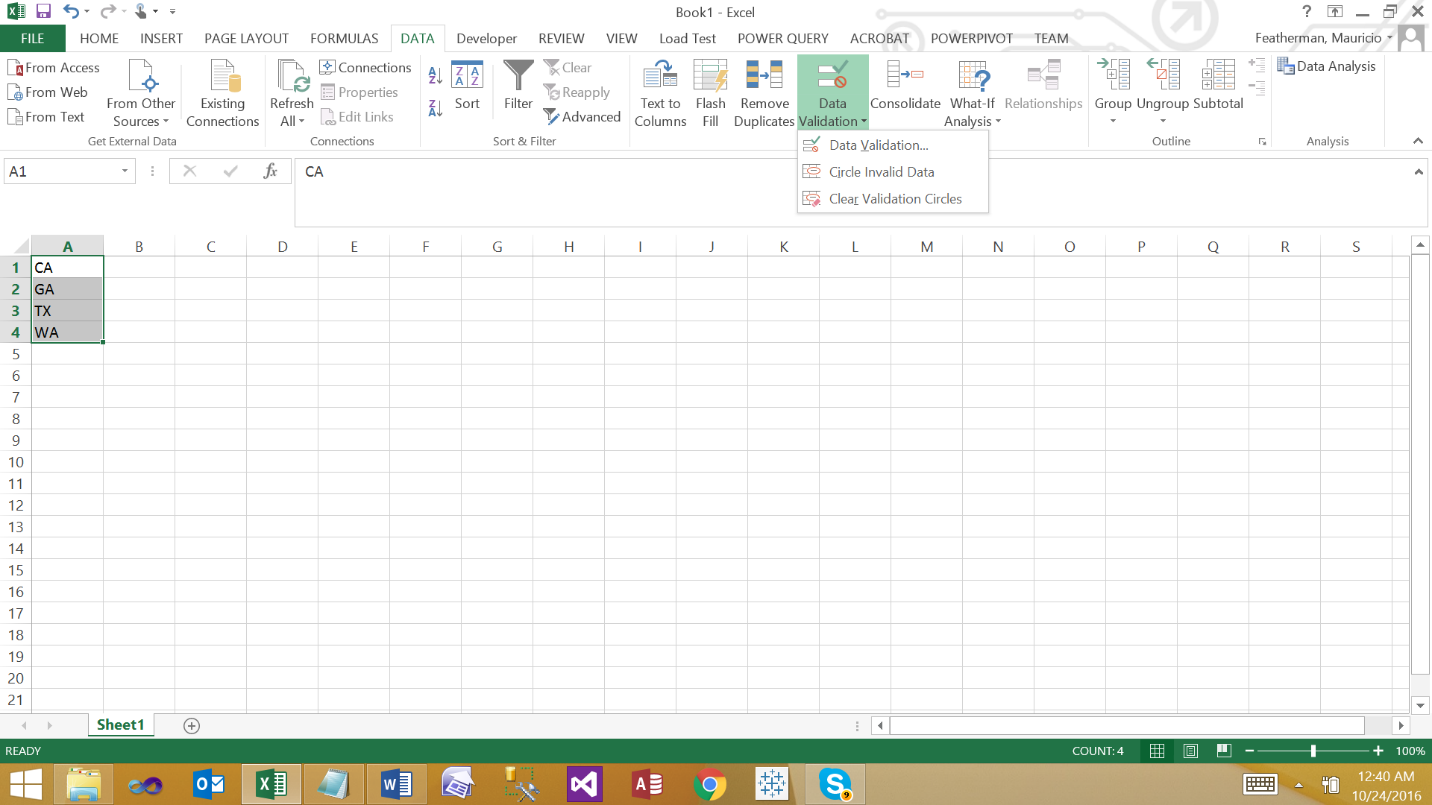
This code above is very simple for the analyst to use and run, just make your command, connection and recordset. Tell the command to run a stored procedure of specified name, then execute the command saving the results into a recordset that is copied into the worksheet. This procedure is very similar to ASP.NET webpages and use of gridviews and datatables which are filled with data and displayed.  
  
The main difference of course is that you are using VBA inside Excel, and you and easily build charts and more analytics in excel.

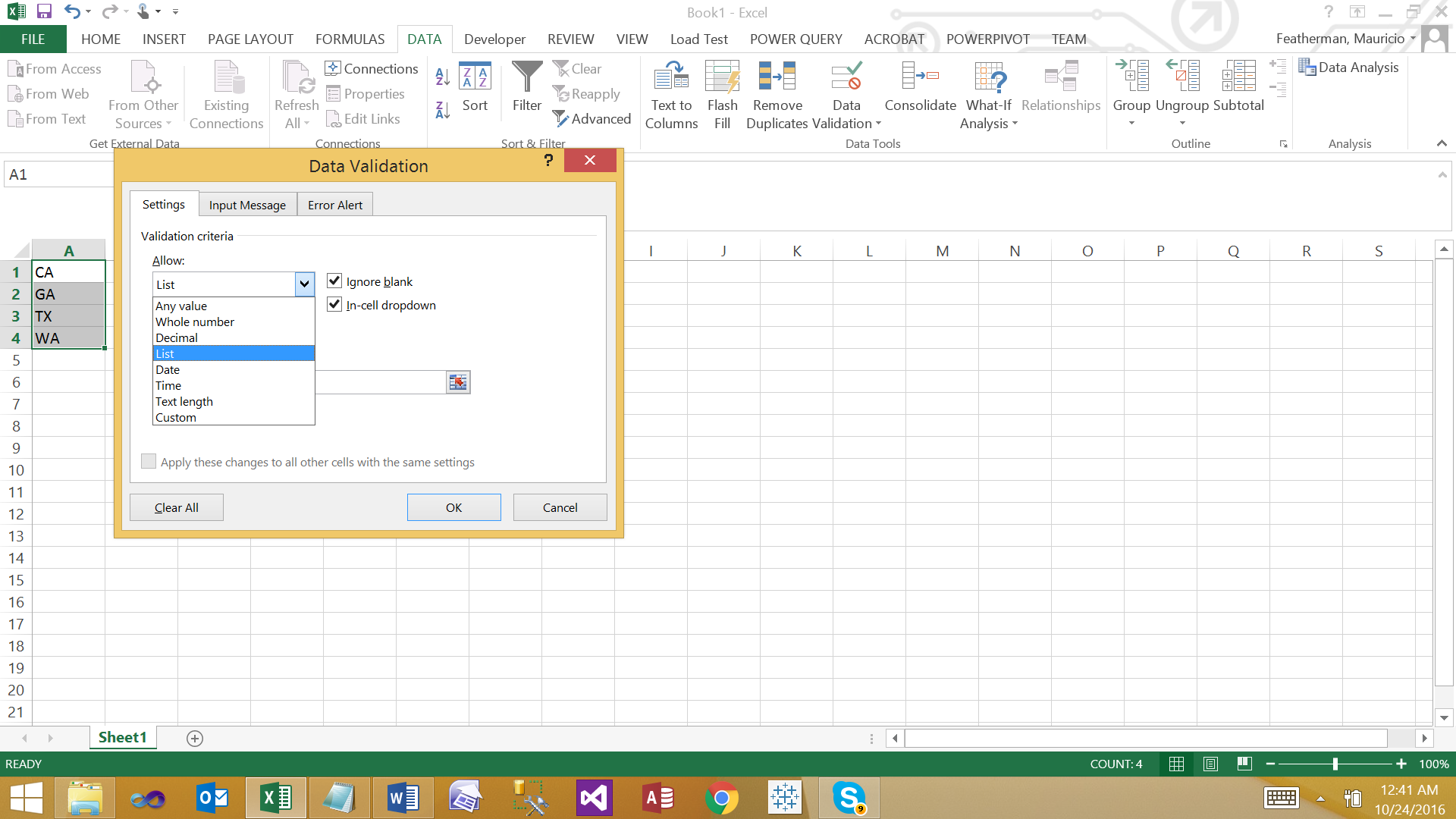
|  |  |
| --- | --- |
| Sub ConnectionTOSP()  Dim con As New ADODB.Connection  Dim rs As New ADODB.Recordset  Dim cmd As New ADODB.Command  con.Open "Provider=SQLNCLI11;Server=cb-ot-devst03.ad.wsu.edu; Database=yourdatabase ;  User Id=yourUserID; Password= your password” | *Here the procedure is explained line by line.*  First a procedure is used to package up lines of code which are in essence instructions to the computer software. A later example will use a button on the form and assign code to the button1\_click event for that button on the form, in essence running the code whenever the button is clicked.  We dimension (create in memory) a connection to our SQL Server database. Next we create a recordset that will hold a set of retrieved database records (this is akin to a table)  Next we create a command which we will use to fire off the SQL stored procedure. Finally we open the connection to the database |
| With cmd  .ActiveConnection = con  .CommandType = adCmdStoredProc  .CommandText = "spCustomerMetrics"  End With | We use a With end with to organize all the properties that need to be set for the same object.  The command will run the stored procedure so we start by telling it what connection to run on (which in turn specifies the database and server). We next specify the command type as stored procedure, and provide the name of the stored procedure – notice it is in quotes as it treated as a string. |
| Set rs = cmd.Execute  Range("A3").CopyFromRecordset rs | Here is where the action occurs.  The assignment operator (=) works from right to left so the cmd.execute operation executes the stored procedure assigning the output from that operation (a bunch of compiled data rows) to the in-memory data container recordset that we created.  Next we copy the data in memory to a range that starts in cell A3 on the current spreadsheet. |
| con.Close  Set con = Nothing  Set cmd = Nothing  End Sub | Now we do memory management we close the connection because SQL Server license are paid for by the number of concurrent users, if a connection to the database remains open then it can’t be used by another individual.  We reset the command and connection freeing up memory for other operations. |

**Demonstration #4** Ok. But what about connecting the code to a button click event or passing parameter values into the stored procedure? Read on brave data analysts! As you can see there is a button on the form which is next to a cell that is shaded to signify that you should type in data there, the Insert Comment command is nice in that it’s easy to give the program user some instructions in a pop-up manner. Go ahead and create a new file, save it as a macro-enable workbook, and add the button and shaded cell. We will simply filter the prior query.



|  |  |
| --- | --- |
| Sub Button3\_Click()  'be sure to use menu command Tools | Reference | to add in the Microsoft ActiveX Data Objects Libray (the highest version, currently 6.1?)  'here we will open a connection, use a command to run a called SQL stored procedure, bringing the data into a recordset (literally a set of records) on the worksheet  Dim con As New ADODB.Connection  Dim cmd As New ADODB.Command  Dim rs As New ADODB.Recordset  Dim WSP1 As Worksheet  ' Remove any values in the cells where we want to put our Stored Procedure's results.  Dim rngRange As Range  Set rngRange = Range(Cells(4, 1), Cells(Rows.Count, 1)).EntireRow  rngRange.ClearContents  ' Log into our SQL Server, and run the Stored Procedure  con.Open "Provider=SQLNCLI11;Server=cb-ot-devst03.ad.wsu.edu; Database=your database; User Id=yourID; Password=your password”  With cmd  'tell the command upon which connection and database to operate, and that that it will be running a delux stored procedure  .ActiveConnection = con  .CommandType = adCmdStoredProc  .CommandText = "spCustomerMetricsParameterized"  ' Add the parameter to the Stored Procedure - the name MUST match the variable name in your Stored procedure (Parameter types can be adVarChar,adDate,adInteger)    .Parameters.Append cmd.CreateParameter("**@strState**", adVarChar, adParamInput, 10, Range("D2").Text)    Set rs = .Execute(, , adCmdStoredProc)  End With  ' Copy the results to cell D1 on the first Worksheet  Set WSP1 = Worksheets(1)  WSP1.Activate  If rs.EOF = False Then WSP1.Cells(4, 1).CopyFromRecordset rs  ‘the rest is cleanup  rs.Close  con.Close  Set rs = Nothing  Set cmd = Nothing  Set con = Nothing  End Sub | Notice that the procedure is called button3\_click – this means that the code will run in response to that event – the button being clicked.  We again create the connection, command and recordset.  We also create a reference to the worksheet, so that we can interact with the worksheet.  The next set of lines declare a range which will be used to display the data. The range starts at (4,1) which is row 4 and column 1 (column A).  Next we activate the connection and set up the command. The new line is very important in that it adds a parameter to the command. Notice that the parameter has the same EXACT name as the variable created in the stored procedure (please verify below).  Note that the value is passed in from Cell D2.  Next the results of the command being run is assigned to the recordset.  The recordset is copied into a range on the first worksheet tab, starting at cell (4,1) i.e. D1  Finally all the objects are removed from memory. |
| USE [Featherman\_Analytics]  GO  CREATE PROCEDURE [your database owner].[spCustomerMetricsParameterized]  **@strState** as char(2)  AS  BEGIN  SELECT \*  FROM [Featherman\_Analytics].[featherman].[CustomerMetricsSSIS]  WHERE [State] = @strState  END | Here is the stored procedure. While it is a very simple query, for this demonstration, the query can actually be quite complex. |

We can make a dropdownlist using the Data | Data Validation option so that you can select a state then press the button.



Set the Range for the input control to the range of options (sorry no idea about multi-select at this point).

A lot more can be done with VBA, I personally hope you take a deeper look at ASP.NET webpages which also draw from the ADO.NET set of classes and functionality.