Article
Nigel Salm · Aug 18, 2021 15m read

Why I love ObjectScript and why I think I might love Python More

Why I love ObjectScript and why I think I might love Python More

I was looking at the thread of messages on the topic of "Performance when constructing a comma-separated string", and I started writing a response but got distracted, the page refreshed, and I lost my text. I couldn't spend the time rewriting my response, so I started writing this document instead.

I started writing MUMPS at the beginning of my career. I wrote very tight and dense code blocks where exercises such as the string example were authentic challenges. We squeezed every last bit of performance out of the Digital DEC or VAX servers, where we planned where a key global would be positioned on a disk platter. When Caché was released, we were still working with M/SQL. There was a period where I was involved in several performance comparisons between Caché against Oracle, Sybase and SQL Server. We would design a schema of a few tables, populate them with several million records and then execute many searches on the resultant database. I used to write two versions of the SQL statements. One version would be a pure SQL statement, and the other would be a custom query which I would write into the class definition. The bulk of the logic goes into the 'Fetch', and I would craft my 'Fetch' method to maximize the indices I had defined and use ^CachéTemp for any interim results complex joins. I would sometimes job off one or more sub-queries that would create the interim temp globals and then resolve the joins once all of the jobbed processes had finished. The result could be summarised as follows:

Inserting data into the database using SQL or Caché Objects was always faster than any other DB. Using pure COS and direct global sets was an order of magnitude faster than SQL, Objects, and any other databases. The resultant database would be roughly half the size of the database created by any of the relational databases.

When I compared the code that I wrote in my 'Fetch' method against the code

generated by the Caché SQL Engine, I used fewer variables, 25% fewer lines of code, and the code was more readable.

The number of physical data block reads would be roughly the same as the code generated by M/SQL. However, the number of logical reads from the Global Buffer Pool would be 20% less than M/SQL.

I made use of every trick in the "MUMPS Developers Cook Book". I used commands such as 'execute', 'job' (effectively creating threads to handle sub-queries in parallel), indirection, and post-conditions. We recommend that developers don't use these language features to write readable and maintainable code by other developers.

I would initialize variables in the form:

```
set (a,b,c,d)="",(x,y,z)=0,p1=+$h,p2=...,pN=99
```

I squeezed as many expressions into one line of code as I possibly could. We believed a cost was incurred when reading each line of code into the "execute buffer". Therefore the number of lines of code executed always had a direct and inverse effect on performance.

When I work on code written by some other developer, and I notice that there are blocks of code consisting of one set command per line, I get somewhat worked up and invariably compress those 30 lines down into one. I fell in love with Caché Objects. Twenty-five years later, that love affair has outlasted two long term relationships and marriage. Class definitions, with precise and very readable property names, bitmap indexing on everything unless find indexing can do better. Parent-Child relationships rather than One-Many when I can. I will use a custom primary key in code tables when bitmap indexing is not required because set record=$g(^global(code)) will always be faster than

```
set record="",rowId=$o(^IndexGlobal("IndexName",code,""))
set:$l(rowId) record=^Global(rowId)
```

There were some forms of SQL select statements that M/SQL either didn't support or performed poorly. In general, Caché was 2-3 times faster than any other database.

Over the years, the SQL engine has significantly improved. Bitmap and iFind indexing were introduced. We use iFind indexing on the Names and Addresses of Patients in a database of 15 million Patients. Every other field is bitmap indexed. When we receive a FHIR Patient Search with several parameters, we support all FHIR specification qualifiers and operators. We construct an SQL statement that starts with a join across all of the entities of FHIR Patient, which we store in persisted classes. I am pushing for us to use the IRIS for Health repository for our next phase of development. IRIS has had two releases and has matured since I first worked with it in version 2019.1. The join is followed by any iFind clauses on Names and Addresses if specified in the search criteria. Then AND/OR clauses are added for any fields in the search criteria that we know are supported with Bitmap Indices. The deterministic or probabilistic searches we perform are so accurate and so fast it still has me jumping around in excitement (at my age!!!).

I must confess that I had never liked SQL when I was one of an ever-shrinking pool of developers that wrote MUMPS code in the late '80s. My peers were all too quick to jump into bed with Oracle or SQL Server, and it was difficult at times not to fall into a state of despair as I listened to the naysayers shouting, "MUMPS is dead.

Then, at the annual MUMPS Conference in Dublin, we woke up one morning to a note pushed under our doors announcing that InterSystems had bought DTM. At a conference held a year later in Birmingham, I was working for InterSystems, and we were showing off Visual Basic forms using the Caché dll that we had acquired when we bought Data Tree. Micronetics were on the stand opposite ours, and they didn't have a dll. Their sound system was louder than ours, but we knew we had won. It would take another year before we had bought DSM from Digital and finally MSM from Micronetics, and then there was no holding back. I remember showing off M/SQL to a customer in Birmingham who wrote accounting software. One of their customers was Barings Bank who had just lost 859 000 000 GBP due to their rogue trader Nick Leason. I couldn't help but set up my example database so that I could run an SQL query that was probably no more complex than "SELECT sum(Total) from Accounts WHERE .... and AccountNumber="666..". The account number was the account number that Nick Leason had used to hide the trading he was doing to rescue his situation that was getting worse with every single ring of the Singapore Stock Market Trading Floor Bell. I remember standing there giggling quietly, partly because of the implicit reference to the Barings Bank Collapse but also because the query actually executed, provided the correct answer and didn't take more than a minute to run (none of these things was certainties then).

That was the only memory that I have of enjoying SQL. I would deal with one audience after another audience of Oracle and SQL Server DBA's where I demonstrated Caché, Caché and VB, Caché Objects and Caché SQL and delighting in Caché Objects: so elegant, so obvious, so malleable, so readable. Object syntax (in any language) is so much more natural to me than any SQL statement and when we did get the opportunity to take a prospective customers application schema and run it through the SQL importer and translate the set of stored procedures that the prospective customer would include in the schema into either Caché Objects or pure Caché Globals I became very acquainted with reading the SQL execution plan and the generated stored query and getting into long conversations with Aviel Klausner about the one SQL query that the prospect customer had given me that wasn't working and that would make the difference between: watching the Oracle DBA's slink out of the conference room back to the safety of their index tuning and the guaranteed 6 hours of downtime that their systems had every day while backups were being done, where they could coax their relational applications back to life in readiness for the next days trading, or the excitement of winning over a customer that we had been pursuing for months who was more interested in the speed of Caché, the Object Orientation, the gateways to .Net or Java, the simple elegance of the CSP broker. I think that the question: "Why write applications INSIDE a DB environment at all?" isn't a question at all. Firstly, I create a database that will contain my code and another for my globals and right there, I have a point of separation. We have all grown up over the last 25+ years thinking of Classes, Objects, ObjectScript and Globals as being all lumped together. I argue that at runtime, the code executing in the code buffer is OBJ code. OBJ code is essentially a mixture of compiled C code, pure machine code optimized for the platform it is running on, and some remnants of the class definition that is required if you are using $classname, $classmethod, $property and other factors. Much of the 'engine' of Caché or IRIS is written in ObjectScript is a testament that ObjectScript is a perfect language to work with. It is a language that can be explicit, can be abbreviated, can be very compact. It contains all of the operators and constructs of any modern language (if, ifelse, else, try - catch, while, for [to be fair our implementation of FOR is wonderful: ¦ for i="apples","pears","Nigel","Fruit" {} ¦ for {} ¦ for i=$$$StartGValue():$$$Increment():$$$EndValue() ¦]). If one of the early MUMPS creators had called $order "$next", it would immediately be recognizable as Next() as found in every other language that iterates through an array. $PIECE is a bit quirky, but only because every other database uses fixed-length fields. The concept of delimited strings used as a database construct is alien to a SQL DBA. When you look at the compiled machine code of either form of database, the machine instructions are moving through the string, character by character, and

either counting the number of characters or looking for a specific field delimiter, whilst counting the characters as it does so.

$list squeezed a little bit of performance gain over $piece but at the cost of an extra byte or two at the beginning of each field but still less space-consuming than fixed-length fields. The reason why all of the system code is written in ObjectScript, even to this day, is because it is a very efficient language, it is a very readable language, and when the core Caché/IRIS developers, Scott Jones, Dave McCalldon, Mo Chung, required something where ObjectScript was inadequate, they would write that in C and bury it in the Kernal.

Next: If I have a table definition and I have fields that require some form of formatting or validation over and above the obvious constraints of type and length, then I want to write that code and keep it nice and close to the field definition itself. Why would I want to go into another language, another environment, to write that validation? The relational databases use stored procedures and triggers to handle such validation using the SQL language to express the validation logic. Find me a programmer who would rather use SQL to write sometimes complex logic rather than Basic or C# or C++ or OjectScript or Python, and I'll buy you a beer when I next pass through Vienna :-)

At university, I learned to program in Fortran and Pascal. Pascal was a perfectly readable usable language and being an easily excitable 19-year-old, it fascinated me that the Pascal Compiler could be written in Pascal. Later on, I learned COBOL. WTF??? And yet, I have a friend who is a developer for Sage Accounting, and he writes COBOL because Sage Accounting was written in COBOL. Page after Page of the most verbose, unreadable, unusable language I have ever come across. In fact, there is a ton of COBOL out there.

You would think that Pascal would have easily surpassed COBAL and even Basic. But it didn't. And why didn't it? Easy, it wasn't used in Banking applications (COBOL was used extensively in large Mainframe Batch Processing Applications such as Accounting). We joked that we couldn't get Banks to buy into the Caché Model because we weren't expensive enough. It wasn't that ObjectScript couldn't do the transactional processing of those Banking applications, and we were demonstrably faster than whatever technology they were using. The problem was that they had spent so much money on the systems they had and the hardware required to run those overnight Batches in time for the Banks to open at 9 am the following day. The expensive server rooms with radon gas and filtration systems

remove even the smallest dust particles lest a particle land on a disk platter or mag tape and render an entire day's worth of account transactions useless.

Pascal should have outlived Basic, and it possibly would have if Microsoft hadn't built Visual Basic and gone into competition with Delphi and Borland. Their IDE looked exactly like VB but used Pascal rather than Basic. And this was all happening while Microsoft brought out C# because they had to accommodate all those C++ programmers, and they certainly weren't going to win over the C++ programmers with Basic. They were also threatening to bring out their version of Java or remove support for Java because it annoyed them that Java ran on hardware platforms that Windows would never be able to run on. It was only when technology advances made the concept of VM's or Containers a realistic deployment option that Microsoft backed off. And so Pascal and Delphi just disappeared. I did a quick search in Google now, and there is a Pascal Interpreter for Android, so it is still out there.

Given that Pascal was just a language as opposed to Basic, which was just a language in one sense. But Microsoft used it for scripting in applications such as Excel and a proprietary connection to SQL Server, which allowed them to bind two intrinsically unsuitable environments together without the pesty hassle of complying with the ODBC and JDBC standards. Standards were heavily backed by Oracle, Sybase and pretty much everyone who had to provide a gateway to their proprietary versions of SQL. And so Basic lived on, and I'm happy that I started my programming career with Pascal, followed by a rude awakening when I wrote COBOL programs for a year working for an insurance company, when I arrived on the wet and grey shores of the UK and walked into my first job, which, just happened to be a MUMPS house. Every evolution of MUMPS to CachéObjectScript, then Objects, followed by Object Gateways to .Net and Java, Caché Basic and MultiValueBasic and now Python. Python takes me full circle, and in a sense, proves the point that ObjectScript is not some aberration lumped onto a non-relational outcast of database technology.

Caché Globals, these multidimensional sparse arrays that are so very convenient to the very nature of Healthcare data to such an extent that no matter how hard Oracle and Microsoft have tried to consume that market space and though they may well have killed off Pascal and Fortran and even basic, they haven't been able to kill off InterSystems. I remember attending an **Oracle** seminar on "**Oracle for Health**". The presenter was going on about Oracle in HealthCare which, she assured us, would take over the Healthcare Market once and for all. I put my hand up and asked, "**Isn't that what you have claimed with every major release for years**

now? You failed then. What makes you think you'll do any better this time around?" She stared at me, "**Who are you?**" she asked. I replied: "**I am from InterSystems. We dominate the Healthcare market and have done for 35 years. We have done so because our technology was born in Massachusetts General Hospital, and guess what. They still run their core systems on our Technologies.**". At which point, two burly security guards removed me from the auditorium.

So you have Oracle with their pSQL, and they own Java. You have Microsoft with SQL Server, C# and tSQL, and when you need to interact with Java, you are constrained to JDBC. Likewise, if you live in Java and have to talk to SQL Server tables, you are constrained to using ODBC, and where do we sit? Well, we have this rather clever idea of having wrappers for .Net and Java. When using ObjectScript, I instantiate an instance of Class A. It doesn't actually matter whether Class A is actually a .Net class or a Java Class, or an ObjectScript Class because I will instantiate those objects using precisely the same syntax in all cases. Then I am going to invoke the instance or class methods to manipulate those objects. It doesn't matter what the insides of those methods look like because the syntax for interacting with those classes and their methods is essentially identical no matter what they contain.

And along comes Python, which shares many features of ObjectScript in that it is an interpreted language as opposed to a compiled language. It is very readable and very usable. Just as Caché ObjectScript found a Niche in Unstructured Data, Python found a Niche in the world of Mathematical Modelling, ML, AI and much, much more. This is not a world that C# or Java are particularly comfortable in, and nor is ObjectScript, for that matter. So InterSystems has concentrated on providing increasingly powerful functionality for manipulating vast amounts of unstructured data, throw in some iFind and iKnow, some very clever indexing techniques and probability matching algorithms, and you then invite Python to come and cuddle up to our multidimensional sparse arrays and bring with it, its millions of baby .py's that do just about everything complex that you'll need. You have a match made in heaven. Oh, and just in case, I forget to mention that several architectures that dominate the world of web page development are all based on JS (Angular.js, REACT.js, Vue.js, Bootstrap (ok, there is no JS, but it is JS in all but name) and Node.js) and JS Arrays. JS isn't going away anytime soon. However, it will be interesting to see where Golang will go if you catch my drift. I have noticed that there have been entries based on JS arrays in the last couple of code competitions. If there is one technology that understands arrays better than any other technology, then it's IRIS.

And I think back to those days, sitting in my office at the company I worked for in the heart of London. The company, at one point, had been full of MUMPS

programmers but then turned them into Relational SQL programmers and then made them Redundant. I remember that feeling of beginning to question whether my faith that MUMPS, being just the best language I had ever encountered, might be wrong? The language and the companies that had built interpretations of that language were going to die. And that made me very sad because by then, I had learned five other programming languages (APL, Basic, Fortran, COBOL and Pascal) before I discovered MUMPS, and MUMPS was just so straightforward. Easy to write, easy to read, easy to deploy. In short, it was as natural to me as English, and it had a rhythm that reminded me of the hymns we sang at the Methodist school I attended:

Onward, Christian soldiers!

Marching as to war,

With the cross of Jesus

Going on before.

Christ, the royal Master,

Leads against the foe;

Forward into battle,

See his banners go!

But it didn't die. The song changed a bit:

Onboard Nigel Saaalllm

Flying off to War

With his ISC CreditCard

Going On Before.

John, the Master, McCormack

Leads against the foe (Microsoft)

Forward into Battle

See his Duty Frees Go

And CacheObjectScript was even better than MUMPS if that was possible, And CacheObjects looked so cool when demonstrated to an audience for the first few times, and CacheSQL left its M/SQL days behind and has become rather good over the years. Still, I don't particularly appreciate writing much SQL, but I have found a nice balance between Objects and SQL and Direct Global references as I have relaxed. And whereas my code was heavily weighted towards Direct Global references, with some OO and minimal SQL. When the products reassured me that I could trust that the generated code was tight, elegant, efficient, and readable, the balance has shifted again. Now I seldom use direct global names, lots of Objects and a reasonable amount of SQL.

Working with Python will require my mind to see different patterns from my ObjectScript Code. There are way too many '_abc'_ and other strange structures, but once I write a few pages of py code and then stand back. Just as I do when painting an oil painting, the patterns will pop out at me. Just as I see music as colour synesthesia, so too will my colour coded ***py*** *programs flowing across the page begin to resemble a little watercolour or even a heavy oil painting. I will be delighted, and all will be right with the world.*

#Coding Guidelines #ObjectScript #Python #InterSystems IRIS