Client/Server, .NET, and ODBC

Mike Gunderloy

In this issue's column, Smart Access eXTRA eNewsletter editor Mike Gunderloy tackles some of the questions he's gotten from readers over the past several years.

I've always been under the impression that when you place an Access MDB on a server and then run a query against a table, all the processing is still done at the client PC. In other words, if I query a 100-record table to get one record, all 100 records go across the LAN, and *then* the SQL is processed by Jet on my machine. This was always explained to me as a drawback of Access being a file-based database product.

I'm wondering if this is still true. If not, was this correct in the past? Can you give me a definitive answer? Does it matter whether DAO, ADO, or RDO is used?

Well, you're partly right: All of the processing does indeed happen on the client. But that doesn't mean that every record has to cross the LAN. It's easy enough to prove that the processing happens on the client: Put an Access database on a server that doesn't itself have Access installed, and you'll discover that you can still open it from a client copy of Access.

But that doesn't mean that the Jet engine has to be stupid about things. Look at it this way: As far as Jet is concerned, there isn't really much difference between an MDB on the client and an MDB on the server. To Jet's way of thinking, a server is just a big hard drive that's sort of slow at delivering information. And, because Jet has an intimate knowledge of the Access file format, it can use search strategies other than simply reading all the records in a table to find a particular record.

Consider a query like this one:

```
SELECT * FROM [Order Details] WHERE OrderID = 10704
```

With an index on the OrderID field, Jet doesn't have to read every row of the table to locate the desired records. Instead, it can read the OrderID index and use that index to locate the correct records in the file. Then Jet needs only read those pages from the file that contain the desired information. Of course, because Jet reads data a page at a time rather than a record at a time, Jet will probably read some extra data. But Jet certainly won't read the entire table to resolve a query such as this one unless there's no useful index or the table is small. With small tables, Jet may decide that it's faster to read the entire table than it would be to bother with reading the index and then reading the table.

Of course, this still doesn't make Jet as efficient as a true client/server database where network traffic is concerned. If you execute the exact same query against a SQL Server database, even less information will cross the wire. Access will send only the SQL statement (or the name of an existing view or stored procedure) to the server. The database server will then locate the appropriate records (probably by performing the same indexed search that I just described) and send only the resulting records back. Thus, the client/server version doesn't have the overhead of downloading the entire index or the other records that are located on the same data pages. It also avoids Jet's two accesses to the database file: once to read the index, once to fetch the data.

So is that why most experienced developers recommend moving the SQL Server for networked databases? Not really. With today's network speeds, it's not often that the download of a single index is going to make a difference that's visible to the end user. Normally, it's not the speed but the number of simultaneous users that dictates a move to a client/server database. This, too, is a consequence between the processing schemes used by a file/server database and a client/server database.

When you have five or 10 (or 50) clients sharing a single file/server database, you have that many programs actually modifying the shared file. The different copies of Jet don't talk to each other, so they have to communicate by locking sectors on the server, and they all have to do their job perfectly. Turn off a computer, or kick out a network cable, and you can leave things in an inconsistent state for everyone.

In contrast, it doesn't matter whether a client/server database has one client or 100, the disk file is only being read and modified by a single program: the database server manager. The database server manager can act as a traffic cop and gatekeeper for all of the clients. The manager has complete knowledge about the database file because it's the only application modifying the file. Turn off your computer and the server will notice and release any locks that it was holding on your behalf. The result is that client/server

databases are inherently more robust than multi-user file-server databases.

We're an ISV selling an application developed in Access 2002. We've got a fairly large investment in Access at this point, but we're hearing more and more about .NET. As we start designing our next version, should we be migrating to .NET or waiting for the next version of Access?

Without knowing a lot more about your application, that's really impossible to answer. But perhaps I can help you out by talking a bit about how .NET deals with Access databases.

First off, I'm going to assume that by .NET you mean the combination of the .NET Framework and Visual Studio .NET. Microsoft appears poised to slap the .NET label on just about everything they put out now, whether or not it has anything to do with the new and elegant software design embodied in the .NET Framework. Don't assume, for example, that something labeled Office .NET (if such a thing exists) has anything more than passing similarities to the real .NET environment.

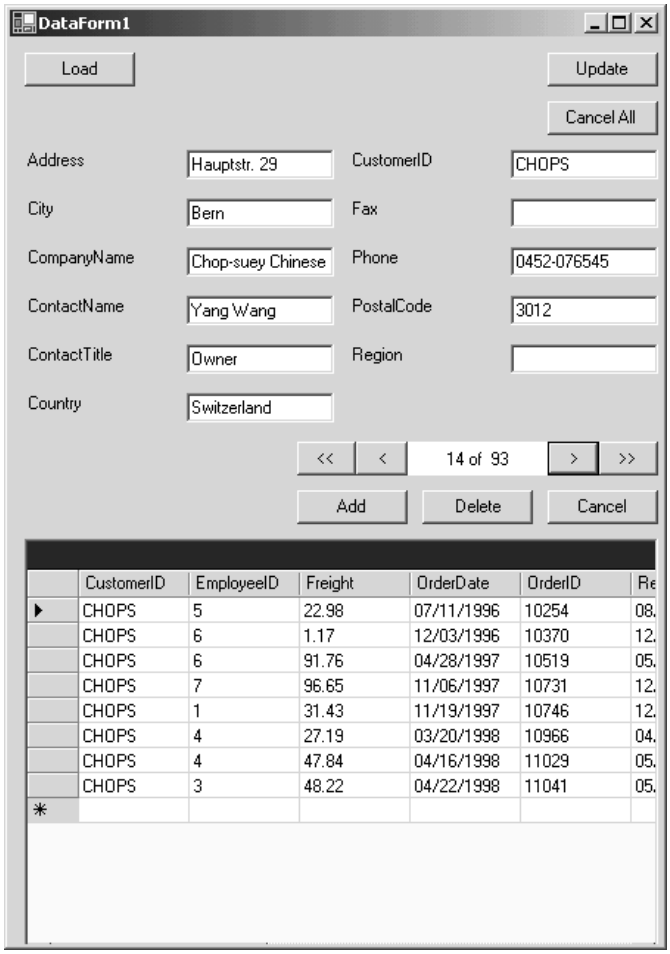

Figure 1. A Data Form built using the wizard included as a part of Visual Basic .NET.

.NET treats Jet as one of the three first class supported databases (the other two being SQL Server and Oracle). You can be confident that you can get your Access data into a .NET application with minimal effort. Visual Studio .NET also includes a Data Form Wizard, which can be very helpful if your user interface depends on bound forms. **Figure 1** shows a Data Form based on the Northwind sample database that I generated without writing any code at all. This may not be the prettiest form in the world, but the VB.NET forms designer is powerful and will be easy for Access developers to use.

If you're feeling constrained by the limitations of Access, you'll find the power of the .NET Framework useful. Pretty much anything that you can do under Windows can be done using the object-oriented Base Class Library. Non-rectangular forms, public-key cryptography, invocation of objects on remote machines, analysis of XML files, easy globalization... that just scratches the surface of what the Framework can do for you.

.NET's support for the Internet is superb. You've no doubt heard about Web Services by now, which allow you to call objects on a remote computer and retrieve the results of your calls by sending XML messages over HTTP. You can access Web Services

from any version of Access, but, from .NET, writing Web Services code is easier in .NET. It's also faster and more functional. ASP.NET also provides a wonderful environment for displaying bound data in the browser. It blows the doors off anything that you can do with Data Access Pages.

On the flip side, though, there are a few things to be aware of. First, of course, the .NET Framework is, at this point, a version 1.0 product. It's too soon for us to really know where the rough spots, limitations, and bugs are lurking (although the long and gargantuan beta program should have stamped most of the bugs out). So you're taking a bit of a chance if you move to .NET.

The .NET Framework is also huge, containing around 1,000 objects to learn about. Expect a severe drop in developer productivity while they come up to speed on this new environment. And, although there are a lot of (probably too many) .NET books on the shelf already, most of them are of marginal use at best. Finding useful information at the level you need can be a challenge.

As a last warning, beware of the glib assertion that ADO.NET is an incremental upgrade to ADO 2.7. No matter how many times you hear or read that, it's just not true. ADO.NET is completely different from previous data access layers, and you'll probably find it a challenge to master. I think it's worth the effort, but an effort it will be. (One thing that can make it easier is David Sceppa's *Microsoft ADO.NET: Core Reference*, from Microsoft Press.)

Don't overlook the client requirements for a .NET application either. Notably, the .NET runtime is 21 MB, which means it may be too large to install if you deliver your products over the Web.

On the whole, I expect many applications will migrate from Access to Visual Basic .NET or Visual C# .NET as, in the past, some Access applications have been migrated to Visual Basic 6. But, if you decide to take the plunge, here's one last word of advice: Investigate the interoperability pieces of the picture. COM-.NET interoperability can allow you to convert only part of the functionality to .NET and leave the rest in COM or Access. For example, you can create a Web Service in .NET code and consume it from Access code. You can also use COM to call a .NET object. If you just want to use some of the new capabilities without migrating your application, these interop pieces may be the answer. It's not a two-way street—while you can call most COM objects from a .NET application, adding a reference to Access in a .NET application is problematic.

I'm an independent system developer using Access 97 (it still works perfectly well for me), but my latest project requires ODBC connectivity to data sources that aren't covered by the standard Access 97 drivers. Are there available drivers for additional sources for Access 97, and, if so, where can they be obtained?

Even though Microsoft wants to move you to all new bits, you don't have to go. Sure, ADO.NET is sexy, and OLE DB provides a good interface for COM-based database access, but Access 97 certainly makes good use of ODBC. Before you give up and upgrade everything to the latest protocols and standards, you might buzz through Ken North's Web site at http://ourworld.compuserve.com/homepages/Ken_North/Homepage.HTM. There, you'll find, among many other things, an extensive list of ODBC drivers and related resources. QuickBooks, SAS, LDAP... there are hundreds of other drivers out there for data sources large and small, if only you know where to look. Of course, not every possible data source is covered, and many of these drivers are nearing the end of their useful life. But it's definitely worth having a look. ▲

Mike Gunderloy is a long-time Access developer and editor of Pinnacle's Smart Access eXTRA e-mail newsletter. He's the author of *ADO and ADO.NET Programming* and the co-author (with Ken Getz and Paul Litwin) of *Access 2002 Developer's Handbook*, both from Sybex. When not writing, he raises waterfowl, peacocks, and other critters on a farm in eastern Washington state. MikeG1@larkfarm.com.

Total Visual Code Tools...

Continued from page 18

the FMS tools for inserting new code templates and when running cleanup against existing code. Your standards can then be saved to a file and shared with team members who also use FMS Total Visual Code Tools. This ensures that all developer-generated code will have a similar look and feel.

But wait—there's more

While it was great of the Microsoft Access development team to include the native tools, it's clear that they left the door open for companies like FMS to assist the developer community with some cool utilities. In addition to what I've already described, Total Visual Code Tools 2002 includes utilities to insert code for message boxes, Select Case statements, and the code to return an ADO or DAO recordset. Is it rocket science?