Artigo
Danusa Calixto · Out. 27, 2022 15min de leitura

Por que eu amo ObjectScript e por que eu acho que poderia amar Python ainda mais

Por que eu amo ObjectScript e por que eu acho que poderia amar Python ainda mais

Eu estava olhando o tópico de mensagens sobre o assunto "Desempenho ao construir uma string separada por vírgulas" e comecei a escrever uma resposta. No entanto, me distraí, a página foi atualizada e perdi meu texto. Não podia gastar tempo reescrevendo minha resposta, então comecei a escrever este documento em vez disso.

Comecei a escrever na linguagem MUMPS no início da minha carreira. Eu escrevia blocos de código bastante concisos e densos em que exercícios como o exemplo da string eram verdadeiros desafios. O desempenho dos servidores VAX ou Digital DEC eram aproveitados até a última gota. Planejávamos onde globais importantes ficariam em um disco. Quando o Caché foi lançado, ainda estávamos trabalhando com M/SQL. Durante um período, estive envolvido em diversas comparações de desempenho entre o Caché e o Oracle, Sybase e SQL Server. Criávamos um esquema de algumas tabelas, preenchidas com milhões de registros, e executávamos várias pesquisas no banco de dados resultante. Eu costumava escrever duas versões de declaração SQL. Uma era uma declaração SQL pura, e a outra era uma consulta personalizada que eu escrevia na definição de classe. A maior parte da lógica está no "Fetch", e eu elaborava esse método para maximizar os indíces definidos e usar ^CachéTemp para qualquer junção complexa de resultados provisórios. Às vezes, eu criava jobs de uma ou mais subconsultas que gerariam os globais temporários e resolvia as junções após a conclusão de todos os processos de jobs. O resultado poderia ser resumido da seguinte maneira:

Inserir dados no banco de dados usando SQL ou Caché Objects sempre foi mais rápido do que qualquer outro DB. Usar COS puro e conjuntos globais diretos era uma ordem de grandeza mais rápida do que SQL, Objects e qualquer outro banco de dados. O banco de dados resultante teria aproximadamente metade do tamanho de um criado por qualquer banco de dados relacional.

Ao comparar o código que escrevi no meu método "Fetch" com o código gerado pelo Caché SQL Engine, vi que usei menos variáveis, 25% menos linhas e o código ficou mais legível.

O número de leituras de blocos de dados físicos seria praticamente o mesmo que o código gerado pelo M/SQL. No entanto, o número de leituras lógicas do pool de buffers globais seria 20% menor que o M/SQL.

Recorri a todos os truques possíveis de desenvolvedores MUMPS. Usei comandos como "execute", "job" (criando threads para lidar com subconsultas em paralelo de forma eficaz), indireção e pós-condições. Recomendamos aos desenvolvedores que não usem esses recursos de linguagem para escrever código legível e capaz de ser mantido por outros desenvolvedores.

Eu inicializaria variáveis desta forma:

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

Espremi o máximo de expressões possível em uma linha de código. Acreditávamos que a leitura de cada linha de código no "buffer de execução" resultava em um custo. Portanto, o número de linhas de código executadas sempre teve um efeito direto e inverso no desempenho.

Quando trabalho com código escrito por outro desenvolvedor e noto blocos de código que consistem em um comando definido por cada linha, fico um pouco agitado e sempre condenso essas 30 linhas em uma. Me apaixonei por Caché Objects. Vinte e cinco anos depois, esse caso durou mais que dois relacionamentos sérios e um casamento. Definições de classe, com nomes de propriedade precisos e bastante legíveis, indexação de bitmap em tudo, a menos que a indexação de busca possa fazer melhor. Quando possível, relacionamentos pais-filhos em vez de um-muitos. Uso uma chave primária personalizada em tabelas de código quando a indexação de bitmap não é necessária porque set record=$g(^global(code)) sempre será mais rápido que

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

Havia algumas formas de declarações SQL select com que o M/SQL não era compatível ou executava mal. Em geral, o Caché era 2 a 3 vezes mais rápido do que qualquer outro banco de dados.

Ao longo dos anos, o mecanismo SQL melhorou significativamente. A indexação de bitmap e iFind foi lançada. Usamos a indexação iFind nos nomes e endereços de pacientes em um banco de dados de 15 milhões de pessoas. Todos os outros campos são indexados por bitmap. Quando recebemos uma Pesquisa de Paciente de FHIR com vários parâmetros, oferecemos suporte a todos os qualificadores e operadores de especificação FHIR. Construímos uma declaração SQL que começa com uma junção em todas as entidades do paciente de FHIR, que armazenamos em classes persistentes. Estou reivindicando o uso do repositório IRIS for Health para nossa próxima fase de desenvolvimento. O IRIS teve dois lançamentos e amadureceu desde a primeira vez que trabalhei com ele na versão 2019.1. A junção é seguida por qualquer cláusula iFind em Nomes e Endereços, se especificada nos critérios de pesquisa. As cláusulas AND/OR são adicionadas para os campos nos critérios de pesquisa que sabemos serem compatíveis com índices bitmap. As pesquisas determinísticas ou probabilísticas que realizamos são tão rápidas e precisas que ainda me fazem pular de animação (na minha idade!!!).

Preciso confessar que nunca gostei do SQL quando fazia parte de um grupo cada vez menor de desenvolvedores que escreviam código MUMPS no final dos anos 80. Meus colegas foram rápidos em aderir ao Oracle ou SQL Server. Às vezes, era difícil não entrar em um estado de desespero quando ouvia opositores gritando: "O MUMPS está morto".

Então, na conferência anual do MUMPS em Dublin, acordamos em determinada manhã com um bilhete enfiado debaixo das nossas portas anunciando que a InterSystems havia comprado a DTM. Em uma conferência realizada um ano depois em Birmingham, eu estava trabalhando para a InterSystems e estávamos apresentando formulários do Visual Basic usando o Caché de dll que adquirimos quando compramos Data Tree. A Micronetics estava no estande em frente ao nosso, e eles não tinham um dll. O sistema de som deles era mais alto, mas nós havíamos vencido. Levaria mais um ano para comprarmos DSM da Digital e, por fim, MSM da Micronetics. Não havia mais como recuar. Lembro de mostrar o M/SQL a um cliente em Birmingham que escrevia um software de contabilidade. Um dos seus clientes era o Barings Bank, que havia acabado de perder 859.000.000 GBP devido ao trader desonesto Nick Leeson. Não pude deixar de configurar meu banco de dados de exemplo para poder executar uma consulta SQL que provavelmente não era mais complexa que "SELECT sum(Total) from Accounts WHERE .... and AccountNumber="666..". A conta tinha o mesmo número daquela usada por Nick Leeson para esconder as negociações que estava fazendo para recuperar sua situação, que piorava a cada toque do sino do pregão no mercado de ações de Singapura. Lembro de ficar rindo silenciosamente, em parte pela referência implícita ao colapso do Barings Bank, mas também porque a consulta foi realmente executada, forneceu a resposta correta e não levou mais de um minuto (nenhuma dessas coisas era uma certeza na época).

Essa é a única memória que tenho de gostar do SQL. Eu lidava com vários públicos de DBA do Oracle e SQL Server e demonstrava o Caché, Caché e VB, Caché Objects e Caché SQL. O Caché Objects me encantava: tão elegante, óbvio, maleável e legível. A sintaxe de objetos (em qualquer linguagem) é muito mais natural para mim do que qualquer declaração SQL. Quando tivemos a oportunidade de pegar o esquema do aplicativo de um cliente em potencial, executá-lo através do importador SQL e traduzir o conjunto de procedimentos armazenados que o cliente em potencial incluiria no esquema em Caché Objects ou Caché Globals puro, me familiarizei muito com a leitura do plano de execução SQL e a consulta armazenada gerada. Entrei em longas conversas com Ariel Klausner sobre a consulta SQL que o cliente em potencial me deu e não estava funcionando, e essa seria a diferença entre: ver os DBAs do Oracle saindo da sala de reuniões de volta para a segurança do ajuste do índice e as 6 horas garantidas de inatividade dos sistemas todos os dias durante os backups, onde eles poderiam trazer seus aplicativos relacionais de volta à vida em prontidão para os próximos dias de negociações, ou a emoção de conquistar um cliente que havíamos buscado por meses e estava mais interessado na velocidade do Caché, Orientação de Objetos, gateways .Net ou Java e a elegância simples do CSP de corretagem. Acredito que "por que escrever aplicativos DENTRO de um ambiente de DB?" não é uma pergunta. Primeiro, crio um banco de dados para meu código e outro para meus globais e, ali mesmo, tenho um ponto de separação. Todos nós crescemos nos últimos 25+ anos pensando em Classes, Objetos, ObjectScript e Globais como todos agrupados. Argumento que, no tempo de execução, o código sendo executado no buffer é OBJ. Basicamente, o código OBJ é uma mistura de código C compilado, código de máquina puro otimizado para a plataforma em que está sendo executado e alguns resquícios da definição de classe necessária se você estiver usando $classname, $classmethod, $property e outros fatores. Grande parte do "mecanismo" do Caché ou IRIS é escrito em ObjectScript, provando que essa é uma linguagem perfeita para trabalho. É uma linguagem que pode ser explícita, abreviada e muito compacta. Ela contém todos os operadores e as construções de qualquer linguagem moderna (if, ifelse, else, try - catch, while, for [para ser justo, nossa implementação de FOR é maravilhosa: ¦ for i="apples","pears","Nigel","Fruit" {} ¦ for {} ¦ for i=$$$StartGValue():$$$Increment():$$$EndValue() ¦]). Se um dos primeiros criadores do MUMPS tivesse chamado $order de "$next", ele seria imediatamente reconhecível como Next(), conforme encontrado em todas as outras linguagens que iteram por uma array. $PIECE é um pouco peculiar, mas só porque todos os outros bancos de dados usam campos de tamanho fixo. O conceito das strings delimitadas usadas como construção do banco de dados é estranho para um DBA do SQL. Ao analisar o código de máquina compilado de qualquer uma das formas de banco de dados, as instruções se movem pela string, caractere por caractere, e contando o número de caracteres ou fazendo isso enquanto procuram por um delimitador de campo específico.

$list conseguiu um desempenho um pouco melhor do que $piece, mas à custa de um ou dois bytes adicionais no início de cada campo. No entanto, ainda consumiu menos espaço do que os campos de comprimento fixo. O motivo pelo qual todo código de sistema é escrito em ObjectScript até hoje é porque a linguagem é muito eficiente e legível. Além disso, quando os principais desenvolvedores do Caché/IRIS, Scott Jones, Dave McCalldon e Mo Chung, precisavam de algo onde o ObjectScript fosse inadequado, eles escreviam em C e enterravam no Kernal.

Segundo, se eu tiver uma definição de tabela e campos que exigem alguma forma de formatação ou validação em cima e além das restrições óbvias de tipo e comprimento, então quero escrever esse código e mantê-lo bastante próximo à própria definição do campo. Por que eu entraria em outra linguagem, outro ambiente, para escrever essa validação? Os bancos de dados relacionais usam procedimentos e gatilhos armazenados para lidar com essa validação usando a linguagem SQL para expressar a lógica de validação. Se encontrar um programador que prefira usar SQL para escrever lógica complexa em vez de Basic, C#, C++, ObjectScript ou Python, eu compro uma cerveja para você na próxima vez que eu passar por Viena :-)

Na universidade, aprendi a programar em Fortran e Pascal. Pascal era uma linguagem utilizável e perfeitamente legível. Como um jovem de 19 anos que ficava facilmente empolgado, o fato de que o compilador Pascal pudesse ser escrito nessa linguagem me fascinou. Mais tarde, aprendi COBOL. WTF??? No entanto, tenho um amigo que é desenvolvedor da Sage Accounting e escreve nessa linguagem porque a Sage Accounting foi escrita em COBOL. Páginas e páginas da linguagem mais detalhada, ilegível e inutilizável que já vi. Na verdade, o COBOL ainda é muito usado.

Você poderia pensar que Pascal ultrapassaria facilmente COBAL e até Basic. Mas isso não aconteceu. Por quê? É simples. Essa linguagem não era usada nos aplicativos bancários (o COBOL foi amplamente adotado em grandes aplicativos de processamento em lotes de mainframe, como Accounting). Brincávamos dizendo que os bancos não queriam comprar o modelo Caché porque não era sofisticado o suficiente. Não era porque o ObjectScript não conseguisse fazer o processamento transacional desses aplicativos bancários. Nós éramos comprovadamente mais rápidos do que qualquer tecnologia que eles estivessem usando. O problema era que eles haviam gastado muito dinheiro nos sistemas que tinham e no hardware necessário para executar esses Lotes durante a noite para os bancos abrirem às 9h do dia seguinte. As salas caras dos servidores com gás radônio e sistemas de filtragem removem até mesmo as menores partículas de poeira para que não caiam em um disco ou uma fita magnética, causando a perda de um dia inteiro de transações de contas.

Pascal deveria ter vivido mais do que o Basic e talvez isso tivesse acontecido se a Microsoft não houvesse criado o Visual Basic e entrado em competição com Delphi e Borland. O IDE parecia mesmo com o VB, mas usava Pascal em vez de Basic. Tudo isso estava acontecendo enquanto a Microsoft lançava o C#, porque eles tinham que acomodar todos os programadores de C++ e certamente não conseguiriam conquistá-los com o Basic. Eles também estavam ameaçando lançar a própria versão do Java ou remover a compatibilidade com ele, porque o Java rodar em plataformas de hardware em que o Windows nunca seria capaz de rodar era algo que os incomodava. A Microsoft só recuou quando os avanços da tecnologia tornaram o conceito de máquinas virtuais ou containers uma opção de implantação realista. Então, Pascal e Delphi simplesmente desapareceram. Fiz uma pesquisa rápida no Google e há um interpretador de Pascal para Android, então ele ainda existe.

Considerando que Pascal era uma linguagem, ao contrário do Basic, que era apenas uma linguagem de certa forma. No entanto, ele foi usado pela Microsoft para os scripts de aplicativos como o Excel e uma conexão de propriedade com o SQL Server. Isso permitiu vincular dois ambientes que eram intrinsecamente inadequados sem o incômodo de atender aos padrões ODBC e JDBC. Os padrões eram fortemente apoiados pelo Oracle, Sybase e praticamente todos que precisavam fornecer um gateway para as versões proprietárias do SQL. Assim, o Basic sobreviveu. Estou feliz por ter começado minha carreira de programador com o Pascal. Depois, tive uma grande surpresa ao escrever programas COBOL por um ano trabalhando para uma empresa de seguros. Cheguei às costas úmidas e cinzentas do Reino Unido e comecei meu primeiro emprego, que, por acaso, usava MUMPS. Toda a evolução do MUMPS para CachéObjectScript, Objects, Object Gateways, .Net e Java, Caché Basic e MultiValueBasic e agora Python. O Python fecha um ciclo e, em certo sentido, prova que o ObjectScript não é uma aberração em uma tecnologia de banco de dados não relacional rejeitada.

Caché Globals são arrays esparsas multidimensionais tão convenientes para a própria natureza dos dados de saúde que não importa o esforço do Oracle e da Microsoft em consumir esse espaço no mercado. Ainda que tenham matado Pascal e Fortran, e até Basic, eles não conseguiram matar a InterSystems. Lembro de participar de um seminário do **Oracle** sobre "**Oracle para a Saúde**". A apresentadora falava sobre o Oracle na área da saúde que, ela nos garantiu, dominaria o mercado da saúde de uma vez por todas. Levantei minha mão e perguntei: "**Não é isso o que vocês afirmam em todos os grandes lançamentos há anos? Vocês fracassaram antes. Por que fariam melhor desta vez?"** Ela olhou para mim e perguntou, "**Quem é você?**". Respondi: "**Sou da InterSystems. Dominamos o mercado da saúde há 35 anos. Conseguimos isso porque nossa tecnologia nasceu no Massachusetts General Hospital. E adivinhe. Eles ainda executam os sistemas centrais nas nossas tecnologias**". Nessa hora, dois seguranças corpulentos me tiraram do auditório.

Então, o Oracle com o pSQL tem o Java. A Microsoft tem o SQL Server, C# e tSQL e, quando você precisa interagir com o Java, fica sujeito ao JDBC. Da mesma forma, com Java, se você precisa conversar com tabelas do SQL Server, é obrigado a usar ODBC. Onde ficamos? Bem, chegamos à grande ideia de ter wrappers para .Net e Java. Ao usar o ObjectScript, eu instancio uma instância da Classe A. Na verdade, não importa se a Classe A é uma classe .Net, Java ou ObjectScript, porque eu instancio esses objetos usando exatamente a mesma sintaxe em todos os casos. Depois, invoco os métodos de classe ou instância para manipular os objetos. O interior desses métodos não é importante, porque a sintaxe para interagir com essas classes e os métodos é basicamente idêntica, independentemente do que elas contenham.

Junto vem o Python, que compartilha vários recursos com o ObjectScript, pois é uma linguagem interpretada, e não compilada. É bastante legível e utilizável. Assim como o Caché ObjectScript encontrou um nicho nos dados não estruturados, o Python encontrou um nicho no mundo da modelagem matemática, ML, IA e muito, muito mais. Esse não é um mundo em que C# ou Java se sintam particularmente confortáveis. Aliás, nem o ObjectScript. Assim, a InterSystems foca em fornecer funcionalidades cada vez mais poderosas para manipular grandes quantidades de dados não estruturados, além de lançar iFind e iKnow, algumas técnicas de indexação muito inteligentes e algoritmos de correspondência de probabilidade. Então, você convida o Python para se aconchegar nas nossas arrays esparsas multidimensionais, trazendo milhões de bebês .py que fazem praticamente tudo o que você precisa. É a combinação perfeita. Por precaução, esqueci de mencionar que várias arquiteturas que dominam o mundo do desenvolvimento de páginas da Web são completamente baseadas em JS (Angular.js, REACT.js, Vue.js, Bootstrap — certo, não há JS, mas ele está em tudo menos no nome — e Node.js) e Arrays de JS. O JS não vai desaparecer tão cedo. No entanto, será interessante ver o que vai acontecer com a Golang, se é que você me entende. Vi entradas baseadas em arrays de JS nas últimas competições de código. Se existe uma tecnologia que entende arrays melhor do que qualquer outra, é a IRIS.

Penso naqueles dias, sentado no meu escritório na empresa onde trabalhava no coração de Londres. Em determinado momento, a empresa estava cheia de programadores MUMPS, mas ela os transformou em programadores SQL relacionais, que depois se tornaram redundantes. Lembro da sensação de começar a questionar se minha fé no MUMPS ser simplesmente a melhor linguagem que já havia encontrado poderia estar errada. A linguagem e as empresas que construíram interpretações dessa linguagem morreriam. Isso me deixou muito triste porque, até então, havia aprendido cinco outras linguagens de programação (APL, Basic, Fortran, COBOL e Pascal) antes de descobrir o MUMPS, e o MUMPS era tão simples. Fácil de escrever, ler e implantar. Em outras palavras, era tão natural para mim quanto o inglês, e tinha um ritmo parecido com os hinos que cantávamos na escola metodista que frequentei:

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!

Mas ele não morreu. A música mudou um pouco:

A bordo, Nigel Saaalllm

Voando para a guerra

Com seu cartão de crédito internacional

Indo à frente.

John, o Mestre, McCormick

Lidera contra o inimigo (Microsoft)

Avançando na batalha

Veja seus duty frees partindo

CacheObjectScript era ainda melhor que o MUMPS se isso era possível. CacheObjects parecia tão legal quando demonstrado para uma audiência pelas primeiras vezes, e CacheSQL deixou seus dias de M/SQL para trás e melhorou muito ao longo dos anos. Ainda assim, eu particularmente não gosto de escrever muito em SQL, mas encontrei um bom equilíbrio entre Objects, SQL e referências globais diretas conforme relaxei. Meu código era bastante orientado para as referências globais diretas, com um pouco de OO e o mínimo de SQL. Quando vi com os produtos que o código gerado era compacto, elegante, eficiente e legível, o equilíbrio mudou novamente. Agora, uso nomes globais diretos raramente, muito Objects e uma quantidade razoável de SQL.

*Para trabalhar com Python, minha mente precisará ver padrões diferentes do meu código ObjectScript. Há muitos "**abc**" e outras estruturas estranhas. No entanto, depois de escrever algumas páginas de código py e me afastar, como faço ao pintar a óleo, os padrões saltarão. Assim como vejo a música como sinestesia de cores, meus programas **py** codificados por cores fluindo pela página também começarão a parecer aquarela ou até mesmo uma pintura a óleo pesada. Ficarei encantado, e estará tudo bem.*

#Globais #Guia de Codificação #ObjectScript #Python #Global Masters #InterSystems IRIS

URL de origem:https://pt.community.intersystems.com/post/por-que-eu-amo-objectscript-e-por-que-eu-acho-que-poderia-amar-python-ainda-mais