For each in depth

Order and performance clauses

GeneXus™

Let's focus on order clauses and how they relate to the optimization of queries.

```
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 [when condition]
      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
```

We know that to order the information to be queried and returned, the syntax of the For each allows us to specify a list of conditional clauses and an unconditional one.

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 [when *condition*]

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

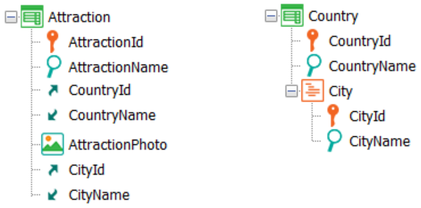
when_duplicate_code

when none

when_none_code

endfor

```
for each Attraction
   order AttractionName
   where CountryId = &countryId
   where CityId = &cityId
      print attractionInfo //AttractionName
endfor
```



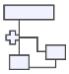
Attraction
- AttractionId
- AttractionName
- CountryId
- CountryName
- AttractionPhoto
- CityId
- CityName

Country
- CountryId
- CountryName
- City
  - CityId
  - CityName

Let's go from what is simpler to what is more complex: let's start by analyzing the case of a single order clause with no conditions, such as this one. We want to run through the table of tourist attractions, filtering those corresponding a given country and city, and to display them sorted by AttractionName, a secondary attribute.

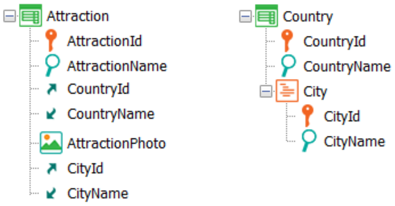
When specifying what we want, it shouldn't matter whether in order to get the required data it should be first obtained and then sorted, or if it is done the other way around, or in some other way. We, the developers, specify what we want and the GeneXus specifier and, above all, the DBMS will solve it.

```
for each Attraction
 order AttractionName
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```



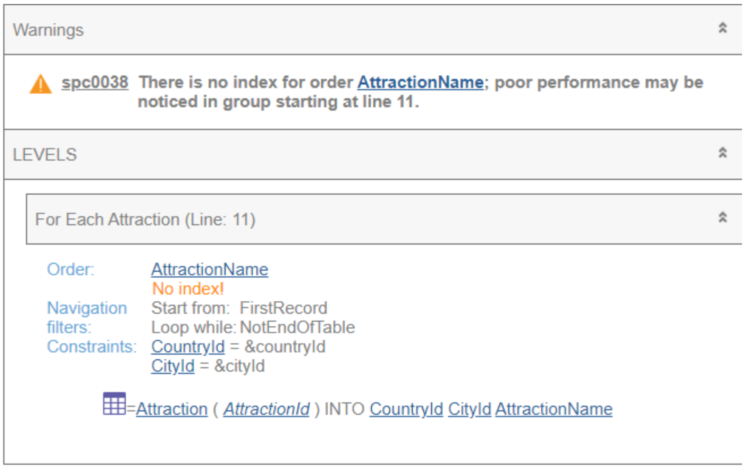
Net - SQL Server

Java - Oracle



- Attraction
  - AttractionId
  - AttractionName
  - CountryId
  - CountryName
  - AttractionPhoto
  - CityId
  - CityName

- Country
  - CountryId
  - CountryName
  - City
    - CityId
    - CityName



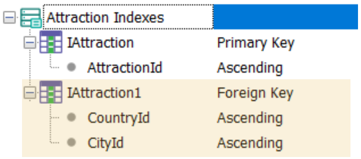
Warnings

⚠ spc0038 There is no index for order AttractionName; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

```
Order:         AttractionName
               No index!
Navigation     Start from:  FirstRecord
filters:       Loop while: NotEndOfTable
Constraints:   CountryId = &countryId
               CityId = &cityId

     =Attraction ( AttractionId ) INTO CountryId CityId AttractionName
```



Attraction Indexes	
IAttraction	Primary Key
AttractionId	Ascending
IAttraction1	Foreign Key
CountryId	Ascending
CityId	Ascending

When we ask GeneXus to specify and generate the program associated with the object containing the For each, we do it for a given environment, that is, in a given programming language, such as Net and for a database managed by a given DBMS, such as SQL Server, for example. It could also be for a Java environment against Oracle, or so many other alternatives.

When a developer writes a For each, they do it in GeneXus, with a certain independence from what the final environment will be, in order to obtain the application in different environments with the same code.

This means that the specific implementation is handled by GeneXus, which is aware of the specific features of each environment. However, its knowledge has a limit: it knows the database structure, but not the data, nor its distribution, quantity, etcetera. This information is held by the DBMS, which records statistics, and caches data and queries in the history of the queries that have been made, builds execution plans, maintains indexes, and so on. The more evolved and intelligent the DBMS is, the less it will need GeneXus to tell it precisely how to perform the query, because GeneXus will never know more than it does.

So, for example, if we ask to specify the object that contains this For each we will see this navigation list, which warns us that there is no index for the attribute by which we want to sort the query and that therefore we could notice a low performance.
"We might" doesn't mean that this will indeed be the case. Why? Because it depends not only on the amount of data in the table, but also on the DBMS and its strategies. The navigation list shows the worst case scenario: here the whole table

must be run through ordered by an attribute for which there is possibly no index (GeneXus doesn't know if the DBMS created it or not, and has no information about it). So, in the worst case scenario, which is that of centralized architectures, it may have to temporarily create the index to solve the query in that order and thus run through the entire table to evaluate each record individually to determine whether it meets the filters or not. This would be the scenario of a poor database management system.

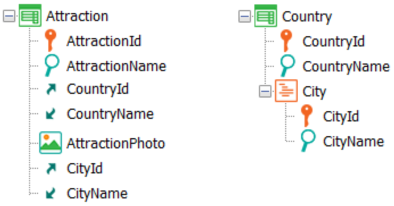
Let's think, for example, that in this case using the foreign key index {CountryId, CityId}, which we know exists because GeneXus forces it to be created, might be a better strategy. Then the records that meet the filters are obtained in an optimal way and, with that result, it is sorted by AttractionName. The best strategy will depend largely on the distribution of the data. If there are only 3 attractions from that country and city out of millions, this seems a better strategy because the cost of sorting 3 records is insignificant. However, if there are millions of records in the table and most of them are from that country and city, using the index by country and city will not significantly reduce the query of the entire table. Therefore, sorting the result by AttractionName will be almost the same as sorting by AttractionName and then evaluating each record individually to determine if it also corresponds to the country/city or not. GeneXus doesn't know the data distribution to make this kind of decisions. In addition, if this same query has already been performed before, the DBMS will probably cache the result and will not have to perform the query in the same way again. This is, of course, if the filters and the data do not change.

Ultimately, then, the navigation list provides information based on the most conservative scenario, with the worst DBMS. However, it is possible that the DBMS will greatly improve the most pessimistic scenario and the query will be optimized.

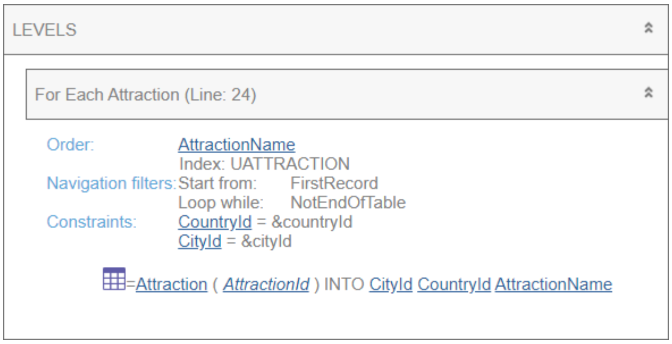
```
for each Attraction
 order AttractionName
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

Net - SQL Server

Java - Oracle



- Attraction
  - AttractionId
  - AttractionName
  - CountryId
  - CountryName
  - AttractionPhoto
  - CityId
  - CityName
- Country
  - CountryId
  - CountryName
  - City
    - CityId
    - CityName

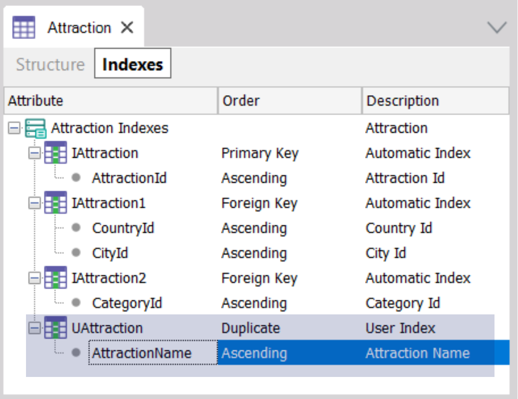


LEVELS

For Each Attraction (Line: 24)

Order: AttractionName
Index: UATTRACTION
Navigation filters: Start from: FirstRecord
Loop while: NotEndOfTable
Constraints: CountryId = &countryId
CityId = &cityId

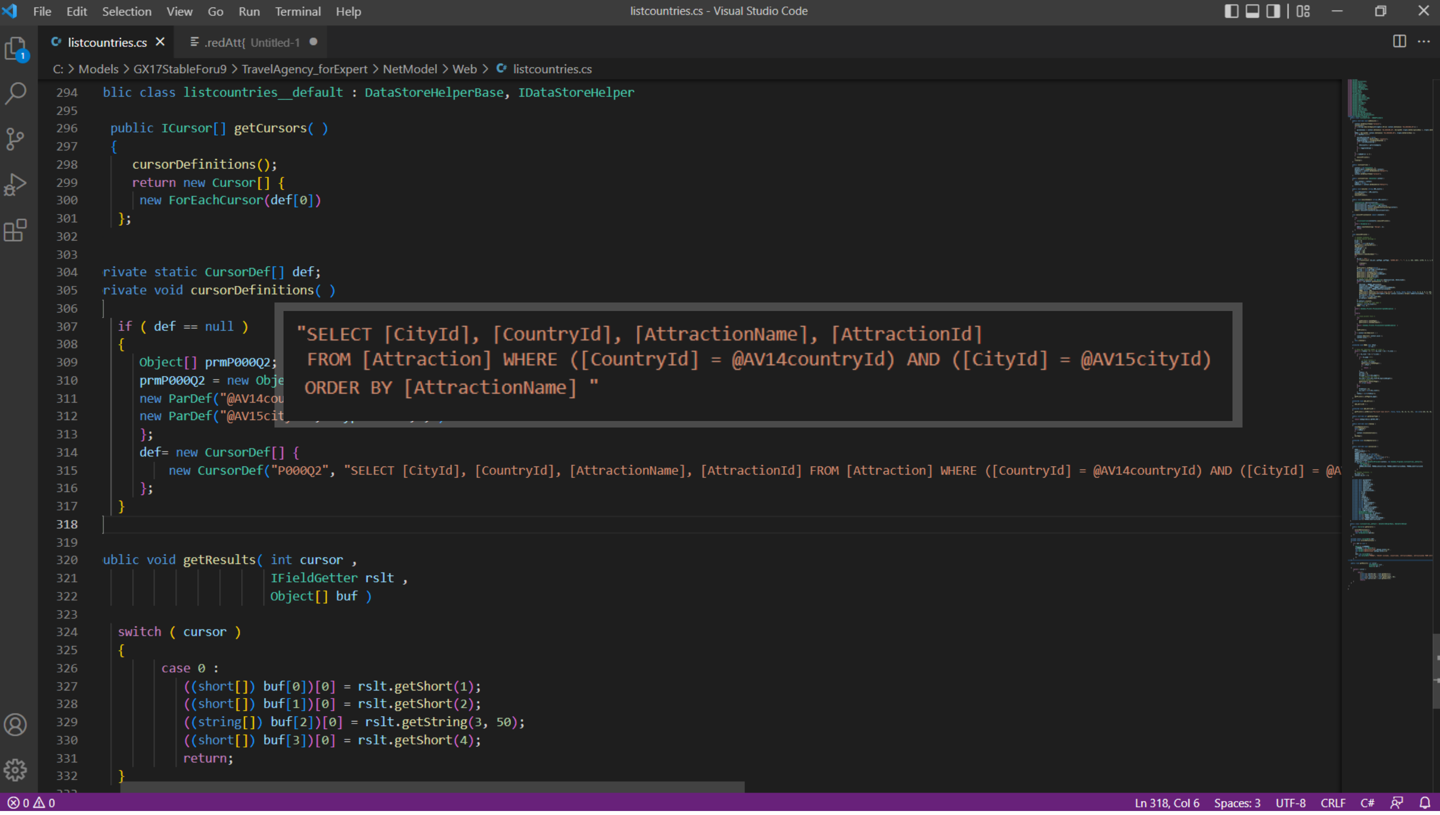
=Attraction (AttractionId) INTO CityId CountryId AttractionName



Attribute	Order	Description
Attraction Indexes		Attraction
IAttraction	Primary Key	Automatic Index
AttractionId	Ascending	Attraction Id
IAttraction1	Foreign Key	Automatic Index
CountryId	Ascending	Country Id
CityId	Ascending	City Id
IAttraction2	Foreign Key	Automatic Index
CategoryId	Ascending	Category Id
UAttraction	Duplicate	User Index
AttractionName	Ascending	Attraction Name

We might be tempted to think that if we know there will be millions of records in the Attraction table, it will be best to instruct the DBMS from GeneXus to create the user index by AttractionName. In that case, the database will be reorganized. When the object is specified again, GeneXus will inform us that it will use the new index to solve the query and the performance warning will no longer be shown.

However, this can be a much worse solution. Precisely, if the DBMS is intelligent, it will not need to be forced to create an index at all. It will do it itself if necessary. So much so that even in this case in which GeneXus itself requested the creation of the index, it doesn't send it to the DBMS when it performs the query from an environment with an intelligent DBMS.


File Edit Selection View Go Run Terminal Help
listcountries.cs - Visual Studio Code
listcountries.cs
.redAtt{ Untitled-1
C: > Models > GX17StableForu9 > TravelAgency_forExpert > NetModel > Web > listcountries.cs
Ln 318, Col 6 Spaces: 3 UTF-8 CRLF C#

```
blic class listcountries__default : DataStoreHelperBase, IDataStoreHelper

 public ICursor[] getCursors( )
 {
    cursorDefinitions();
    return new Cursor[] {
     new ForEachCursor(def[0])
  };


rivate static CursorDef[] def;
rivate void cursorDefinitions( )

  if ( def == null )
  {
     Object[] prmP000Q2;
     prmP000Q2 = new Obje
     new ParDef("@AV14cou
     new ParDef("@AV15cit
     };
     def= new CursorDef[] {
         new CursorDef("P000Q2", "SELECT [CityId], [CountryId], [AttractionName], [AttractionId] FROM [Attraction] WHERE ([CountryId] = @AV14countryId) AND ([CityId] = @A
     };
  }


ublic void getResults( int cursor ,
                       IFieldGetter rslt ,
                       Object[] buf )

  switch ( cursor )
  {
        case 0 :
           ((short[]) buf[0])[0] = rslt.getShort(1);
           ((short[]) buf[1])[0] = rslt.getShort(2);
           ((string[]) buf[2])[0] = rslt.getString(3, 50);
           ((short[]) buf[3])[0] = rslt.getShort(4);
           return;
  }
```

```
"SELECT [CityId], [CountryId], [AttractionName], [AttractionId]
 FROM [Attraction] WHERE ([CountryId] = @AV14countryId) AND ([CityId] = @AV15cityId)
ORDER BY [AttractionName] "
```

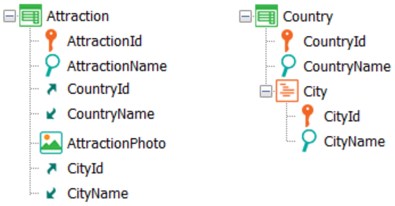
Just look at the source generated for the Net environment against SQL Server. In the Select statement no information is being sent about the index to be used. Why would it send it if the DBMS knows of its existence? If it needs it, it will use it. And if it doesn't, it's because it will use a better strategy.

We may suppose that the case would be different if the DBMS was not intelligent.

```
for each Attraction
 order AttractionName
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

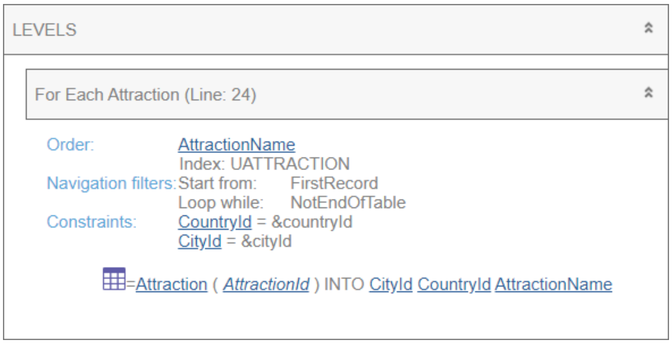
Net - SQL Server

Java - Oracle



- Attraction
  - AttractionId
  - AttractionName
  - CountryId
  - CountryName
  - AttractionPhoto
  - CityId
  - CityName

- Country
  - CountryId
  - CountryName
  - City
    - CityId
    - CityName

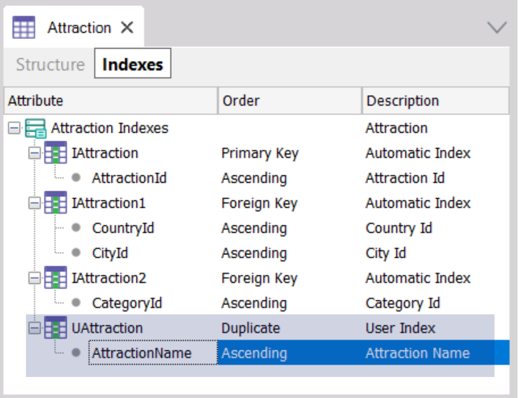


LEVELS

For Each Attraction (Line: 24)

Order: AttractionName
Index: UATTRACTION
Navigation filters: Start from: FirstRecord
Loop while: NotEndOfTable
Constraints: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CityId CountryId AttractionName



Attraction

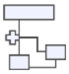
Structure | **Indexes**

Attribute	Order	Description
Attraction Indexes		Attraction
IAttraction	Primary Key	Automatic Index
AttractionId	Ascending	Attraction Id
IAttraction1	Foreign Key	Automatic Index
CountryId	Ascending	Country Id
CityId	Ascending	City Id
IAttraction2	Foreign Key	Automatic Index
CategoryId	Ascending	Category Id
UAttraction	Duplicate	User Index
AttractionName	Ascending	Attraction Name

We must take into account that creating an index is costly, not only when creating it, but also later on when maintaining it, during the whole life cycle of the table. Every time the table is updated, a small price is paid to maintain it.

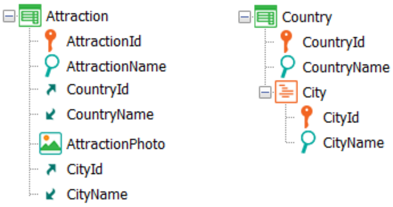
That's why creating user indexes doesn't seem to be a good practice, unless they are needed to control uniqueness; that is, to define candidate keys, such as unique indexes.

```
for each Attraction
 order AttractionName
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```



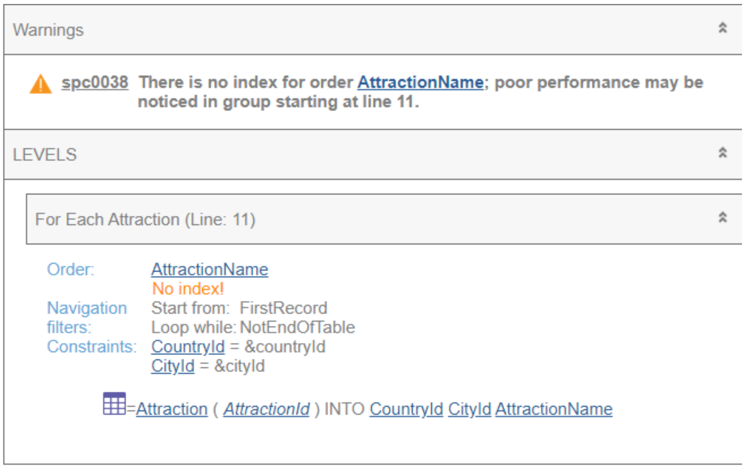
Net - SQL Server

Java - Oracle



- Attraction
  - AttractionId
  - AttractionName
  - CountryId
  - CountryName
  - AttractionPhoto
  - CityId
  - CityName

- Country
  - CountryId
  - CountryName
  - City
    - CityId
    - CityName



Warnings

spc0038 There is no index for order AttractionName; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

```
Order:        AttractionName
              No index!
Navigation    Start from:  FirstRecord
filters:      Loop while: NotEndOfTable
Constraints:  CountryId = &countryId
              CityId = &cityId

        =Attraction ( AttractionId ) INTO CountryId CityId AttractionName
```

So, as we said, the GeneXus specifier does the best it can with the information it has and with its current intelligence (it is expected that this intelligence will increase as GeneXus evolves). It sends the query as optimized as possible to the DBMS without providing obvious information, knowing that in the worst case it will take it but in general it will improve it.

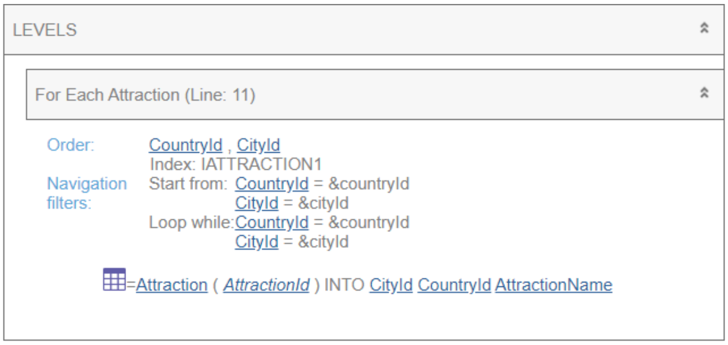
In short, we can never be sure that what is shown in the navigation list will be what the DBMS will finally do if it is intelligent. What we do know is that it will be that or something better.

Let's look at examples of the current intelligence of the GeneXus specifier, regardless what the DBMS ends up doing later.

We know that the order in which the resulting records will be returned is determined based on the developer's specification in the order clause, but also on internal optimization algorithms.

```
for each Attraction

 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```



LEVELS

For Each Attraction (Line: 11)

```
Order:       CountryId , CityId
             Index: IATTRACTION1
Navigation   Start from: CountryId = &countryId
filters:                 CityId = &cityId
             Loop while: CountryId = &countryId
                         CityId = &cityId

          =Attraction ( AttractionId ) INTO CityId CountryId AttractionName
```

For example, if we didn't care about how the attraction names will be sorted we could write the For each without the order clause. In general, we knew that if we did this the query would be sorted by primary key. That is to say, a Select with AttractionId order was sent to the DBMS. However, in this case something different will happen, as we can see in the navigation list. We know that the database has an index by CountryId and CityId, the two equality filters. We know this because they form a foreign key. Clearly, then, it will be preferable to use that index and return the query sorted by those values.
That's why if we look at the query sent by the source to the DBMS, we find this ORDER BY. If not specifying the order clause meant that we wanted the queries to be sorted by primary key, in this case we will have to make it explicit.

Similarly, if now the query is this other one, where we are asking for the attractions to be sorted by city identifier, and we are only filtering by country identifier, we will see that the navigation list will not ask to sort by CityId, but by the pair. In this way, it optimizes the query as it knows about the existence of the composite index (because it is a foreign key, precisely), without failing to achieve the sorting result requested by the developer.

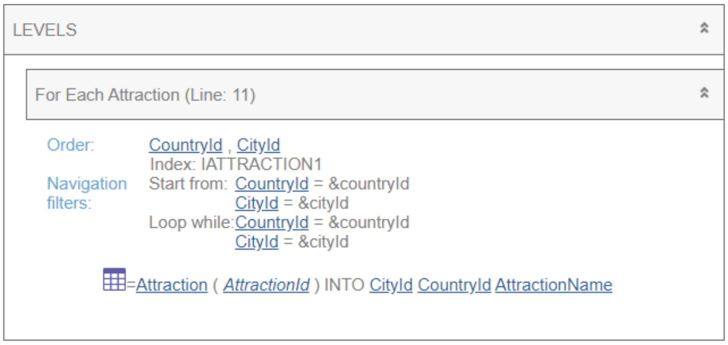
In short, with the order clause the developer indicates the order in which they want the records to be returned, but for this the specifier could alter this clause, complementing it with contextual information (if there are implicit or explicit equality conditions and there is an index that contains them, in addition to the attributes of the order) in order to optimize the query, although the DBMS will have the last word.

It is important to understand that the data will be returned in the order specified by the developer, even if other criteria are used to solve the query.

```
for each Attraction
 order AttractionId
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

"SELECT [CityId], [CountryId], [AttractionName], [AttractionId] FROM [Attraction] WHERE [CountryId] = @AV14countryId and [CityId] = @AV15cityId ORDER BY ~~[CountryId], [CityId]~~"

[AttractionId]



LEVELS

For Each Attraction (Line: 11)

Order: CountryId , CityId
Index: IATTRACTION1

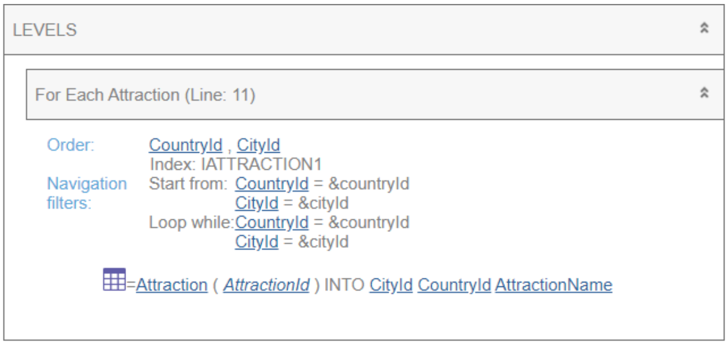
Navigation filters: Start from: CountryId = &countryId
CityId = &cityId
Loop while: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CityId CountryId AttractionName

If not specifying the order clause meant that we wanted the queries to be sorted by primary key, in this case we will have to make it explicit.

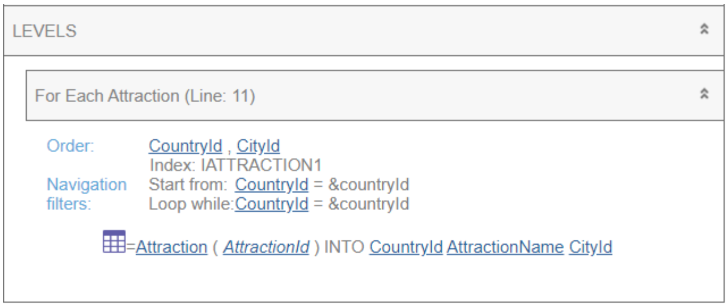
```
for each Attraction

 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```



```
LEVELS

For Each Attraction (Line: 11)

Order:       CountryId , CityId
             Index: IATTRACTION1
Navigation   Start from: CountryId = &countryId
filters:                 CityId = &cityId
             Loop while: CountryId = &countryId
                         CityId = &cityId

=Attraction ( AttractionId ) INTO CityId CountryId AttractionName
```

```
for each Attraction
 order CityId
 where CountryId = &countryId
  print attractionInfo //AttractionName
endfor
```



```
LEVELS

For Each Attraction (Line: 11)

Order:       CountryId , CityId
             Index: IATTRACTION1
Navigation   Start from: CountryId = &countryId
filters:     Loop while: CountryId = &countryId

=Attraction ( AttractionId ) INTO CountryId AttractionName CityId
```

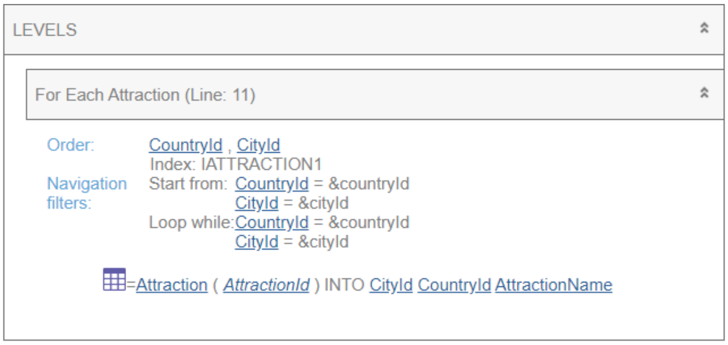
Similarly, if now the query is this other one, where we are asking for the attractions to be sorted by city identifier, and we are only filtering by country identifier, we will see that the navigation list will not ask to sort by CityId, but by the pair. The reason is that this optimizes the query, since it knows about the existence of the composite index (because it is a foreign key, precisely), without failing to achieve the sorting result requested by the developer.

In short, with the order clause the developer indicates the order in which they want the records to be returned, but for this the specifier could alter this clause, complementing it with contextual information (if there are implicit or explicit equality conditions and there is an index that contains them, in addition to the attributes of the order) in order to optimize the query, although the DBMS will have the last word.

It is important to understand that the data will be returned in the order specified by the developer, even if other criteria are used to solve the query.

```
for each Attraction

 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

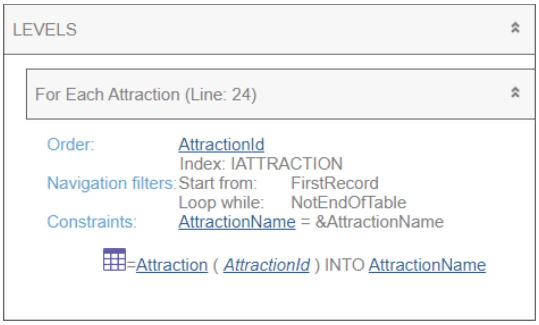
```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```



```
LEVELS
For Each Attraction (Line: 11)
Order:        CountryId , CityId
              Index: IATTRACTION1
Navigation    Start from:  CountryId = &countryId
filters:                   CityId = &cityId
              Loop while:  CountryId = &countryId
                           CityId = &cityId
=Attraction ( AttractionId ) INTO CityId CountryId AttractionName
```

```
for each Attraction

 where AttractionName = &attractionName
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [AttractionName], [AttractionId]
FROM [Attraction] WHERE [AttractionName] =
@AV8AttractionName ORDER BY [AttractionId]"
```



```
LEVELS
For Each Attraction (Line: 24)
Order:              AttractionId
                    Index: IATTRACTION
Navigation filters: Start from:   FirstRecord
                    Loop while:   NotEndOfTable
Constraints:        AttractionName = &AttractionName
=Attraction ( AttractionId ) INTO AttractionName
```

In this case where no order was specified, since there is an index that allows optimizing these conditions, it is chosen instead of the index by primary key.

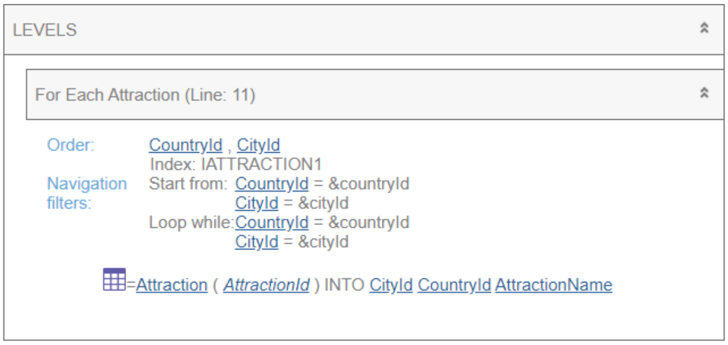
When there is no index, it chooses the primary key. Here we see the SQL statement created by GeneXus.

```
for each Attraction

 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```

```
for each Attraction
 order NONE
 where AttractionName = &attractionName
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [AttractionName], [AttractionId]
FROM [Attraction] WHERE [AttractionName] =
@AV8AttractionName ~~ORDER BY [AttractionId]"~~
```



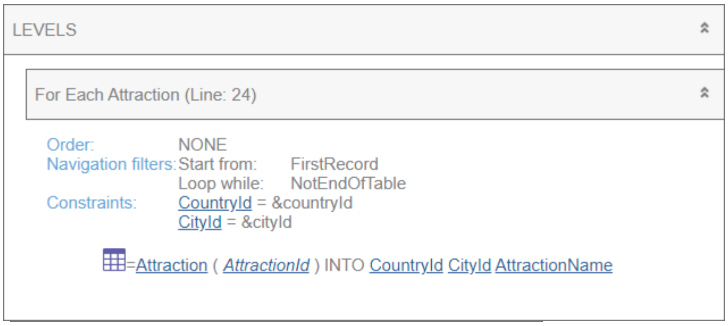
LEVELS

For Each Attraction (Line: 11)

Order: CountryId , CityId
Index: IATTRACTION1

Navigation filters: Start from: CountryId = &countryId
CityId = &cityId
Loop while: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CityId CountryId AttractionName



LEVELS

For Each Attraction (Line: 24)

Order: NONE

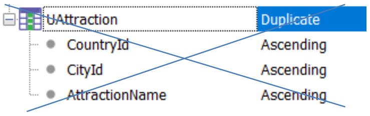
Navigation filters: Start from: FirstRecord
Loop while: NotEndOfTable

Constraints: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CountryId CityId AttractionName

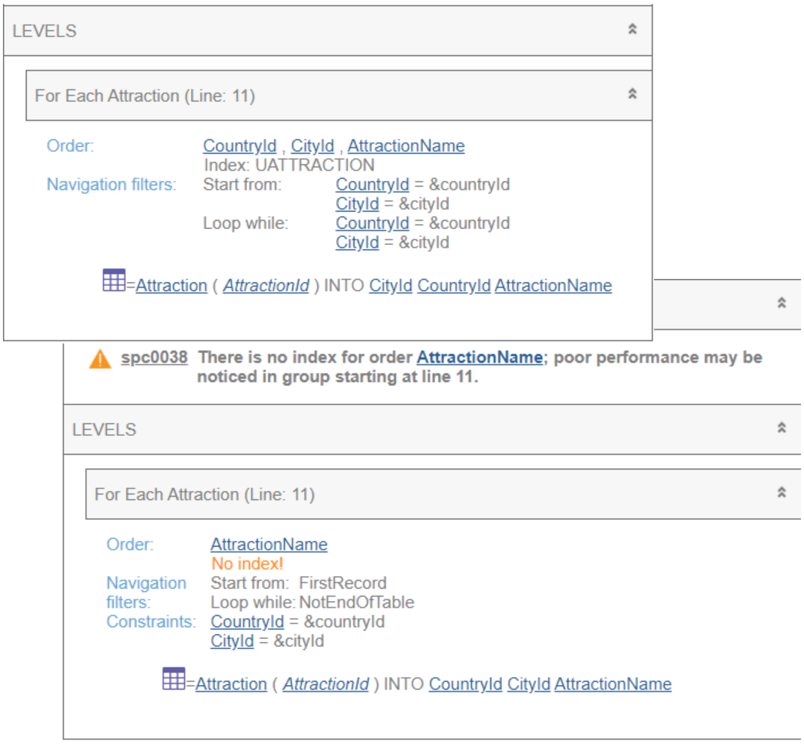
Unless we specify an Order none clause, in which case we leave it up to the DBMS to choose the order. ORDER BY is not added to the SQL statement created by GeneXus.

```
for each Attraction
 order AttractionName
 where CountryId = &countryId
 where CityId = &cityId
  print attractionInfo
endfor
```



UAttraction	Duplicate
CountryId	Ascending
CityId	Ascending
AttractionName	Ascending

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE ([CountryId] =
@AV14countryId) AND ([CityId] = @AV15cityId)
ORDER BY [AttractionName] "
```



LEVELS

For Each Attraction (Line: 11)

Order: CountryId , CityId , AttractionName
Index: UATTRACTION
Navigation filters: Start from: CountryId = &countryId
CityId = &cityId
Loop while: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CityId CountryId AttractionName

spc0038 There is no index for order AttractionName; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

Order: AttractionName
No index!
Navigation filters: Start from: FirstRecord
Loop while: NotEndOfTable
Constraints: CountryId = &countryId
CityId = &cityId

=Attraction (AttractionId) INTO CountryId CityId AttractionName

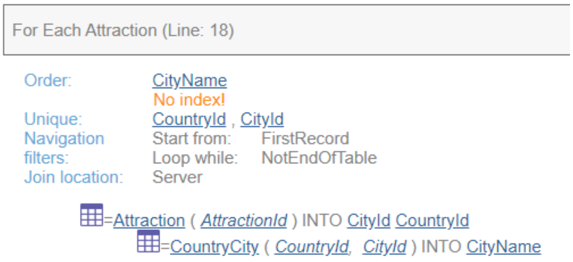
Let's go back to this example. If GeneXus asked to create this composite user index, then it will propose it in the navigation list (even if in the end it doesn't send it to the SQL Server, because the SQL Server already knows about its existence, so why tell it something it already knows). What's remarkable is that it notifies us that at least this optimization will be performed by the DBMS. It will be like this or better.

On the other hand, if the user index doesn't exist, the navigation list will show us this other one, even if the source is exactly the same.

So? Let's say it again: the navigation list shows the worst case scenario. If the index exists, we know that the worst case scenario will be pretty good. That's if the index was previously created for some other reason, so we take advantage of it. Creating the index just to make sure this navigation is optimized doesn't seem to be a good idea if we're using an intelligent DBMS. And neither if the DBMS was not intelligent but the table had few records. In short, create indexes only after noticing performance issues and evaluating the pros and cons.

```
for each Attraction
   order CityName
   unique CountryId, CityId
      print relevantInfo //CityName
endfor
```

```
"SELECT DISTINCT T1.[CityId], T1.[CountryId],
T2.[CityName] FROM ([Attraction] T1 INNER JOIN
[CountryCity] T2 ON T2.[CountryId] = T1.[CountryId] AND
T2.[CityId] = T1.[CityId])
ORDER BY T2.[CityName] "
```

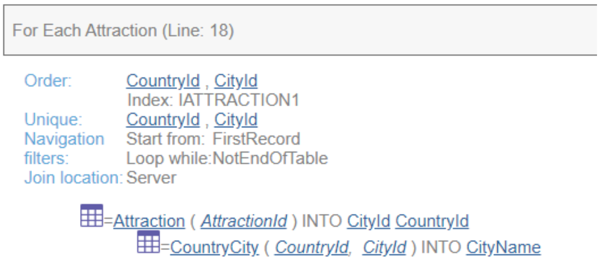


For Each Attraction (Line: 18)

```
Order:          CityName
                No index!
Unique:         CountryId , CityId
Navigation      Start from:    FirstRecord
filters:        Loop while:    NotEndOfTable
Join location:  Server

   =Attraction ( AttractionId ) INTO CityId CountryId
      =CountryCity ( CountryId,  CityId ) INTO CityName
```

```
for each Attraction
   //order CityName
   unique CountryId, CityId
      print relevantInfo //CityName
endfor
```

```
"SELECT DISTINCT T2.[CityName], T1.[CityId],
T1.[CountryId] FROM ([Attraction] T1 INNER JOIN
[CountryCity] T2 ON T2.[CountryId] = T1.[CountryId] AND
T2.[CityId] = T1.[CityId])
ORDER BY T1.[CountryId], T1.[CityId] "
```



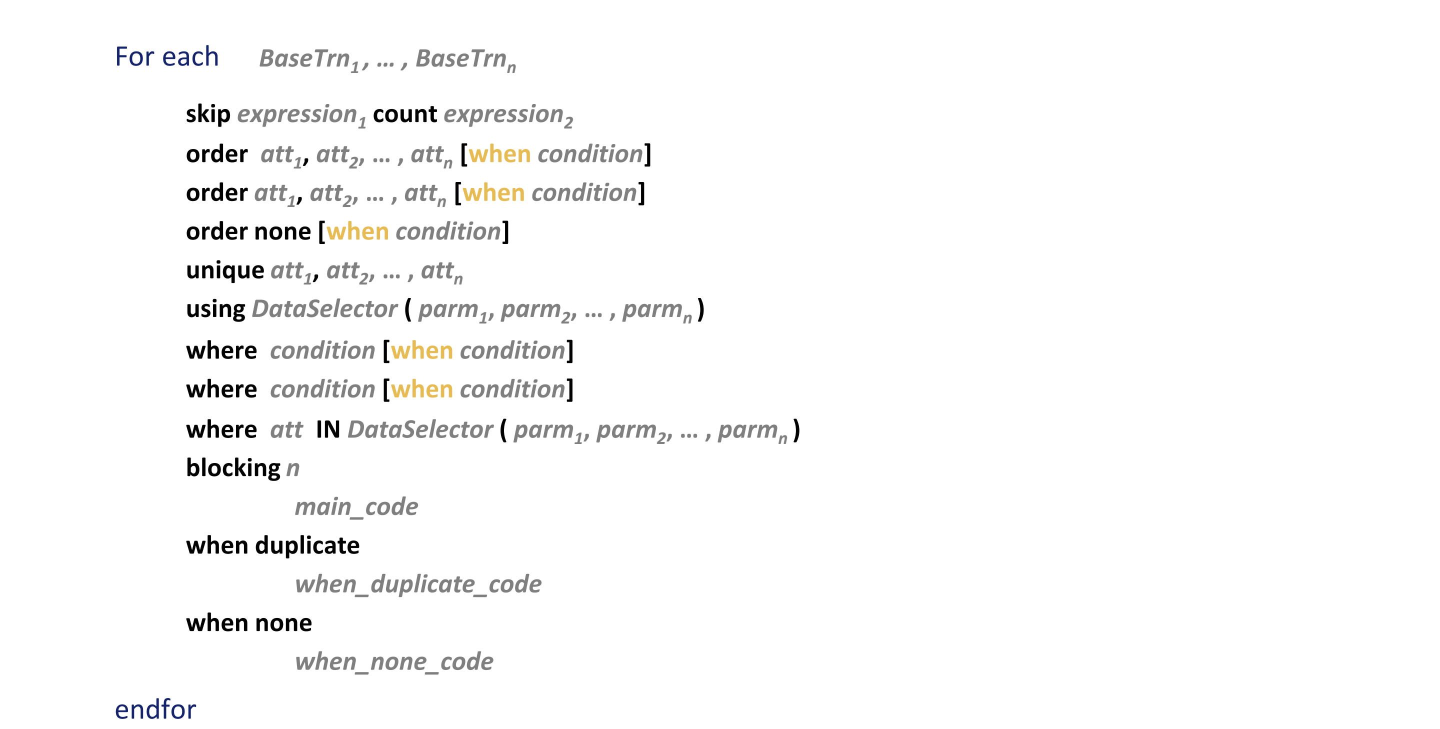
For Each Attraction (Line: 18)

```
Order:          CountryId , CityId
                Index: IATTRACTION1
Unique:         CountryId , CityId
Navigation      Start from:  FirstRecord
filters:        Loop while:NotEndOfTable
Join location:  Server

   =Attraction ( AttractionId ) INTO CityId CountryId
      =CountryCity ( CountryId,  CityId ) INTO CityName
```

Here is another example of the way GeneXus tries to make improvements:

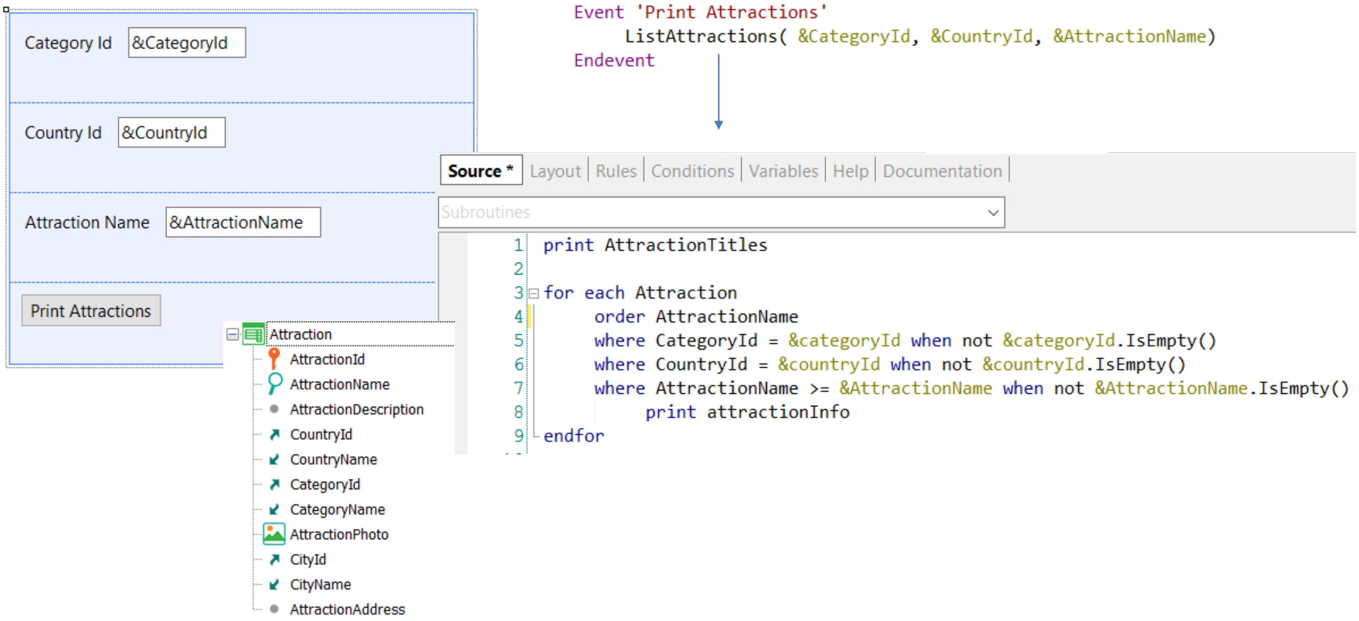
If we want to get all the city names for which there are tourist attractions, we use the unique clause by CountryId, CityId, so that of all the attractions that share a country and city only one is left, in order to list its city name in the output. If we want this output to be sorted by city name, we place the order clause and see that in the navigation list the specifier writes exactly that order, for which it doesn't know any index. It will be up to the DBMS to optimize this query.

On the other hand, if we don't care about the order in which those cities are displayed in the output, then let's see that by not writing it, the specifier chooses to sort by the attributes that we are asking to be unique, since it has an index by them.



```
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 [when condition]
    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
```

Now let's review the conditional order clauses.



Category Id &CategoryId

Country Id &CountryId

Attraction Name &AttractionName

Print Attractions

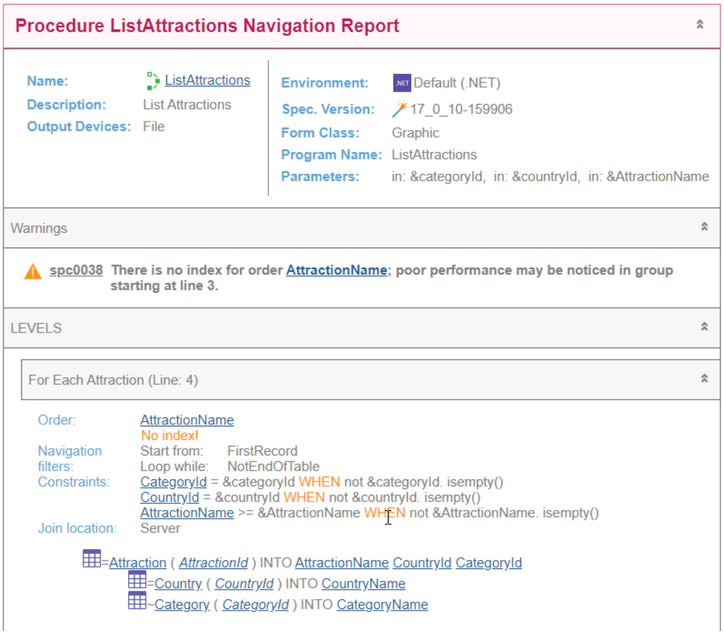
```
Event 'Print Attractions'
     ListAttractions( &CategoryId, &CountryId, &AttractionName)
Endevent
```

```
print AttractionTitles

for each Attraction
     order AttractionName
     where CategoryId = &categoryId when not &categoryId.IsEmpty()
     where CountryId = &countryId when not &countryId.IsEmpty()
     where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
          print attractionInfo
endfor
```

From this Web Panel, we want to list the tourist attractions while allowing the user to filter those of a given category—such as Tourist Site—of a given country, and whose name comes after a given value. So by pressing the button we call this procedure, passing it the three variables.

If the user doesn't enter a value in one of the filter variables, we won't want that filter to be applied, and that's why we condition the three Where clauses.

If, regardless of the filters applied, we wanted the attractions to be displayed sorted by attraction name, then we would write a single unconditional order clause.



Procedure ListAttractions Navigation Report

Name: ListAttractions
Description: List Attractions
Output Devices: File

Environment: Default (.NET)
Spec. Version: 17_0_10-159906
Form Class: Graphic
Program Name: ListAttractions
Parameters: in: &categoryId, in: &countryId, in: &AttractionName

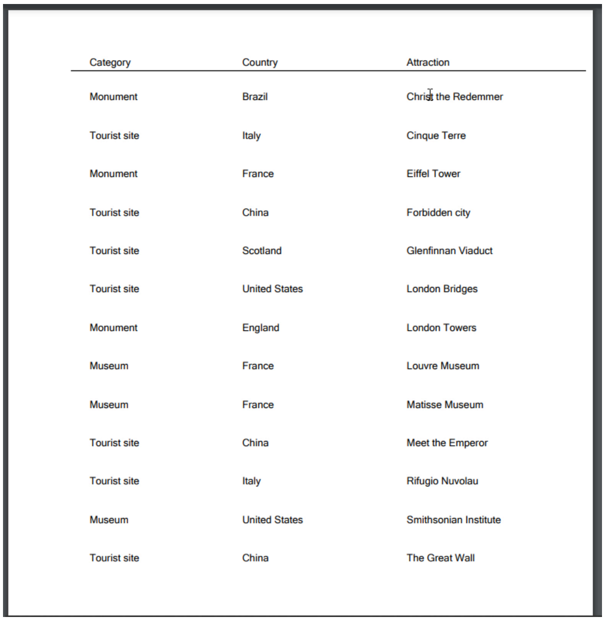
Warnings

spc0038 There is no index for order AttractionName; poor performance may be noticed in group starting at line 3.

LEVELS

For Each Attraction (Line: 4)

```
Order:          AttractionName
                No index!
Navigation      Start from:    FirstRecord
filters:        Loop while:    NotEndOfTable
Constraints:    CategoryId = &categoryId WHEN not &categoryId. isempty()
                CountryId = &countryId WHEN not &countryId. isempty()
                AttractionName >= &AttractionName WHEN not &AttractionName. isempty()
Join location:  Server

    =Attraction ( AttractionId ) INTO AttractionName CountryId CategoryId
        =Country ( CountryId ) INTO CountryName
        ~Category ( CategoryId ) INTO CategoryName
```

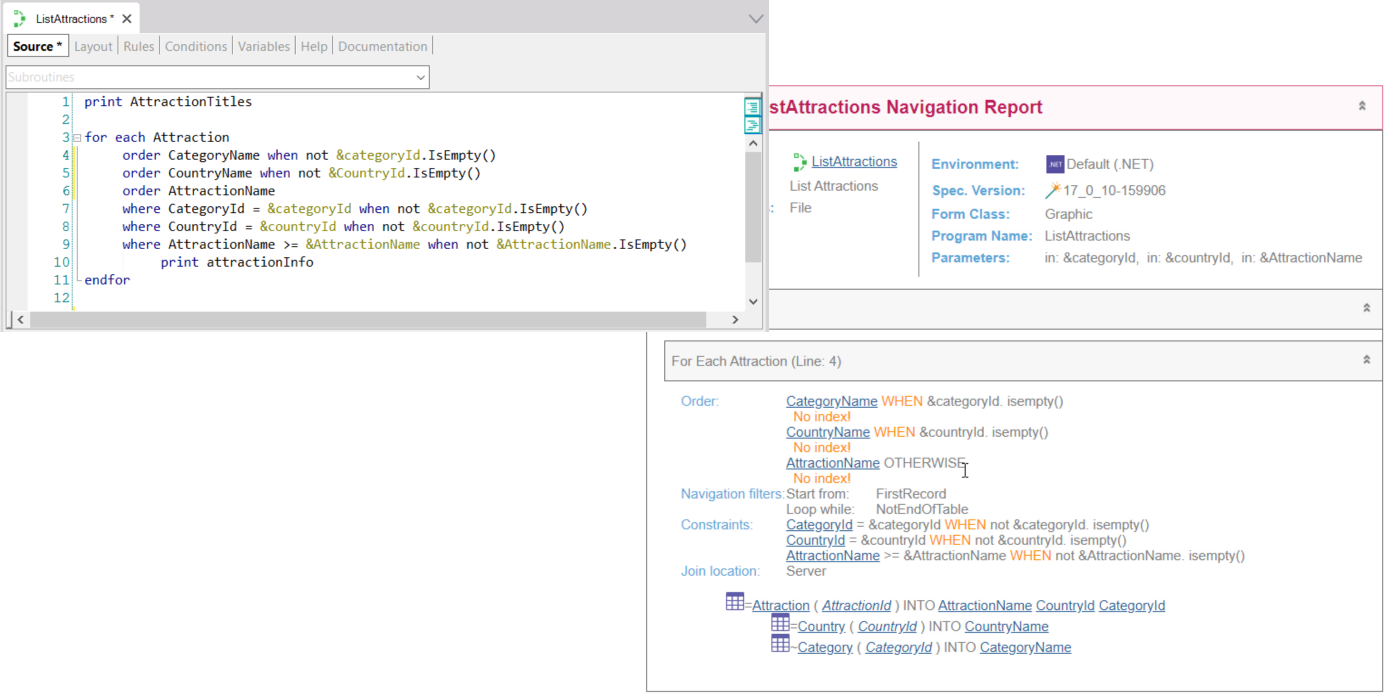


Category	Country	Attraction
Monument	Brazil	Christ the Redemmer
Tourist site	Italy	Cinque Terre
Monument	France	Eiffel Tower
Tourist site	China	Forbidden city
Tourist site	Scotland	Glenfinnan Viaduct
Tourist site	United States	London Bridges
Monument	England	London Towers
Museum	France	Louvre Museum
Museum	France	Matisse Museum
Tourist site	China	Meet the Emperor
Tourist site	Italy	Rifugio Nuvolau
Museum	United States	Smithsonian Institute
Tourist site	China	The Great Wall

Let's run the web Panel that we set as main to make it easier to run.

In the navigation list, we see that the filters are shown in the Constraints section. This is not only because there is no index for optimizing, but also because they contain the When conditions. All Where conditionals will be displayed in the Constraints section, but that doesn't mean that the query will not be optimized. We will come back to this.

Here is the list of all attractions, since none of the 3 filters will have been applied. Note they are listed sorted by attraction name, as requested.
If we now ask to list the attractions in category 3 which is Tourist Site, the result is also shown sorted by AttractionName and not sorted by country.



ListAttractions *
Source * | Layout | Rules | Conditions | Variables | Help | Documentation
Subroutines

```
print AttractionTitles

for each Attraction
      order CategoryName when not &categoryId.IsEmpty()
      order CountryName when not &CountryId.IsEmpty()
      order AttractionName
      where CategoryId = &categoryId when not &categoryId.IsEmpty()
      where CountryId = &countryId when not &countryId.IsEmpty()
      where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
            print attractionInfo
endfor

```

stAttractions Navigation Report

ListAttractions
List Attractions
File

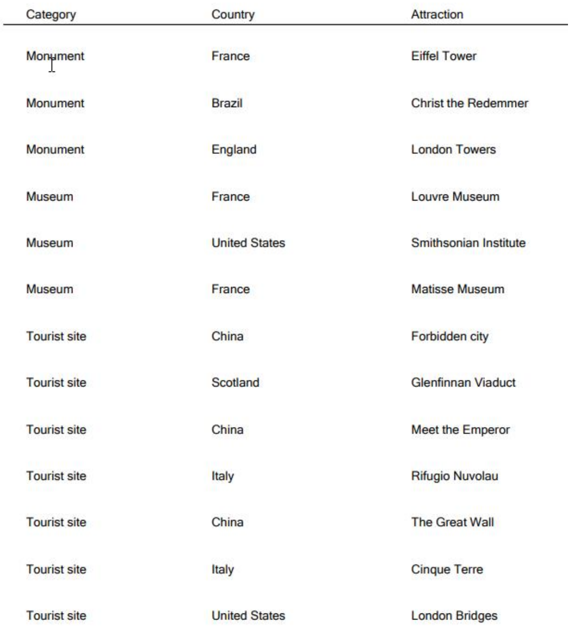
Environment: Default (.NET)
Spec. Version: 17_0_10-159906
Form Class: Graphic
Program Name: ListAttractions
Parameters: in: &categoryId, in: &countryId, in: &AttractionName

For Each Attraction (Line: 4)

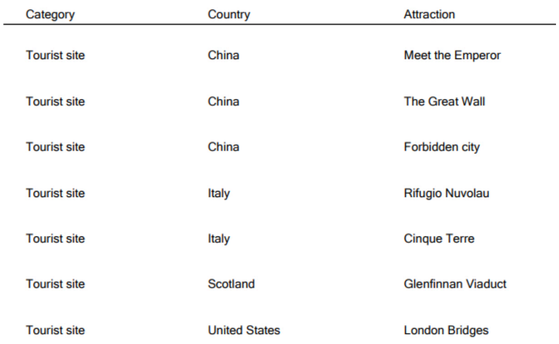
```
Order:             CategoryName WHEN &categoryId. isempty()
                    No index!
                   CountryName WHEN &countryId. isempty()
                    No index!
                   AttractionName OTHERWISE
                    No index!
Navigation filters:Start from:    FirstRecord
                   Loop while:    NotEndOfTable
Constraints:       CategoryId = &categoryId WHEN not &categoryId. isempty()
                   CountryId = &countryId WHEN not &countryId. isempty()
                   AttractionName >= &AttractionName WHEN not &AttractionName. isempty()
Join location:     Server

      =Attraction ( AttractionId ) INTO AttractionName CountryId CategoryId
          =Country ( CountryId ) INTO CountryName
          ~Category ( CategoryId ) INTO CategoryName
```

But let's suppose that if we don't filter by category—that is, this variable is empty—we want them sorted by category name and that, instead, if a value is selected for &categoryId—for example, Tourist Site—we want it to be sorted by country name if no country was selected—that is, this variable was left empty. Also, only in the opposite case—when filtering by category and country—we want it to be sorted by attraction name.

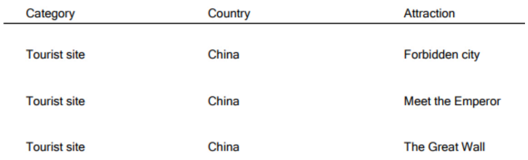
Note that the navigation list shows the conditional order clauses, where the last one is unconditional. Unlike the where clauses that do an AND between them, only one of the order clauses will be applied. For this, the first condition that is True will make its order clause the chosen one. It will only be ordered by the unconditional one if none of the conditions of the previous order clauses are satisfied. Of course, we might not place an unconditional order clause, and there the order would not be defined if none of the conditions are met.



Category	Country	Attraction
Monument	France	Eiffel Tower
Monument	Brazil	Christ the Redemmer
Monument	England	London Towers
Museum	France	Louvre Museum
Museum	United States	Smithsonian Institute
Museum	France	Matisse Museum
Tourist site	China	Forbidden city
Tourist site	Scotland	Glenfinnan Viaduct
Tourist site	China	Meet the Emperor
Tourist site	Italy	Rifugio Nuvolau
Tourist site	China	The Great Wall
Tourist site	Italy	Cinque Terre
Tourist site	United States	London Bridges

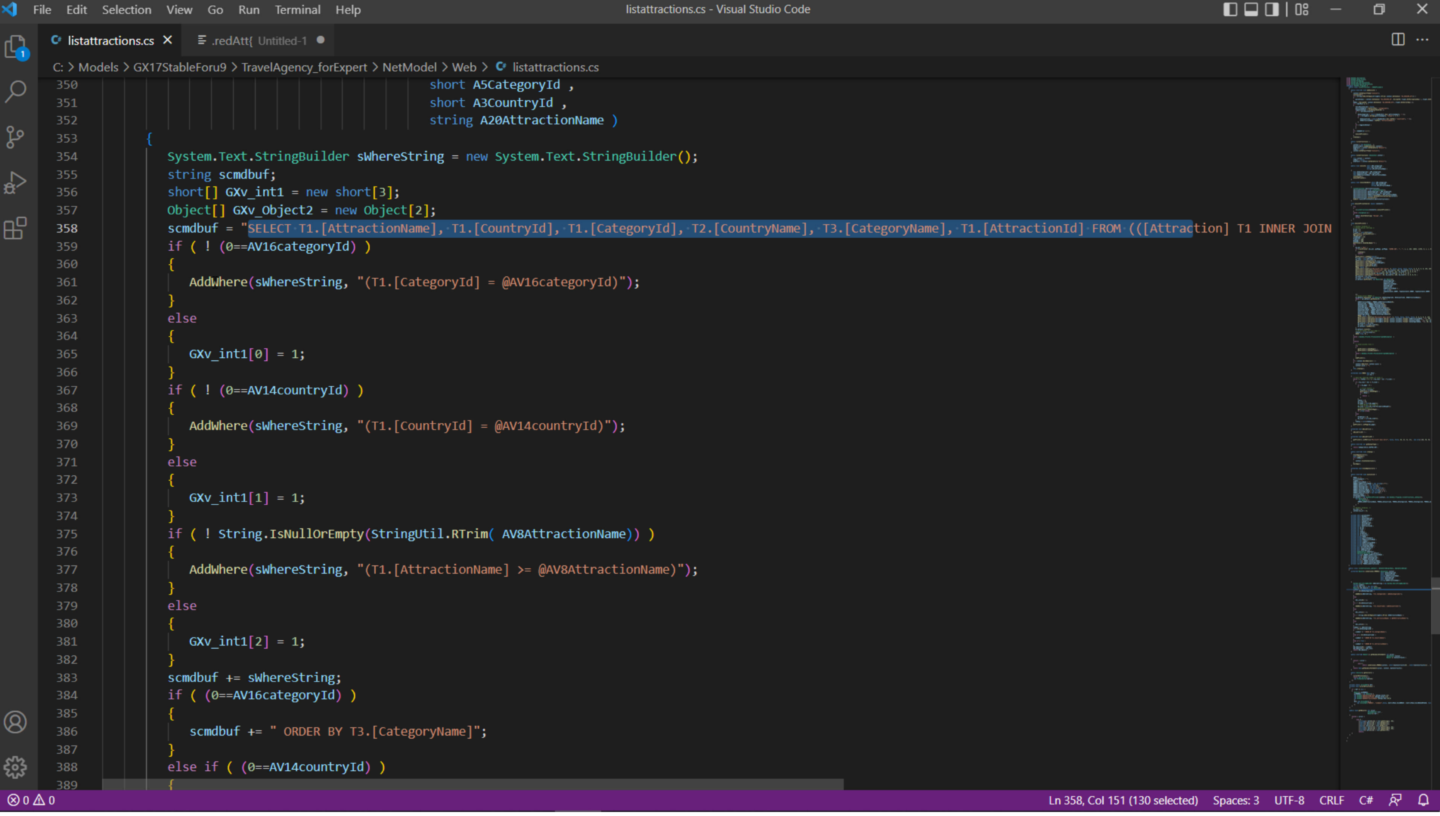


Category	Country	Attraction
Tourist site	China	Meet the Emperor
Tourist site	China	The Great Wall
Tourist site	China	Forbidden city
Tourist site	Italy	Rifugio Nuvolau
Tourist site	Italy	Cinque Terre
Tourist site	Scotland	Glenfinnan Viaduct
Tourist site	United States	London Bridges



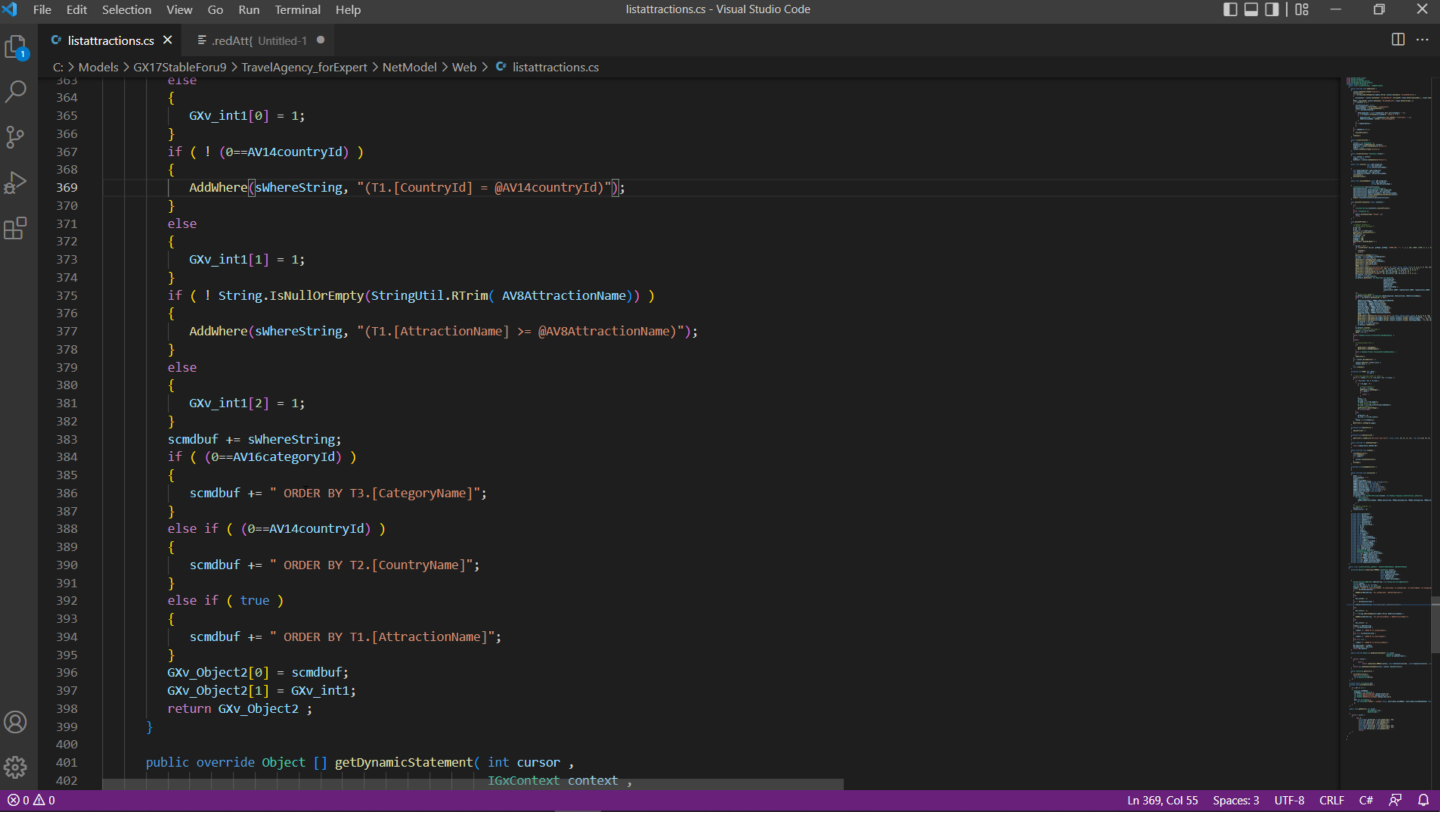
Category	Country	Attraction
Tourist site	China	Forbidden city
Tourist site	China	Meet the Emperor
Tourist site	China	The Great Wall

We can quickly see that if we don't select a category, it will be sorted by it.
On the other hand, if we do select a category and leave the country unselected, then it will be sorted by country and not sorted by the rest.
If we now select category and country, it will be sorted by attraction name.


File
Edit
Selection
View
Go
Run
Terminal
Help
listattractions.cs - Visual Studio Code
listattractions.cs
.redAtt{ Untitled-1
C: > Models > GX17StableForu9 > TravelAgency_forExpert > NetModel > Web > listattractions.cs
⊗ 0 ⚠ 0
Ln 358, Col 151 (130 selected)
Spaces: 3
UTF-8
CRLF
C#

```
                                                short A5CategoryId ,
                                                short A3CountryId ,
                                                string A20AttractionName )
   {
      System.Text.StringBuilder sWhereString = new System.Text.StringBuilder();
      string scmdbuf;
      short[] GXv_int1 = new short[3];
      Object[] GXv_Object2 = new Object[2];
      scmdbuf = "SELECT T1.[AttractionName], T1.[CountryId], T1.[CategoryId], T2.[CountryName], T3.[CategoryName], T1.[AttractionId] FROM (([Attraction] T1 INNER JOIN
      if ( ! (0==AV16categoryId) )
      {
         AddWhere(sWhereString, "(T1.[CategoryId] = @AV16categoryId)");
      }
      else
      {
         GXv_int1[0] = 1;
      }
      if ( ! (0==AV14countryId) )
      {
         AddWhere(sWhereString, "(T1.[CountryId] = @AV14countryId)");
      }
      else
      {
         GXv_int1[1] = 1;
      }
      if ( ! String.IsNullOrEmpty(StringUtil.RTrim( AV8AttractionName)) )
      {
         AddWhere(sWhereString, "(T1.[AttractionName] >= @AV8AttractionName)");
      }
      else
      {
         GXv_int1[2] = 1;
      }
      scmdbuf += sWhereString;
      if ( (0==AV16categoryId) )
      {
         scmdbuf += " ORDER BY T3.[CategoryName]";
      }
      else if ( (0==AV14countryId) )
      {
```

If we look at the source, to see how the SQL statement sent to the manager is built... we see that first it assembles the first fixed part of the select (that of the attributes to be selected and from which tables with the joins to access the extended one...) but then it dynamically complements the Where part from the evaluation of the variables (adding Where when the variables are not empty).



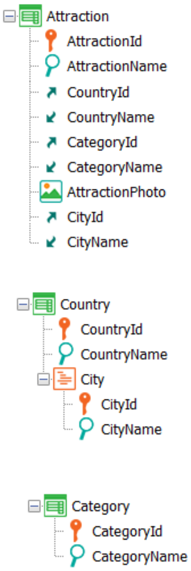
File Edit Selection View Go Run Terminal Help
listattractions.cs - Visual Studio Code
listattractions.cs
.redAtt{ Untitled-1
C: > Models > GX17StableForu9 > TravelAgency_forExpert > NetModel > Web > listattractions.cs
0 0
Ln 369, Col 55 Spaces: 3 UTF-8 CRLF C#

```
      else
      {
         GXv_int1[0] = 1;
      }
      if ( ! (0==AV14countryId) )
      {
         AddWhere(sWhereString, "(T1.[CountryId] = @AV14countryId)");
      }
      else
      {
         GXv_int1[1] = 1;
      }
      if ( ! String.IsNullOrEmpty(StringUtil.RTrim( AV8AttractionName)) )
      {
         AddWhere(sWhereString, "(T1.[AttractionName] >= @AV8AttractionName)");
      }
      else
      {
         GXv_int1[2] = 1;
      }
      scmdbuf += sWhereString;
      if ( (0==AV16categoryId) )
      {
         scmdbuf += " ORDER BY T3.[CategoryName]";
      }
      else if ( (0==AV14countryId) )
      {
         scmdbuf += " ORDER BY T2.[CountryName]";
      }
      else if ( true )
      {
         scmdbuf += " ORDER BY T1.[AttractionName]";
      }
      GXv_Object2[0] = scmdbuf;
      GXv_Object2[1] = GXv_int1;
      return GXv_Object2 ;
   }

   public override Object [] getDynamicStatement( int cursor ,
                                                  IGxContext context ,
```

And for the ORDER BY of the SQL statement it does something similar, only with nested if, to reflect exactly what we said before: only an order clause will be added to the Select.

These evaluations to obtain the final SQL statement that is sent to the DBMS to solve the query are done dynamically at runtime. Every time this list is run, this section of code must be executed to build the final query.

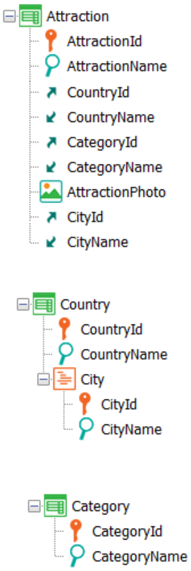
```
For each

    For each Attraction

        order CategoryName when &categoryId.IsEmpty()
        order CountryName when &countryName.IsEmpty()
        order AttractionName

        where CategoryId = &categoryId when not &categoryId.IsEmpty()
        where CountryId = &countryId when not &countryId.IsEmpty()
         where AttractionName >= &attractionName when not &attractionName.IsEmpty()

         print info // CategoryName, CountryName, AttractionName

    endfor

endfor
```



```
Attraction
    AttractionId
    AttractionName
    CountryId
    CountryName
    CategoryId
    CategoryName
    AttractionPhoto
    CityId
    CityName

Country
    CountryId
    CountryName
    City
        CityId
        CityName

Category
    CategoryId
    CategoryName
```

If this For each were included in another repetitive structure run for millions of records, the cost of dynamically building the query could be significant.

```
For each Attraction

   order CategoryName, CountryName, AttractionName

    where CategoryId = &categoryId when not &categoryId.IsEmpty()
    where CountryId = &countryId when not &countryId.IsEmpty()
     where AttractionName >= &attractionName when not &attractionName.IsEmpty()

     print info // CategoryName, CountryName, AttractionName

endfor
```



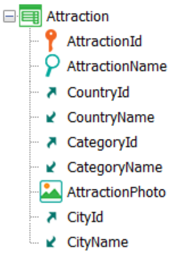
Attraction
- AttractionId
- AttractionName
- CountryId
- CountryName
- CategoryId
- CategoryName
- AttractionPhoto
- CityId
- CityName

Country
- CountryId
- CountryName
- City
  - CityId
  - CityName

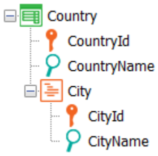
Category
- CategoryId
- CategoryName

In the previous example, we used conditional orders because we wanted to display the information sorted differently based on conditions. That is, the order clauses fulfilled a logical requirement of the query. You could say they were part of the wording of the problem, even though they were not necessary in this case. So, let's think that it was enough to choose this unconditional order to meet the requirement.

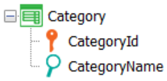
```
For each Attraction
    order CategoryId when not &categoryId.IsEmpty()
    order CountryId when not &countryId.IsEmpty()
    order AttractionName when not &attractionName.IsEmpty()

    where CategoryId = &categoryId when not &categoryId.IsEmpty()
    where CountryId = &countryId when not &countryId.IsEmpty()
     where AttractionName >= &attractionName when not &attractionName.IsEmpty()

     print info // CategoryName, CountryName, AttractionName

endfor
```



```
Attraction
    AttractionId
    AttractionName
    CountryId
    CountryName
    CategoryId
    CategoryName
    AttractionPhoto
    CityId
    CityName
```



```
Country
    CountryId
    CountryName
    City
        CityId
        CityName
```

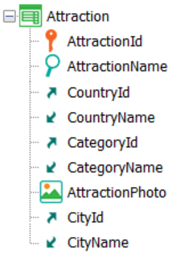


```
Category
    CategoryId
    CategoryName
```

But many times, as we saw for the case of a single unconditional order clause, it is specified for optimization purposes and is not a requirement. In such cases, choosing filter-compatible orders is usually a good practice, especially in the case of unintelligent DBMSs.

For example, if we didn't care about the order in which the information would be listed, we could place these other order clauses. This will translate dynamically as follows: if &categoryId is not empty, then we know the query will be similar to....

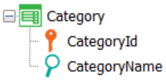
```
For each Attraction
    order CategoryId
    where CategoryId = &categoryId
    where CountryId = &countryId when not &countryId.IsEmpty()
     where AttractionName >= &attractionName when not &attractionName.IsEmpty()
     print info // CategoryName, CountryName, AttractionName
endfor
```



- Attraction
  - AttractionId
  - AttractionName
  - CountryId
  - CountryName
  - CategoryId
  - CategoryName
  - AttractionPhoto
  - CityId
  - CityName

```
For each Attraction
    order CategoryId
    where CategoryId = &categoryId
    where CountryId = &countryId
     where AttractionName >= &attractionName
     print info // CategoryName, CountryName, AttractionName
endfor
```

```
For each Attraction
    order CategoryId
    where CategoryId = &categoryId
    where AttractionName >= &attractionName
     print info // CategoryName, CountryName, AttractionName
endfor
```



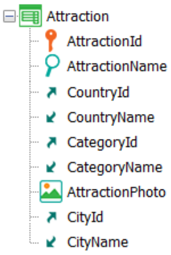
- Category
  - CategoryId
  - CategoryName

```
For each Attraction
    order CategoryId
    where CategoryId = &categoryId
    where CountryId = &countryId
     print info // CategoryName, CountryName, AttractionName
endfor
```

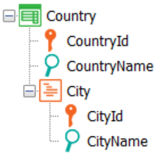
...this one, where depending on whether &countryId is empty or not, and whether &attractionName is empty or not, the final query will look like this, like this, or like this.

Note that, in any case, since there is an index by CategoryId, at least the first Where clause will be optimized.

```
For each Attraction
   ~~order CategoryId when not &categoryId.IsEmpty()~~
   order CountryId when not &countryId.IsEmpty()
   order AttractionName when not &attractionName.IsEmpty()

   ~~where CategoryId = &categoryId when not &categoryId.IsEmpty()~~
   where CountryId = &countryId when not &countryId.IsEmpty()
    where AttractionName >= &attractionName when not &attractionName.IsEmpty()

    print info // CategoryName, CountryName, AttractionName

endfor
```



Attraction
- AttractionId
- AttractionName
- CountryId
- CountryName
- CategoryId
- CategoryName
- AttractionPhoto
- CityId
- CityName



Country
- CountryId
- CountryName
- City
  - CityId
  - CityName

```
For each Attraction
   order CountryId
   where CountryId = &countryId
    where AttractionName >= &attractionName
    print info // CategoryName, CountryName, AttractionName
endfor
```

```
For each Attraction
   order CountryId
   where CountryId = &countryId
   print info // CategoryName, CountryName, AttractionName
endfor
```

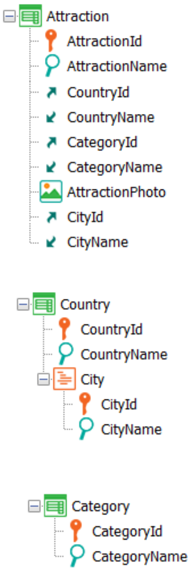
On the other hand, if &categoryId is empty, then if &countryId is not, the query will look like this or like that, depending on whether &attractionName is empty or not.

In either case, it will be optimized in relation to the filter by CountryId, since it has an index because it is a foreign key.

```
For each Attraction
    ~~order CategoryId when not &categoryId.IsEmpty()~~
    ~~order CountryId when not &countryId.IsEmpty()~~
    order AttractionName when not &attractionName.IsEmpty()

    ~~where CategoryId = &categoryId when not &categoryId.IsEmpty()~~
    ~~where CountryId = &countryId when not &countryId.IsEmpty()~~
     where AttractionName >= &attractionName when not &attractionName.IsEmpty()

     print info // CategoryName, CountryName, AttractionName

endfor
```

```
For each Attraction
    order AttractionName
    where AttractionName >= &attractionName
     print info // CategoryName, CountryName, AttractionName
endfor
```

```
For each Attraction
   print info // CategoryName, CountryName, AttractionName
Endfor
```

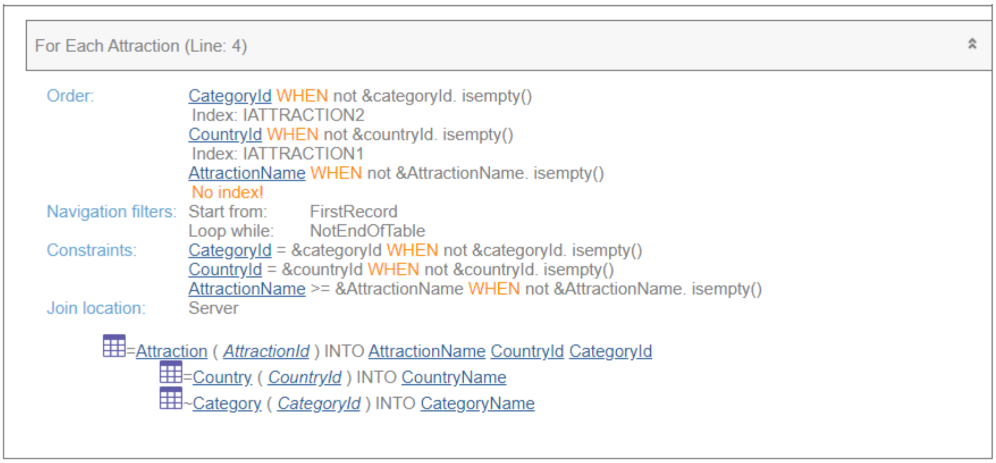


```
Attraction
    AttractionId
    AttractionName
    CountryId
    CountryName
    CategoryId
    CategoryName
    AttractionPhoto
    CityId
    CityName

Country
    CountryId
    CountryName
    City
        CityId
        CityName

Category
    CategoryId
    CategoryName
```

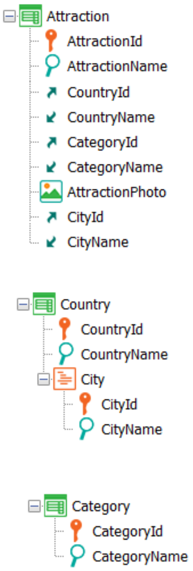
If &countryId is empty as well, then if &attractionName is not, the query will look like this. And if it is, it will look like this but the order will be undefined. This means that it can vary depending on the DBMS and even between successive executions.

In the first case, as we are not aware of the existence of an index by AttractionName, we don't know how optimized the query will be.

```
for each Attraction
   order CategoryId when not &categoryId.IsEmpty()
   order CountryId when not &CountryId.IsEmpty()
   order AttractionName when not &AttractionName.IsEmpty()
   where CategoryId = &categoryId when not &categoryId.IsEmpty()
   where CountryId = &countryId when not &countryId.IsEmpty()
   where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
      print attractionInfo
endfor
```

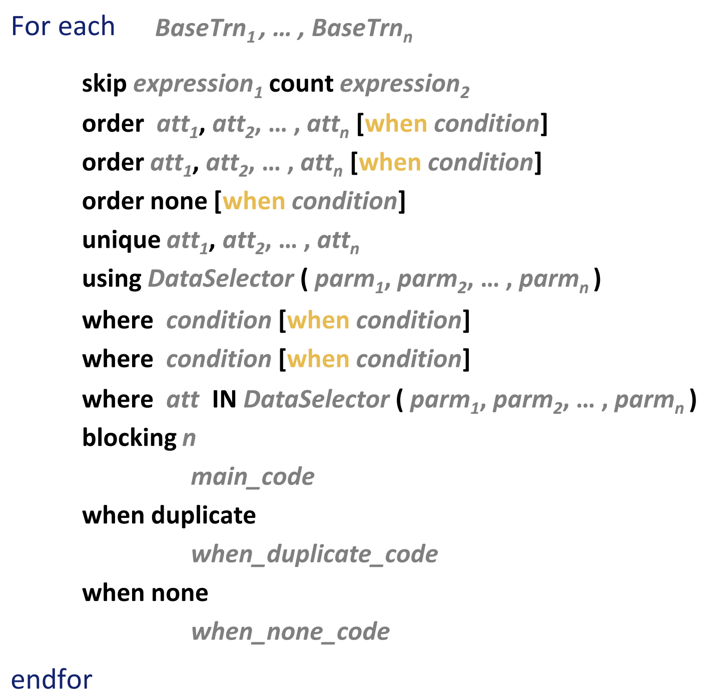


```
For Each Attraction (Line: 4)

Order:               CategoryId WHEN not &categoryId. isempty()
                      Index: IATTRACTION2
                     CountryId WHEN not &countryId. isempty()
                      Index: IATTRACTION1
                     AttractionName WHEN not &AttractionName. isempty()
                      No index!
Navigation filters:  Start from:    FirstRecord
                     Loop while:    NotEndOfTable
Constraints:         CategoryId = &categoryId WHEN not &categoryId. isempty()
                     CountryId = &countryId WHEN not &countryId. isempty()
                     AttractionName >= &AttractionName WHEN not &AttractionName. isempty()
Join location:       Server

     =Attraction ( AttractionId ) INTO AttractionName CountryId CategoryId
        =Country ( CountryId ) INTO CountryName
        ~Category ( CategoryId ) INTO CategoryName
```



```
Attraction
   AttractionId
   AttractionName
   CountryId
   CountryName
   CategoryId
   CategoryName
   AttractionPhoto
   CityId
   CityName

Country
   CountryId
   CountryName
   City
      CityId
      CityName

Category
   CategoryId
   CategoryName
```

The navigation list shows that the filters are still displayed in the Constraints section, although we know that depending on the values of the variables some of them should be displayed in the Navigation filters. This is because the navigation list doesn't perform the breakdown we did before. We must understand, then, that the scenario will be better than it may seem at first glance without taking into account all of the above.

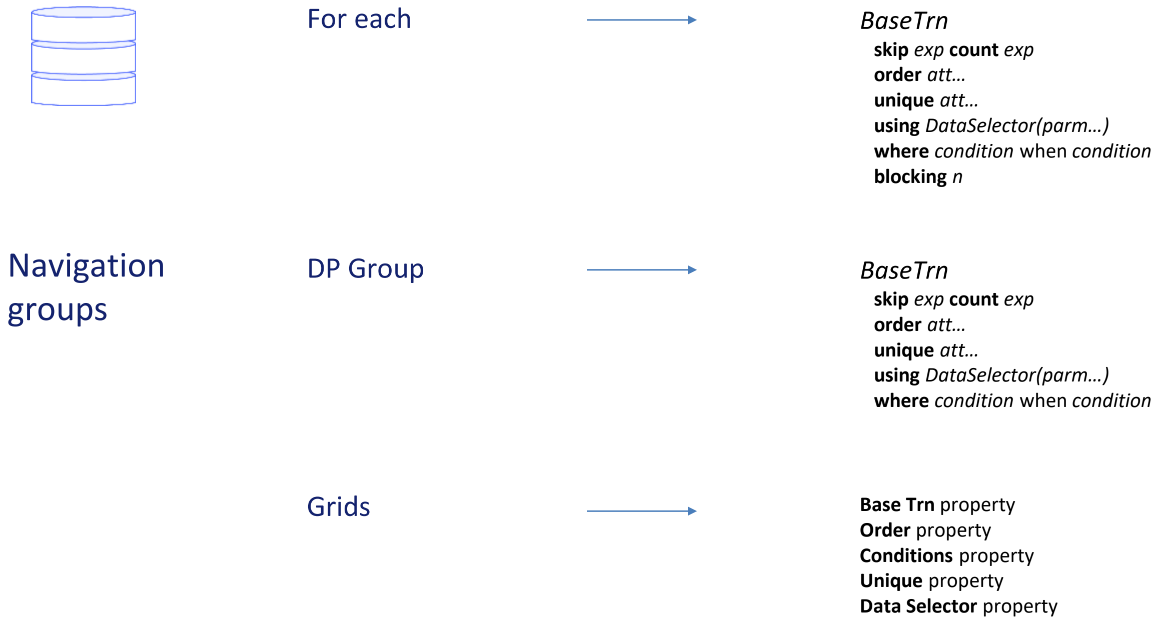


```
For each   $BaseTrn_1$ , … , $BaseTrn_n$
        skip expression$_1$ count expression$_2$
        order  $att_1$, $att_2$, … , $att_n$ [when condition]
        order $att_1$, $att_2$, … , $att_n$ [when condition]
        order none [when condition]
        unique $att_1$, $att_2$, … , $att_n$
        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
```

What else can be said about conditional orders?

They are not supported in control breaks.
They do not apply to legacy Cobol and RPG generators.
If the conditions have attributes, they are considered as instantiated; that is, they are evaluated before starting the navigation and do not change in the process.

This is the end of our exploration of query orders.



Navigation groups

For each

BaseTrn
skip *exp* **count** *exp*
order *att…*
unique *att…*
using *DataSelector(parm…)*
where *condition* when *condition*
blocking *n*

DP Group

BaseTrn
skip *exp* **count** *exp*
order *att…*
unique *att…*
using *DataSelector(parm…)*
where *condition* when *condition*

Grids

Base Trn property
Order property
Conditions property
Unique property
Data Selector property

Of course, what we saw for the For each is valid for groups of Data Providers and grids with a base table, as well as for queries with In in Data Selectors.