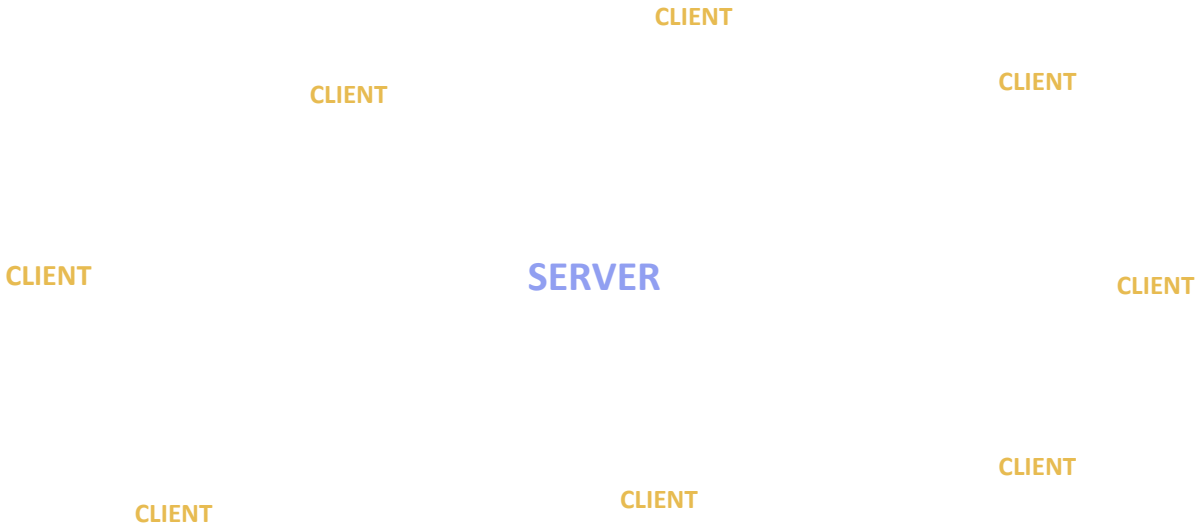
For each em profundidade

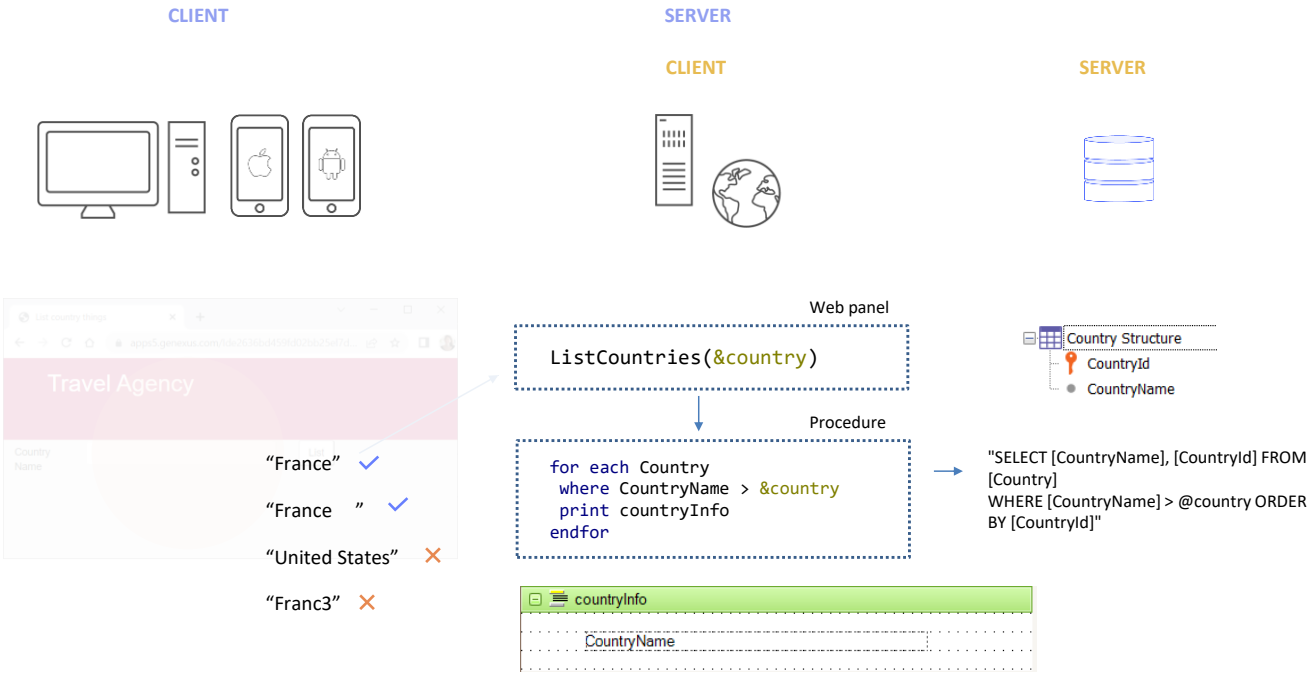
Onde é resolvida a consulta?

GeneXus™

Queremos revisar e aprofundar na lógica do acesso à base de dados, para o que será conveniente colocar um pouco de luz sobre algumas questões que não foram ditas até agora.



Por exemplo, falamos sobre cliente e servidor em muitas ocasiões, mas dependerá do contexto que nos referimos com estes termos e será necessário esclarecê-los. São termos relacionados: se existe um servidor é porque existem clientes aos quais este servidor fornece.



Para ficar mais claro: pensemos em uma aplicação desenvolvida para ambiente web onde temos um Web panel que chama um procedimento que lista os países da base de dados.

O programa que comanda o Web Panel é executado no servidor da aplicação, embora invocado a partir do cliente, através do Browser. Quando o usuário insere um valor no campo e pressiona o botão, a parte do programa do web panel no cliente ativa o programa no server que executa o código do evento, invocando o procedimento, que também é atendido por esse servidor da aplicação.
O procedimento deve executar o for each. Isso significa que deve, por sua vez, chamar o DBMS para que o DBMS realize a consulta necessária à tabela de países. O DBMS está localizado em outro servidor, o de base de dados (embora ambos, o da aplicação e este, o de base de dados, eventualmente possam estar localizados na mesma máquina).

Então quando o procedimento é gerado, é construída uma instrução SQL que depois, quando o procedimento é executado, é enviada para o DBMS no servidor de base de dados, para que o DBMS a resolva (claro, supondo que estamos utilizando um DBMS que fala a linguagem SQL).

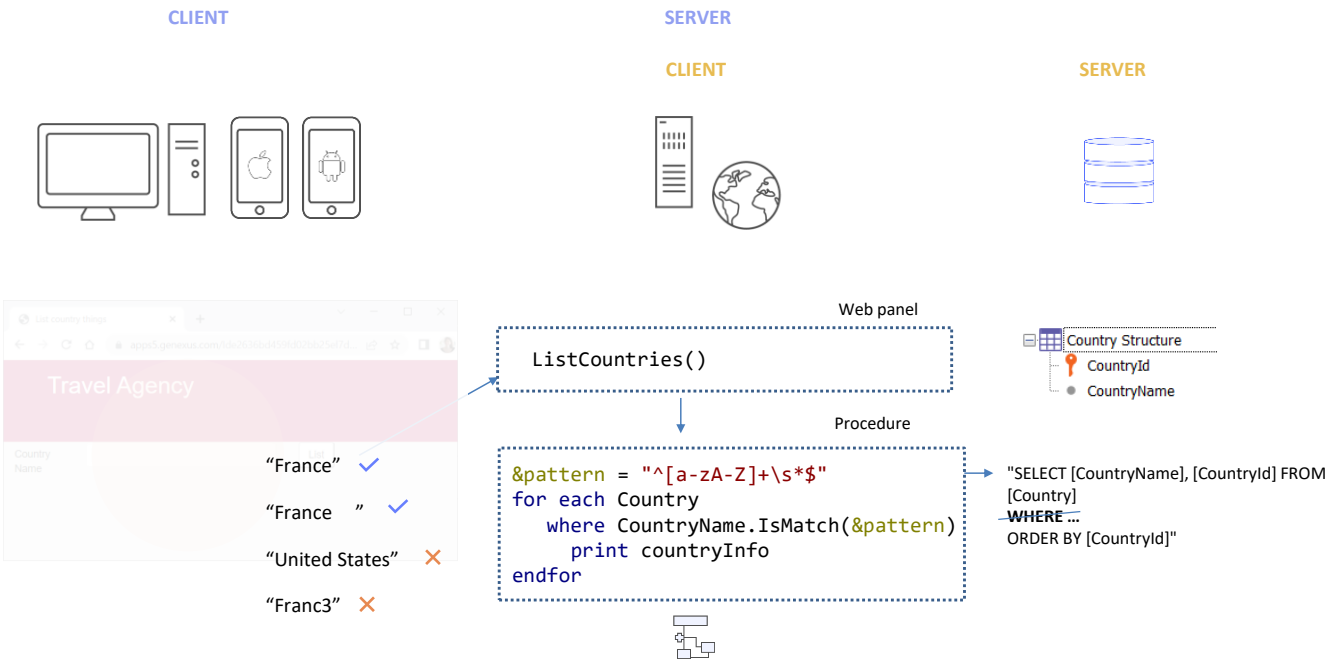
Deste ponto de vista temos o **servidor** de base de dados, e o **cliente** desta consulta é o procedimento, no servidor da aplicação.

Fazer esta distinção pode parecer desnecessário à primeira vista, mas veremos que não é quando a coisa fica mais complexa.

De forma simplificada, vamos pensar assim: no exemplo estamos pedindo para percorrer a tabela Country, filtrando por CountryName (suponhamos que no printblock mostramos apenas o valor desse atributo). É muito diferente enviar a consulta como um Select para o DBMS, que realiza a consulta e retorna para o cliente o resultado já filtrado e ordenado, do que enviar de uma forma em que o DBMS retorne todos os países e o filtro por CountryName tenha que ser feito no cliente, por exemplo. A quantidade de informação que viaja do servidor de base de dados para o cliente pode tornar-se desequilibrante.

Aqui nos parece óbvio que a partir do for each do procedimento, o especificador de GeneXus criará um fonte com esse select que contém um where, para que seja executado pelo DBMS, resolvendo a consulta em uma única operação... mas isto nem sempre será assim.

Por exemplo, se não queremos os países com nome superior a uma string, mas aqueles cujos nomes satisfazem uma expressão regular (por exemplo, que só pode ser uma palavra, composta pelas letras de “a” a “z”, maiúsculas ou minúsculas, mas nenhuma outra possibilidade, ou seja, que “France” satisfaça esse padrão, ou mesmo “France” seguida de espaços em branco, mas não “United States”, pois após o primeiro espaço vem outra palavra, e claro que não “Franc3”, pois contém um dígito)… bem, se quiséssemos isto, teríamos que programar o Source assim:



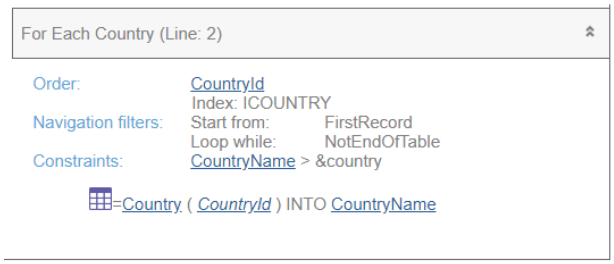
...onde na variável do tipo character estabelecemos a expressão regular (aqui não vamos estudá-las, mas com esta sintaxe estamos indicando que deve começar com uma letra entre a "a" minúscula e a "z" minúscula, ou entre a "A" maiúscula e a "Z" maiúscula, e que será uma repetição de caracteres destas classes, e então poderá ter zero ou mais espaços em branco seguidos e terminar. Ou seja, não pode ser composta de várias palavras, nem de palavras com outros caracteres que não sejam esses.

Tendo esta expressão regular sempre podemos aplicar a um atributo ou variável o método **IsMatch** para saber se seu conteúdo corresponde ou não a esse padrão. Por exemplo, se no atributo CountryName tivermos France, corresponderá, ou France com espaços, mas se em vez disso tivermos "United States", não corresponderá, nem esta outra.

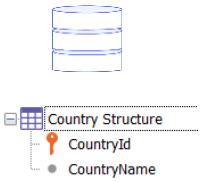
Portanto aqui estamos querendo listar apenas os países que têm uma única palavra em seu nome.

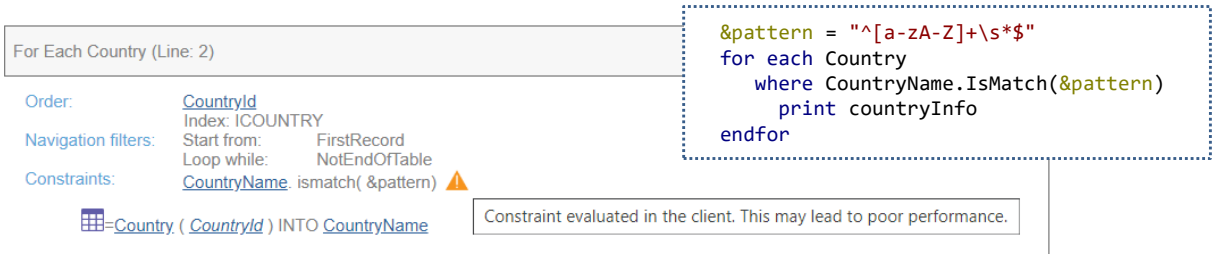
Poderíamos pensar que no Select que enviamos ao DBMS haverá um WHERE como existia no outro caso, de forma que o DBMS filtre os registros de Country de acordo com esse método **IsMatch**. No entanto, se o DBMS for SQLServer, ele não entenderá esse método. Não existe em sua linguagem. GeneXus sabe disso, então a instrução Select que envia ao Server de base de dados deverá ser SEM o where. E será o fonte gerado que, com todos os países retornados pela consulta (ou seja, todos os da tabela), realizará o filtro. Ou seja, o filtro será realizado no cliente, que é o que pode executar o método IsMatch.

No SQLServer são muito poucos os métodos que não podem ser resolvidos pelo DBMS. Este é um deles. Dependerá do DBMS utilizado quais funções e métodos pode resolver e quais não pode. O interessante é que não precisamos saber com antecedência, pois a lista de navegação nos avisará.




```
for each Country
 where CountryName > &country
 print countryInfo
endfor
```



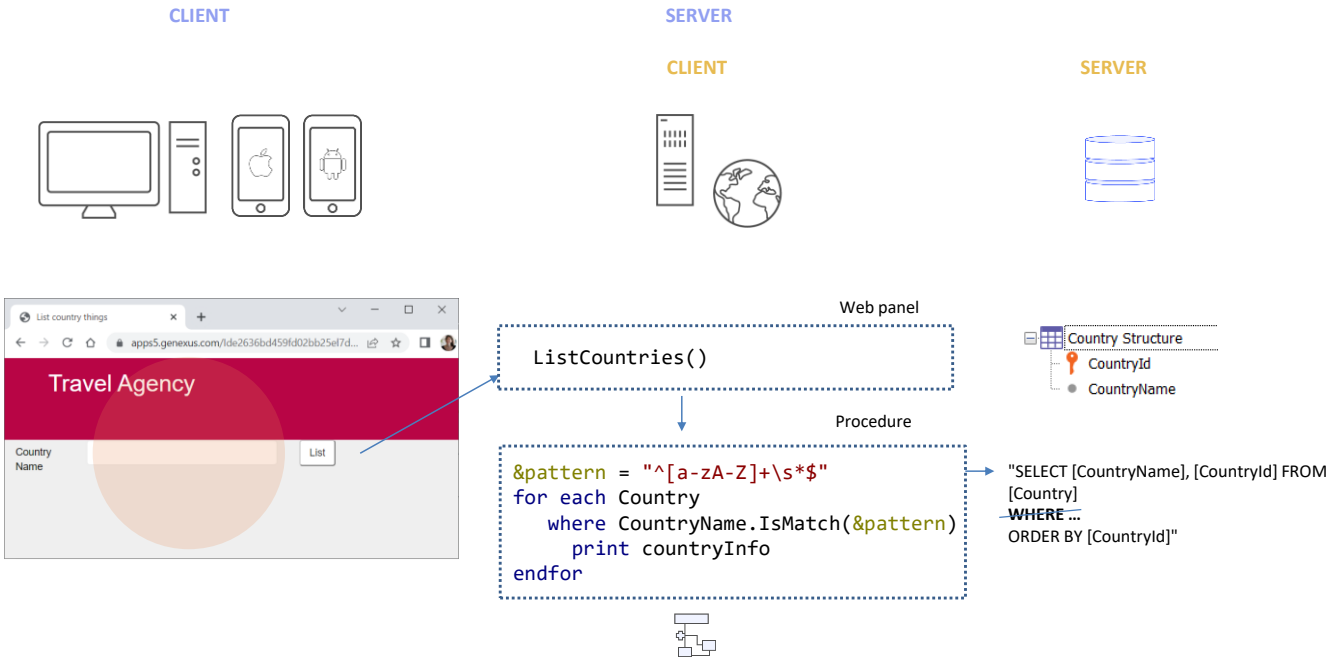



```
&pattern = "^[a-zA-Z]+\s*$"
for each Country
   where CountryName.IsMatch(&pattern)
     print countryInfo
endfor
```

Assim, se compararmos a lista de navegação do primeiro caso com o segundo, vemos que no segundo nos é mostrado um aviso que nos indica precisamente que o filtro não poderá ser aplicado no Server de base de dados, mas em seu cliente, ou seja, no programa correspondente ao procedimento e que isso poderia levar a um mau desempenho.

Pensemos no caso extremo em que há apenas um país que satisfaz o filtro, mas na tabela de países houvesse milhões de registros. Esses milhões viajariam do DBMS para o procedimento e, além disso, o procedimento teria que percorrer todos, um por um, para finalmente ficar com **o** registro para apresentar na saída.

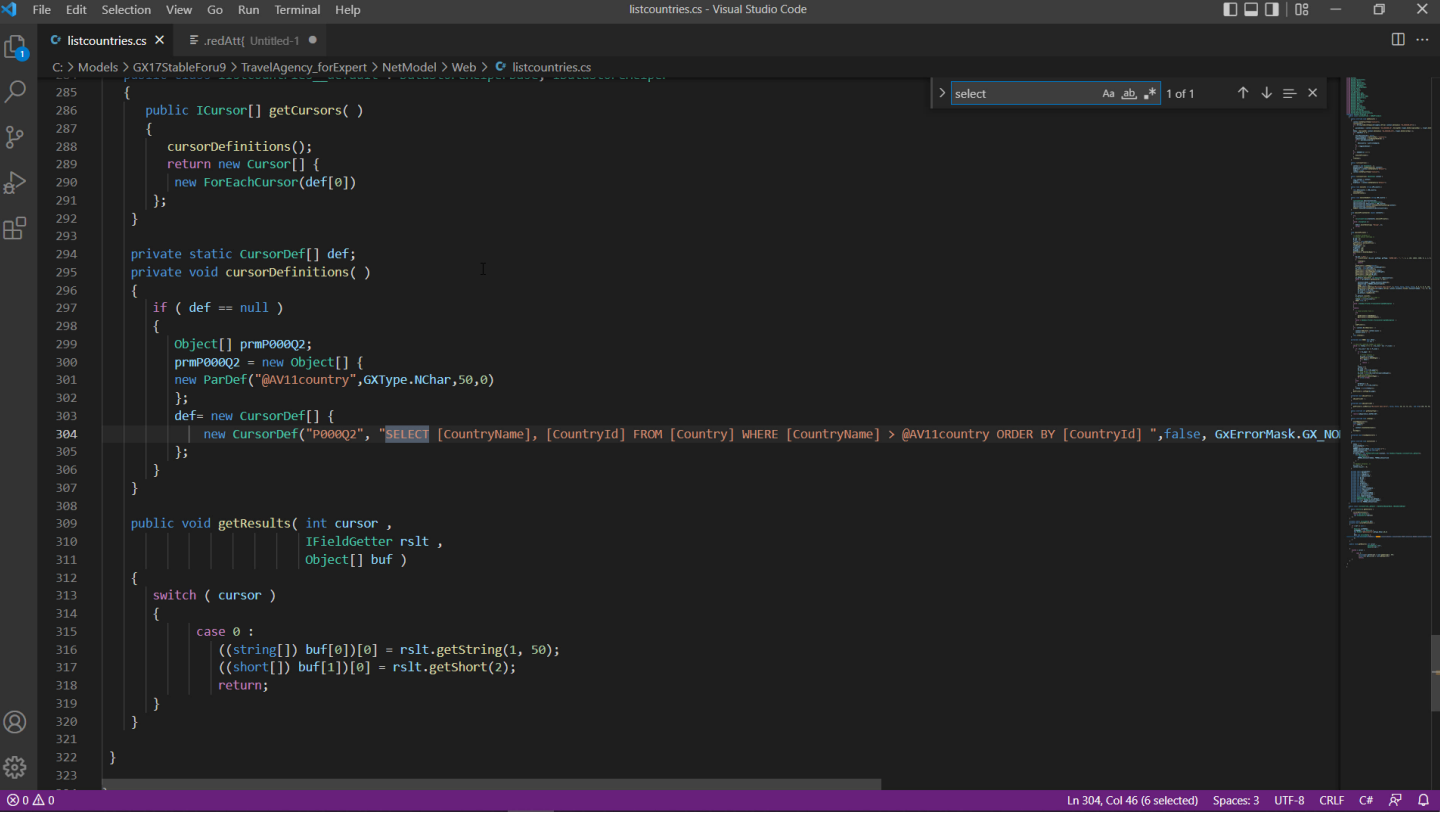
Nesse caso, pode ser que queiramos melhorar as coisas procurando alguma estratégia que mitigue esse impacto.



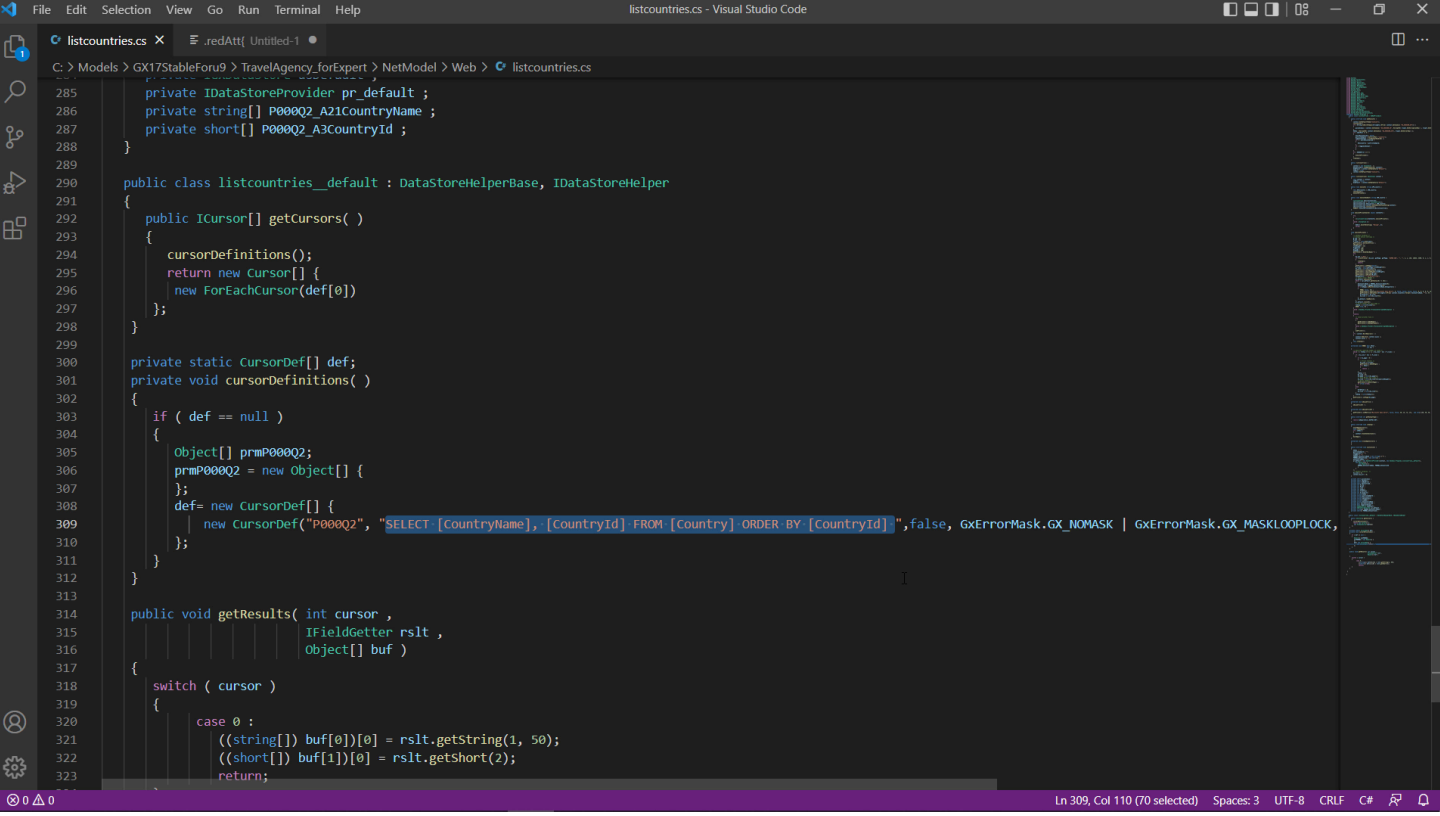
O que queríamos mostrar com estes exemplos é que por um lado temos o servidor da aplicação cujo cliente é o Browser no dispositivo físico do usuário final, e por outro o servidor de base de dados, cujo cliente é a aplicação que é executada no servidor de aplicações, e que a maneira como programamos os acessos à base de dados terá impactos no desempenho.

Nós desenvolvedores GeneXus escrevendo um for each no Source, e então, sabendo para qual environment será construída a aplicação, o especificador GeneXus virá a determinar como deverá ser construído o fonte. Lá toma decisões que dependem da plataforma de programação, mas também do DBMS com o qual se conectará.

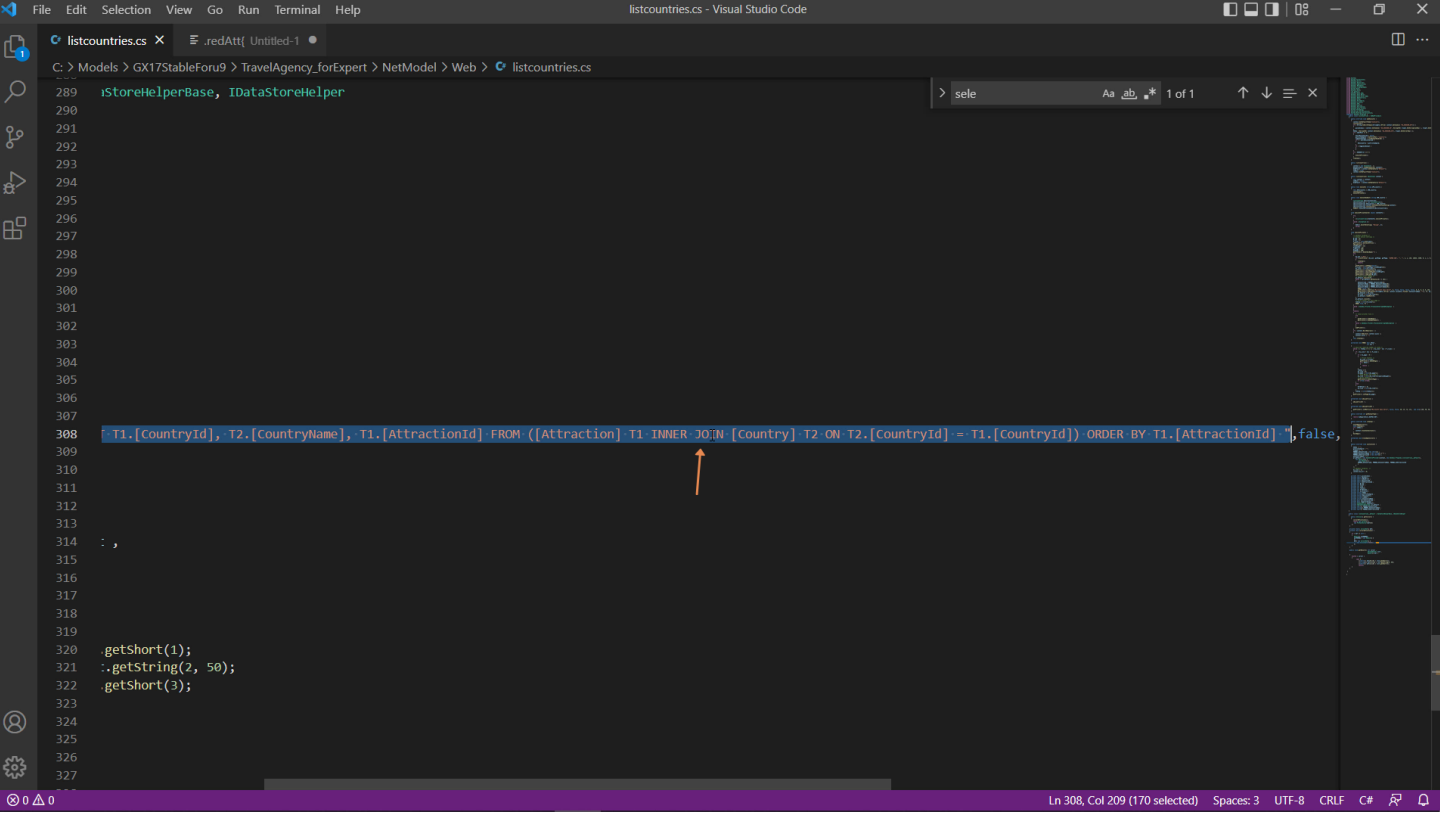
Se estivermos curiosos, sempre podemos inspecionar o fonte e ver a instrução sql.



Por exemplo, aqui temos a primeira proposta… Pedimos que construa o programa fonte… o procuramos no diretório do environment (que é Net com SqlServer), o abrimos… e procuramos o select… aqui o vemos. Existe a consulta completa.



Se em vez disso modificarmos o source para a segunda proposta e fizermos o mesmo… vemos que o select não está incluindo o where… o que significa que é dentro deste fonte que está sendo resolvido o filtro.



E agora se, por exemplo, mudarmos a tabela base do for each para que seja Attraction, de forma que imprima todos os nomes de país das atrações (claro que serão repetidos se várias tiverem o mesmo país)...

Vemos na lista de navegação que como a tabela percorrida será Attraction, então terá que acessar a tabela Country para obter para cada atração seu país, e assim poder imprimir o valor do CountryName para essa atração.
Esta operação nas bases de dados relacionais é chamada de Join. E vemos que por isso aparece esta indicação de Join location, que nos diz onde é realizado esse join, se no servidor de base de dados ou no cliente. Aqui indica que é no servidor, então se procurarmos o fonte gerado, veremos que é enviado o select inteiro para a base de dados para que ali seja resolvido o join. Isso é muito menos custoso do que se tivesse que ser resolvido pelo cliente, ou seja, este procedimento.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$*, *att$_2$*, … , *att$_n$* **[when** *condition***]**
order *att$_1$*, *att$_2$*, … , *att$_n$* **[when** condition**]**
order none
unique *att$_1$*, *att$_2$*, … , *att$_n$*
using *DataSelector* **(** *parm$_1$*, *parm$_2$*, … , *parm$_n$* **)**
where *condition* **[when** *condition***]**
where *condition* **[when** *condition***]**
where *att* **IN** *DataSelector* **(** *parm$_1$*, *parm$_2$*, … , *parm$_n$* **)**
blocking *n*
main_code
when duplicate
when_duplicate_code
when none
when_none_code

endfor

Tendo tudo isto em mente, vamos agora repassar a sintaxe do comando For each e para que é utilizada cada coisa, com a intenção de integrar o que já sabemos e ir um pouco além.