
SOCIETY OF ACTUARIES

Article from:

# CompAct

May 2004 – Issue 17

# Using Excel to Prototype Relational Database Applications

*by Dave Snell*

Excel is often dismissed by information technology (IT) professionals because it is viewed by them as a toy—suitable only for single-user applications. Actuaries know the value of spreadsheets in rate calculations and what-if scenario testing; but most actuaries have never realized that their convenient little spreadsheet tool is also capable of lifting some rather heavy weights in the database environment.

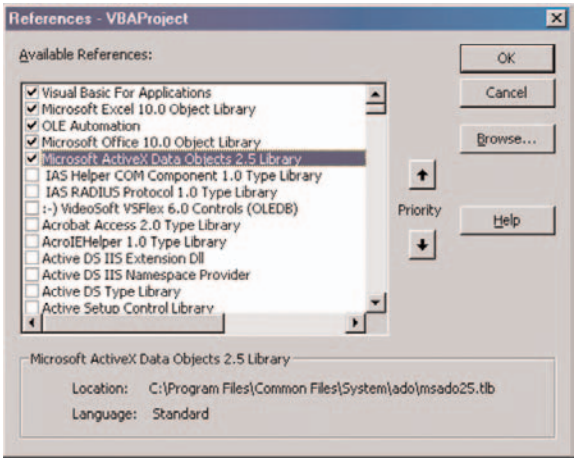
In my opinion, Microsoft Excel is a highly functional programming language that happens to come with an awesome built-in grid control. It is the one of the most cost-effective tools I know of for making high-quality small applications that can be used on almost any corporate desktop. It can also be great for prototyping larger database applications.

I propose to show in this article that you can develop and test relational database projects from your spreadsheets. You can do it rapidly, and with less angst or cost than you might feel by going through the learning curve of the big database tools, like Microsoft's SQL Server or Oracle. I'll assume you have some familiarity with Structured Query Language (SQL), which we won't cover here. I also assume you've some exposure to Visual Basic for Applications (VBA) and that you may have even heard of Active Data Objects (ADO); but this can be minimal. Once you follow the instructions for entering the code below, you can embellish it at your leisure. This article is to help get you started using SQL from within Excel.

## Using SQL with Excel sources

Excel has the capability to contain multiple tables and ranges, and it can be used as a source for SQL queries.

An easy way to accomplish this is via ADO. First, you must add a reference to ADO in your project in the VBA mode, which you can reach quickly from the spreadsheet mode via Alt-F11 (hold down the Alt key and press the F11 key). Once in VBA mode, click on Tools, References.




There are several ADO libraries. I recommend using the lowest numbered one that meets your needs so that your workbook will be more transportable. In this case, I chose version 2.5, which probably came with Excel 97.

Next, you are going to need a little VBA help in the background to let you effectively use this conduit to SQL capability. Click Insert, then Module to get a working area. Then copy and paste the code in from this article.

The first item to copy is a Sub to read from an Excel database via Active Data Objects (ADO).

```
Sub CopyFromXLDatabaseADO( _
   sDBRangeBook As String, _
   sSQLRangeName As String, _
   sDestinationSheet As String, _
   Optional bolClearSheet As Boolean = True, _
   Optional bolShowFieldNames As Boolean = True, _
   Optional sDestRange As String = "A1")
'written by Dave Snell
'This Sub allows you to perform SQL queries against
'tables (named ranges) in an Excel workbook

Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim fld As ADODB.Field
Dim lngOffset As Long
Dim lngStartTime As Long
Dim cmd As New ADODB.Command
Dim s As String
Dim sSQL As String
Dim I As Integer

   lngStartTime = Timer

   Sheets(sDestinationSheet).Activate
   If bolClearSheet Then
      Cells.Select
      Selection.ClearContents
   End If

   'populate with new information from the Access database extract
   With cn
      'PROVIDER STRING follows (change for different source type e.g., Oracle)
      .Provider = "Microsoft.Jet.OLEDB.4.0" 'If you use a 'different
      'version of 'the Jet engine, change the 4.0 to whatever you use.
      .Properties("Extended Properties").Value = "Excel 8.0" 'If you
      'use Excel '2000, this is correct; Excel 2002 is 9.0, etc.
      .Open ActiveWorkbook.Path & "\" & sDBRangeBook
   End With
   With cmd
      Set .ActiveConnection = cn
      'build a SQL query from the text on the SQL worksheet
         With Sheets("SQL")
            s = "  "
            sSQL = ""
            While Len(s) > 0
            s = .Cells(.Range(Range(sSQLRangeName).Address).Row + I, _
               .Range(Range(sSQLRangeName).Address).Column).Value
               sSQL = sSQL & " " & s
               I = I + 1
            Wend
         End With
         .CommandType = adCmdText
         .CommandText = sSQL
         Set rs = .Execute
      End If
   End With

   If Not rs.EOF Then
      With ActiveSheet.Range(sDestRange)
         If bolShowFieldNames Then
            'write field names as column headers
            For Each fld In rs.Fields
               .Offset(0, lngOffset).Value = fld.Name
               lngOffset = lngOffset + 1
            Next fld
         End If
      End With
      'copy the rest of the records one row below the headers
```

• continued on page 14 •

```
        ActiveSheet.Range(sDestRange).Offset(1, 0).CopyFromRecordset rs
        ActiveSheet.UsedRange.EntireColumn.AutoFit
    Else
        MsgBox "Error: No records read", vbCritical
    End If

    Range(sDestRange).Select
    MsgBox "Done!" & vbCrLf & "Elapsed time = " & Timer - lngStartTime & " seconds"

ExitRoutine:
    rs.Close
    cn.Close
    Set cn = Nothing
    Set rs = Nothing
    Set cmd = Nothing

End Sub 'CopyFromXLDatabaseADO
```

This Sub should be usable for many situations. Note however that for readability of the sample code, I removed a lot of error checking. You will want to add error checking in various portions of the Sub. Since this approach uses ADO for the data access, you can also use it to query Access databases, Oracle databases, etc. with a mere change of the Provider string. In practice, I have written a longer version of this macro that determines the source type from parameters I set and changes the provider string accordingly (a future article). Rather than get complicated here though, let's just go ahead and try an application showing how you might make use of the supplied Sub.

Let's create a very simple calling routine:

```
SubTest_CopyFromXLDatabase()
    CopyFromXLDatabaseADOThisWorkbook.Name, "SQL_Example", "Example_Output"
End Sub 'Test_CopyFromXLDatabase
```

This tells the Sub CopyFromXLDatabase to look in the current workbook (we could also refer to any other workbook) for the Range Name "SQL_Example." Then, it reads down the column from cell SQL_Example and builds the SQL query until it encounters a cell that is blank. The final set of results is then copied to the Excel spreadsheet "Example_Output."

All of this looks OK, but obviously nothing is going to happen unless we actually have a range named "SQL_Example" somewhere in the workbook. Let's assume we have a simple workbook with just three worksheets in it. I'll name my three sheets Example_Source, SQL and Example_output.

In Example_Source we'll place our tables. Assume we have two tables of data as shown in Figure 1 on page 15.

Note that I have the Applicants table highlighted. Likewise, I named the Apps table (cells G4 through I25) "Apps." The easy way to accomplish this naming process is to highlight the desired area with your mouse, then type the desired Range name in the input box just above cell A1. In this screen shot you can see Applicants in this input box.

Next, we look at the SQL sheet, Figure 2 on page 15.

Cell A4 on this sheet is the one I named SQL_Example. I also made sure I have another sheet for my output. This one is named Example_Output.

Everything is in place. Now, let's run the macro (Figure 3, page 15).

Clicking on Tools, Macro then Macros (or just pressing the shortcut key Alt-F8), we see a list of the macros available (in this case just the one test macro).

Run the macro Test_CopyFromXLDatabase and voila! You have the SQL query output right away on the Example_Output sheet. (Figure 4, page 16).

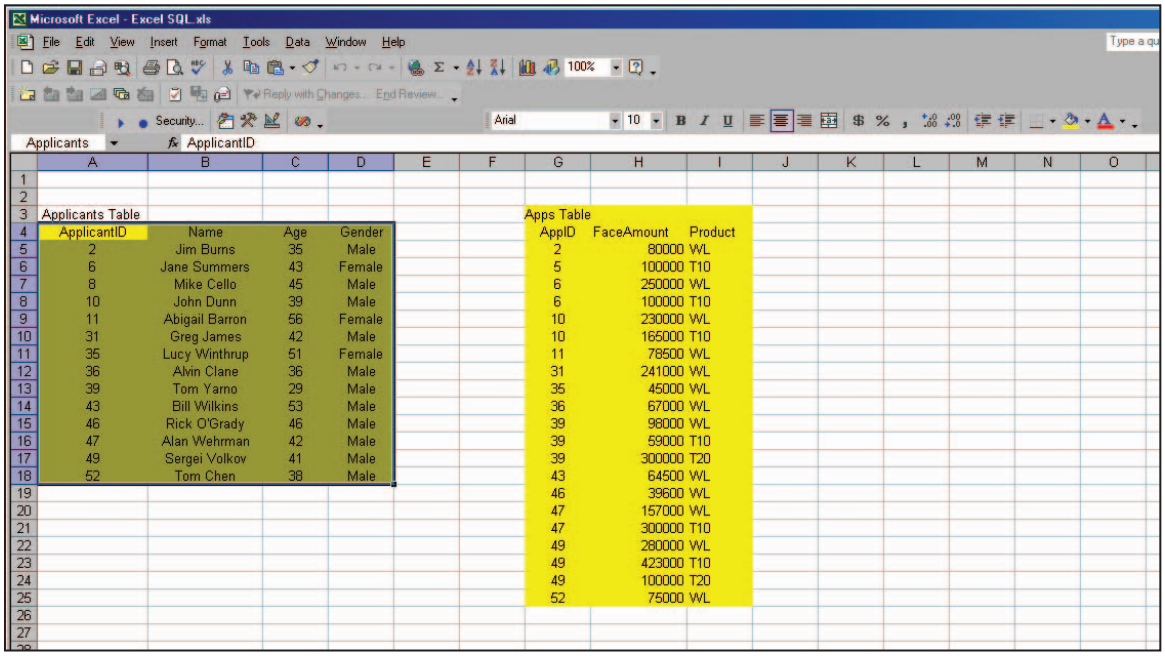



Figure 1

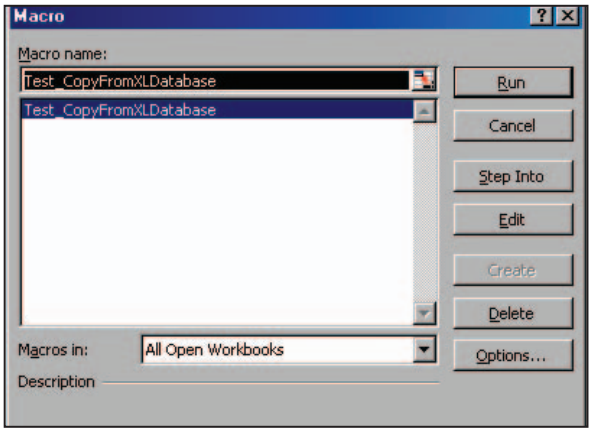



Figure 3

Note the Range Name "SQL_Example" assigned to cell A4 of this sheet.

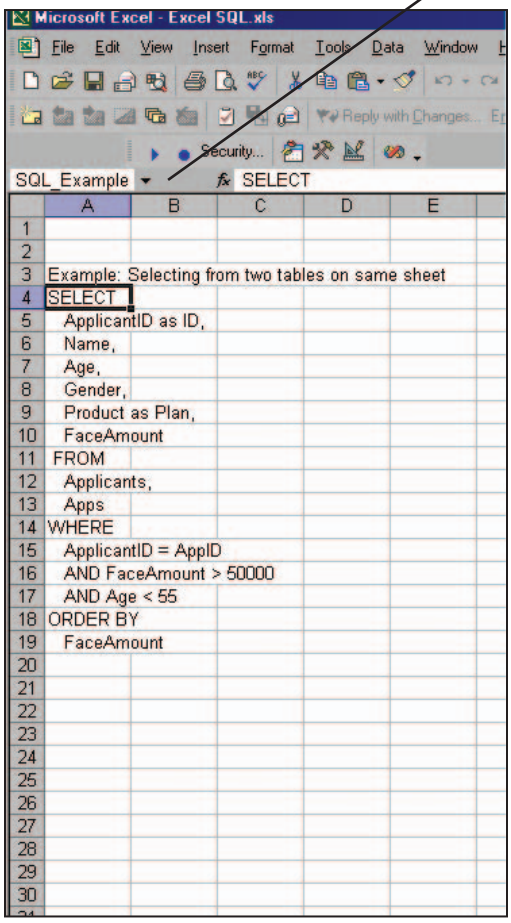



Figure 2

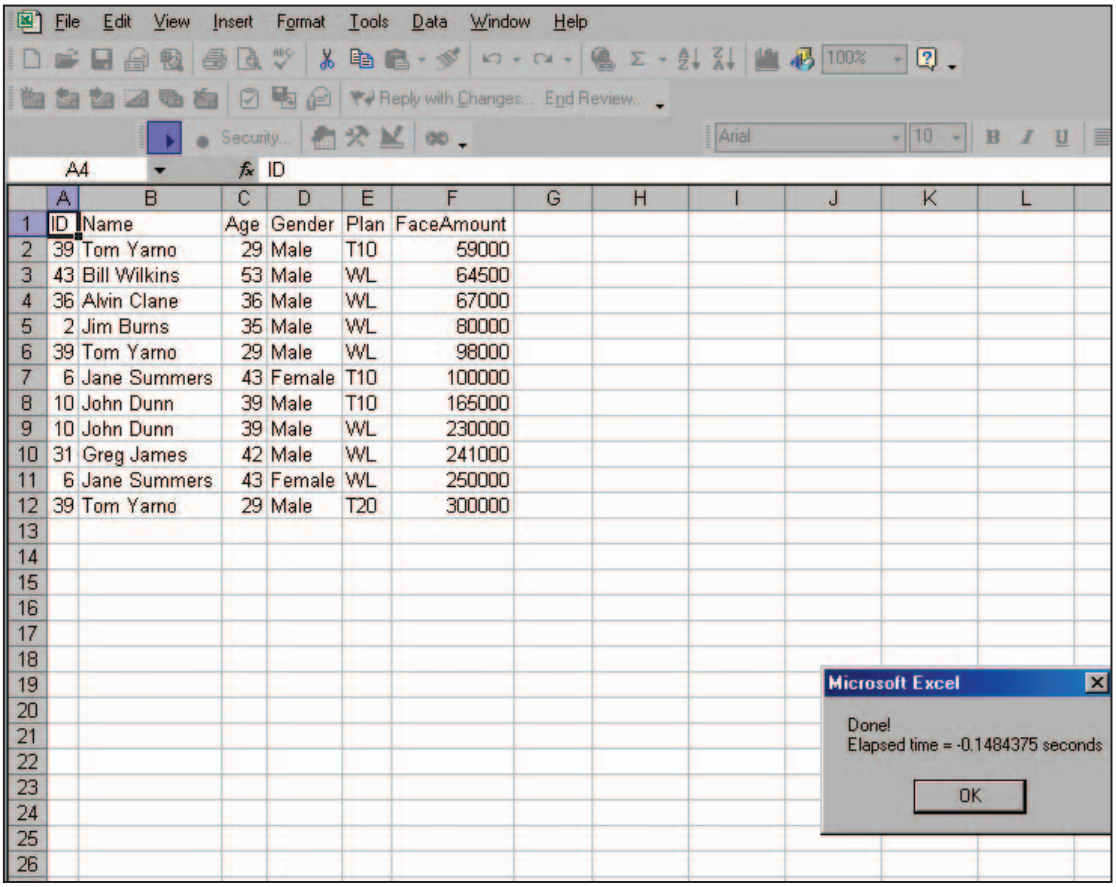
## Bigger and Better

Let's face it. This is a very simple example. It's hardly worth writing a VBA Sub to accomplish this task. You could have obtained similar results with Excel's Data, Filter, Advanced Filter feature or with its built-in QueryBuilder (accessed via Data, Import External Data, New Database Query) capability.

I wrote (and use) the VBA approach for a couple of reasons. First, it allows me to get very flexible with my SQL statements—beyond the limits of the built-in Excel tools. In a report I wrote

• continued on page 16 •

Using Excel continued from page 17




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | ID | Name | Age | Gender | Plan | FaceAmount |
| 2 | 39 | Tom Yarno | 29 | Male | T10 | 59000 |
| 3 | 43 | Bill Wilkins | 53 | Male | WL | 64500 |
| 4 | 36 | Alvin Clane | 36 | Male | WL | 67000 |
| 5 | 2 | Jim Burns | 35 | Male | WL | 80000 |
| 6 | 39 | Tom Yarno | 29 | Male | WL | 98000 |
| 7 | 6 | Jane Summers | 43 | Female | T10 | 100000 |
| 8 | 10 | John Dunn | 39 | Male | T10 | 165000 |
| 9 | 10 | John Dunn | 39 | Male | WL | 230000 |
| 10 | 31 | Greg James | 42 | Male | WL | 241000 |
| 11 | 6 | Jane Summers | 43 | Female | WL | 250000 |
| 12 | 39 | Tom Yarno | 29 | Male | T20 | 300000 |

Figure 4

for one of our foreign offices in Asia, I use a SQL query more than 100 lines long that involves a 16-table join with dozens of special criteria. Second, I like the fact that once I debug my routine on a small scale against an Excel or Access database, I can just change the provider string in my Sub and use exactly the same SQL and VBA code to scale this up to a huge Oracle database.

In my presentation on relational databases at the SOA Annual Meeting in Orlando last year, I showed a video of how much faster a big chainsaw can cut a big black walnut log than an electric circular saw or a hand-held scroll saw can. In the beginning of the video though, I show that if you use the chainsaw the same way you use a handsaw, by rubbing it back and forth against the wood, it gives very disappointing results. My point here is that the bigger database tools, like Oracle and SQL Server, have a serious learning curve associated with their efficient usage. The actuary does not have to become a database administrator (DBA) in order to intelligently prototype a large database project. Excel, supplemented with VBA and ADO, provides a very convenient tool for trying out your ideas, and also gives you a scalable solution you can later turn over to your IT associates as a working set of specifications for the big tool implementation.

Happy prototyping!

SOCIETY OF ACTUARIES
475 N. Martingale Suite 600 • Schaumburg, IL • 60173 • www.soa.org