

A Quick Look at InterSystems' Caché Database

By Gary Walker and Lloyd Work

Way back at the dawn of time, dinosaurs ruled the earth (you know, 40 years ago). These dinosaurs used the cutting-edge data architectures known fondly as hierarchical and network data models. These models worked well for their intended purposes (giving fast access to the applications data), but were limited in the sense that only the application knew how to access the data (putting the various pieces together as needed) and didn't help other systems, users, and others make use of the data without going through the application guarding the data, or replicating the logic to navigate through the data. Thus was born the relational model for data access incorporating tables, tuples, DDL, DML, and all the other things we take for granted in a relational database.

Now, if you are like me, you occasionally hit the wall when trying to use cutting-edge object technologies based upon a typical relational database. Mapping internal object classes to their relational table equivalents and working out the underlying details can often be quite a mess.

A significant portion of your code is related to marshalling the data to and from the database into the internal object representation of your code. OODB (Object Oriented Database) vendors like to call this impedence mismatch. So, you either deal with all of the marshalling code, or you tend to deal with the data in a non-object oriented manner to avoid the marshalling code. You probably also find that you spend a lot of time writing code that performs a left outer join on a dozen tables just to get the information for the query. I often find that the update SQL code is considerably worse than the select SQL. And somehow the so-called SQL standard is just not useful enough to handle the real world that was supposed to make it all worthwhile, because to actually do real work, you are forced to either A) use the vendor extensions and code a mound of triggers and stored procedures, or B) accept substandard performance from the database and embed all of the business logic in the application.

These are not new problems as it was recognized in the mid-eighties, though it has grown worse over the years as we demand more from computers. With the growing popularity of Object Oriented Programming Language (OOP), vendors realized a new way to model their data; thus was born the OODB. The OODB's of the mid-eighties unfortunately repeated the problems of the earlier network database, where the only way to access the data was through the underlying OOP. There were also performance problems, vendor instability, and similar problems as typical of new technologies.

The best that I am able to figure out, InterSystem's new Caché 5 post-relational database enters the picture with the goal of presenting the best of relational and object databases in a unified model without sacrificing performance, scalability, or maintainability. Let's see how well they did.

The Database

Caché 5 boasts a wealth of features that any relational database programmer or DBA would appreciate, including:

A **multidimensional data engine** that uses an advanced storage mechanism to ensure ultra-fast transactional performance, even in high-volume environments. This effectively eliminates the need to do advanced joins as in typical relational databases. Caché's built-in transactional bitmap indexing mechanism delivers excellent query performance while maintaining update performance equal to that of typical RDBMS indexes. Traditional indices are also available, but the bitmapped index is just the ticket for a data-mining application.

- A **multi-tier architecture** that allows separation of application servers from data servers, or data-mining servers from transaction servers, or replication servers all in an easily administered environment. InterSystems ECP (Enterprise Cache Protocol) is the secret to making this work.
- **Unified data** architecture that allows data to be simultaneously represented as both objects and tables, giving developers the option to work with the model they are most comfortable with. Developers can also model their data as objects, tables, or multidimensional arrays. Defining an object property automatically results in a corresponding column and vice-versa. Caché includes a powerful IDE (Integrated Development Environment) for creating Caché Object Classes, and can also import its data from external sources such as DDL or Rational Rose.
- **Open data access** architecture that supports many technologies and protocols including ODBC, SOAP, ActiveX, .NET, XML, HTML, C++, COM, Java, EJB, and JDBC. There are native binding for Java and C++, but other programmers do not have to feel left out.
- **Web architecture** that allows developers to create cutting-edge Web applications using either their own Web technologies or Caché's own high-performance Web technology known as Caché Server Pages (CSP). In addition, Caché has built-in support for Web Services; any Caché object method or stored procedure can be exposed as an SOAP service or an XML object.
- **Cross-Platform support**—Red Hat & Suse Linux, AIX, HP/UX, Solaris, Microsoft Windows, as well as some other platforms for older versions of Caché. Caché Web technology also works on various Web servers; IIS and Apache are both supported in release 5.0.

I have looked at OODB before, and I must say that after looking at Caché, I am favorably impressed. Caché has taken great strides in object-relational technology, providing a comprehensive solution that is both powerful and flexible enough for even the most demanding environments. Other environments that attempt to mix OODB and relational technology simply attempt to graft one on top of the other (which is native and which is grafted depends on the vendor). Unlike grafted branches on a fruit tree, the grafted DB fruit does not taste right and often has additional compromises. Performance is often a forced tradeoff for the ability to use the grafted technology and you are usually relegated to a second-rate implementation of the grafted technology, forcing the programmer to use unfamiliar object routines or jump through extra hoops to access the data. With Caché, you really do get the best of “both” worlds, without having to compromise performance for flexibility.

Installation

Besides being powerful and flexible, Caché also comes with a very small footprint and runtime requirements, making it perfect for embedded applications. In fact, about 80% of Caché users are using it in vertical market applications that came with Caché technology embedded in their products. In fact, Caché happens to be the #1 embedded database technology used in the health care industry. Overall, Caché users number over four million in 88 countries world-wide.

I found Caché’s installation to be straightforward and hassle-free. Because of its Caché Server Page (CSP) extensions, it prompted me to stop my Internet Information Server (IIS) from running so it could install support for CSP, but as soon as it was done, IIS was up and going again.

InterSystems claims that the majority of their users do not have a database administrator, and frankly I believe it. Although Caché does not to my knowledge have the ability to be tuned and tweaked like Oracle, I suspect most sites can save enough money to buy a little more hardware from the money they save on a system administrator. Finally, if your application is engineered using an object model (as opposed to a pure relational model), you are likely to run faster in Caché than under the equivalent Oracle relational model due to the implementation efficiencies of having the data model match actual usage.

The Environment

Though not perfect, Caché provides the basic tools you need to get going. It provides a terminal interface (command line oriented), a control panel and configuration manager (to avoid having to edit control files by hand), Explorer (which is a form of object browser), SQL Manager (for a traditional relational database interface), Studio (which is the Integrated development environment), Documentation (local searchable Web engine), and at least one Windows and taskbar applet to give you quick access to all of these pieces.

Nothing was as slick as the best commercial offerings for such utilities, but they were not slouches either, with clean interfaces, wizards, and so on. Studio is all new, but it has some rough edges (I found the wizard for adding database properties one at a time was cumbersome). It’s a long way from having the code create tables and alter table statements by hand, though.

The Scripting Languages

There is no shortage of ways to manipulate your Caché database. Native bindings for Java and C++ will appeal to some, but if you program in VB or Delphi, you are not left in the cold because you can use ActiveX to use the object features in a basically native manner. Failing that, you have two choices of scripting languages, Caché ObjectScript and Caché Basic. ObjectScript was the original scripting language, but Basic has extended the same benefits. You can write scripts (stored procedures) in either language and compile them to the internal p-code form, which then runs at full speed within the database server (in the database virtual machine). Java and C# programmers will find this computation model quite familiar. With either scripting language, you can invoke methods, set properties, and so forth in a very natural manner (if you are an object-oriented or VB script programmer, at least). And, ObjectScript can call Basic and vice-versa (sounds like .NET to me). Studio includes the debugger for these scripting languages.

For example, here is a very simple method in Basic straight from their online documentation.

```
Method MyMethod() As %Integer [language = basic]
{
  For i = 1 to 10
    person = new Sample.Person()
    person.Name = "John" & i
    person.%Save()
  Next
}
```

Notice that the method is declared as %Integer; the % indicates that it comes from one of the built-in classes (system). Notice also that %Integer is a class type itself, just as in other pure OOPLs. I should also point out that the above method is a standalone method, not belonging to any class (not allowed by Java and some other OOPLs). So, the scripting language is not a completely pure OOPL (though I’m never sure if this is good or bad). The VBScript programmer will find the BASIC language very natural.

What Do You Mean by Post-Relational Database?

The skinny on creating an object class is actually pretty simple. Create a project (MyApp) and a class (Person) that persists in the database.

```
Class MyApp.Person extends %Persistent [ClassType = persistent]
{
  Property Name As %String(MAXLEN=50);
  Property Home as Address;
}
```

But what is Address? Well, it is simply another class.

```
Class MyApp.Address extends %SerialObject [ClassType = serial]
{
  Property City As %String;
  Property State As %String(MAXLEN=2);
}
```

%Persistent basically means that you store this in the database (a row)

%SerialObject means that you are defining an embeddable object that is stored with the Person row, so the database does not have to go searching for the Address when it has the row. This is very much like hierarchical and network data models, except that you can still use relational SQL to get to the same data. Thus, the relational model is not broken; you can still run ad-hoc reports and so forth, using standard tools. But, your mainline applications do not pay for this extra processing complication to join one table every time the relational information that forms the object is brought together. Relational databases in fact do not keep the relationship; the relationship is artificially constructed on demand, based on key comparisons each time it is needed. OODB actually keeps the relationship. Caché gives it to you whichever way you prefer.

Unexpected Surprises — (The good kind)

ΩA few things just struck me as, “That’s cool.”

1) SQL extensions—In a syntax familiar to any C programmer, I can write

```
Select ID,Name,Contact->Name,Contact->Phone from Vendor
where Vendor->Region->Name = ‘Midwest’
```

Now, I could also write the equivalent join syntax, but the code above just seems more natural and readable to me.

2) Inherited table definitions

Define a contact with properties such as Name and Phone. Then, define Employee as extends contact, and add properties for Hire Date and SSN. Add a couple of contacts, and a couple of employees, and then you have:

```
select Name from contact
```

```
Nicholson, Jack
Eastwood, Clint
Walker, Gary
Work, Lloyd
```

```
select Name from employee
```

```
Walker, Gary
Work, Lloyd
```

Caché is smart enough to recognize all four records are contacts, but that we don’t employ the famous actors.

3) Multiple Inheritance. People frequently bad-mouth multiple inheritance as unnecessary, confusing, and so forth (and I’ll agree if you talk about MI in C++). But, using MI for mix-in classes and the like is quite useful, especially when someone else has done the hard work creating the mix-in classes. So, I cracked a smile when I saw MI mix-in classes for XML.

4) The SOAP interface is very slick and easy to use—almost scary.

The Experience

I ran through the samples, and information from the reviewers guide, and then it happened, a fatal problem. The reviewer’s CD included a two-million record sample database file to be used for assisting me in evaluating performance. As luck would have it, when I tried to open it, I got the following message: Error 5075 Class dictionary out of date, please run upgrade utility \$system.OBJ.upgrade(). Of course, this meant nothing to me beyond the high probability that the 2,000,000 record sample was in an older format, not recognized by the latest version of Caché. Not wanting to type in two million records for the test, I decided it would be best to upgrade. Naturally, reviewers get special privileges with contacts deep into the heart of upper-tier tech support (try saying that with a straight face), but I thought why not use the provided documentation, which appeared to be quite good instead. Sure enough, in a couple of minutes I had my answer, and was on my way. Personally, this impressed me about as much as anything involved in the product sniff test. Let’s face it, documentation typically is horrid, and the only thing worse than reading documentation is writing it. But, there it was on the CD, good documentation for a product, searchable and everything in a simple browser interface.

The only other problem I ran into during my admittedly limited testing was that the Web interface was not working correctly, but once I figured out it was running on the wrong port (via installation default), it was easily corrected.

The Real World

Nothing kills a simple product like contact with the real world. We all know the effect; it looks cool in demos, but you hit real problems and you find out the simple stuff does not work anymore, and you are forced to break the model to accomplish real tasks. SQL is like that; tables are easy to understand, columns make sense, but why do I have to learn left outer join and triggers and stored procedures to make SQL actually work? The answer is, of course, that the real-real is hard. Okay, I can live with that. I'm used to such things. But go back in time to Visual Basic 3, or PowerBuilder 3, HTML or ... Now your mind is filled with running into brick walls. Yes, simple things are simple, but real problems are suddenly really hard to solve. Consider HTML; it was a simple thing, a beautiful thing, until someone declared, hook it up to a database and maintain state for the client. I'm not saying I won't find the brick wall somewhere, but I did not even though I am used to looking for them.

"For we walk by faith, and not by sight." 2 Cor 5:7. Unfortunately, there is no time to explore everything you would like to in a product such as Caché. I cannot build everything necessary to stress test the product, test replication servers, and so forth. I did test performance (let's just say it looked good, outrunning an equivalent test on Oracle), but building a real-world app is just not possible. Fortunately, Caché does have some white papers and customer testimonials on their site. I also read such with a bit of skepticism, but unless Intersystems is outright lying, some people are using their systems for big projects in the real world—in mission-critical systems where lots of real money is involved if the system were to be broken. You gotta have faith that Intersystems is not run by scam artists or pathological liars because they have been in business too long, with too many customers and seats to believe otherwise. Needless to say, if half the stuff in their testimonials is true, Caché is ready for the real world, or at least in some real-world cases.

How Much?

You all know the feeling. The brochure is slick, the product looks great, the salesperson is good-looking, has fresh breath, and acts like your friend, but no one will tell you how bad the damage is going to be. Once you do find out, crest-fallen, you slink into the distance, trying to find a second-rate product that almost meets your needs, but that you can afford. Caché is not free, but starting at \$200 for single-user licenses and \$1,000 for multi-user licenses, it's not that painful. You can download an evaluation copy for Windows or Linux, but this does not give you a license to deploy or develop an application. But, you can kick the tires and see whether it might be right for you.

Conclusions

Caché is not magic, but I find myself intrigued by some of the possibilities. Some that struck me in particular as possible for my own future development are:

- Embedded application development for users without a database administrator
- Fast development of systems using complex, related objects
- Fast development of Web database applications
- Low-cost deployment for large-scale databases

Download a copy at www.intersystems.com and try it yourself.

Gary Walker and Lloyd Work
Astra Digital, Inc.



www.Intersystems.com