

Use InterSystems Caché with Visual FoxPro

Visual FoxPro works well as a client with Caché as the database server. Learn about Caché and how to make it work with VFP.



By Ryan Jentsch

People are, by nature, impatient. Wanting an application to run faster and respond better is a popular feature request from users. FoxPro's hallmark characteristic has always been speed. I remember an ad for DOS FoxPro that consisted of a chart comparing its query speed to the popular DB2 and Sybase mainframe database systems. (That a DOS-based database would dare compare itself with the big mainframe database systems of the day was amazing.) Microsoft has made sure VFP carried on this tradition of unsurpassed performance.

Similarly, the InterSystems Caché database system claims to be a database with superior speed, reliability, and performance. The similarities don't stop there. VFP has a grassroots beginning and is a Microsoft niche product with a loyal user base. Caché is a flexible database system that can be used as a database for many industries, but has found itself a niche as the most popular database system in the healthcare industry.

VFP works well as a client with Caché as the database server. If you're looking to convert your application to a client-server architecture or you're starting a project and want to build it as a client-server solution, Caché is a database server to consider.

Caché is not as expensive as SQL Server or Oracle, but Caché has more features and "feels" more professional than "free" or open source databases such as MySQL. Caché fits in best with VFP if you're developing a client-server solution and can't afford the expensive high-end server, but you don't trust the

lower-end servers—or the lower-end servers may fall short in features you need.

What is Caché?

As a VFP developer, I was interested in seeing if I could effectively use Caché with VFP. I also wanted to see if Caché could live up to InterSystems' performance claims.

When you install Caché, an icon that looks like a solid blue Rubik's Cube appears in the system tray. Right-clicking on the cube brings up a context menu that lets you start and stop the Caché service, explore objects in the server, and manage other aspects of the Caché system, similar to SQL Server or the Microsoft Desktop Engine (MSDE).

InterSystems doesn't refer to Caché as a relational database; instead, it calls it "a multi-dimensional data engine." This means that, internally, Caché stores data differently than most database systems do. Traditional database systems (including VFP) store data in a linear manner, as a stream of data. Caché keeps data in "sparse... multidimensional arrays." This eliminates the processing overhead needed for traditional joins.

Think of these sparse arrays as a hypercube. You can store data in this space with any number of subscripts of any data type. This is similar to how VFP handles arrays. Each element in a VFP array can be of a different type. Consider this code:

```
** Define a structure for invoice line
** items
**
DEFINE CLASS InvLines AS Session
    InvoiceNum = 0
```

VERSIONS
Visual FoxPro 8/7/6/5/3

Ryan Jentsch has been programming since 1982. He works for Ingenix, a publishing company, and also does independent consulting. Ryan is a Microsoft Certified Professional and is working toward a Microsoft Certified Solution Developer credential. ryan.jentsch@ingenix.com.

```

Item = ''
Price = 0
Quantity = 0
ENDDDEFINE

** Create an array to hold each line
** item object
**
DIMENSION aLineItems(2)

** First invoice line item
**
aLineItems[1] = CREATEOBJECT('InvLines')
aLineItems[1].InvoiceNum = 123882
aLineItems[1].Item = '1223'
aLineItems[1].Price = 10.00
aLineItems[1].Quantity = 1

** Create array to hold invoice header information
**
DIMENSION Invoices(n,4)
Invoices[1,1] = 123882           && Invoice#
Invoices[1,2] = {03/14/2003}   && Invoice Date
Invoices[1,3] = 12.32          && Invoice Amount
Invoices[1,4] = 12.32          && Amount Paid
Invoices[1,5] = aLineItems     && Line item array

```

This code creates an invoice with two detail lines in arrays (in memory). However, this code breaks in VFP at the line:

```
Invoices[1,5] = aLineItems
```

This doesn't work because VFP doesn't support nested arrays (also called a hyper-cube or sparse arrays). You can't have an array element as another array.

But, this is how Caché works. When you query the Caché system, it loads the requested data into arrays (into memory) and manipulates the data in a hyper-cube structure. For example, the following SQL command loads the data from the hard drive for invoice 123882:

```
SELECT * FROM Invoices WHERE invoicenum = 123882
```

It gathers the data into an array, and includes the associated invoice line items as a sub-array. Caché has an internal function similar to an ASCAN that lets it quickly search arrays and sub-arrays for requested data. This is similar to VFP's Rushmore technology that can load relevant data from the hard drive into memory and manipulate it at amazing speeds. Caché takes a similar approach by loading requested data into memory, but it organizes the data as arrays and sub-arrays.

In an application, the request for invoice header information would typically be followed by a request for the invoice line item details. Because Caché knows the invoice header and line item data are related, a second query, such as the following, would execute quickly because the line item data was already loaded into sub-arrays from the query against the invoice header table:

```
SELECT * FROM InvoiceLines WHERE invoicenum = 123882
```

Aside from the claim of great speed, one of the intriguing things about Caché is that it's class-based (i.e., object-oriented). So, all data structures in Caché, including table structures, are actually classes. When you create or change a table structure, you must compile the table/class for your changes to take effect.

Caché exploits the power of inheritance at the database level. Because tables are classes, new table classes can inherit existing table class definitions and behavior. For example, you could create a base table (class) named People to collect address information. You could also create Employee and

Customer tables that inherit their structure and behavior from the People table. You could expand the Employee table to include a Salary column and the Customer table to include a CreditLimit column. If you later decide you need to collect a phone number for both employees and customers, you can add a phone number field to the People table. The Employee and Customer tables would automatically inherit the phone number column as part of their structure. This is one of the advantages of class-based design.

Caché also has its own Java-like language you can use to create object-oriented stored procedures. I'll describe class-based table definitions later in the article.

I need my space

Much as you would use a database container (DBC) in VFP to contain your database objects, Caché uses namespaces to contain databases and database objects. Think of a namespace as a project that contains one or more databases. When you install Caché, some system namespaces are already set up. Before you can create a database in Caché, you must create a namespace. The database inherits the behaviors and defaults you set for the namespace.

To add a namespace, just right-click on the Caché cube and select Configuration Manager (figure 1). The Namespaces tab has a list of existing namespaces. Clicking on the Add button launches a wizard to guide you through creating the namespace. You can either create a new database or map an existing database to the namespace. I decided to name my namespace the same as my database.

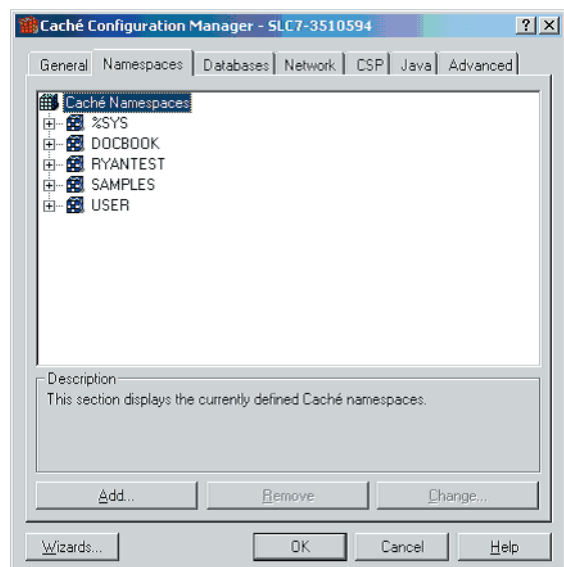


Figure 1: Namespace wizard—Clicking on the Add button on the Namespaces Configuration Manager launches a wizard that walks you through adding new namespaces and database objects.

Connecting: Getting to Caché

Connecting Caché to VFP is easy because Caché comes with both an ODBC driver and an OLEDB provider. VFP 7 uses ODBC to connect to foreign database systems. The easiest way to set up an ODBC connection is to create a domain name

system (DNS). You can create a Caché DNS by going to Control Panel > Administrative Tools > Data Sources (ODBC). From the Data Sources applet, click on the Add button and select the InterSystems ODBC driver; this brings up the DNS configuration window. When accessed through ODBC, Caché's table objects work just like standard RDBMS tables.

As an example, I created a new DBC in VFP and named it CACHE.DBC. Then, I created a connection object with the command:

```
CREATE CONNECTION Caché ;
DATASOURCE CACHE ;
DATABASE RyanTest
```

If you prefer a DNS-less connection, you can use SQLSTRINGCONNECT() using the connection string:

```
DRIVER={InterSystems ODBC};
SERVER=127.0.0.1;PORT=1972;DATABASE=RYANTEST;
UID=_SYSTEM;PWD=MyPassword
```

Show me the data

I wanted to create some tables and populate them with data quickly. SQL Server 2000 comes with a sample PUBS database, so I used SQL Server's Data Transfer Service (DTS) to transfer the PUBS database to Caché. After transferring the tables and data, I could view the newly created objects using the Caché Explorer (figure 2). The Caché Explorer is available by right-clicking in the Caché cube in the system tray and selecting Explorer.

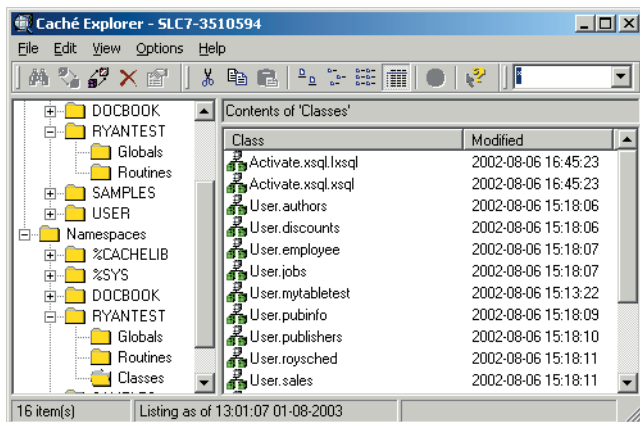


Figure 2: Caché Explorer—Here, you can view and modify database objects.

To test my connection and verify the data was there, I ran:

```
OPEN DATABASE Caché
nCache = SQLCONNECT('Caché ')
SQLEXEC(nCache, "SELECT * FROM authors",;
"CurAuthors")
SELECT CurAuthors
BROWSE
```

The SELECT command worked great, and I was able to query and browse the entire authors table. Caché also accepts Data Definition Language (DDL) commands, which you can use to create or alter database structures. For example, here's how you'd use DDL to add a new column to a table or create a new table or a database:

```
SQLEXEC(nCache, 'CREATE TABLE ' +;
MagicSpells' +;
'(myid int, spelltext VARCHAR(30))')
```

This code creates a table named MagicSpells. I checked to see if database objects are case-sensitive:

```
? SQLEXEC(nCache, "INSERT MaGiCsPeLLs (myId, sPeLLtext) "
+ ;
"VALUES (1, 'Firebolt')")
```

Again, no problem. Database objects aren't case-sensitive; standard ODBC functionality works properly. Only one DDL command failed: CREATE DATABASE. This is probably because the database is mapped to a namespace, a concept standard DDL doesn't support. This isn't a big problem because it's easy to create databases using Caché's namespace wizard. Plus, it's rare to have to create a database on-the-fly in an application, particularly with a client-server system.

DDL works great for creating tables, but the Caché Explorer offers more finite control over the table definition. It's interesting to look at the table editor in Caché Explorer because the table is defined not by DDL commands but as a class in a Java-like language (figure 3).

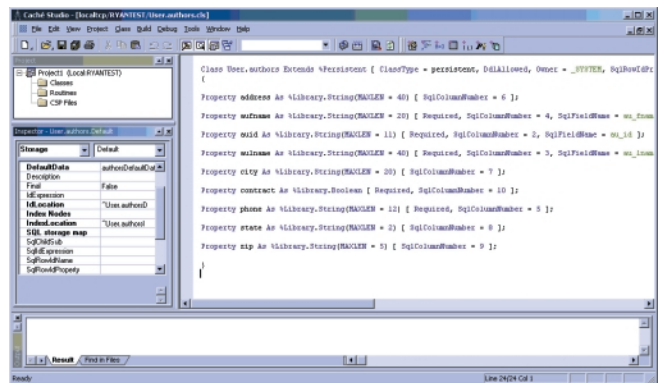


Figure 3: Table definitions—Tables in Caché Explorer appear as classes; each column is a property.

To add a new column to a table in Caché, just type a new property definition. You can further define the class/table by adjusting the values in the inspector window (much like a property sheet).

When you've made all your changes, compile the class. If it doesn't encounter any errors, Caché re-instantiates a table instance that reflects the changes. I'm a big fan of object-oriented programming, and Caché's method of making tables class-based with columns as properties makes it easy to understand and manipulate objects.

Testing speed and performance

My company has a fairly large SQL Server 2000 database. I transferred a subset of 70,000 records from this database to MSDE, Caché, and a free DBF file for comparison purposes.

I needed another table so I could benchmark join performance. I copied the first 20,000 records of the table into a new table. When I was done, I had a Caché database with two tables: SOTRAN (with 70,000 records) and SOTRAN2 (with 20,000 records). I had the exact same tables and data in MSDE 2000 and VFP 8.0.

In the three database systems, I created indexes on the primary key for all the tables. I also wrote a program called

BENCHMARK.PRG to test the performance of common SQL commands.

My first runs of BENCHMARK.PRG Caché performed fairly well. However, given InterSystems' performance claims, I was sure there was something I could do to increase performance. After some exploring, I found that Caché has what's called a bitmap index. A bitmap index is similar to SQL Server's clustered index. You can only have one bitmap index per table, and any query that references the bitmap-indexed field runs faster than if it were a normal index. I modified the table in Caché to use a bitmap index, and performance significantly increased.

Table 1 shows the results of BENCHMARK.PRG.

Table 1: Results (in seconds)—In most cases, Caché gives sub-second response times.

Test	MSDE	Caché	VFP
Simple one-record SELECT	0.234	0.187	0.031
Multiple records SELECT range	0.843	2.359	0.172
Simple JOIN	3.047	0.046	0.000
JOIN with filter	1.470	0.047	13.203
INSERT multiple records	5.157	0.157	N/A
SUB SELECT	0.937	0.344	60.422
Simple one-record UPDATE	3.219	1.580	6.266
Multiple-record UPDATE	0.375	2.375	0.062

I did eight types of tests for performance:

Simple select—Returns one record based on a primary key value.

Select range—Returns multiple records using a BETWEEN clause.

Simple join—Joins one table with another without using a WHERE clause.

Filtered join—Joins two tables using a WHERE clause.

Insert—Inserts multiple records via INSERT INTO SELECT FROM.

Subselect—Queries using a sub-query with an IN clause.

Simple update—Updates one record based on a single primary key value.

Multiple update—Updates multiple records using UPDATE...WHERE BETWEEN.

VFP doesn't support the INSERT INTO...SELECT FROM syntax, so I didn't gather statistics on that.

Many database systems—Caché included—use what's called a cached query plan to improve performance. When a SQL request comes to the server, the database system analyzes the request, determining what indexes to use and how much memory and other internal resource are necessary, and then compiles this information into a query plan. The database system temporarily saves the query plan in memory so if a similar SQL request comes through, the server can reuse the cached query plan. The result is improved performance because the system doesn't have to reanalyze the SQL request.

Continued

AMI - Network

I performed the tests on an IBM machine running Windows 2000 at 1.2GHz, with 256MB RAM and two 30GB hard drives. The OS is installed on my C: drive. I stored the data for testing on MSDE, Caché, and VFP 8.0 databases on my D: drive, which has 16 GB of free space.

The statistics in table 1 indicate Caché performs well with applications that have a lot of inserts and updates. It also does well when querying data that returns one record or is a result from a simple join. In these cases, Caché was as fast as (or faster than) SQL Server.

Caché is an appropriate database choice for general business applications such as order-entry and contact and sales management. Caché is a particularly good choice for applications that need to collect real-time data quickly. InterSystems has an example of this type of application at <http://www.cachedatabase.com/analysts/2002/trakus.html>

where hockey players and a puck were fitted with position transponders and positional data was fed into the Caché database. Users could view players and game statistics (such as distance skated, player's average speed, etc.) in real time.

Caché didn't perform as well against the other databases when returning large groups of multiple records. However, keep in mind that when it was returning multiple records, there were about 5,000 records in the result set; Caché returned these in under 2.5 seconds, which is acceptable for most applications expecting large result sets.

Gotchas

I had very few problems with Caché. Where I found problems, I could usually work around them or handle them through Caché's administrative tools.

Other features of Caché

Caché supports a large number of data types. Internally, Caché defines data types as classes. Table 2 shows a list of common data types and how they map to traditional SQL systems.

Table 2: Common data types—Here's how common data types map to traditional SQL systems.

Class Name	Holds	Analogous SQL Type(s)
%Binary	binary data	BINARY, BINARY VARYING, RAW, VBINARY
%Boolean	a Boolean value	N/A
%Currency	a currency value	MONEY, SMALLMONEY
%Date	a date	DATE
%Float	a floating point value	DOUBLE, DOUBLE PRECISION, FLOAT, REAL
%Integer	an integer	BIT, INT, INTEGER, SMALLINT, TINYINT
%List	data in \$List format	N/A
%Name	a name in the form "Lastname,Firstname"	N/A
%Numeric	numeric values of varying precision	DEC, DECIMAL, NUMBER, NUMERIC
%Status	an error status code	N/A
%String	a string	CHAR, CHAR VARYING, CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING, NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, NATIONAL VARCHAR, NCHAR, NVARCHAR, VARCHAR, VARCHAR2
%Time	a time value	TIME
%TimeStamp	a value for a time and date	TIMESTAMP

Here are some of the benefits Caché offers:

- Users can define their own data types, deriving them from Caché's base classes.

- Built-in support for SOAP, FTP, HTTP, and e-mail.
- Server runs on multiple platforms:
 - Alpha (OpenVMS, Tru64 UNIX)
 - HP (HP/UX 11i)
 - IBM P (AIX 4.3.3, 5.1)
 - Linux (Red Hat—Intel, SuSE—Intel)
 - Sun Solaris (Intel, SPARC 2.8)
 - Windows (95, 98, ME, NT 4, 2000, XP)
- Security model with users, roles, and privileges.
- Support for distributed server processing (called Federated servers in SQL Server).
- Takes advantage of servers with multiple processors with increased concurrency.
- Tight integration of the database with the Web.
- A Caché Server Page (CSP) language (similar to Microsoft's Active Server Page technology, but CSP is tightly integrated with the database).
- Support for user-defined functions and common built-in functions such as SUM(), COUNT(), AVG(), etc.
- Multiple client access protocols: ODBC, ActiveX, JDBC, Java, and TELNET.
- Shadow server capability (similar to SQL Server replication).
- Object extensions to SQL. For example, say you have a Vendor class (table) that references two other classes: Contact and Region. You can refer to properties (columns) of related classes using the reference operator:

```
SELECT ID, Name, ContactInfo->name
FROM Vendor
WHERE Vendor->Region->Name = 'Antarctica'
```

- GUI tools to administer, query, and program the Caché system.
- Web-based (local CSP) documentation with full search capability.
- SQL Gateway (called Linked Servers in SQL Server).

Use InterSystems Caché with Visual FoxPro

I did find an issue with how Caché specifies the default database owner. Oracle and SQL Server both designate the database owner as DBO—for example:

```
SELECT Authors.au_fname, authors.au_lname
FROM dbo.authors
```

When you're using Caché, this statement would look like this in the View Designer:

```
SELECT Authors.au_fname, authors.au_lname
FROM SQLUser.authors
```

This might not seem like a big deal, but I have utility programs that assume the use of DBO and I had to modify these to get them to work with Caché. It isn't a flaw with Caché, just a difference to be aware of and a good thing to think about if you're designing utility programs!

The View Designer in Visual FoxPro 8 handles most Caché queries well. I added tables and joined them in the designer. I encountered a "connection busy" problem when I ran the query from the designer and then tried to add a new field to the selection. I was able to solve this by changing the "Maximum records to fetch" setting to All, or not running the query in design mode.

The two biggest problems I ran into had to do with server-side views and how Caché concatenates strings. It uses the underscore (_) to join strings, like this:

```
SELECT au_lname "_", "_ au_fname FROM Authors
```

This is a legal query for Caché. When you run it, it displays the authors in "lastname, firstname" format. But the View Designer doesn't know how to work with this syntax. From the View Designer's point of view, the underscore characters are never operators, and their presence in the SELECT statement doesn't look right.

To overcome this, I tried to create a server-side view, but I ran into a number of problems here, too. Using DDL commands or the SQL Manager returned various error codes when I tried to create the view. InterSystems says it's still working on the server-side view feature and that it should be available in the next release.

Caché cow

Microsoft's SQL Server and Oracle are the obvious database heavyweights on the market. With Caché's professional look and feel, it could become a contender with the heavyweights. It installed on my computer without problems and performed well in my tests. I was working with a reviewer's copy and the documentation is still a work in progress, but what was available was concise and helpful. I was able to get Visual FoxPro to connect and start using the system in a short time. The price for a single-user license is about US\$300.

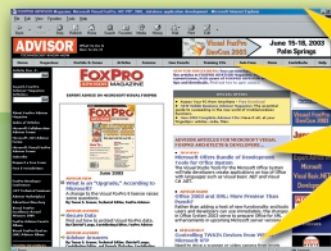
It was fun to experiment with this new technology and put it to the test. There are many Visual FoxPro programmers who would like to start using client-server technologies or migrate their existing projects to a client-server solution, but have found the heavyweight database systems carry a heavy cost and other database alternatives may not have the rich features they want such as stored procedures, views, and user-definable functions or fast performance. Caché seems to offer a nice middle ground with a fast, stable, feature-rich server that's reasonably priced. **ADVISOR**

FOXPRO ADVISOR is Now

Online

250+ Technical Articles Are Ready for You

Right Now!



**4 YEARS
for the
price of 1**

NEW: Pro-Subscription to FoxPRO ADVISOR gives you every new issue on paper & ONLINE.

PLUS: You get full access to the **FoxPRO ADVISOR ARCHIVE** with years of essential how-to articles & downloads!

With a **FOXPRO ADVISOR PRO-SUBSCRIPTION** you can read every new issue's articles online, typically before you receive the magazine in the mail. Plus, you'll find extras online, such as exclusive how-to tips, tricks, and downloads in the ever-growing FoxPRO ADVISOR ARCHIVE.

The entire FoxPRO ADVISOR ARCHIVE is organized by date, keyword, expert author—and it's full-text indexed so you can find anything instantly. You get it all when you have a Pro-Subscription to FoxPRO ADVISOR Magazine.

To start your Pro-Subscription, or add online access to your current subscription, go to www.FoxProAdvisor.com, or contact us today!

www.FoxProAdvisor.com
800.336.6060 • 858.278.5600
Subscribe@Advisor.com