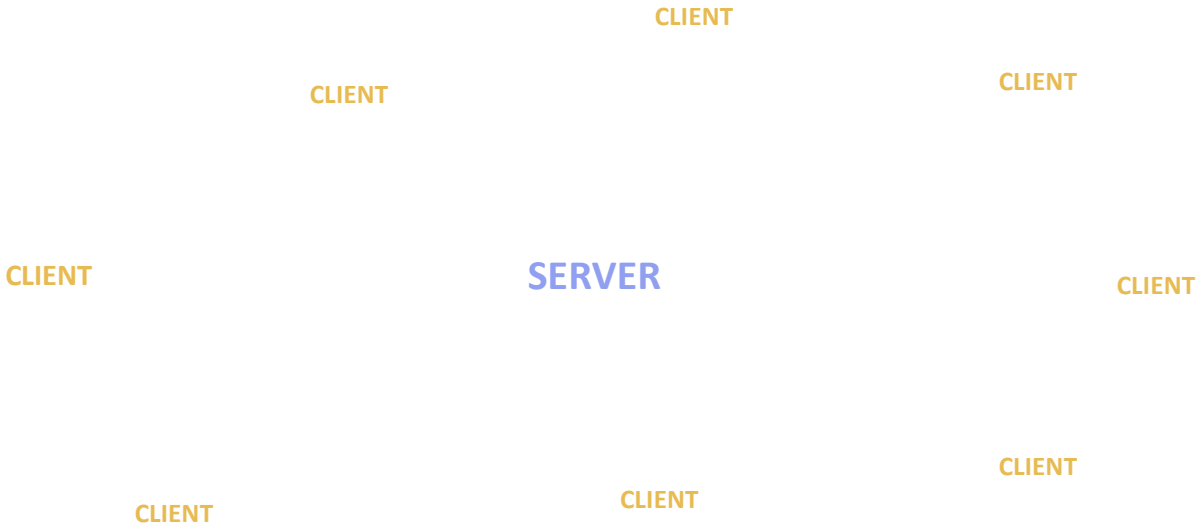
For each command in depth

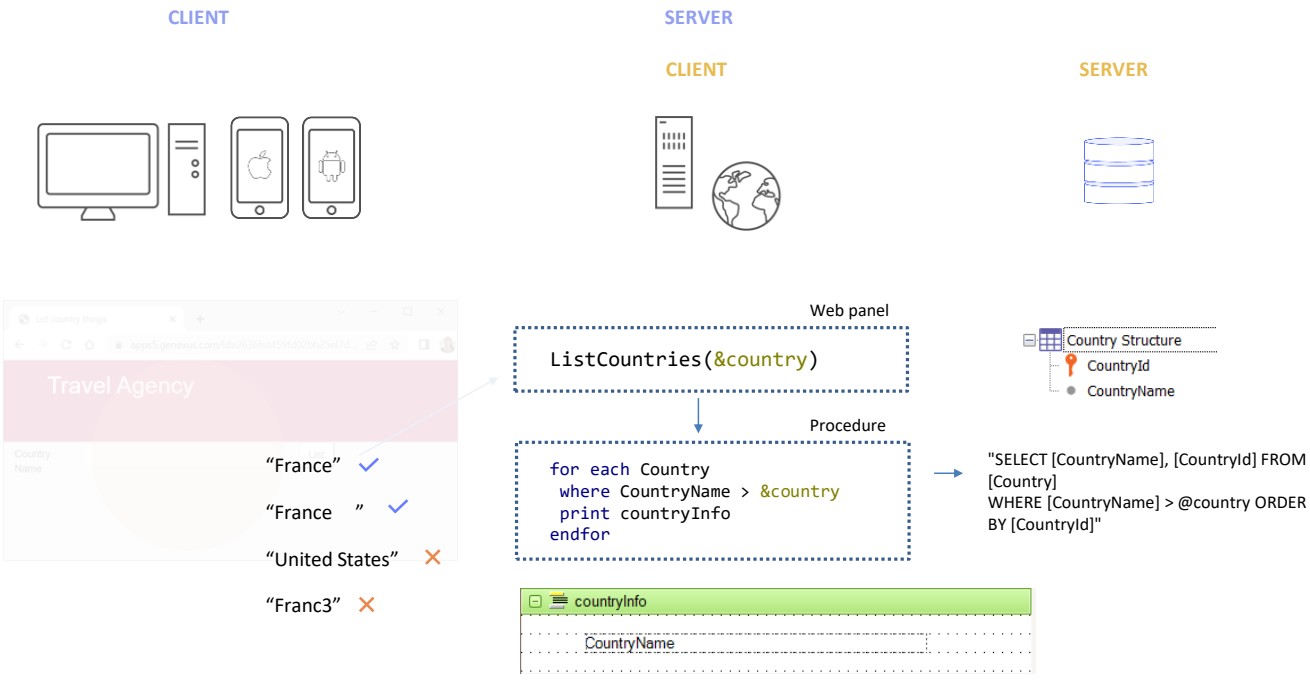
Where are queries resolved?

GeneXus™

We want to review and dive deeper into the logic of database access, and to do so it will be convenient to clarify some aspects that haven't been mentioned so far.



For example, we have talked about client and server on many occasions, but what we mean by these terms will depend on the context and will need to be clarified. They are related terms: if there is a server, then there are clients that it serves.



To make it clearer: let's think of an application developed for a web environment which has a Web panel that calls a procedure to list the countries in the database.

The program that commands the Web panel is executed on the application server, although it is invoked from the client, through the browser. When the user enters a value in the field and clicks the button, the web panel part on the client activates the program on the server that executes the event code, invoking the procedure, which is also served by that application server.
The procedure must run the For each. This means that it must, in turn, call the DBMS so that the DBMS makes the required query to the country table. The DBMS is located on another server—the database server (however, both the application server and the database server may occasionally be located on the same machine).

When the procedure is generated, a SQL statement is built. Then, when the procedure is executed, it is sent to the DBMS on the database server, for the DBMS to resolve it—of course, assuming that we are using a DBMS that understands SQL.

From this perspective, we have the database **server**, and the **client** of this query is the procedure, on the application server.

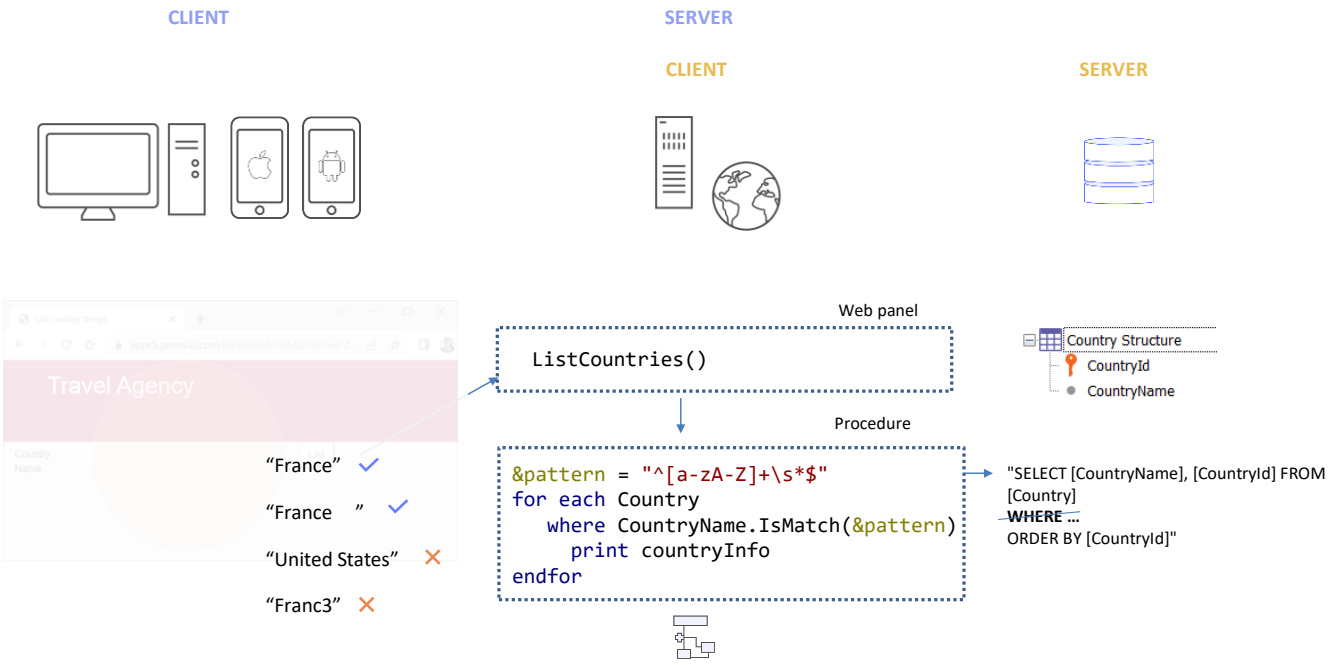
Making this distinction may seem unnecessary at first glance, but we will see that it is not so when things become more complex.

In a simplified way, let's think of it like this: in the example we are asking to run through the Country table, filtering by CountryName (let's assume that in the

printblock we only show the value of that attribute). It is very different to send the query as a Select to the DBMS, which makes the query and returns the result to the client already filtered and sorted, than to send it in a way in which the DBMS returns all the countries and the filter by CountryName has to be done in the client, for example. The amount of information that travels from the database server to the client can be overwhelming.

Here it is obvious that, from the For each of the procedure, the GeneXus specifier will create a source with that select containing a where, to be executed by the DBMS resolving the query in a single operation... but this will not always be the case.

For example, if we don't want the countries whose name is greater than a string, but those whose names satisfy a regular expression (for example, that it can only be a word, consisting of letters from “a” to “z,” uppercase or lowercase, but without anything else; that is, “France” satisfies this pattern, or even “France” followed by blanks, but not “United States,” because after the first space comes another word. And certainly not “Franc3,” because it contains a digit)... well, if we wanted this, then, we would have to program the Source like this:



...where in the variable of character type we establish the regular expression. We will not study them here, but with this syntax we are indicating that it must begin with a letter between lowercase “a” and lowercase “z,” or between uppercase “A” and uppercase “Z,” and that it will be a repetition of characters of these classes. Also, it can have a zero or more blank consecutive spaces and end. In other words, it cannot consist of several words, nor of words with characters other than these.

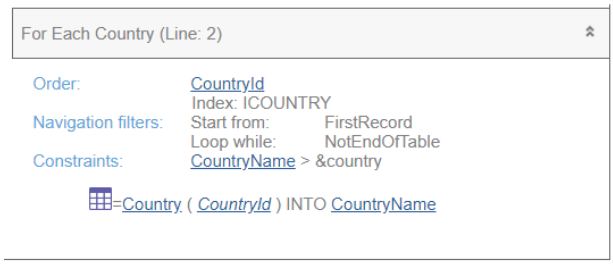
With this regular expression, we can always apply the **IsMatch** method to an attribute or variable to know if its content matches this pattern or not. For example, if in the CountryName attribute we have France, or France with spaces, it will match. However, “United States” will not match, and neither will this other one.

So here we want to list only the countries whose name consists of a single word.

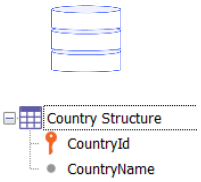
We might think that in the Select that we sent to the DBMS there will be a WHERE as it existed in the other case, so that the DBMS filters the Country records according to that **IsMatch** method. However, if the DBMS is SQL Server, it will not understand that method. It doesn't exist in its language. GeneXus knows this, so the Select statement that it sends to the database Server will have to be WITHOUT the where. And the generated source will be the one that performs the filter with all the countries returned by the query (all the countries in the table). That is to say, the filter will be performed in the client, which is the one that can execute the IsMatch method.

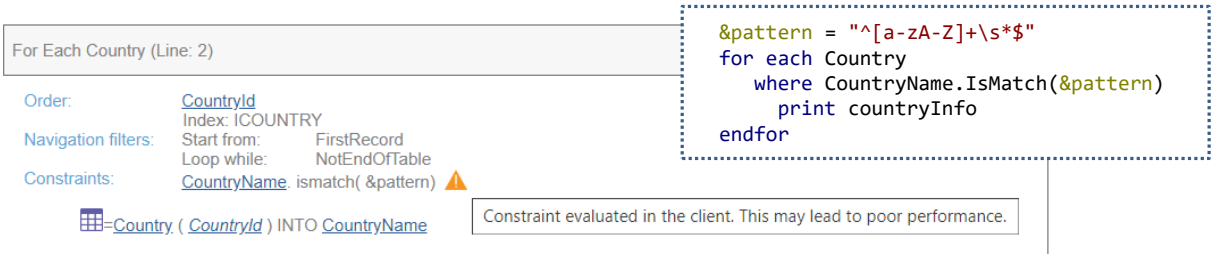
In SQL Server there are very few methods that cannot be resolved by the DBMS.

This is one of them. The functions and methods that can and cannot be resolved will depend on the DBMS used. Interestingly, we don't need to know this beforehand as the navigation list will warn us.




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



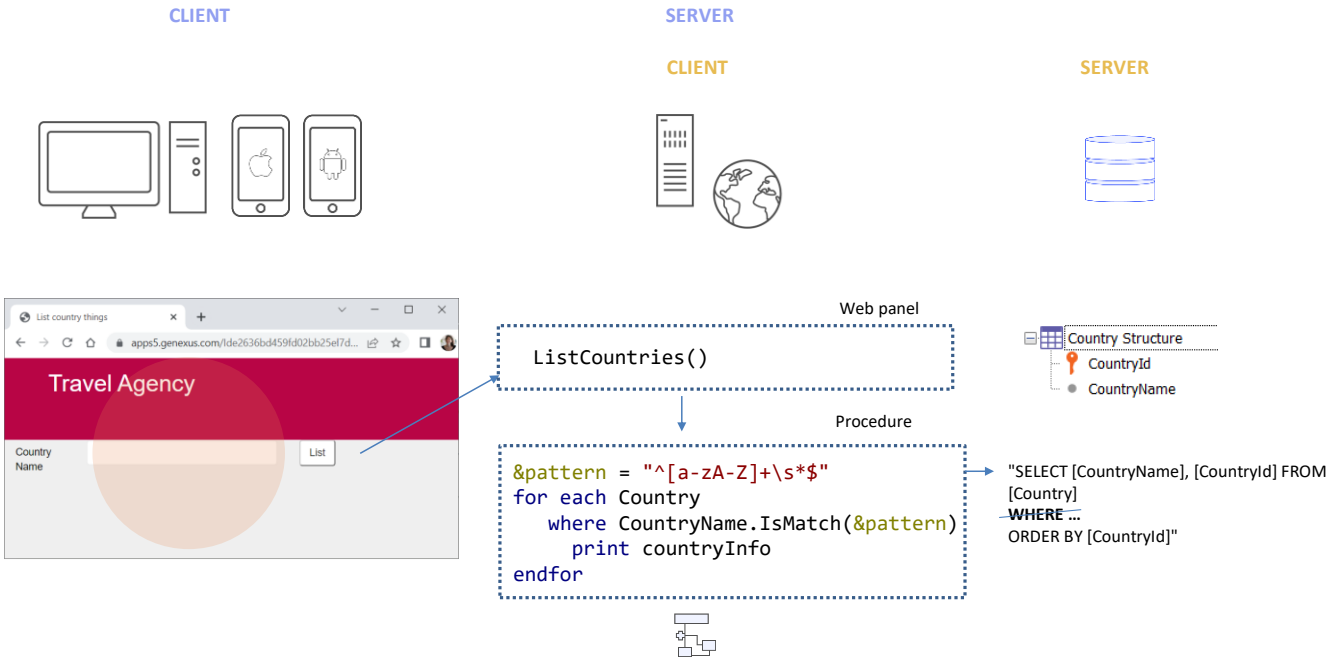



```
&pattern = "^[a-zA-Z]+\s*$"
for each Country
   where CountryName.IsMatch(&pattern)
     print countryInfo
endfor
```

Thus, if we compare the navigation list of the first case with the second one, we see that the second one shows a warning indicating that the filter cannot be applied in the database server, but in its client—that is, in the program corresponding to the procedure, and that it could lead to poor performance.

Consider an extreme case where there is only one country that satisfies the filter, but there are millions of records in the country table. Those millions would travel from the DBMS to the procedure, and also the procedure would have to run through them all, one by one, to finally keep **the** record to display in the output.

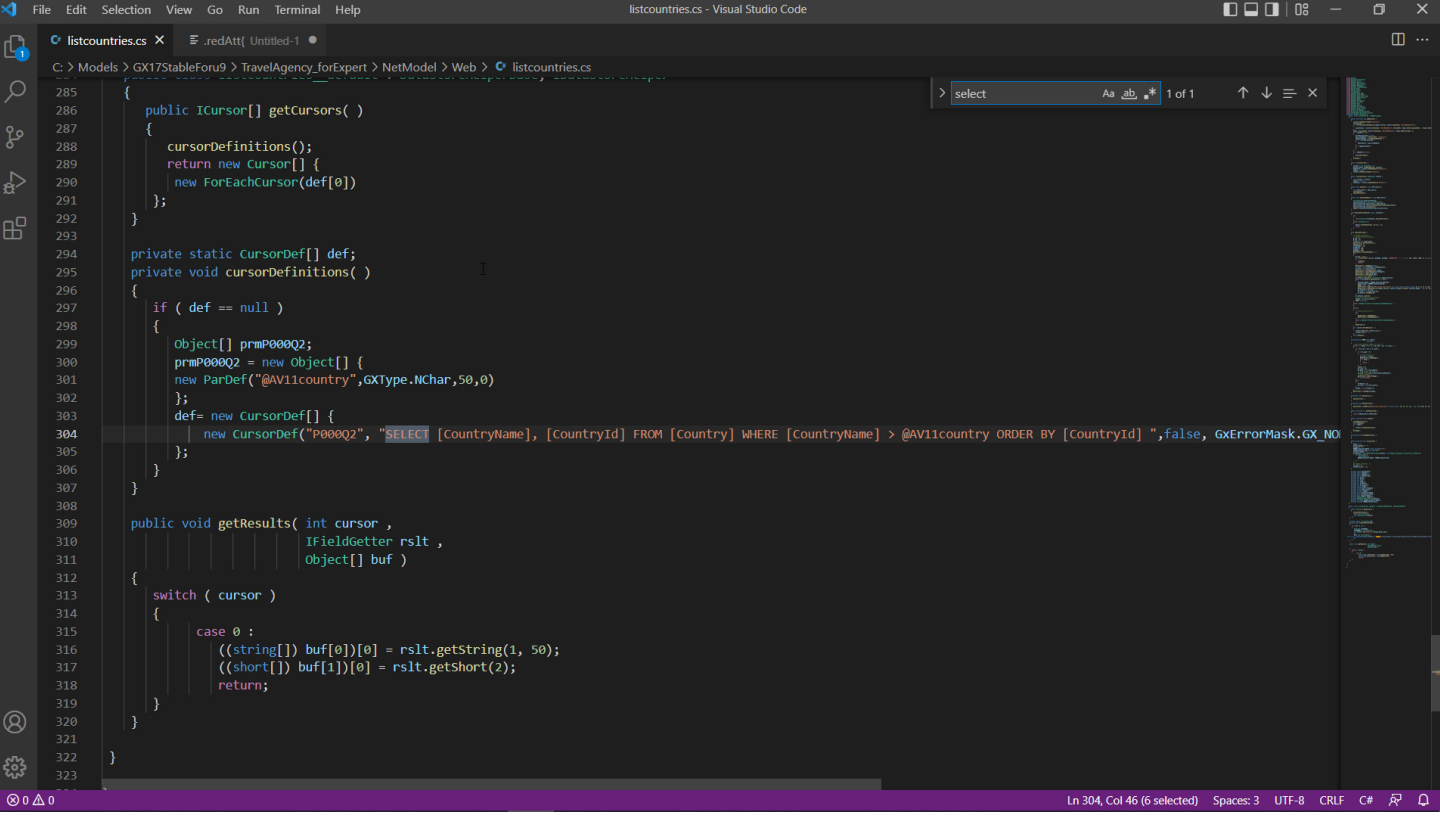
In that case, we might want to improve things by looking for some strategy to reduce that impact.



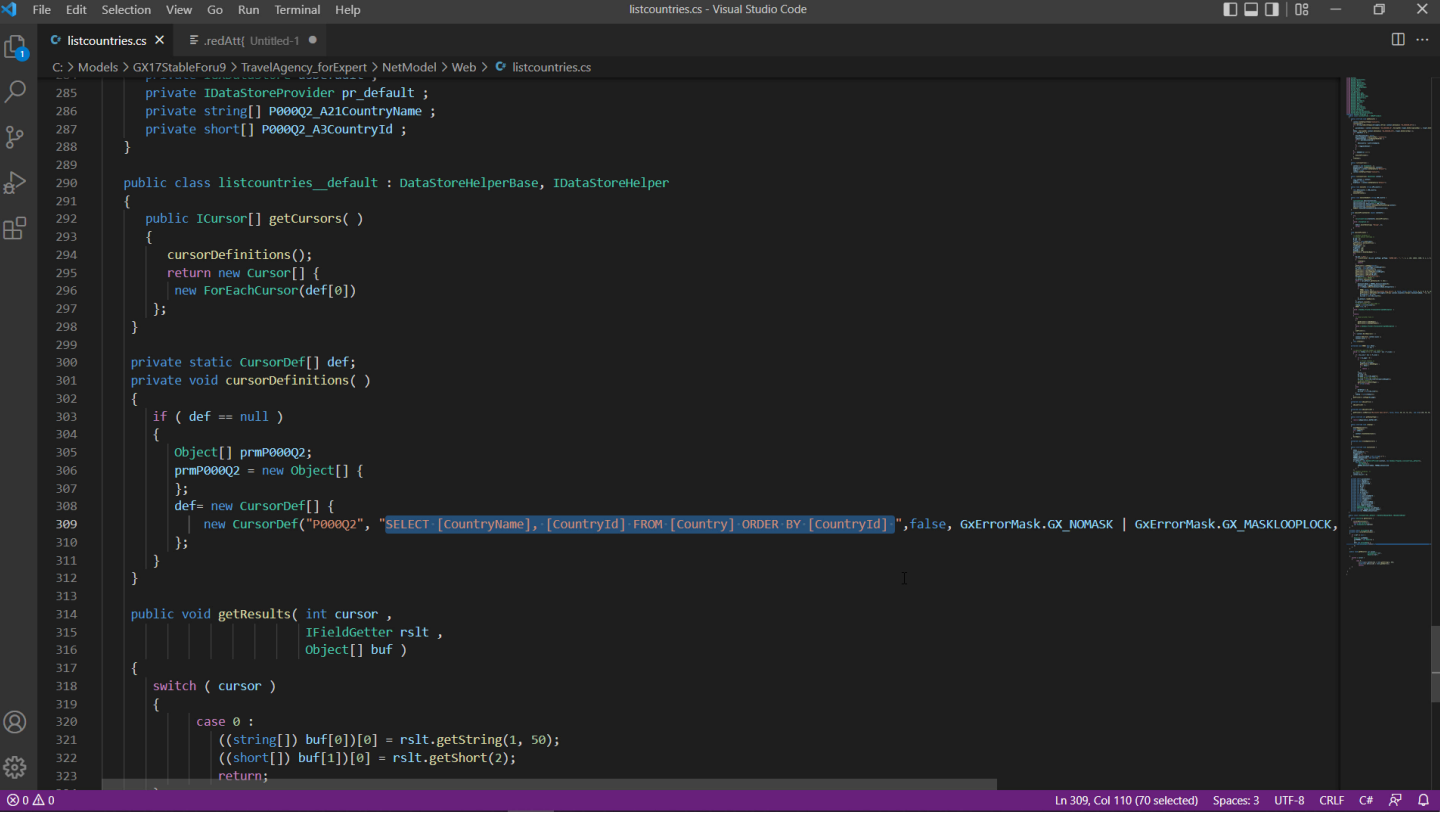
With these examples we wanted to show that, on one hand, we have the application server whose client is the browser on the end user's physical device and, on the other hand, we have the database server whose client is the application running on the application server. The way we program the database accesses will have an impact on performance.

GeneXus developers write a For each in the Source, and then, knowing the environment for which the application will be built, the GeneXus specifier will determine how the source should be built. There it makes decisions that depend on the programming platform, but also on the DBMS with which it will connect.

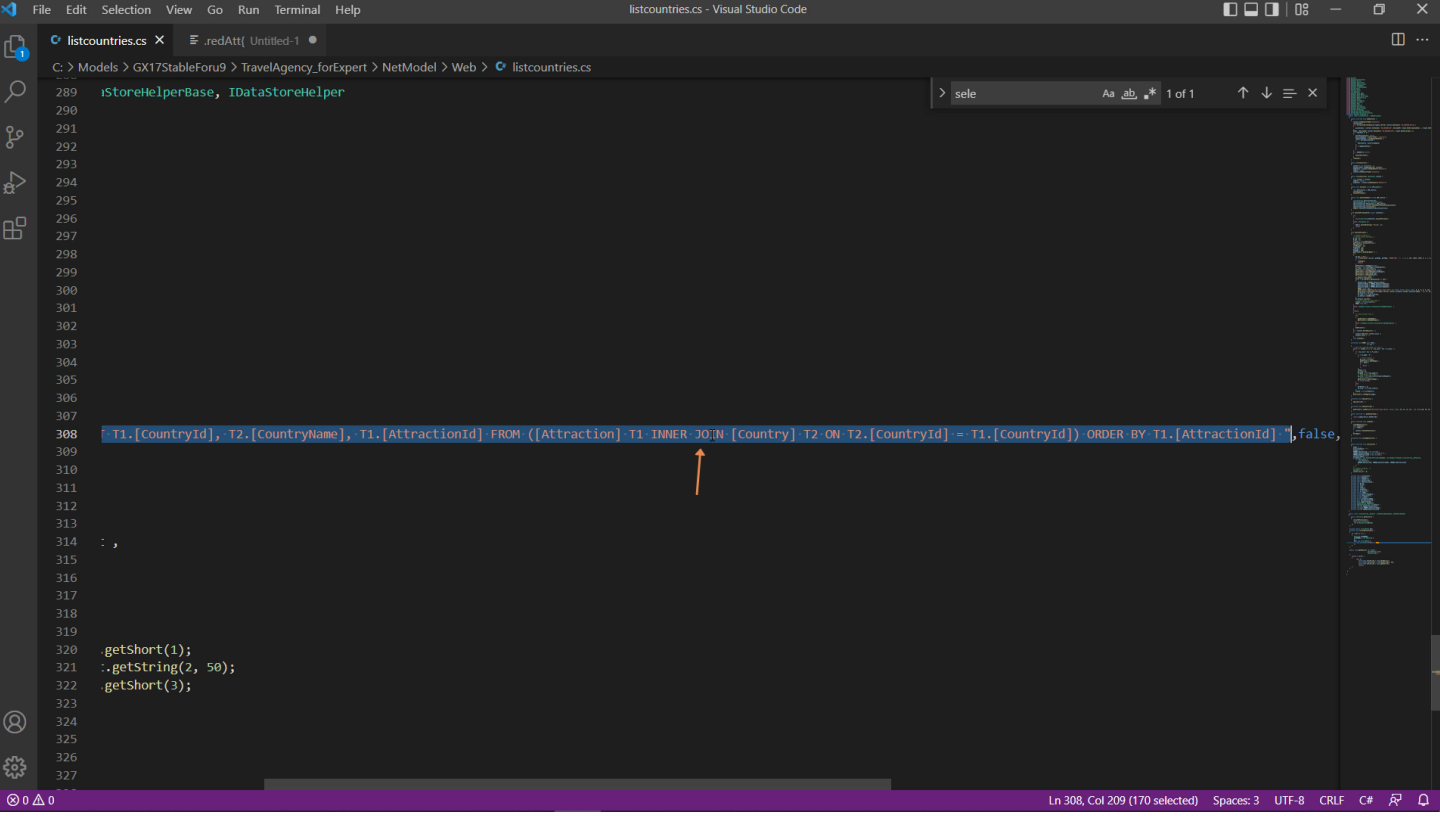
If we are curious, we can always inspect the source and see the SQL statement.



For example, here we have the first proposal... We ask it to build the source program... we look for it in the environment directory (which is .Net against SQL Server), we open it... and look for the select... here we see it. There is the complete query.



Instead, if we modify the source for the second proposal and do the same... we see that the select doesn't include the where... which means that the filter is being resolved inside this source.



And now, for example, if we change the base table of the For each to Attraction, so that it prints all the names of the attractions' countries (of course they will be repeated if several have the same country)...

In the navigation list, we can see that as the table run through will be Attraction, then it will have to access the Country table to obtain each attraction's country, and thus be able to print the value of the CountryName for that attraction.
This operation in relational databases is called a Join. That's why the Join location is indicated, which shows where the Join is performed, whether in the database server or in the client. Here it tells us that it is on the server, so if we look for the generated source, we will see that the entire select is sent to the database so that the Join is resolved there. That is much less costly than if it had to be resolved by the client, that is, this procedure.

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

With all this in mind, we will review the syntax of the For each command and what everything is used for, in order to incorporate what we already know and go a little further.