“We Want To Go To SQL”

Whil Hentzen

Never before in the history of Fox has there been such a disparity in the sophistication of deployed applications. While there are plenty of extremely high end systems in place now, built on complex frameworks, processing data over large networks, it's still not unusual to find companies running their operations with a folder full of FXPs that consist of @SAY/@GET commands and the occasional READ thrown in there for good measure.

Thus, while the average reader of this magazine has been using SQL commands and accessing SQL back ends for two decades, that's not a universal situation.

I get a call probably once a month with these specific words: "We want to go to SQL." This isn't as clear cut a request as it seems, though, and is full of potential pitfalls.

The purpose of this article is to help you address, strategically and technically, their request to 'go to SQL', and to help you help them open their pocketbooks.

Nearly 20 years ago I merged a dozen in-house Fox 1.x and 2.x systems with three MS-SQL applications, porting all of the Fox data into a fourth, for a small multinational that ran all of their sales, orders, inventory, invoices, marketing and collections operations. If you count back on your fingers, yes, you've got it right, this was a Y2K project as well, so it won't shock you when I tell you that there were a couple of late nights on that gig. We shipped on time, and in fact, they were able to take their holiday shutdown as scheduled.

I've been dealing with projects like that (sans the Y2K requirements) ever since.

Unfortunately, that project was somewhat of an anomaly. They knew they were facing a rewrite, and had prepared for it. Most folks people expect a magic bullet (as they do with everything) that will take their app, crafted over decades, replete with all sorts of custom constructs, and move to a SQL backend over a weekend, because they heard a MSFT presentation 20 years ago promising that by using VFP's new-fangled “Views", you could "flip a switch and 'voila, SQL!'" Sure, they'll admit that they didn't really quite understand the 'views' part, but still, you can do that, right?

They're uniformly disappointed to find this is not true. And disappointed people don't spend money (with you) as readily as optimistic people.

What does 'go to SQL' mean?

Before we get into the nitty gritty, let's take a step back and look at the bigger picture.

We're pretty smart folks, that's why we're developing software instead of digging ditches or making sure the french fry machine is clean. But this also means that we have a tendency to jump ahead in the conversation, to move past the mundane parts and go straight to the interesting stuff. So when someone says, "We want to go to SQL", well, we're liable to assume that they mean "A SQL database backend."

But this isn't necessarily the case.

I've run into more than one situation where their desire to 'go to SQL' actually meant that they simply wanted to convert their XBASE-style data access to SQL commands, replacing

```
append blank
replace name with m.name
```

or

```
append blank
gather memo memvar
```

with

```
insert into MyTable (name) values (m.name)
```

Of course, this process is part of the the larger process of moving from DBFs to a SQL backend, but again, let's just make sure.

So, just what do they mean when they say "go to SQL"?

You've likely run into folks whose grasp of technology is a little suspect. They've been entrenched in the @SAY/@GET world for so long that just what “SQL” is isn't exactly clear. "Is it a

floor wax or a dessert topping? It's both!" They just know they want “SQL”, because all the other cool kids on the golf course are using it, and they don't want to be left out, not now in 2016.

This statement always reminds of the cartoon where the PHB tells Dilbert "We should build an SQL database." Dilbert thinks, "Does he understand what he said or was it something he saw in a magazine?" So he asks, "What color do you want that datbase?" and the PHB, in his all-knowing manner, answers, "I think mauve has the most RAM."

Do they understand what they're saying?

First, in email, it's tough to decipher. Are they thinking, "Ess-Que-Elle" or "SEE-kwell"? I know, technically 'SQL' can be pronounced either way, and thus their intent can't be decisively determined by how they pronounce it. Yet, for some reason, I've generally heard the first pronunciation, S-Q-L, used when referring to the data manipulation language, while 'sequel' is more often used as shorthand for a database product (most likely Microsoft SQL Server, but possibly a competitor, such as MySQL or PostgreSQL.)

Interestingly enough, peers of mine have run into the exact opposite interpretations. Go figure!

In the first situation, I'd be inclined to think they mean converting their code to use SQL constructs such as SELECT, INSERT UPDATE and DELETE. Obviously, then, the second situation is that they have in mind the conversion of their data from DBFs to another backend, along with the necessary code modifications.

Given these two meanings, your next job is to determine which they mean. This can be a challenge, they may not even understand themselves that there is a difference.

Thus, your very first task is ask is to confirm what they mean. Difficulty arises when they don't know what they mean, when they're basically parroting terms they read about in a magazine (yes, it really does happen) or heard from one of their buddies. "Yeah, we moved to SQL this spring." (Puffs out chest a bit.) "It's so much more robust, and tolerance to faults has gone up substantially."

I've had conversations with several potential customers who, even after repeated probing, were unable to specify whether they meant a new backend or simply rearchitecting their existing code. Kind of like overhearing a couple of teens talk about sex, if they haven't actually done it themselves, they're simply repeating buzzwords from an ethereal realm.

How to determine what they mean?

Since changing to a SQL-based backend includes rearchitecting the code to use SQL instead of record based logic, addressing the issues specific to the backend acquisition is the way to determine which they mean. Direct technical questions, however, may not do you any good - asking *which* database they are thinking of may result in a blank stare and a response of "I just told you – SQL!"

Asking about their budget for licenses, for example, is a great way to find out if they understand what they're talking about. If they give you the deer in the headlights look, they probably don't have a database product in mind.

Another key question to ask is who will act as their DBA is another tactic to unearth information about their intentions and goals. Folks who are purposefully and knowledgeably going to a backend database will often have an idea of how it's going to be managed.

Why is this difficult?

Regardless of their intent, they may well be under the impression that this is simply a matter of exchanging one set of commands for another, as described earlier.

Not so much.

It's a huge job, fraught with peril. There is no single cookbook to deal with constructs like this:

```
append blank
gather memvar a,b,c,d,e,f
copy to temp2
scatter memv
append blank
copy to TEMP3
do while !(expression)
  something
  else
  something
enddo
select TEMP1
gather memv c,d,e
skip
```

and Fox 1.x and 2.x (and a lot of VFP apps) are based on code like this.

Even worse, "back in the day", nobody ever heard of n-tier, so the idea of segregating data processing logic from the user interface was never done. Never. So we had code that looked like this:

```
append blank
gather memvar a,b,c,d,e,f
a=a+b
@say a
copy to temp2
scatter memv
b=b+c
@say b
append blank
copy to TEMP3
do while !(expression)
```

```
  something
  else
  something
enddo
select TEMP1
gather memv c,d,e
c=c+d
@say c
skip
```

And... notice the complete lack of comments in that code. Look familiar? Right!

So they better have a really good reason for doing so.

What are their reasons?

Now that we have this cleared up, realize that one is a subset of the other. To go to a backend, you'll need to cleave the intermingled UI and data access first, and then, once you have data access converted to SQL commands, incorporate those those commands into backend connections.

This is a lot of work. A LOT of work, as systems that were working just fine are now torn apart, with parts all asunder. There better be good reasons for all this work and risk. And the peril. Did I mention the peril?

To be sure, there **are** a number of very good reasons to undertake this chore.

The first that comes to mind is the size limit of Visual FoxPro tables. A single file (DBF, FPT, or CDX) can't be larger than 2 gigabytes, due to internal architecture of VFP. Database servers have limits that boggle the mind, perhaps even more than those 2 GB limits boggled twenty years ago.

So if their system is running into problems with size limits, or they've had to implement one workaround after another to avoid said problems, an investment in moving to a new backend could be very well worth it.

A second reason oft cited is security. VFP tables are simply files on disk. If you've got an ODBC driver, or the appropriate VFP-enabled application (like Word or Excel), you can open up one of those VFP tables and cause all sorts of mayhem. Worse, anyone else can too.

A related reason is the reliability of the VFP data structure. While VFP goes to great lengths to protect the structure of the DBF file system, a table (or its related index and/or memo files) can be corrupted, causing data loss and much anguish. Database servers, because of the way they're built and maintained, are inherently more reliable. Not perfect, mind you, but a properly maintained SQL backend will never generate the dreaded "Not a table" error message ever again.

Just because these are good reasons for someone doesn't mean that they're good reasons **for them**. Don't assume, ask *them* why. And then, what's the effort worth to them?

Preparing for the carnage

As mentioned earlier, there is no cookbook for converting 1/2.x code. Between the intermingling of UI and data access, and the lack of direct mapping between xbase and SQL constructs, there are simply too many variables.

That said, a potential customer isn't going to be happy with "We have no idea" when they ask how much it's going to cost to do the conversion.

Regardless of whether the process entails "simply" converting to SELECTS and INSERTS, or rewriting every file interaction via SPT or views is a little irrelevant; they're close enough for our purposes now. In both cases, you're going to need to touch every instance in the xBASE collection of SKIP, GOTO, REPLACE and GATHER MEMVAR style commands. It can be a daunting task, the difficulty of which is not easily communicated to the people writing the check or allocating the hours.

To get a handle on the situation, and to educate your customer, I've found it useful to do a size and scope of the work involved. The next article will provide a quick rubric for doing so, plus a simple utility for digging up and organizing the necessary data. Let me emphasize that the purpose of this tool is a rough size and scope - not an exact analysis. The goal is to move from 'I have no idea' to an estimate within maybe a factor of 2 or 3.

Stay tuned!

Author Profile

Whil Hentzen is an independent software developer based in Milwaukee, Wisconsin (as opposed to Milwaukee, Minnesota, as many people think.) His writing has killed many trees over the years, but none since 2007. He has realized he really sort of misses it. You can reach him at whil@whilhentzen.com