RECORD, Volume 30, No. 3[*]

Annual Meeting and Exhibit
New York, NY
October 24-27, 2004

Session 71 TS
Visual Basic for Applications – Black Belt in the Office

Track: Computer Science

Moderator: David L. Snell

Panelists: Charles S. Fuhrer
David L. Snell

Summary: Most courses in Excel, Word, PowerPoint or other Microsoft Office products stop just short of teaching you the highly productive things you can do with Visual Basic for Applications (VBA), the powerful language that comes free with your purchase of Microsoft Office. This session starts where the introductory classes finish and takes you to a new level of expertise — a black belt in programming the many products that incorporate VBA behind the scenes.

MR. DAVID L. SNELL: Charles Fuhrer is going to present some basic techniques using Excel and Visual Basic for Applications (VBA), including writing macros for that. After that, I'm going to show you some things you can do with Excel and give you insight into how you can do those. VBA can combine different office applications: Word and Excel together, Excel and Project together, PowerPoint and Word together. There are lots of different things you can do to get extra power.

Our first speaker is Mr. Fuhrer. He was an FSA in 1977. He works for the Siegel Company, which is a national benefit-consulting firm. He's the chief health actuary there and has been for six years. Before that, for 25 years, he was with four health companies. He's a Society meeting speaker, usually in the health actuarial research area. He's co-editor of the *Actuarial Research Clearing House*. He has been on many committees over the years. He's joined us on the computer science section. He previously was chair of the educational research section and the health section.

[*]Copyright © 2005, Society of Actuaries

He's written tax-sheltered annuity (TSA) papers. He won the Actuarial Foundation's practitioner's prize in 1988 for a method for calculating aggregate stop/loss premiums.

MR. CHARLES FUHRER: I taught myself to do Excel spreadsheets with VBA. This is a teaching session, but we're not going to go into the nuts and bolts of programming. Basic programming in VBA is pretty straightforward. It's not a whole lot different from whatever else you learned to program on. I want to give the flavor of what it can do for actuaries. I want to tell you some things that are important that you might not think of. I'm going to talk a little bit about some things that are unique to VBA and Excel, like how to move data into and out of spreadsheets, into the program and from the program, and a few more programming suggestions.

It's a tremendous advantage to actuaries to be able to use Excel with VBA. It allows you to write programs that do calculations that other people in your company couldn't possibly do on their own. It also allows you to create looping programs that would not fit into an Excel spreadsheet. It's very convenient because the people in your firm know Excel to some extent. And the Excel VBA program is carried along with the spreadsheet. The people you have to send this to get the spreadsheet and can invoke the VBA program and get results that pop up in their own spreadsheets. There's no need to send out separate executable files and no need for the input to be on separate formatted input files — which we would normally use for freestanding programs. There's no need to design output forms. The output appears right on the spreadsheet. They can then use that for further calculations within the spreadsheet. And finally, when they're all done, they can save the spreadsheet, and they'll have everything stored — both their input and output together. It becomes a very good tool, particularly when you're the actuary working for an insurance company or a consulting firm and you need to get calculations out to other people who aren't actuaries who won't understand them, perhaps.

Some of the other advantages to the VBA program include looping calculations. Another thing is speed. Generally, the VBA program will execute quicker than putting formulas into cells. Size is another advantage. When I first started doing stuff in Excel, I didn't realize how to take advantage of the VBA program to cut down on the size of the spreadsheet. So I put a lot of formulas into the spreadsheet. In fact, the original demo sheet was 15 megabytes, which isn't all that bad, except that it causes the document retrieving and storing system on our network to time out. You have the capability of putting your VBA program formula in and have the program copy it (just like you would if you were in a spreadsheet), and then change the formulas back to values. It's a simple device to do what amounts to calculations within Excel, but you don't have to pay the price of having a large number of formulas in a relatively large spreadsheet.

Another thing you can do with VBA is change the format according to the input. Excel allows you to do that with conditional formatting. But the opportunities to do it within VBA are much more extensive. You can change the size of input areas. You could change what they look like in terms of instructions on the sheet. There are all kinds of things you can do. You can be as creative as you want to be. The normal Excel situation gives you much less power.

Let me mention that Excel has a lot of power within it. There are a couple of things you can do within Excel that are really foolish to do in VBA. One thing is sum calculations, such as matrix inversion. I would like to meet the person who programmed that for Microsoft because it is faster than I think is possible. It can do 10 x 10 matrices and get the inverse within fractions of a second. And considering the number of floating point operations that are done, it's incredible. You don't want to write a program to do that. You want to take advantage of the built-in capabilities.

There are a lot of conditional formats you can put in. There's also data validation within Excel. So you can specify the types of values the user can put in. You can do a lot within the spreadsheet itself, which is not to say you shouldn't do both. They certainly can complement and supplement each other.

I have a demo spreadsheet that I've done some work on. It is a calculation I did for a company. It calculates claim reserves. For the people who are not familiar with health insurance, you get a triangle of claims by incurred and paid month, and you use that to establish a pattern of payments. It's a relatively simple calculation and didn't really need the macro. But this was a case where the size came into consideration. The triangle might be hundreds of entries by hundreds of entries. And if you put all of the calculations into the program, the size got huge.

I just want to talk about some of the things you could do. For example, on one sheet, the user could put the year he wants the input to be. And then, if he switches over to the claim-input tab, he'll see that it starts at the same month we put in. It also could format the cells in yellow for input that is legitimate, so you couldn't have claims paid before they were incurred. Then he could go back and change it. It's instantly changed. When he wants to do the calculations, there is a simple button here that says, "Click to calculate all." When that's clicked, it's changed.

In the project platform, you can do programming. It's interesting that Excel gives you pre-programmed areas, where you can put results from spreadsheet data. For the worksheet, you get a whole bunch of events that occur. And when those events occur in that sheet and the sheet is activated (before a double click, before a right click, before you hit calculate or when you hit calculate), if you make any change to the sheet (if you deactivate it or if you make a selection change on the sheet) you can put checks in place. My worksheet says it changed. Then I check to see if I changed the particular cell that had the dates in it. If so, I change the format of the other page where the input is entered.

One of the things it's important to examine is object-oriented programming. You can create your own objects by inserting class modules. It's not quite as variable or extensive of an object-maker as you would have in C++, but it's pretty good. However, what's more important is that they give you a lot of pre-established objects, which you can use to do your programming. There is an object browser. This is gives all of the pre-programmed classes and remembers those classes in Excel VBA. Many of these are useful for your programming. So you can refer to it. If you click on it, it tells you a little something about the properties of these classes.

I want to talk a little bit about programming. First of all, it's extremely useful to do structured programming so other people can check and you don't use "go-tos." If anything needs to be executed, it should be executed as a function or a sub-routine. We recommend name conventions. You use lowercase prefixes to indicate what kind of variables they are. You have to put the option explicitly into Excel at the top of each module, otherwise Excel "thinks" a new name is a variable and you just haven't bothered defining it. Of course, misspelling something is a way to make errors. I strongly recommend using the words "option explicit." You just put those words in at the top of the module.

Let's talk a little bit about moving data into and out of the spreadsheets. One of the most useful objects is the range object. The range object allows you to address any spreadsheet range. The syntax for that is "range," and then, in parentheses, you put the identifier for that range. You can either use normal Excel ranges — say A1:B2 — which would give you the cells in that range. You put those in quote marks. However, I strongly recommend you use named ranges. You create named ranges in Excel by entering them in a little drop-down box, or you can go to the name editor. The reason for doing that is if you put addresses into the VBA program, they won't get updated if you later move that range somewhere else or add rows or columns. That happens automatically with your relative addresses. In fact, that happens if you move even the fixed addresses. If you put a range name in when you add the rows, it automatically will keep track of where that named row is and your program will continue to function.

The other thing that I want to mention is there is a relatively easy and quick way to move data out of a spreadsheet and into the program. If you assign a range to a variant variable (a variant is a default variable; it can be anything) it automatically puts that in as a 2 x 2 array. You can move a whole spreadsheet into the program. Everything is in there in a matter of seconds. It's very quick. If you step through the array for some reason, Excel VBA's execution tends to be very slow. So that's a device worth mentioning.

These are six books I purchased when learning VBA. "Excel 2000 VBA Programmer's Reference" is a good reference. It doesn't have a lot of text to get you started. "Using Excel Visual Basic for Applications" is by far the best, in my opinion. It has tremendous reference, plus it's got excellent instructions. I strongly recommend it. You also might want to look into "Microsoft Excel 97 Developer's

Handbook, VB & VBA in a Nutshell, Writing Excel Macros and Microsoft Excel 2000 Power Programming with VBA." Computer books tend to be expensive. Although, if they're a prior year's edition, sometimes you can get them at a discount.

That concludes my portion. Mr. Snell is the 2004 chairman of the computer science section. He's called a "technology evangelist" for the RGA Reinsurance Co. He has a lot of fun designing customized solutions for RGA's Asia-Pacific operation. He was the chief architect and the co-inventor of RGA's multilingual underwriting expert system, Aura, which is in daily use in English, French, Spanish, Chinese and Korean. He's a frequent speaker at actuarial and computer-science events. In past Society of Actuary annual meetings, he has presented about relational databases automated underwriting, internationalization issues for application of software development, the rapid-application development approach and prototyping in science and in art.

MR. SNELL: Some presenters like to use sports analogies. I'm not very good at sports analogies, so I like to use tools analogies. When I talk about useful ways to become more productive in the office, Excel and Word and the other programs are flexible tools that are easy to learn (not necessarily to master). I liken them to a jigsaw. Most of us never master the creative sculpting capability of a jigsaw. But just about everybody knows how to push the "on" button and saw through a board. It's not a very difficult procedure. In a similar manner, I'm constantly amazed by the richness of Excel and Word features. Relative to some folks, I'm an expert on the usage of these tools, but I feel like I've merely scratched the surface of the many capabilities available.

Somewhere in one of the many VBA books I've read, an author made a profound statement that the best code, or at least the fastest code, is no code. There are some exceptions to that statement. But it's ironic that once a person learns to program macros, every Excel task starts to become a macro task. It's kind of a variant of the old adage that if the only tool you have is a hammer, every problem starts to look like a nail.

A very simple example of what I mean is the data-validation feature. I used to write VBA code to restrict input to certain values. I would create a list box of available values. The person running my macro would not have to type in (and possibly incorrectly type) the input parameters. I discovered the data-validation feature, which is really neat and built-in. You can take any particular cell and make it a list item. I didn't have to program it to do that. And frankly, it's a lot better than some of the ones that I did program to do that. If I were to type in "Jane" instead of "Dave," it would automatically tell me that that's not allowable and switch it back to "Dave." Likewise, there's an advanced filtering capability. I can say I want to filter a list with my list range and other criteria. For example, issue age is greater than 24 and death benefits will be less than whatever that comes out to be. We could even use unique records only. A lot of times, you want to find how many unique records or rows you have on a spreadsheet. And it's as easy as using the check box.

Sometimes, you need to access more automation features or to harness the power of associated applications like PowerPoint or Word or Project. Every actuary knows Excel has a 65,536-row limitation. And you know that because that's exactly 2^{16}, and that's been a limitation that's been an annoyance to us for a long time. Yet, we often have databases with hundreds of thousands or even millions of rows. Later in this session, I'll show you how you can have Access or Oracle store those larger databases for you. Still, you'll be able to use Excel features with them. I'll even show you a trick I've developed to squeeze all of that data into an Excel workbook, so you won't even need the original data source afterward.

My tool analogy is a circular saw. A few years ago, an actuary friend of mine helped me to build a 20-by-40 foot deck on the back of our house. And when you have to make a lot of straight cuts like that, you don't want to use a jigsaw. It's just not consistent. It's not fast enough. However, if you look closely at my deck, you'll be able to spot a couple of places where I made a mistake and cut further than I meant to. The circular saw is not quite as easy to use as the jigsaw. Access seems to have a higher learning curve for people compared to spreadsheets. PowerPoint and VBA also require some extra lessons before you get to a comfort level with them.

Lots of administrative assistants in your offices are proficient with many of the individual tools of the Office Suite. Very few, however, take the next step to write code that helps to automate them and coordinate their capabilities. This is where you encounter the chainsaw in my cutting-tool analogy.

There's a learning curve associated with VBA. If you get by the learning curve, you can do a lot of great things. But don't expect it to be as intuitive as a spreadsheet when you first get into it. When you're dealing with VBA, we talk about things like "subs" and "functions" and "classes" and application programming interface (API) calls and OADB and open database connectivity (ODBC) and active data object (ADO) and .NET, etc. Let's quickly try to demystify a few of these.

Once you're in Excel and you want to get into VBA, there's a shortcut key for that. You hold down the "Alt" key and press "F11." That automatically gets you in there. The first thing I might pull up is a macro that the recorder made when I did data validation. We have a sub that made the macro. It said I recorded it on Oct. 25. We could go to the cell "V2." And with that selection, we're using the validation. "End with" will allow you to do lots of properties without having to do extra typing. We added the type of validate list. This shows where the list is, tells you when to drop down, where to show up, etc. If there's an error, it will show the input, too. And when we're all done, we select the range.

Basically, with a sub, all that is required is a sub, a name and an end sub. We didn't really have to put anything in there to make it a valid sub. It's nice if you do, but that's all that's necessary. Once you do that with the macro recorder, you could also rename your sub. I renamed the second one as “advanced filter.” After that, we have the range, .select. We have what we were going to do and what the criterion range was. If we look at the macro recorder that did all of the work on a larger scale, the macro recorder will work for you. But it creates a lot of “junky” code, and it’s very specific code. It’s not what you want to end up with. But for me, it’s a great way to start out. If I start out with a macro recorder, I can look at that and say that’s OK. I can do so much better than that. But it’s nice to have that as a starting point, especially when you’re using something like PowerPoint, etc. When you’re not familiar with the object model, the recorder can be very handy.

The next thing I mentioned was classes. Classes are subsets that use something called “encapsulation.” It’s fancy jargon. It means what you’re doing is combining both programming code or method and data. You’re putting them together so you’re protecting the data using those methods. If you have a sub that you’re using over and over again (and in lots of different spreadsheets and you’re copying that over), sooner or later, you’re going to run into the problem of using the same variable name in that program as you did in this sub or in the transfer. And it’s going to mess things up. So if you’re going to reuse data a lot, a class is a good way to go.

API calls are calls to the operating system. They’re application program interface calls. Those are really neat because you can find out the exact machine name of the PC that’s running something or the user name on a network. With that in your program, you can determine what level of privileges a person should have. APIs are great for that, but I have to give you a caution that they’re going out of fashion. It used to be that this was the mark of honor. The badge of a real geek was if you knew there were thousands of API calls, and you can do all kinds of things with them. But you also can get into trouble because they take you out of the safety zone that VBA normally provides for you. It’s like a big sandbox. And you can’t do terrible things to the operating system from VBA, unless you use something like API calls. And then, you can get into such trouble that it’s time to reformat the hard drive. That is generally not optimal.

The meaning of OADB and ODBC acronyms changes over the years, so it’s difficult to say what they stand for. Object-link embedding was a methodology that Microsoft used to transfer things from one program to another. When they had data that was being transferred, they would use object-link embedding for databases and other database connections with ODBC. The only important thing about OADB and ODBC is that when they’re set up right by your network administrator, they work. You don’t really have to understand how they work; you just have to spell them correctly when they give you the screen for the connection. And if you do that, and the program still doesn’t work, then you need to know who the network person is so you can tell him that it didn’t work and ask him to load the right ODBC driver.

ADO allows you to separate the database from your program logic. There are certain generic things you do with the database. You might want to add a record, delete a record. You might want to update a record. And those are fairly straightforward. You can write code for those. But in the code, you don't want to have to tell it that if you are using an Oracle database, you've got to do an add or an update a certain way, as opposed to the way it is done with a Sequel Server or Access or Excel. That's the type of information you don't want to have in your program. And ADO allows you to separate the logic of your program from the mechanics of what kind of database it is and how to get at the data there. On some of our projects, we start very small. We might use an Access database to start with. And when we want to scale it up to Sequel Server or Oracle, all we have to do is change the connection string on ADO. And all of the rest of the code basically works. .NET is coming along. Some of you might be using it already. Eventually, .NET will replace VBA. As nice as VBA is, it has lots of shortcomings, too. Eventually, you'll have to learn a new "chainsaw."

There are lots of built-in tools to get you started with VBA for Excel, Word, PowerPoint and Project, and others. The first one is the macro recorder. Getting into the macro recorder is very simple. If you use your Visual Basic toolbar (which is one of the toolbars available in Excel), there is a button that starts you on the macro recorder. You can use that as a starting point for your own code. There are some Office and other programs that use VBA but don't yet have the macro-recorder capability. And for those, I have to use the object browser. There's also something called Intellisense. As you start to type, the operating system or the VBA environment — the integrated development environment (IDE) — is trying to figure out what you are trying to do. You work with a range or with an application. The IDE will give all the options, to prevent misspelling. You just get cues from the drop-down menu.

When you start to price a new type of product or do an actuarial evaluation that has to comply with the accounting rules in another country or many other actuarials for the first time, you normally do some research to get up to speed.

I will use the example of a photo database that I sometimes create. When I'm doing that, I make what I call "thumbnails." This will allow me to put anywhere from one to eight photos across on a page. The program will ask me how many pictures I want to import. It will go out to the folder that I'm in at the time and ask me to cue. I'll tell it what folder to use. I can tell it which ones I want to actually print out. It's a very easy code. To get in, I'll use that "Alt,""F11" key and then I'll try to find the modules. I'll click "load picture from folder." I used the macro recorder for that, so I didn't have to know any of that information. Then I convert inches to points because I don't think in terms of a 72nd of an inch. I programmed it to ask how many pictures I want across each row with the maximum number of pictures. Then I had to do some empirical programming to get the right scale factors to make it look decent. Trial and error is not such a bad technique on low volume. The key for this thing is selection in line shapes and add picture. This "add picture" feature is the method by which it grabs each one of the pictures that I have in the directory and puts them into a table. Then I multiply by the scale factor. And when I'm all done, I tend to put in a message box telling me that I'm done. Because on a longer macro, I might not know whether I'm hung at that point or if something good actually happened.

When you do your macros, it's good to have some sort of status as you're going along, especially if it's a longer one. I wrote a program for our general agents one time that took about 30 seconds. I had optimized the program and it did lots of calculations. But it took 30 seconds to make a long illustration, and they were complaining. They have an attention span of about three seconds, and this was 10 times that. Just on a whim, I took out the optimized part that turned off the screen refresh. So now, as it was going along, it typed at blitzing speed across the screen. It took 45 seconds instead of 30 seconds, but they loved it. Anyway, keep the user in mind. There's a book I bought titled "Programming as if People Mattered." The book itself wasn't that great, but I bought it just for the cover.

You can do a PowerPoint version of the same thing. It will do the same thing. It can go out to the same directory and create a slide show from the pictures that I have in the folder. Then I use a "randomize" on the transition.

You could go back into the code and show the pictures. I used code that tells it that some of the pictures might be .jpg, some might be .bmps, some might be .tiffs, etc. I want to grab any of those and put them in there. As we go through, we're using that slide range, shapes, add pictures, etc.

Where do you find which words to use when programming? I try to take the easiest route possible. There are too many things I have memorized over the years. And this is not my favorite thing. I go into anything that has a macro recorder and try to tell it how I would add a picture into a slide. Using PowerPoint, I turn on the macro recorder. I go into "insert." That gives me "picture." I bring that in. Then I look at the code I generated. I clean up that code a little bit. From that point on, I don't have to worry about the syntax. That's where I usually get it. If it doesn't have a macro recorder, then I have to use the object browser. If I can't figure it out from the object browser, then I'll get onto MSN or Google. It gives you a whole bunch of help files. I put in some arcane thing like, "How do you insert a picture programmatically into PowerPoint using VBA."

Those were all single-application examples. A lot of people probably use Microsoft Project or are subjected to Microsoft Project on big, multi-user tasks. And if you're the one who has to update the project, and then you have to distribute it to a lot of people, there's a common complaint that you always get: "Why can't I get this in Excel? Because I don't have Project on my machine or I don't have that version?" Then, you export it to Excel, which you can do from Project. But they lose the indentation, for example, because the export wasn't all that great. I use Excel to grab those details from Project and do the indentation for me. In order to do that, I first go to “tools/references” and put in a reference to the Microsoft Project — in this case 10.0 Object Library. It wouldn’t have worked if I didn’t do that. The reason it wouldn’t have worked is in my actual code, I’m defining things as resources or tasks. And Excel doesn’t know what a resource is. It doesn’t know what a task is. But if I put in that object module as part of the references on tools, then suddenly Excel gets so much “smarter” about everything to do with Microsoft Project. And I can put those in and cycle through the task for the specific people. I can even go into the task-outline level and do the indentation. As the result, they were indented accordingly.

I tend to make a lot of presentations and usually don’t have a lot of time to prepare. So I’ll scratch out some notes on PowerPoint regarding things that I want to talk about as I’m preparing the slides. Afterward, if I were to print out the slides with notes, I hate those. It means I have to spend a whole page on notes. And if I’ve got 50 slides, I’ve got this stack of paper that I drop halfway through. It just doesn’t work out for me. And the print is too small. I wrote a macro in Word to go into my PowerPoint presentation and create something I like better. I can follow along; it’s got big print. And I’ve got thumbnails of the slides. As I go through, I will look over every so often to see if I’ve missed something really big.

I also can define a variable as a PowerPoint application. Now, I couldn’t do that in Excel if I didn’t first go to tools/references and add Microsoft PowerPoint to the object library. Now that I’ve added that, it knows about PowerPoint objects. And now that we know about the objects, we can do the same kind of thing with page setup. In the “slide sorter” view, I copy one slide at a time and adjust the height and the width of those. I grab all of the text, stick it into my table and loop through until there aren’t any more slides. It takes about two or three minutes to run the thing. But that’s a lot faster than me copying over those notes or trying to use the PowerPoint output, which I just don’t like that much.

I study Mandarin Chinese. It’s important in Chinese to know the difference between the traditional character and the simplified character, and the stroke order. Because when you make a character, you’re supposed to do these strokes in a very specific order. If you don’t know the stroke order, it makes it really difficult to look up something in a dictionary, unless you happen to know the character ahead of time. And then, if you knew the character ahead of time, why would you have to look it up? Anyway, I wrote a Word routine that went out to Access. I store my characters in Access, and it has double-byte capability, which means it can store the really funny characters beyond “ABC” and our number set. But then, Access doesn’t have a great formatting capability. I wanted very precise formatting because I was trying to squeeze a whole lot of stuff onto these flash cards I make. And Word gave me that capability.

Early last year, I was in our Tokyo office. While I was there, the office staff asked me for some help. Their underwriting support staff had to stay until midnight each night to get the work out. Since we are a U.S.-based company, our basic data was on an Excel spreadsheet in English. And in the United States, we tend to go "month, day, year," which is really unusual for the rest of the world. The rest of the world usually uses "day, month, year" or "year, month, day." In Tokyo it's "year, month, day." But they give the year, and then there's a year symbol, and then there's the month, and then month symbol, etc. When doing that conversion and translating into Japanese and doing some of the calculations and translating that into Japanese, it became time-consuming. They were very good at this because they spoke Japanese and English. But it would take them about five minutes to make these custom quotes. They would do more than 100 per day. They would have to stay until midnight to do these things. Midnight was kind of the "pumpkin time," because that was when the last subway left that particular station. And you really didn't want to be on the last subway.

I looked at that problem and played around with combinations. First, I wrote a macro to find out all of the combinations and phrases they had in their Japanese templates for the various companies. Then I made a pseudo-dictionary, a phrase dictionary from English to Japanese and Japanese to English. I then did the conversion on the dates so all they had to do is suffer through my security notices. They would go into one of these companies and double-click on any particular person, then press a button. It would give them our customized quote. It took about two or three seconds. This saved them about 500 minutes a day. They can go home now at 10 p.m., and they're thrilled.

Now I will talk about using Excel with data from Excel, Access and Oracle. I wrote an article on that for Compaq for the *Computer Science Newsletter*. I showed that Excel can be used as a source for a relational database. It is a souped-up version of a spreadsheet, basically. I use a cool-data validation program. I chose whether I want to get my data from Access, Excel or Oracle. Then I choose a particular sequel that I set up for Access. I just double-click on that and my macro will do the work.

I use Intellisense. In the immediate mode, I type in "application." It tells me some of the choices. I might start to type "cursor" because I hate to see that hourglass. I might choose "equals." And then it gives me the choices of what I can use. I go back and I get my regular cursor again. I get out of Access mode and go to Excel. On Excel, there's a source where I've got a table called "Applicants" and another one called "Aps." I could join these two tables. I want to get all of the rows from "applicant" and "aps" for which "applicant ID" is equal to the "ap ID" and the age is greater than 35 and the face amount is greater than 200,000.

You can do more comprehensive things. I was using active data objects. This allows you to have hundreds of thousands of records in Access. But you can use millions of records in Oracle and Sequel Server. Using one of those ODBC or OADB connections, you can use ADO to grab information from large databases.

I assume just about everybody has used pivot tables. If you've got Excel 2000 or later, you also have an option of pivot charts. In my mind, the spreadsheets were a dramatic leap of an order of magnitude of power that you had over data compared to the old ledgers. And the Excel pivot charts capability is that next leap beyond that. Once you start using these, you're bound to be hooked. I have a spreadsheet. It's called "Retro.xls." It's 65k. That's a small spreadsheet.

It is a chart I have created. But it's not just any old chart, it's a pivot chart. If I save it, I'm saving a huge Excel file. This Excel file doesn't have 65,536 rows in it. It has all the data from more than 500,000 rows from that original database. It's so big that we can't show it on any one of the spreadsheets.

It's up to 130 megabytes, which is a big spreadsheet, even for actuaries. It's basically a chart. This was created for retrocession for a piece of business for an office. And I changed some of the data so I wouldn't be using private information. I used pool years. I'm going to drag the benefit category right off of this chart. Everything is reflected in that now. Let's say we actually want this to be sorted by gender. I'll bring gender over. Then we have our breakdown by gender by the same information by pool year. But we've got two pool years. How often does it happen that one is wrong when you're looking at half a million lines of data? Anyway, with a chart like this, anomalies jump right out at you. Because now I've been plotting by pool year, and I have some really weird pool years. You use a pivot chart that is exactly synchronized with everything you do with that chart is a pivot table. And in that pivot table, I can see the year 2020 has maybe 5 million of a sequence number. The sequence number shows you how many rows we had. I could bring in something more reasonable, like the reinsurance space amount. We could sort the reinsurance space amount by year in that pool. Now it's 800,000 in year 2020. I can double-click on that and get down to the actual records. I can't fit those records on a spreadsheet, but I can drill down to them and actually show them. For the 999, I could do the same thing. I'll go to 1,546 and drill down. And again, this record is for a single individual. There was an entry error. The entry error probably occurred years ago. I can tell from the entry field. It spots exactly what the anomaly is.

FROM THE FLOOR: Could you edit those records like that?

MR. SNELL: No, I'm disconnected at this point from the actual database. But I can turn to whoever has responsibility for editing the data and give that information over. And it's good that I can't edit here, because you want some measure of integrity in your database. Actuaries tend to mess things up on that.

I'm going to get rid of those anomalies on the chart because they're embarrassing. Now, you have a nice, clean data situation. Say we didn't want the sum of the face amount in a certain cell; we wanted the average. The chart changes right away. We had some weird stuff going on in 2002, probably some automatic block. But let me get back to sum again because it looks more reasonable. I'm going to pull off "pool year" entirely and focus on our total budget under code. Let's bring an underwriter rating. There are so many underwriter ratings that are obscuring my view. I can double-click on a button and tell the program that I just want to show the influence of the top 10. If you didn't want the results in the form that comes up, you can separate those and move them. The 100-percent rating might be skewing our results. I really want to see what the substandard stuff looks like. So let's take the 100 out of the equation, and now we see what the others really look like. You also can create lots of different graphs from that information.

Anyway, pivot charts are not part of VBA macros by themselves, but I programmed one recently with a VBA macro. It took me until 5 a.m. But after that, I ended up with a small module. Why did that take so long? It was only about a page of code. Basically, I defined some stuff as activated data object database (ADO DB) connection. Another is a record set. Another is a pivot cache. Another is a pivot table and a chart. Before I could use this ADO DB, I had to go to tools/references and put in the active data object library. You usually have a bunch of those to choose from. I usually choose something that's not the most recent. Because if I'm sending it to somebody else, he might not have that version. I want something that's in this decade. So 2.5 is usually a decent choice. I've got the Microsoft Office Object Library, all the automation. Most of these objects get put in for you. If we wanted to combine objects with PowerPoint, we could stick that in there. But it's not terribly long to follow through. I open a connection. Here my provider is Access, so we're using the Microsoft Jet OADB provider. I open that record set using my Sequel Server call. And my Sequel will grab everything from the table. It's not very exotic. I set a pivot cache. That pivot cache is this cool thing that is inside the Excel Workbook. It's so big that we can't put it into the spreadsheet. If, back on the Excel sheet, I pick something really large and I double-clicked on that, it probably would come back with some kind of error message saying I wasn't specific enough. There are more than 65,536 rows in that particular thing. We can make it very granular just by dragging and dropping more objects into that. I can break it up by gender. I can break it up by fund type within gender — whether it's ordinary or super granulation or pension. I can get treaty codes. It is a tremendous tool with tremendous capability. And you can program it with VBA and make it turnkey. Typically, when I start showing actuaries this feature, they are impressed. They want to see things sorted by issue agent or major benefit, for example.

FROM THE FLOOR: You mentioned "option-explicit" and "naming conventions." Would you also agree that it's a good idea to declare the variable by type, excise integer, and so on? Otherwise, everything is going to be a variant, which you probably don't want.

MR. FUHRER: Yes, of course. And, in fact, when you put "option explicit" in, you have to declare it by type.
FROM THE FLOOR: A question on the technique of setting a variable equal to a range. It's an object that you use, but it's not two-dimensional …

MR. SNELL: Yes, you can actually define that as a range, that’s a data type. If you use variables, it also will work. But a range will be a little bit more efficient.

FROM THE FLOOR: Two other things that I would like to mention: The terminal-emulator software that we use for our Legacy mainframe follows the Microsoft object convention. As a result, you can use Excel VBA or any other applications of VBA for that. And it’s really great for interfacing with the mainframe. Another thing that wasn’t mentioned but I found useful is using Windows scripting, you can get the objects that Excel or whatever software is on your machine provides, and use that to access the object directly without even necessarily running Excel visibly on the screen.

MR. SNELL: That’s true, Window scripting; it’s a stripped-down version of VB. It’s not as powerful as regular VB, but it has a lot of good features, and it’s getting closer and closer to VB. There are also lots of other applications that have VBA built in that are non-Microsoft applications. Corel, with its wonderful graphics suite, now has VBA built in, plus they license that for Microsoft.

FROM THE FLOOR: But you can get the objects available with whatever application you want using VBA. You have to know what the name of the object is. And I’m not sure how easy that is to find. But in the case of Excel, for instance, it’s Excel.sheet, and then you’ve got all of the VBA that goes along with Excel right there to use with your Windows scripting.