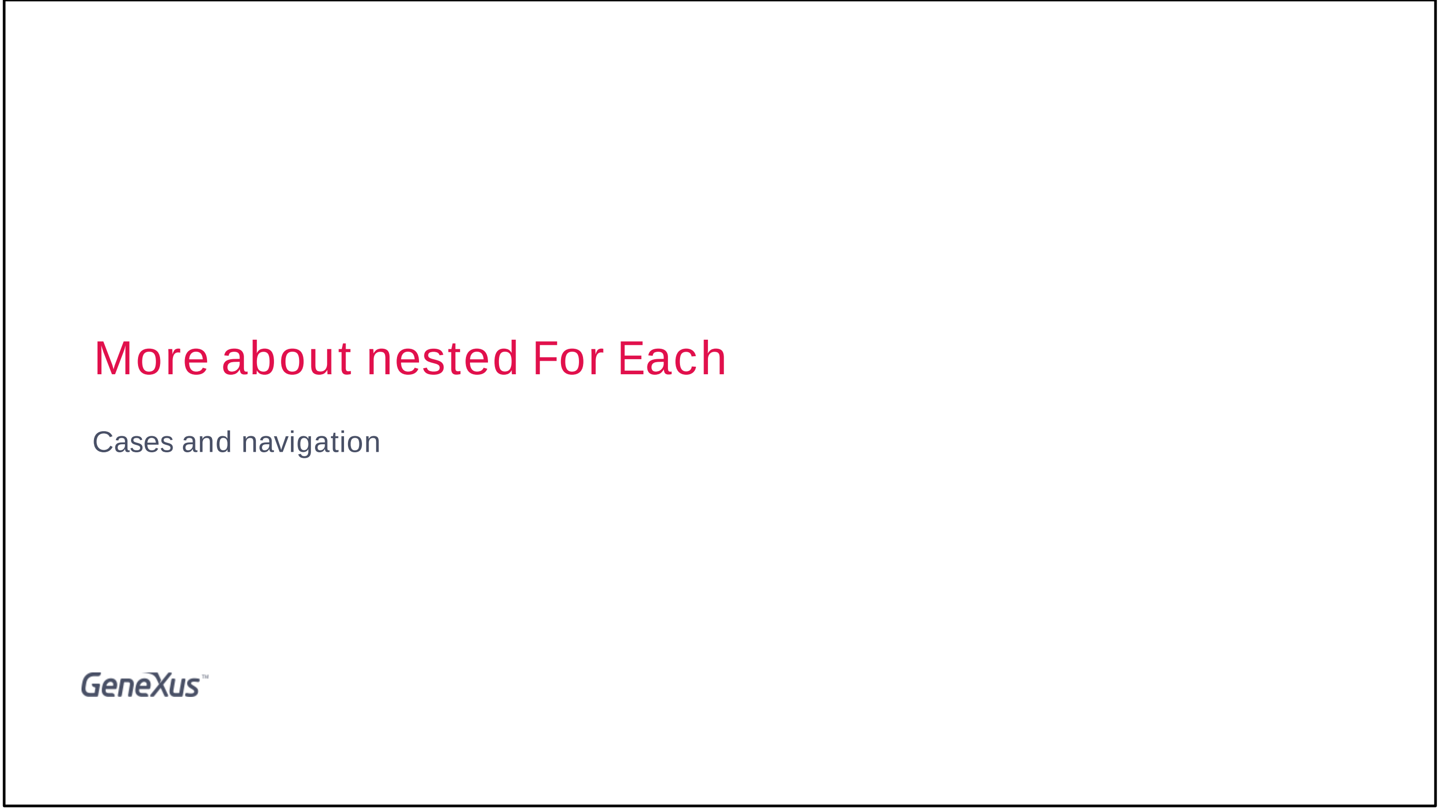


More about nested For Each

Cases and navigation

GeneXus™

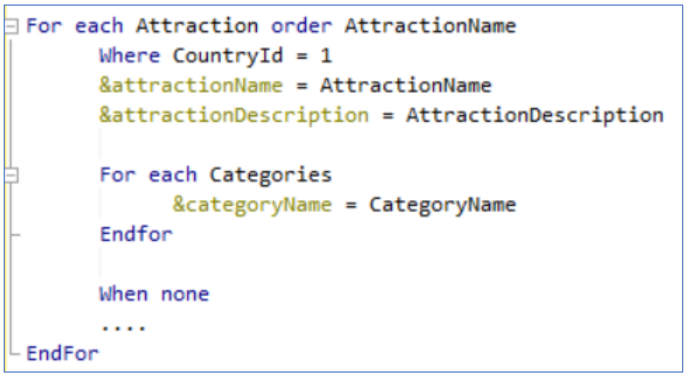
When there are nested For Each commands, GeneXus must first determine the base tables of each one of them.

When the developer indicates the Base Transaction, it is done right away. Otherwise, GeneXus has to determine each base table according to the attributes included in every For Each command. This case is more complex and will not be addressed in this video.

Next, GeneXus defines the necessary navigation to solve the multiple query. One of three options will be applied:

-Join
-Cartesian product
-Control break

Nested For Each

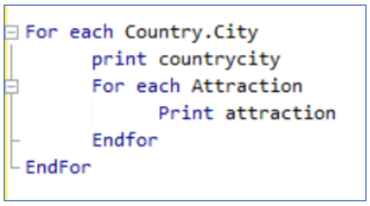


```
For each Attraction order AttractionName
        Where CountryId = 1
        &attractionName = AttractionName
        &attractionDescription = AttractionDescription

        For each Categories
                &categoryName = CategoryName
        Endfor

        When none
        ....
EndFor
```

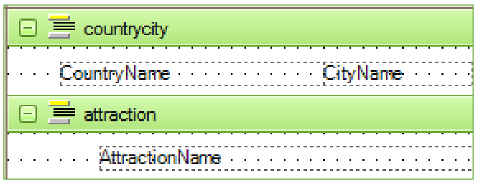
As we have said, the first thing GeneXus does when it finds a couple of nested For Each commands is determine the base table of each one of them, in an ordered manner and from the outside in, starting from the outermost one. Only then it determines how navigation will take place.

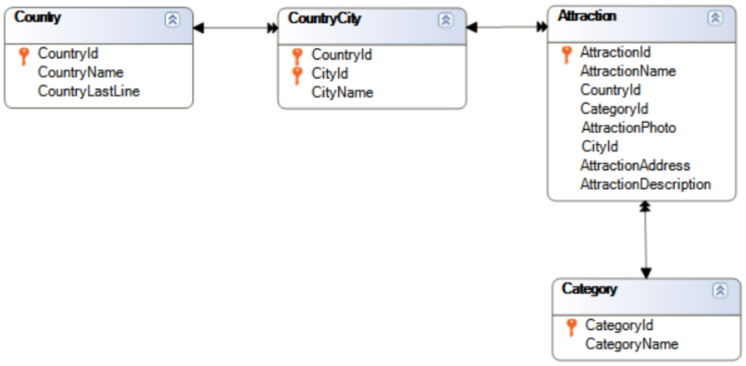
In every For Each command, the indicated base transaction and only the attributes that belong to this For Each command come into play: from the Order, Where, etc. as well as those in its body, except for those included in a nested For Each command. That is to say, removing the nested For Each command, the base table is determined as in a simple For Each command. The attributes of the When none clause are never taken into account. All attributes must belong to the extended table of the base table found, which matches the base table associated with the base transaction. The attributes that don't meet this condition will not be “instantiable” because they can't be reached.

Nested For Each



```
For each Country.City
        print countrycity
        For each Attraction
                Print attraction
        Endfor
EndFor
```





In the example, it is done in this order:

1) The base table of the external For Each command is determined. To this end, the indicated base transaction is considered; that is to say, Country.City, and it is checked whether the attributes included in the printblock (CountryName and CityName) belong to its extended table. Otherwise, a warning is displayed in the navigation list to inform the user that some attributes cannot be instantiated, because they can't be reached from the extended table of that For Each command. In this case, CountryName and CityName belong to the extended table of CountryCity, the base table of the For Each command.

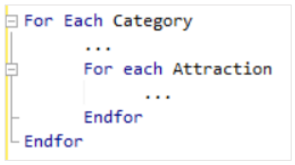
2) The base table of the nested For Each command is determined. The Attraction base transaction is considered, as well as the AttractionName attribute included in the printblock. If a base transaction hadn't been written, something related to the external For Each command attributes would be considered to determine the base table of the internal For Each command, but this is not the case. Thus, its base table is determined as if it was a standalone For Each command. Therefore, its base table will be Attraction.

Nested For Each

Different Base Table

External For each ⟷ Nested For each

Join



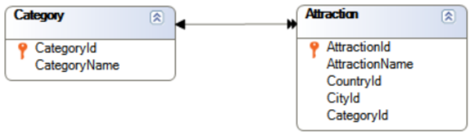
```
For Each Category
        ...
        For each Attraction
            ...
        Endfor
Endfor
```

Table Category

CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist site

Table Attraction

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2



The three examples of nested For Each commands that we've seen before are based on the determination of base tables. Now we will examine them more closely.

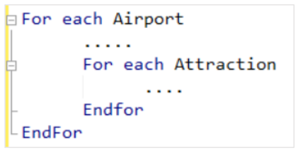
When the base tables are different, there are two possibilities: whether or not there is a direct or indirect 1 to N relationship between them. In the first case, for each record of the main For Each, the nested For Each will run its instructions only for the N associated records. The operation that cuts the data from a table by that of another one is known as a Join.

Nested For Each

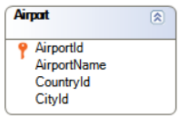
Different Base Table

External For each ~~⟶~~ Nested For each

Cartesian product



```
For each Airport
      .....
      For each Attraction
            ....
      Endfor
EndFor
```



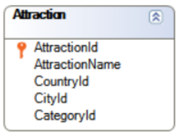


Table Airport

AirportId	AirportName	CountryId
1	Guarulhos	2
2	Charles de Gaulle	1
3	Tegel	3

Table Attraction

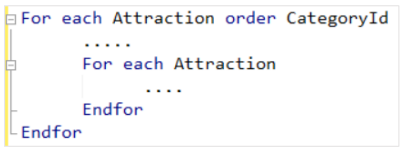
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2

In the second case, when there is no relationship, for each record considered in the main For Each, the nested For Each will run its instructions for all the records of the other table because it has found no relationship between them. This operation is known as Cartesian Product.

Nested For Each

Equal Base Table

Control break



```
For each Attraction order CategoryId
      .....
      For each Attraction
            ....
      Endfor
Endfor
```

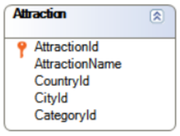


Table Attraction

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
6	The Centre Pompidou	2	1	1
7	Quai Branly	2	1	1
3	Eiffel Tower	2	1	2
5	Christ the Redemmer	1	2	2
2	The Great Wall	3	1	3
4	Forbidden city	3	1	3

When the base tables are the same, an operation known as Control Break is performed: it takes place when we need to group the data from a table, run certain instructions that consider the group’s common data and run through each member, and then run other instructions to move on to the next group and repeat the process. In this case we must indicate the attributes that make up the group using the order clause.

This case occurs because the same base transaction was specified for the external For Each command and for the nested one.
If the base transactions of the For Each commands have not been specified, as we said, GeneXus will calculate them from the attributes it finds within those For Each commands. But it will only infer that you want to implement a control break if the base tables found by it are the same. We will not go into details about it in this course.

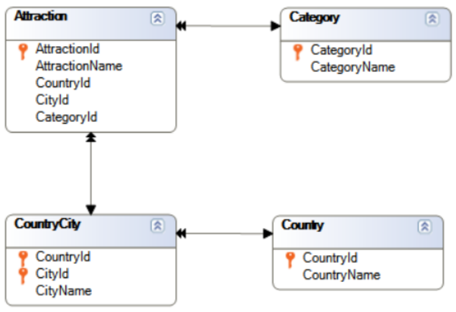
Let's look at this other case.

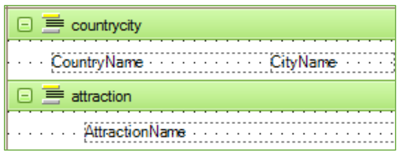
Nested For Each

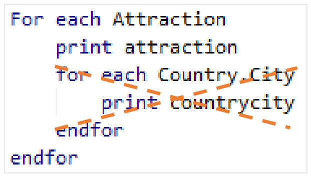
Different Base Table

External For each ⟷ Nested For each

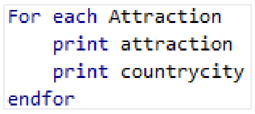
Control break







```
For each Attraction
    print attraction
    ~~for each Country.City~~
        ~~print countrycity~~
    ~~endfor~~
endfor
```



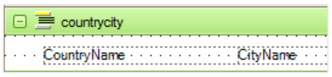
```
For each Attraction
    print attraction
    print countrycity
endfor
```

Note that here the second For Each command would be unnecessary, because for every attraction selected at a given moment in the external For Each command, there is only one related CountryCity record. So, this would be the same as not writing the second For Each command, and sending to print the CountryCity printblock that contains CountryName and CityName, which both belong to the Attraction extended table.

Nested For Each

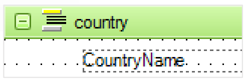
1

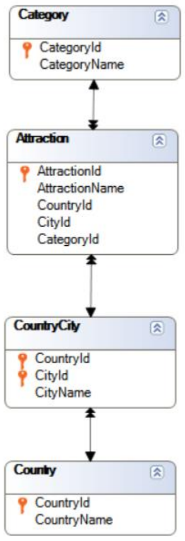
```
For each Country.City
    Print countrycity
                    Direct 1 - N
   For each Attraction
       Print attraction
   Endfor
Endfor
```


countrycity
CountryName CityName

2

```
For each Country
    Print country
                    Indirect 1 - N
   For each Attraction
       Print attraction
   Endfor
Endfor
```


country
CountryName



Let's examine these two cases of 1 to N relationship between the For Each commands, and therefore, of JOIN.

The first one is direct. Note that the base tables of the external and nested For Each are CountryCity and Attraction, respectively; they are related by a 1 to N relationship.

The name of the country and city will be printed, and for each pair, their attraction names.

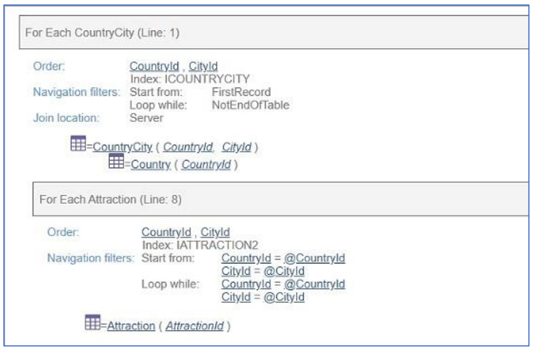
The second is indirect. The base tables of the external and nested For Each are Country and Attraction. If we look closely, each country has N cities, each one with N attractions. The base tables of the For Each commands do not have a direct 1 to N relationship, but they do have an indirect one, through the CountryCity table. In other words, note that the base table of the first For Each, Country, is included in the extended table of the base table of the nested For Each: Attraction. Then a join will be made.

Nested For Each

```
For each Country.City
    Print countrycity
```

Direct 1 - N

```
    For each Attraction
        Print attraction
    Endfor
Endfor
```

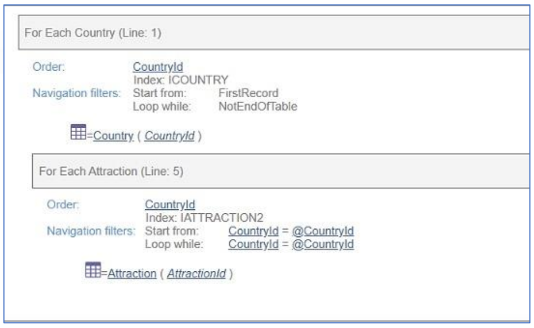


```
For Each CountryCity (Line: 1)
  Order:              CountryId , CityId
                      Index: ICOUNTRYCITY
  Navigation filters: Start from:   FirstRecord
                      Loop while:   NotEndOfTable
  Join location:      Server

        =CountryCity ( CountryId,  CityId )
              =Country ( CountryId )

  For Each Attraction (Line: 8)
    Order:              CountryId , CityId
                        Index: IATTRACTION2
    Navigation filters: Start from:   CountryId = @CountryId
                                      CityId = @CityId
                        Loop while:   CountryId = @CountryId
                                      CityId = @CityId

          =Attraction ( AttractionId )
```

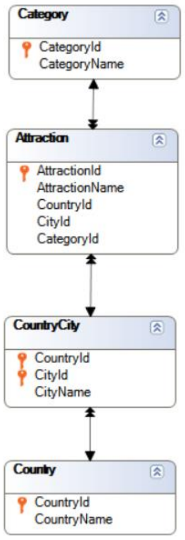
```
For each Country
    Print country
```

Indirect 1 - N

```
    For each Attraction
        Print attraction
    Endfor
Endfor
```



```
For Each Country (Line: 1)
  Order:              CountryId
                      Index: ICOUNTRY
  Navigation filters: Start from:   FirstRecord
                      Loop while:   NotEndOfTable

        =Country ( CountryId )

  For Each Attraction (Line: 5)
    Order:              CountryId
                        Index: IATTRACTION2
    Navigation filters: Start from:   CountryId = @CountryId
                        Loop while:   CountryId = @CountryId

          =Attraction ( AttractionId )
```



Let's look at the navigation lists. For the nested For Each, which in both cases navigates Attraction, the entire table is not run through. Note that in the first case, where the relationship is direct, the at sign (@) indicates that it will be filtered by the composite foreign key that establishes the relationship, CountryId and CityId. In the second case, we see that it will be only by CountryId, which is part of the composite foreign key.

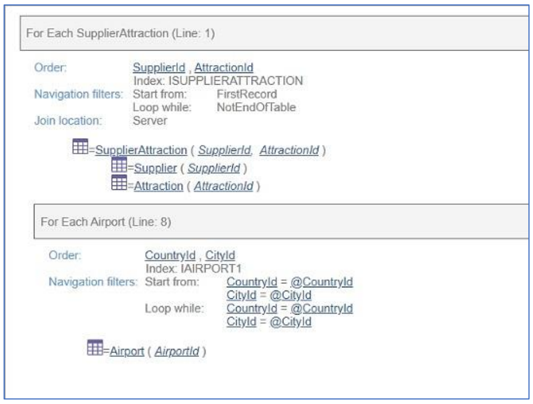
Note that in both cases instead of ordering the navigation by the primary key of Attraction, which is AttractionId, it does so by the relation attribute or set of attributes. To do so, it has an index automatically created by foreign key. In this way, the database access will be optimized.

Therefore, after determining that it will make a Join, GeneXus tries to optimize its navigation.

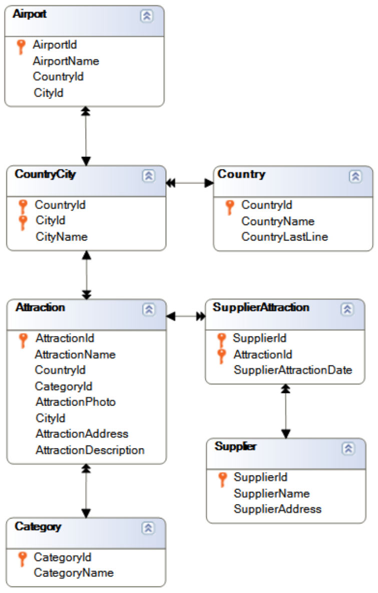
Nested For Each

```
For each Supplier.Attraction
    print info
            ↕  Indirect 1 - N
    For each Airport
        print airports
    endfor
endfor
```

Join



```
For Each SupplierAttraction (Line: 1)

Order:              SupplierId , AttractionId
                    Index: ISUPPLIERATTRACTION
Navigation filters: Start from:   FirstRecord
                    Loop while:   NotEndOfTable
Join location:      Server

    =SupplierAttraction ( SupplierId,  AttractionId )
        =Supplier ( SupplierId )
        =Attraction ( AttractionId )

  For Each Airport (Line: 8)

  Order:              CountryId , CityId
                      Index: IAIRPORT1
  Navigation filters: Start from:   CountryId = @CountryId
                                    CityId = @CityId
                      Loop while:   CountryId = @CountryId
                                    CityId = @CityId

      =Airport ( AirportId )
```



Here is a third example. In this case, suppliers have been added that offer tourist attractions, and therefore two more tables appear: Supplier and its subordinate, SupplierAttraction.
In addition, we have the airports, which belong to a country and city.
Then if we look at the For Each, we see that the base table of the external For Each is SupplierAttraction and the base table of the nested For Each is Airport. Let's assume for this example that the printblock info shows the name of the supplier, the name of the attraction and the date on which that attraction will be shown, and the second printblock shows the name of the airport.

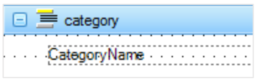
Note that there is an indirect 1 to N relationship. That is to say, for every SupplierAttraction record, there will be only one Attraction record, given that we obtain only one from CountryCity, which in turn is related to N records of Airport (the N airports found in that country/city, even though, in general, there is actually only one. In this model there may be several).

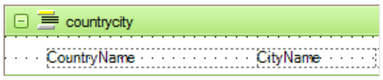
Here GeneXus finds attributes in common between the extended table of the main For Each command and the base table of the nested For Each command. Which ones? The pair {CountryId, CityId}. It will make the Join through them.

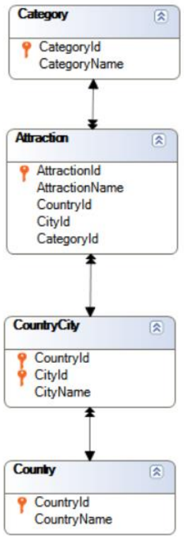
For optimization reasons, we can see that it chooses the index by foreign key of the Airport table. In short, this is a case where the relationship between the base tables is not directly 1 to N, but indirect, through intermediary tables.

Nested For Each

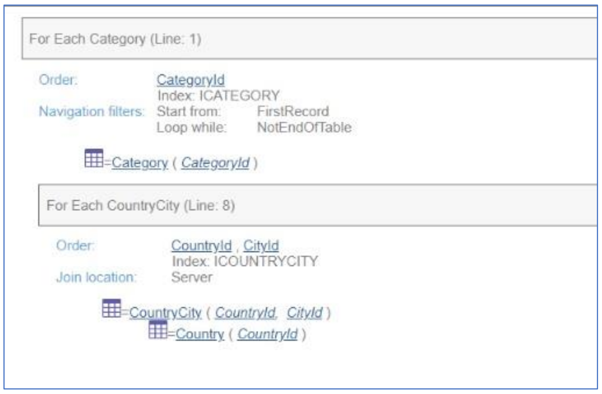
```
For each Category
    Print
    
    For each Country.City
        Print
    Endfor
Endfor
```







Cartesian product



```
For Each Category (Line: 1)

Order:              CategoryId
                    Index: ICATEGORY
Navigation filters: Start from:   FirstRecord
                    Loop while:   NotEndOfTable

    =Category ( CategoryId )

    For Each CountryCity (Line: 8)

    Order:          CountryId , CityId
                    Index: ICOUNTRYCITY
    Join location:  Server

        =CountryCity ( CountryId, CityId )
            =Country ( CountryId )
```

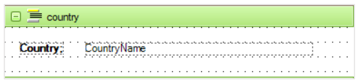
Let's see now this case of nested For Each commands, with different and unrelated base tables.

Here the first for each navigates Category, while the nested one navigates CountryCity. In this case, there is no 1 to N relationship of any kind, contrary to what it might seem. Looking at the table diagram, we see that a Category has N attractions, each of which has one and only one CountryCity. One would think that for each category, the CountryCity of each of its attractions would be listed. That would be true if the base table of the nested For Each were Attraction. But it is not, it is CountryCity. So GeneXus understands that we are not looking for the records related by attraction. If we were looking for those, it would be enough to specify Attraction as the base transaction for the second For Each. This can be a misleading case, so it is essential to read the navigation list carefully to avoid confusion. In the listing we can clearly see that no join is being made.

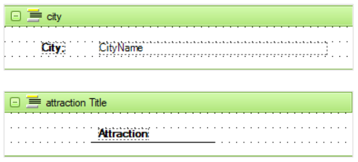
In this case, GeneXus can't find a direct or indirect 1-N relationship between the tables; therefore, it doesn't apply implicit filters to the nested For Each records. That is to say, it makes a Cartesian Product between the tables: for each base table record of the external For Each (Category), it considers all records of the nested For Each base table (CountryCity).

Nested For Each

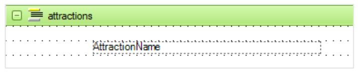
```
For each Attraction order CountryName
          Print
          For each Attraction order CityName
                    Print
                    Print
                    For each Attraction
                              Print
                    Endfor
          Endfor
Endfor
```

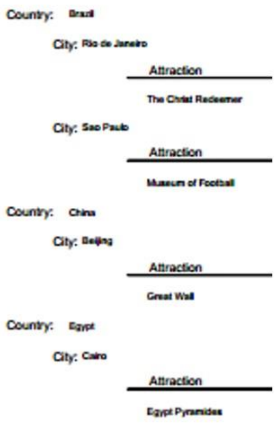


Control Break



Control Break





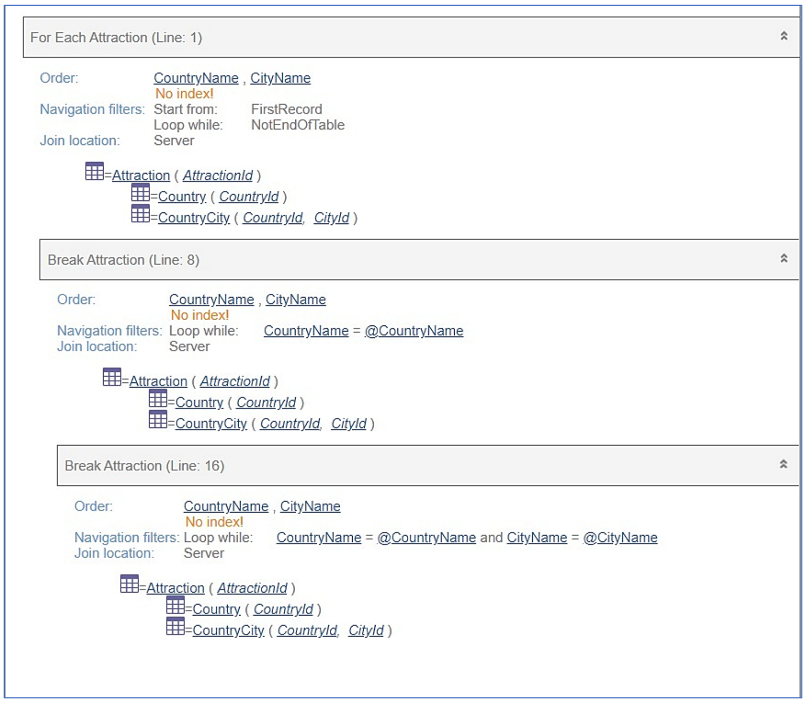
```
Country:   Brazil
      City: Rio de Janeiro
                   Attraction
                   The Christ Redeemer
      City: Sao Paulo
                   Attraction
                   Museum of Football
Country:   China
      City: Beijing
                   Attraction
                   Great Wall
Country:   Egypt
      City: Cairo
                   Attraction
                   Egypt Pyramides
```

In this case we want to list all countries; for each one of them we want to list their cities, and for each city we want to list their attractions. The only restriction is that we want to do it only for the countries and cities that have tourist attractions recorded.

That is to say, we will have to implement a double control break, in which first we group by country, and within it we will then group by city. Within the latter group, we will show the names of all attractions. To do so, we will:

Define the grouping criteria using order clauses. Remember that the order is very important in a control break; it not only indicates the attribute or attributes used to list data, but also sets how to group it.

We could indicate an order for the innermost For Each, but this order will only be used in the conventional manner. That is, it will be used only for ordering purposes. We'll see this soon.



```
For Each Attraction (Line: 1)

Order:               CountryName , CityName
                     No index!
Navigation filters:  Start from:   FirstRecord
                     Loop while:   NotEndOfTable
Join location:       Server

        =Attraction ( AttractionId )
              =Country ( CountryId )
              =CountryCity ( CountryId,  CityId )

Break Attraction (Line: 8)

Order:               CountryName , CityName
                     No index!
Navigation filters:  Loop while:   CountryName = @CountryName
Join location:       Server

        =Attraction ( AttractionId )
              =Country ( CountryId )
              =CountryCity ( CountryId,  CityId )

Break Attraction (Line: 16)

Order:               CountryName , CityName
                     No index!
Navigation filters:  Loop while:   CountryName = @CountryName and CityName = @CityName
Join location:       Server

        =Attraction ( AttractionId )
              =Country ( CountryId )
              =CountryCity ( CountryId,  CityId )
```

Control Break

Control Break

We have a double control break, which implies three For Each Commands
If we look at the navigation list, we can see the word Break for every internal For Each, indicating the same base table, Attraction, and therefore, a control break.

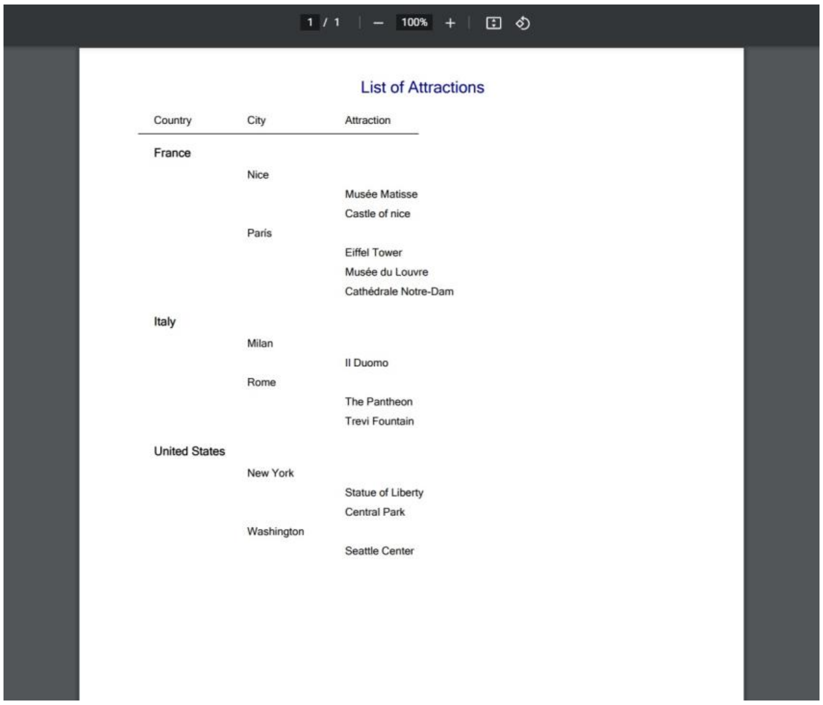
In addition, it will run through this base table only once. To do so, it needs to order by the concatenation of the attributes included in the orders of the For Each commands. That's why it chooses CountryName, CityName.

Note that in the second For Each, the break is performed by country, iterating on the country it is positioned in the first For Each. In the third For Each, the break is performed by country and city, iterating on the city it is positioned in the second For Each.

Here we see an example of how it would be displayed if this were the data. First we see the country, and for it we see the first of its cities according to its name, and the tourist attractions of that country and city. Then we see the next city of the same country, and its attractions, and only when there are no more cities of that country, the next one is shown, in alphabetical order, and it starts all over again. Country, city, attractions.

Think about how the previous list will be executed if instead of ordering the first For Each by CountryName and the second by CityName, we would have sorted both or just the first one by the pair CountryName, CityName.

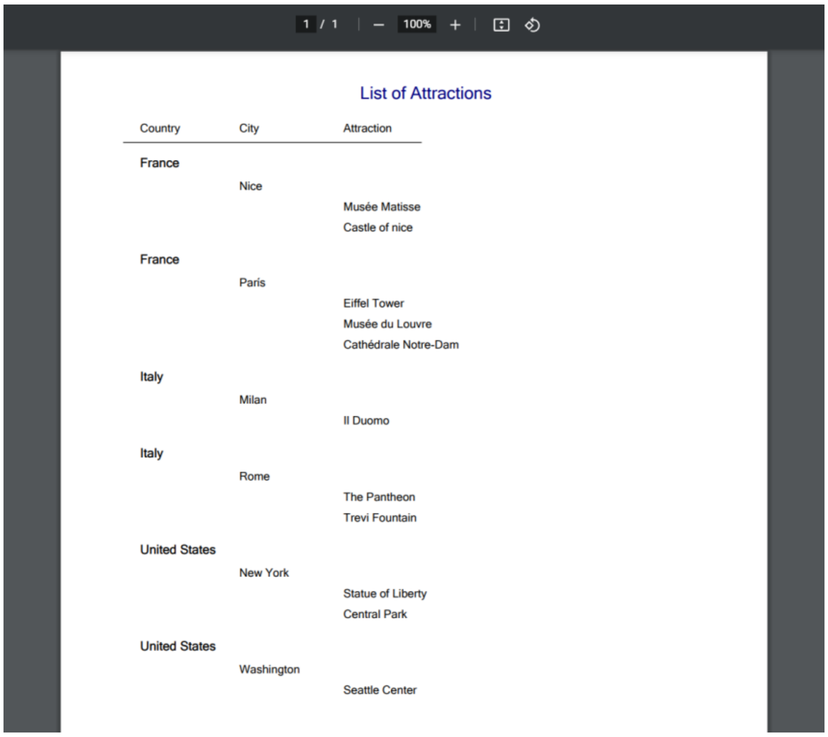
Note that in this case the navigation list will be different from the one shown above, in the second For Each. There, Loop while will read "CountryName = @CountryName and CityName = @CityName".



List of Attractions

Country	City	Attraction
France		
	Nice	
		Musée Matisse
		Castle of nice
	París	
		Eiffel Tower
		Musée du Louvre
		Cathédrale Notre-Dam
Italy		
	Milan	
		Il Duomo
	Rome	
		The Pantheon
		Trevi Fountain
United States		
	New York	
		Statue of Liberty
		Central Park
	Washington	
		Seattle Center

This means that at runtime, now the same information will be seen in this other way. That is, the country comes out... its first city... and its tourist attractions. Then, the same country and its second city... and its tourist attractions, and so on until the country changes and the same thing happens again for the next one.
Clearly the information is being presented differently just by having placed CityName in the first or in the second For Each.



List of Attractions

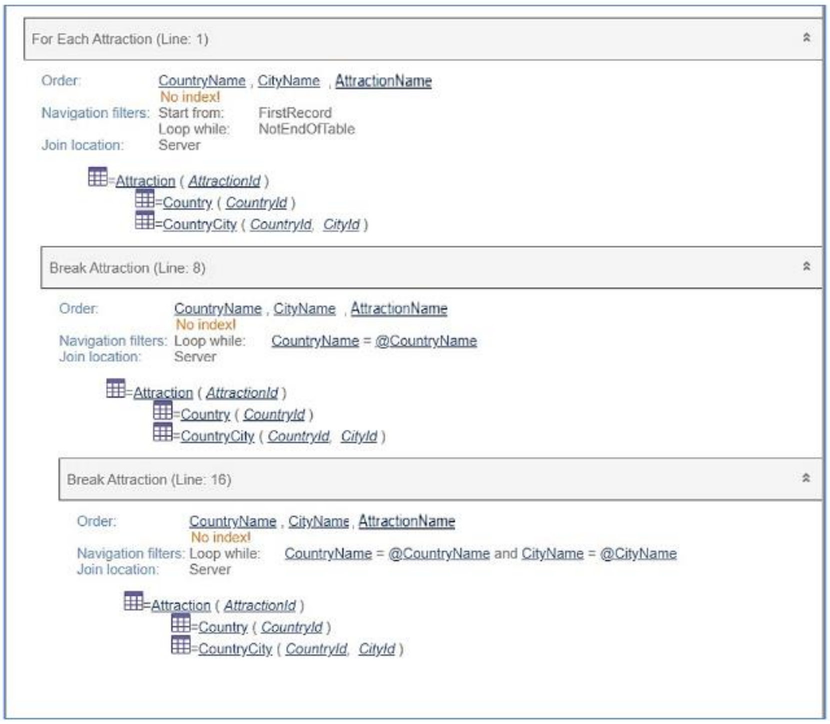
Country	City	Attraction
France		
	Nice	
		Musée Matisse
		Castle of nice
France		
	Paris	
		Eiffel Tower
		Musée du Louvre
		Cathédrale Notre-Dam
Italy		
	Milan	
		Il Duomo
Italy		
	Rome	
		The Pantheon
		Trevi Fountain
United States		
	New York	
		Statue of Liberty
		Central Park
United States		
	Washington	
		Seattle Center

Now let's suppose that in the first example, in which the break is made by country and then by city, to then list the attractions, we add to the last For Each, the innermost one, an order by AttractionName. That is to say, we will want the attractions of a country and city to be ordered by attraction name.

The navigation list will change, and AttractionName will be added to the order of the three For Each commands, leaving CountryName, CityName and AttractionName.

And if we run with this change, we see exactly what we were saying: the break is being made by country name, and that is why France comes out first and Italy much later. Within France, the break is made by CityName, and that is why Nice, which is alphabetically before Paris, comes out first; within each of these subgroups, the attractions are ordered alphabetically:

This is contrary to what happened in the first case, when we did not have the order by AttractionName in the last For Each.



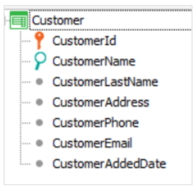
```
For Each Attraction (Line: 1)

Order:              CountryName , CityName , AttractionName
                    No index!
Navigation filters: Start from:     FirstRecord
                    Loop while:     NotEndOfTable
Join location:      Server

        =Attraction ( AttractionId )
            =Country ( CountryId )
            =CountryCity ( CountryId, CityId )

  Break Attraction (Line: 8)

  Order:              CountryName , CityName , AttractionName
                      No index!
  Navigation filters: Loop while:     CountryName = @CountryName
  Join location:      Server

          =Attraction ( AttractionId )
              =Country ( CountryId )
              =CountryCity ( CountryId, CityId )

    Break Attraction (Line: 16)

    Order:              CountryName , CityName , AttractionName
                        No index!
    Navigation filters: Loop while:     CountryName = @CountryName and CityName = @CityName
    Join location:      Server

            =Attraction ( AttractionId )
                =Country ( CountryId )
                =CountryCity ( CountryId, CityId )
```

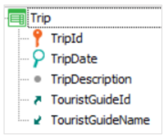
As we have seen, the break criteria are in Navigation Filters. Here we are giving these instructions: order the run by the set CountryName, CityName, AttractionName and execute the body of the first For Each, where only the CountryName is printed. Next, execute the second For Each, which, as long as the CountryName does not change, must print the printblock that contains only CityName, but then immediately execute the internal For Each, where as long as the countryName cityName cityName pair does not change, it must print the AttractionName.

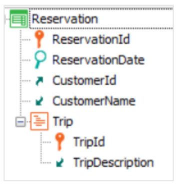
That is to say, we can see that the filters remain exactly the same as when we didn't order the innermost For Each by AttractionName, so, as we said, this order that we have just entered is only useful for sorting, and not for making a break.

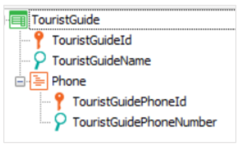
Case Study

Transaction Design

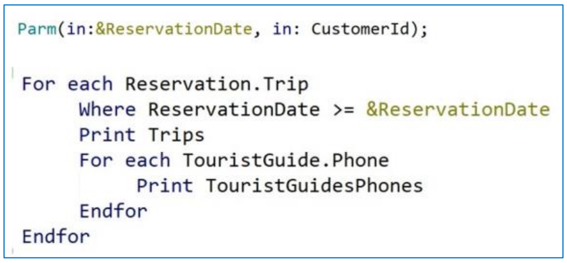

Customer
- CustomerId
- CustomerName
- CustomerLastName
- CustomerAddress
- CustomerPhone
- CustomerEmail
- CustomerAddedDate


Trip
- TripId
- TripDate
- TripDescription
- TouristGuideId
- TouristGuideName


Reservation
- ReservationId
- ReservationDate
- CustomerId
- CustomerName
- Trip
  - TripId
  - TripDescription


TouristGuide
- TouristGuideId
- TouristGuideName
- Phone
  - TouristGuidePhoneId
  - TouristGuidePhoneNumber

Source


```
Parm(in:&ReservationDate, in: CustomerId);

For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```

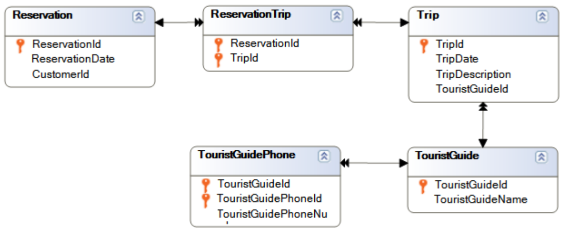
In a previous video we analyzed a case study to determine how GeneXus applies filters when they are received as an attribute in the Parm rule.

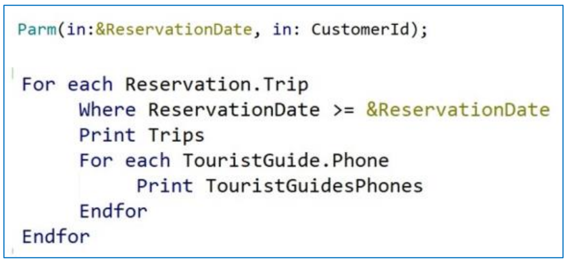
Going back to that same case, we will see how to prevent that filter from being applied.

We then start from the design of transactions design that is displayed, and we needed to obtain a list showing, for a given client, and from a given date, all the tours he has booked, and for each of them the contact phone numbers of the tour guide in charge.

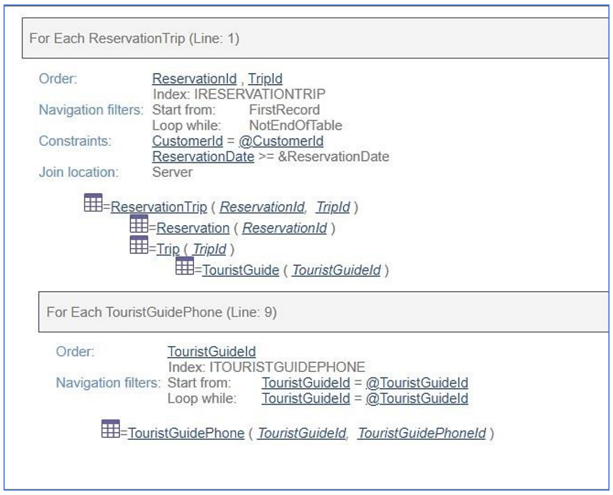
To achieve this, the source shown is proposed.

Case Study





```
Parm(in:&ReservationDate, in: CustomerId);

For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```



```
For Each ReservationTrip (Line: 1)

Order:              ReservationId , TripId
                    Index: IRESERVATIONTRIP
Navigation filters: Start from:    FirstRecord
                    Loop while:    NotEndOfTable
Constraints:        CustomerId = @CustomerId
                    ReservationDate >= &ReservationDate
Join location:      Server

=ReservationTrip ( ReservationId, TripId )
   =Reservation ( ReservationId )
   =Trip ( TripId )
      =TouristGuide ( TouristGuideId )

For Each TouristGuidePhone (Line: 9)

Order:              TouristGuideId
                    Index: ITOURISTGUIDEPHONE
Navigation filters: Start from:    TouristGuideId = @TouristGuideId
                    Loop while:    TouristGuideId = @TouristGuideId

=TouristGuidePhone ( TouristGuideId, TouristGuidePhoneId )
```

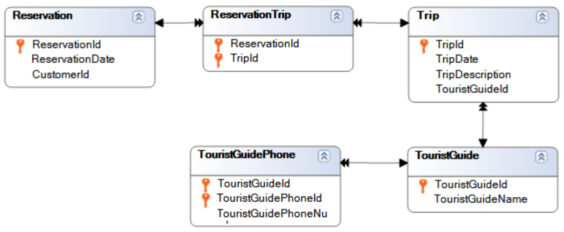
Knowing that the base table of the external For Each command is RESERVATIONTRIP, and that the base table of the internal For Each command is TOURISTGUIDEPHONES, does it establish implicit filters for the information it will use? Yes, it will show the phones of each tour guide.

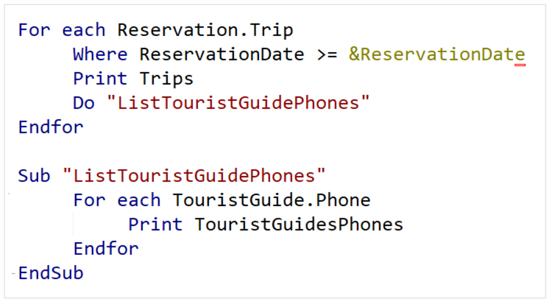
Why? GeneXus looks for a relationship between the extended table of the external For Each command and the base table of the nested For Each command:

It's another way of looking for a 1 to N relationship, although in this case it is an indirect one. If each RESERVATIONTRIP has a TouristGuideId, and in the table to be navigated there is also a TouristGuideId, then GeneXus understands that due to the relationship between the information, it will be the same.
That´s why it will make a Join.

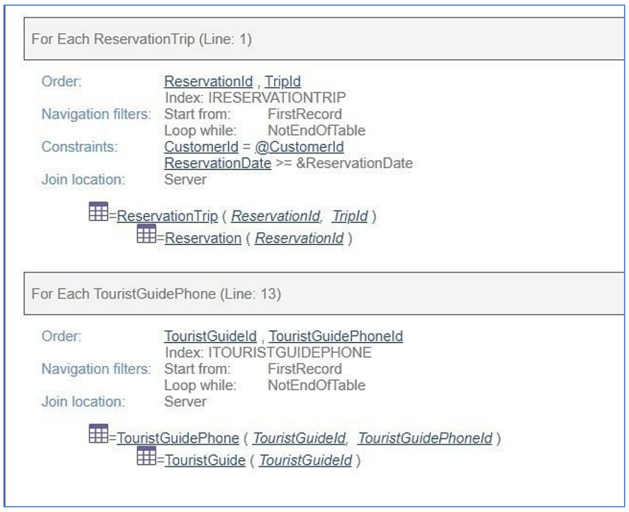
It can be clearly seen in the navigation list, where the @ symbol always indicates context data (note the @CustomerId, which refers to the value received in the CustomerId attribute of the Parm rule). This @TouristGuideId refers to the value of the attribute with that name in the TRIP table accessed by the first For Each command.

Case Study





```
For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     Do "ListTouristGuidePhones"
Endfor

Sub "ListTouristGuidePhones"
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
EndSub
```



```
For Each ReservationTrip (Line: 1)

Order:               ReservationId , TripId
                     Index: IRESERVATIONTRIP
Navigation filters:  Start from:    FirstRecord
                     Loop while:    NotEndOfTable
Constraints:         CustomerId = @CustomerId
                     ReservationDate >= &ReservationDate
Join location:       Server

     =ReservationTrip ( ReservationId,  TripId )
          =Reservation ( ReservationId )

For Each TouristGuidePhone (Line: 13)

Order:               TouristGuideId , TouristGuidePhoneId
                     Index: ITOURISTGUIDEPHONE
Navigation filters:  Start from:    FirstRecord
                     Loop while:    NotEndOfTable
Join location:       Server

     =TouristGuidePhone ( TouristGuideId,  TouristGuidePhoneId )
          =TouristGuide ( TouristGuideId )
```

If this is not what the programmer wants, is there a way to avoid these automatic filters?

Yes, using a subroutine to encapsulate the code of this second For Each command.
If you do this, the navigation list will show it.

Subroutines are blocks of code, which are defined in an object by means of the Sub command, which can be called later, within the same object and as many times as we want, by means of the Do command.

In the next videos we will explain in detail how to implement them.

training.genexus.com
wiki.genexus.com