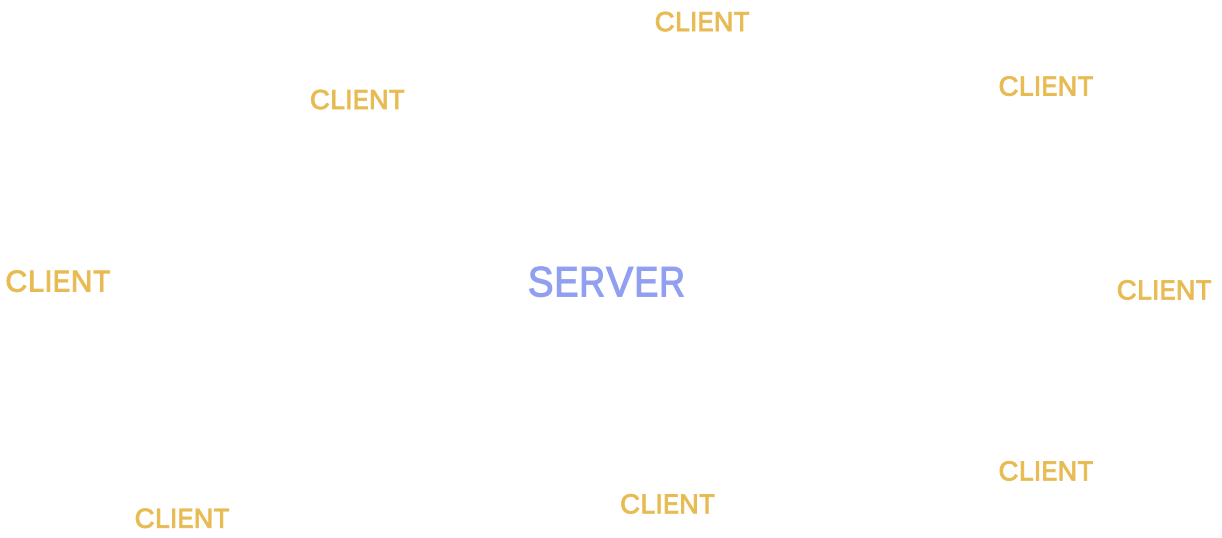
For each en profundidad

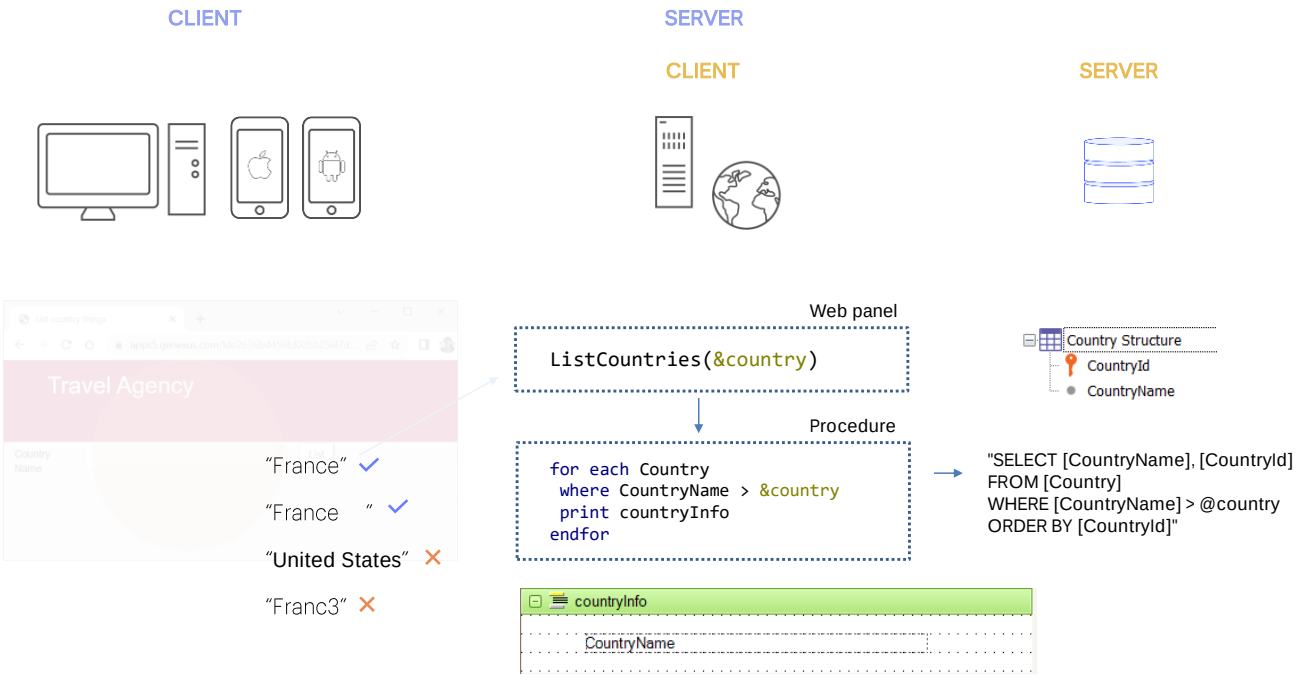
¿Dónde se resuelve la consulta?

GeneXus™

Queremos repasar y profundizar en la lógica del acceso a la base de datos, para lo que será conveniente echar un poco de luz sobre algunas cuestiones que no se han dicho hasta el momento.



Por ejemplo, hemos hablado de cliente y servidor en muchas oportunidades, pero va a depender del contexto a qué nos referimos con estos términos y será preciso aclararlos. Son términos relacionados: si hay un servidor es porque hay clientes a los que este servidor provee.



Para que sea más claro: pensemos una aplicación desarrollada para ambiente web donde tenemos un Web panel que llama a un procedimiento que lista los países de la base de datos.

El programa que comanda al Web panel se ejecuta en el servidor de la aplicación, aunque invocado desde el cliente, a través del Browser. Cuando el usuario ingresa un valor en el campo y presiona el botón, la parte del programa del web panel en el cliente activa al programa en el server quien ejecuta el código del evento, invocando al procedimiento, que también es servido por ese servidor de la aplicación.
El procedimiento debe ejecutar el for each. Eso signfica que debe, por su parte, llamar al DBMS para que el DBMS realice la consulta requerida a la tabla de países. El DBMS se encuentra en otro servidor, el de base de datos (aunque ambos, el de la aplicación y éste, el de base de datos, eventualmente puedan encontrarse en la misma máquina).

Entonces cuando el procedimiento es generado, se construye una sentencia SQL que luego, cuando el procedimiento es ejecutado le envía al DBMS en el servidor de base de datos, para que el DBMS la resuelva (claro, suponiendo que estemos utilizando un DBMS que habla lenguaje SQL).

Desde esta óptica tenemos al **servidor** de base de datos, y el **cliente** de esta consulta es el procedimiento, en el servidor de la aplicación.

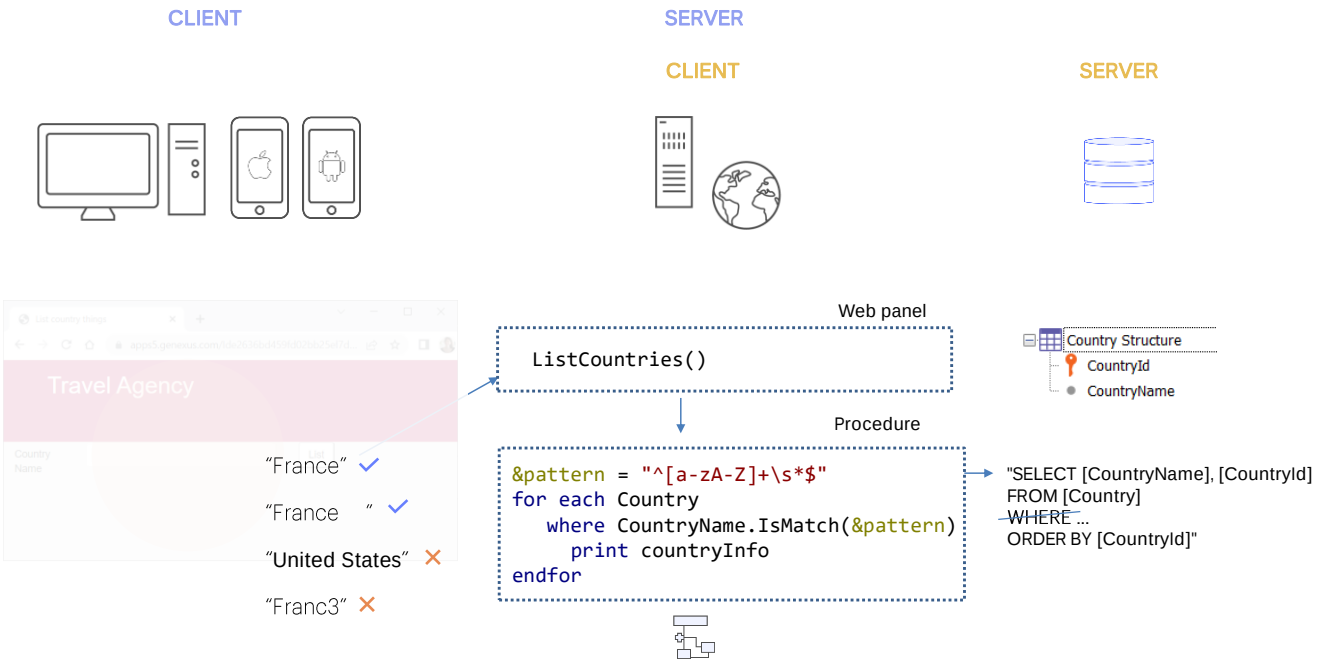
Hacer esta distinción puede parecer innecesario a primera vista, pero ya

veremos que no cuando la cosa se complejice.

De manera simplificada pensémoslo así: en el ejemplo estamos pidiendo recorrer la tabla Country, filtrando por CountryName (supongamos que en el printblock solamente mostramos el valor de ese atributo). Es muy distinto enviar la consulta como un Select al DBMS, que realiza la consulta y devuelve al cliente el resultado ya filtrado y ordenado, que enviarla de un modo en el que el DBMS devuelve todos los países y el filtro por CountryName se tenga que hacer en el cliente, por ejemplo. La cantidad de información que viaja del servidor de base de datos al cliente puede llegar a ser desequilibrante.

Aquí nos resulta una obviedad que del for each del procedimiento el especificador de GeneXus creará un fuente con ese select que contiene un where, para que sea ejecutado por el DBMS resolviendo la consulta en una sola operación... pero esto no siempre será así.

Por ejemplo, si no queremos los países de nombre mayor a un string, sino aquellos cuyos nombres satisfagan una expresión regular (por ejemplo, que solo pueda tratarse de una palabra, compuesta por letras de la "a" la "z", mayúsculas o minúsculas, pero ninguna otra posibilidad, es decir, que "France" satisface ese patrón, o incluso "France" seguida de espacios en blanco, pero no "United States", pues después del primer espacio viene otra palabra, y por supuesto no "Franc3", pues contiene un dígito)... bueno, si quisiéramos esto, entonces, tendríamos que programar el Source así:



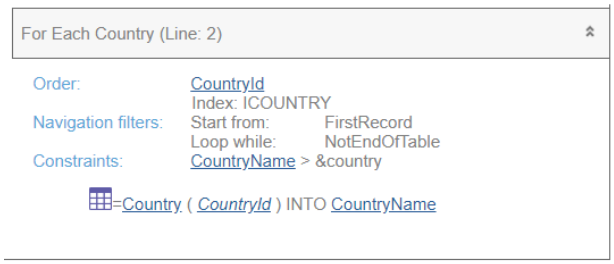
...donde en la variable de tipo character establecemos la expresión regular (aquí no las estudiaremos, pero con esta sintaxis estamos indicando que debe comenzar por una letra entre la "a" minúscula y la "z" minúscula, o entre la "A" mayúscula y la "Z" mayúscula, y que será una repetición de caracteres de estas clases, y luego podrá tener cero o más espacios en blanco seguidos y terminar. O sea, no puede componerse de varias palabras, ni de palabras con otros caracteres que no sean esos.

Teniendo esta expresión regular siempre podemos aplicar a un atributo o variable el método **IsMatch** para saber si su contenido matchea o no con ese patrón. Por ejemplo, si en el atributo CountryName tenemos a France, matcheará, o a France con espacios, pero si en cambio tenemos a "United States", no lo hará, como tampoco esta otra.

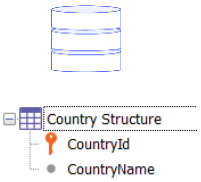
Por tanto aquí estamos queriendo listar únicamente los países que tienen una sola palabra en su nombre.

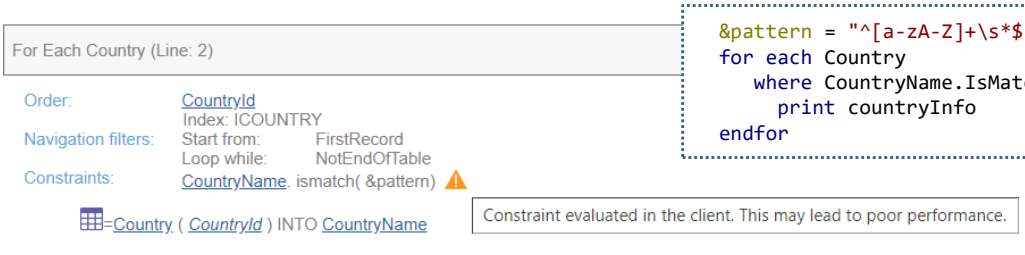
Podríamos pensar que en el Select que enviamos al DBMS habrá un WHERE como existía en el otro caso, de modo que el DBMS filtre los registros de Country de acuerdo a ese método **IsMatch**. Sin embargo, si el DBMS es SQLServer, éste no comprenderá ese método. No existe en su lenguaje. GeneXus lo sabe, por lo que la sentencia Select que envía al Server de base de datos tendrá que ser SIN el where. Y será el fuente generado el que con todos los países devueltos por al consulta (es decir, todos los de la tabla), realizará el filtro. Es decir, el filtro se realizará en el cliente, que es el que puede ejecutar el método IsMatch.

En SQLServer son muy pocos los métodos que no puede resolver el DBMS. Este es uno de ellos. Va a depender del DBMS utilizado qué funciones y métodos puede resolver y cuáles no. Lo interesante es que no tenemos por qué saberlo de antemano ya que el listado de navegación nos lo advertirá.




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



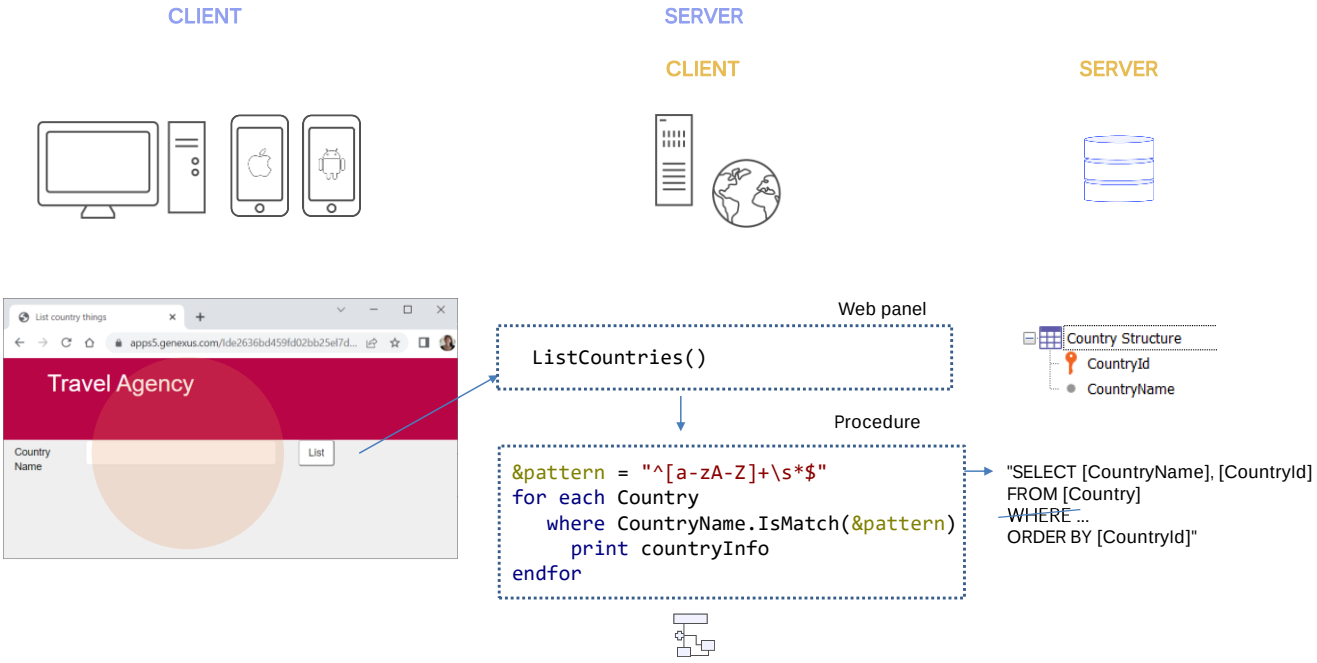



```
&pattern = "^[a-zA-Z]+\s*$"
for each Country
   where CountryName.IsMatch(&pattern)
     print countryInfo
endfor
```

Así, si comparamos el listado de navegación del primer caso con el segundo, vemos que en el segundo se nos muestra una advertencia que nos indica justamente que el filtro no podrá aplicarse en el Server de base de datos, sino en su cliente, es decir, en el programa correspondiente al procedimiento y que esto podría conducir a una mala performance.

Pensemos en el caso extremo en que haya un único país que satisfaga el filtro, pero que en la tabla de países hubiera millones de registros. Esos millones viajarían del DBMS al procedimiento, y además el procedimiento tendría que recorrerlos todos, uno por uno, para finalmente quedarse con **el** registro para presentar en la salida.

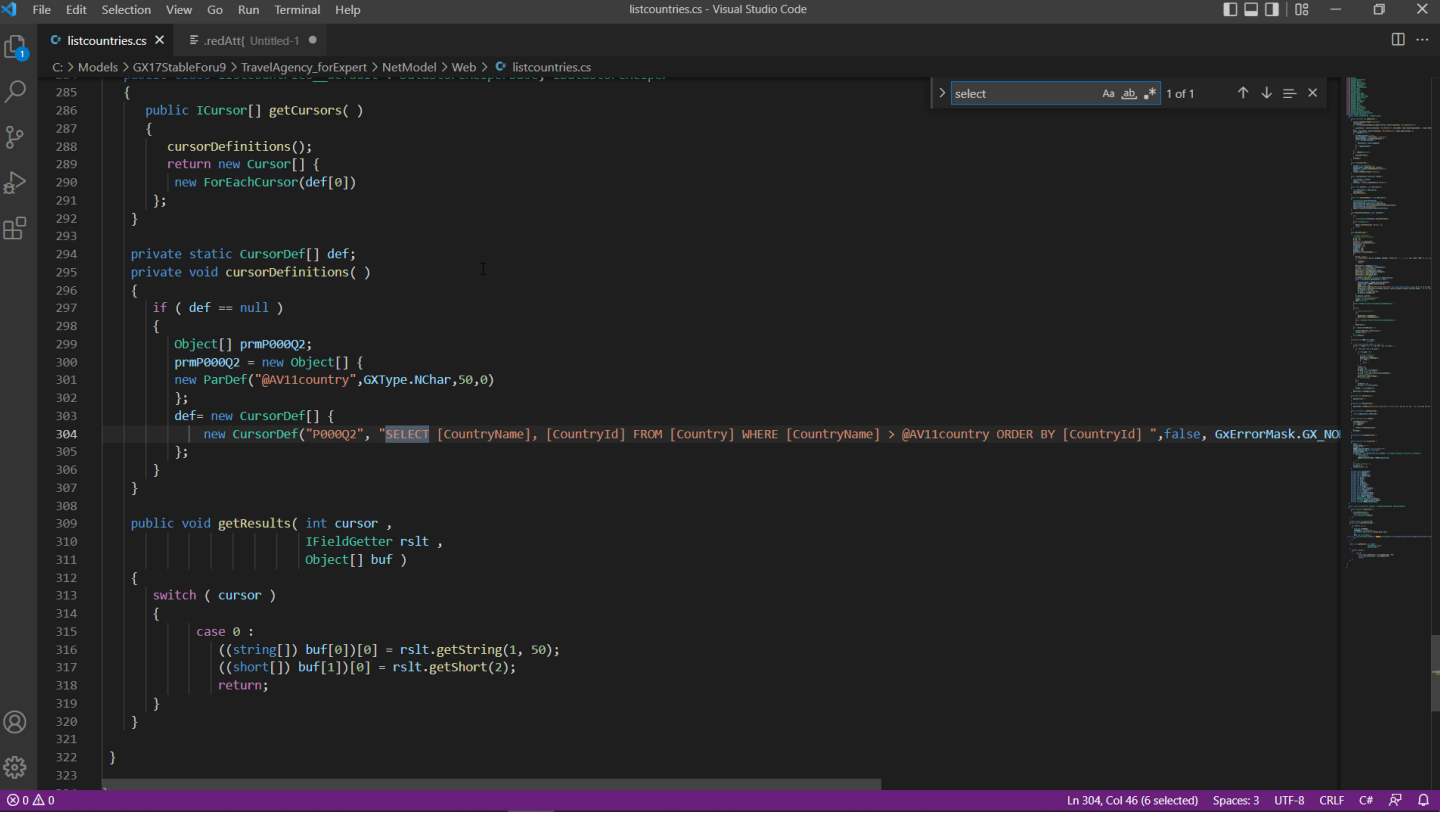
En ese caso puede ser que queramos mejorar las cosas buscando alguna estrategia que mitigue ese impacto.



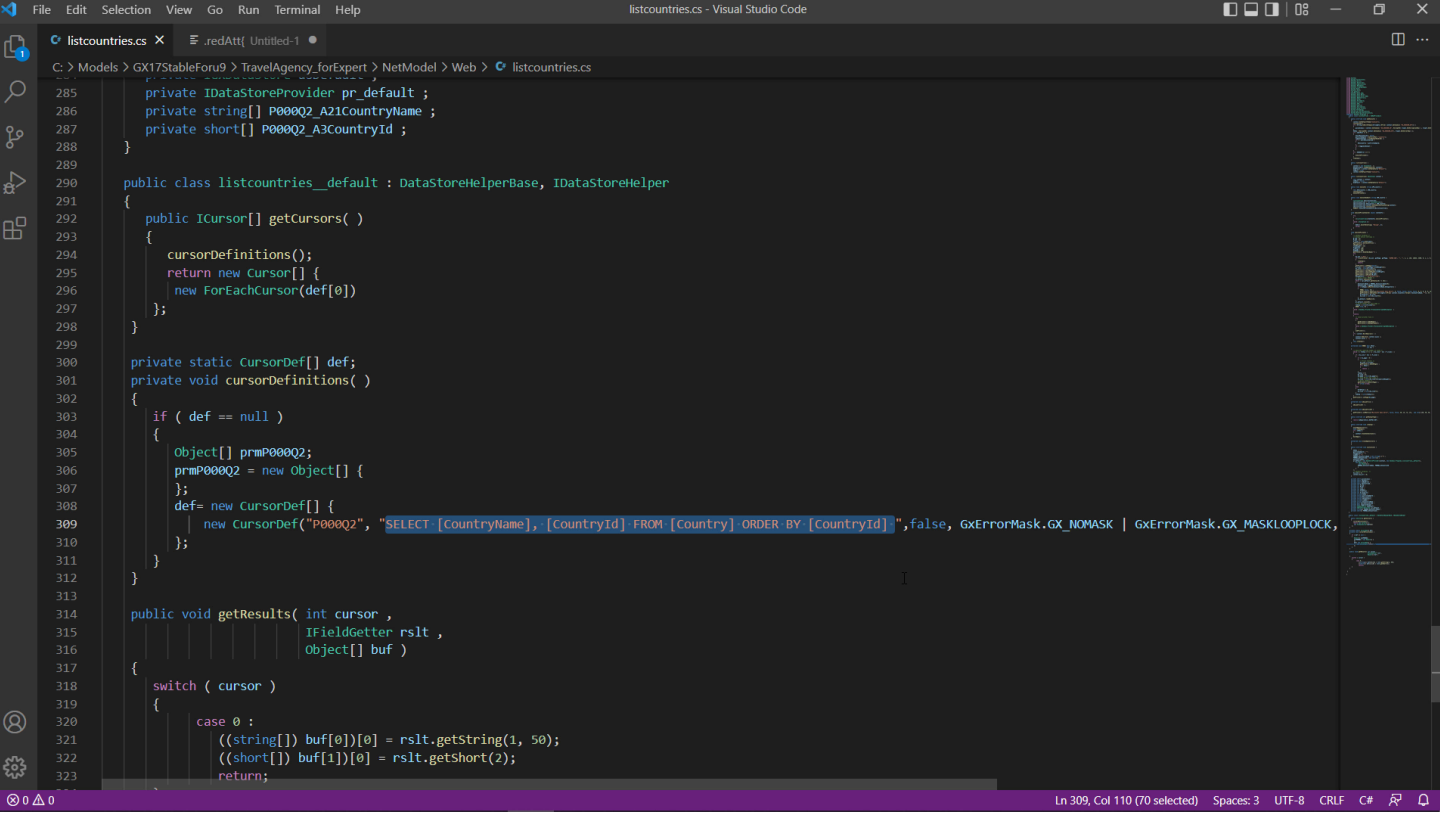
Lo que queríamos mostrar con estos ejemplos es que por un lado tenemos el servidor de la aplicación cuyo cliente es el Browser en el dispositivo físico del usuario final, y por otro el servidor de base de datos, cuyo cliente es la aplicación que se ejecuta en el servidor de aplicaciones, y que la forma en que programemos los accesos a la base de datos tendrán repercusiones en la performance.

Los desarrolladores GeneXus escribimos un for each en el Source, y luego, sabiendo para qué environment se construirá la aplicación, vendrá el especificador de GeneXus a determinar cómo deberá construir el fuente. Allí toma decisiones que dependen de la plataforma de programación, pero también del DBMS con el que se conectará.

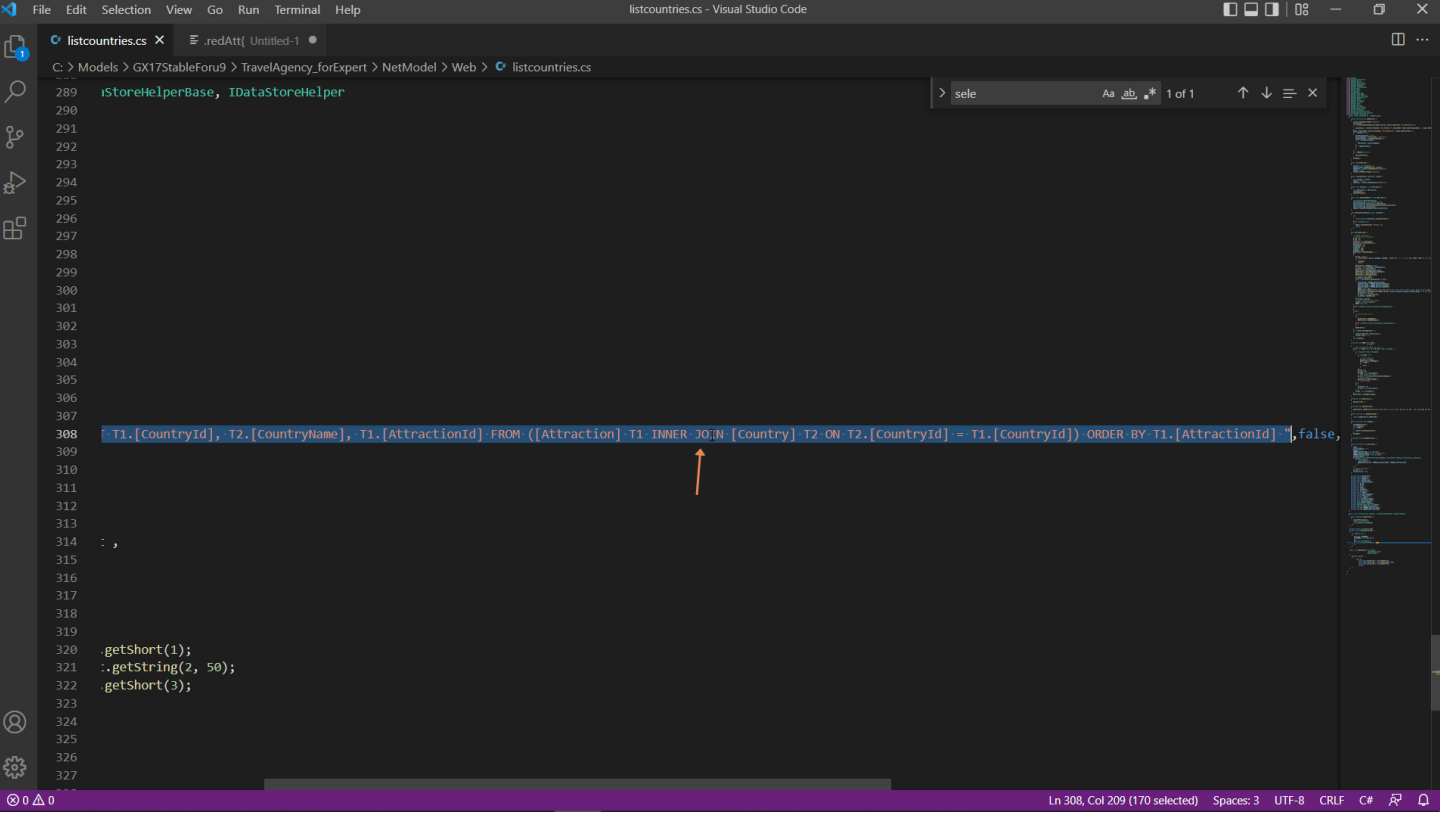
Si nos da curiosidad, siempre podemos inspeccionar el fuente y ver la sentencia sql.



Por ejemplo, aquí tenemos la primera propuesta... Pedimos que construya el programa fuente... lo buscamos en el directorio del **environment (que es Net contra SqlServer**), lo abrimos... y buscamos el **select**... aquí lo vemos. **Está la consulta completa.**



Si en cambio modificamos el source para la segunda propuesta y hacemos lo mismo... vemos que el **select no está incluyendo el where**... lo que significa que es dentro de este fuente que se está resolviendo el filtro.



Y ahora si, por ejemplo, cambiamos la tabla base del for each para que sea Attraction, de modo que imprima todos los nombres de país de las atracciones (por supuesto saldrán repetidos si varias tienen al mismo país)...

Vemos en el listado de navegación que como la tabla recorrida será Attraction, entonces tendrá que acceder a la tabla Country para obtener para cada atracción su país, y así poder imprimir el valor del CountryName para esa atracción.
Esta operación en las bases de datos relacionales se llama Join. Y vemos que por eso aparece esta indicación de Join location, que nos dice dónde se realiza ese join, si en el servidor de base de datos, o en el cliente. Aquí nos indica que es en el servidor, por lo que si buscamos el fuente generado, veremos que se envía el select entero a la base de datos para que allí se resuelva el join. Eso es mucho menos costoso que si lo tuviera que resolver el cliente, es decir, este procedimiento.

```
For each  BaseTrn1 , ... , BaseTrnn
    skip expression1 count expression2
    order  att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where  condition [when condition]
    where  condition [when condition]
    where  att  IN DataSelector ( parm1, parm2, ... , parmn )
    blocking n
            main_code
    when duplicate
            when_duplicate_code
    when none
            when_none_code
endfor
```

Teniendo todo esto en mente, pasemos ahora a repasar la sintaxis del comando For each y para qué se utiliza cada cosa, con la intención de integrar lo que ya sabemos e ir un poco más allá.