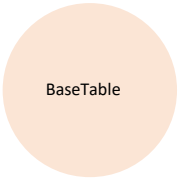
Logic of query to database with GeneXus

Determining base tables

GeneXus™

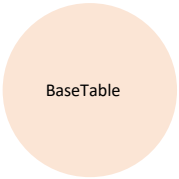



```
For each BaseTrn1, ... , BaseTrnn
    skip expression1 count expression2
    order att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where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

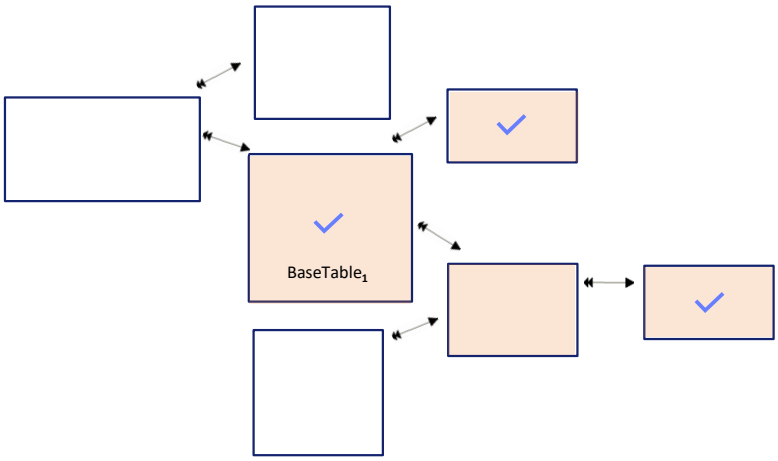
extended table

We shall focus on the For Each command. When this command has a defined base transaction, then the problem is already solved. The question is: what happens when it doesn't?

How should the base table be found? GeneXus searches for the attributes existing in all these places, and based on them it searches for an extended table that contains them all.

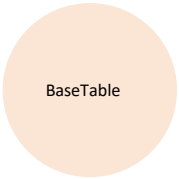



```
For each BaseTrn1, ... , BaseTrnn
    skip expression1 count expression2
    order att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where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

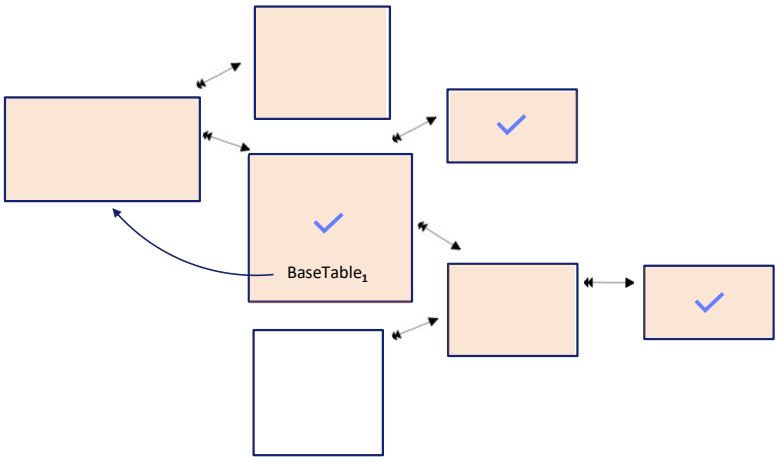


For example, if the attributes that appear in these places are from these tables, which will be the base table of the For each?

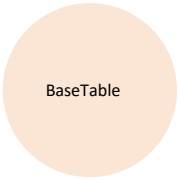
It will be this one, because its extended table contains them all …



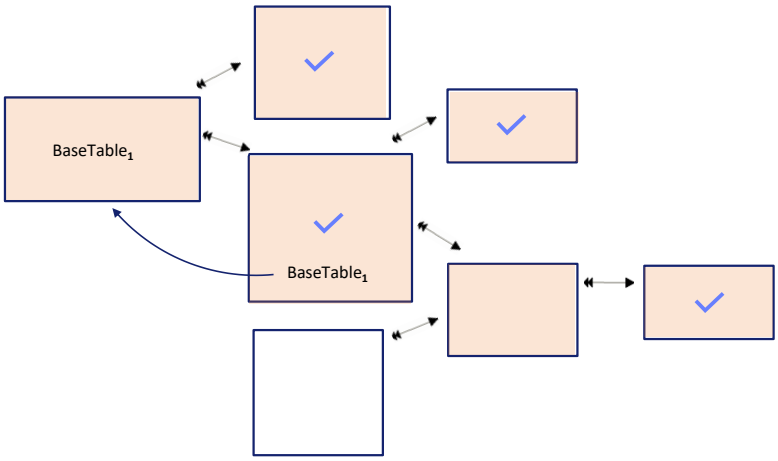

```
For each BaseTrn1, ... , BaseTrnn
    skip expression1 count expression2
    order att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where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



It could be possible to argument that, if the choice was this other one as base table, its extended table would also contain all the attributes, so, it becomes necessary to add a condition: it should be the minimum table of all the extended tables that contain **all** the attributes.
That solves the problem, and the table will be this one.

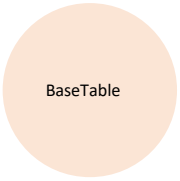



```
For each BaseTrn1, ... , BaseTrnn
    skip expression1 count expression2
    order att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where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



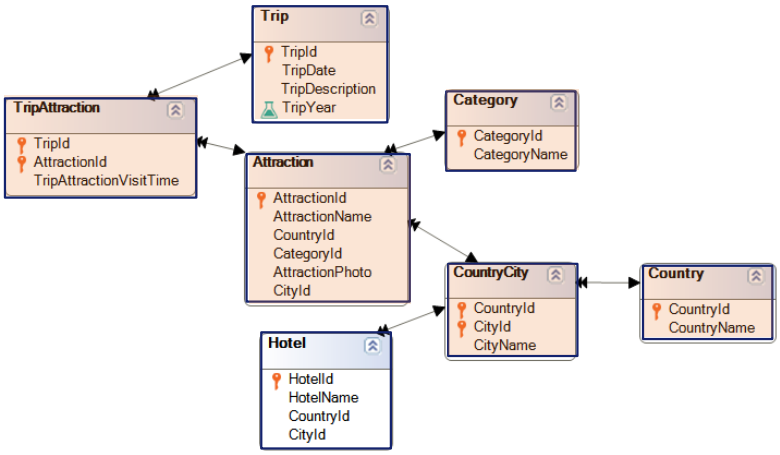
Obviously, in the case that an attribute of this table appeared, then the base table would be a different one.

Consider it in this example…




```
for each
order TripDate
where CategoryName = "Monument"
where CountryName = "France"
   print PB1 //AttractionName
endfor
```

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



There is no base transaction. This attribute is here, this other one is here, and this other one here; and this other one appears in the main code. The navigation list will show…

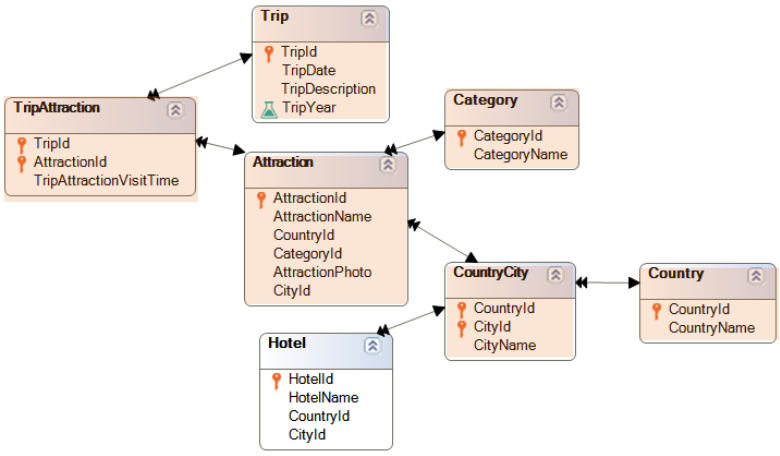
BaseTable

```
for each
order TripDate
where CategoryName = "Monument"
where CountryName = "France"
   print PB1 //AttractionName
endfor
```

For Each TripAttraction (Line: 43)

Order: TripDate
No index!
Navigation filters: Start from: FirstRecord
Loop while: NotEndOfTable
Constraints: CategoryName = "Monument"
CountryName = "France"
Join location: Server

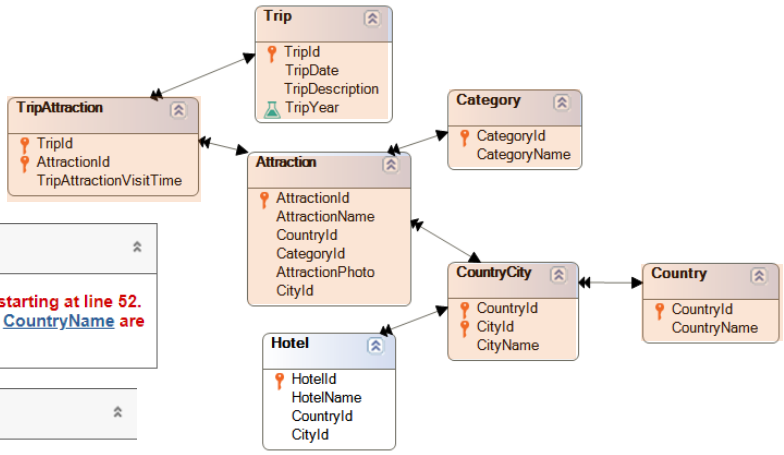
=TripAttraction (*TripId*, *AttractionId*) INTO TripId AttractionId
=Trip (*TripId*) INTO TripDate
=Attraction (*AttractionId*) INTO CountryId CategoryId AttractionName
=Country (*CountryId*) INTO CountryName
=Category (*CategoryId*) INTO CategoryName

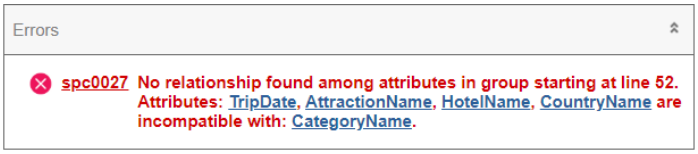


You can see that it is in fact selecting TripAttraction as base table.

BaseTable

```
for each Trip.Attraction
order TripDate
where CategoryName = "Monument"
where CountryName = "France"
   print PB1 //AttractionName, HotelName
endfor
```





But, what happens if there is no extended table containing them all?

For example, if you add HotelName in the printblock, GeneXus will show an error and it will not be possible to generate the object.

Note that if, for example, TripAttraction had been specified as base table, then instead of an error there will be a warning about the HotelName attribute being unreachable, but the base table will be perfectly determined by the base transaction.

BaseTable

```
for each
order TripDate
where CategoryName = "Monument"
where CountryName = "France"
   print PB1 //AttractionName
endfor
```

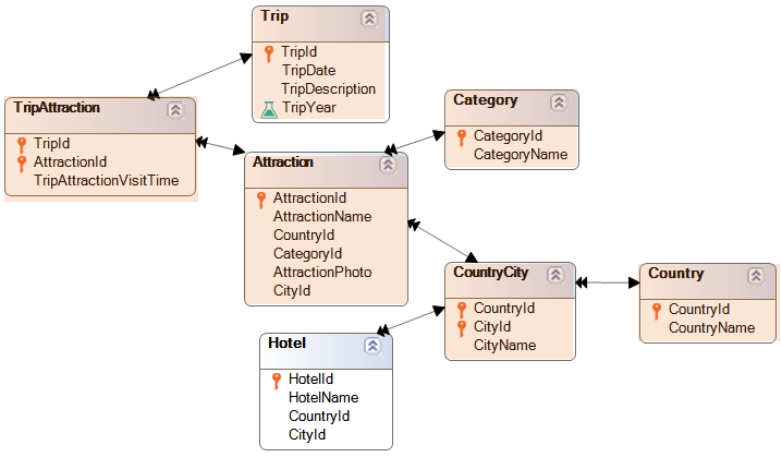
For each

Grid with base table
(with base table)

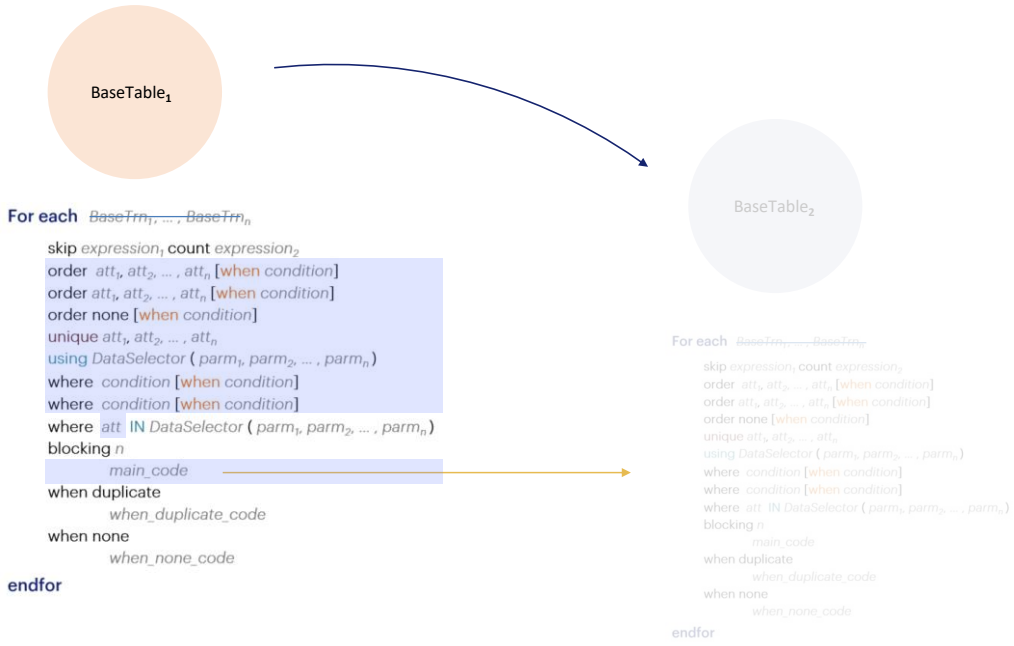
Data Provider Group
(with base table)

Data Selector

Formula



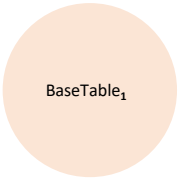
All this related to a For each also applies to a Grid (with base table), as well as to a group of Data Providers. And the same goes for a Data Selector executed as an independent query, or a formula.



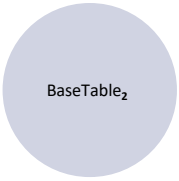
Now, what happens in the case of a nested For each?

The base table of the main For each is determined without taking the nested For each into account at all. That is to say, as if it didn't exist in the main code.

How is the base table of the nested For each determined? It is always defined **after** determining the one of the For each that contains it.
The case that should be analyzed is when there is no base transaction.



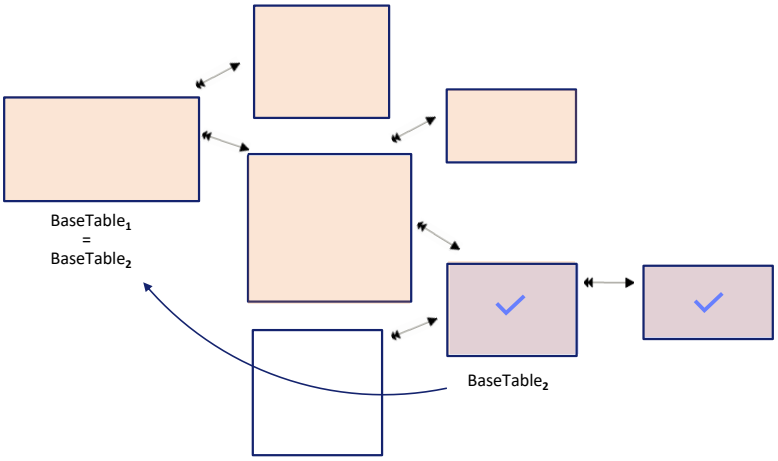

```
For each BaseTrn1, ... , BaseTrnn
    skip expression1 count expression2
    order att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    where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




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

It could be considered in a similar manner, taking the attributes found here…

```
for each
   …
   for each
      …
   endfor
   …
endfor
```

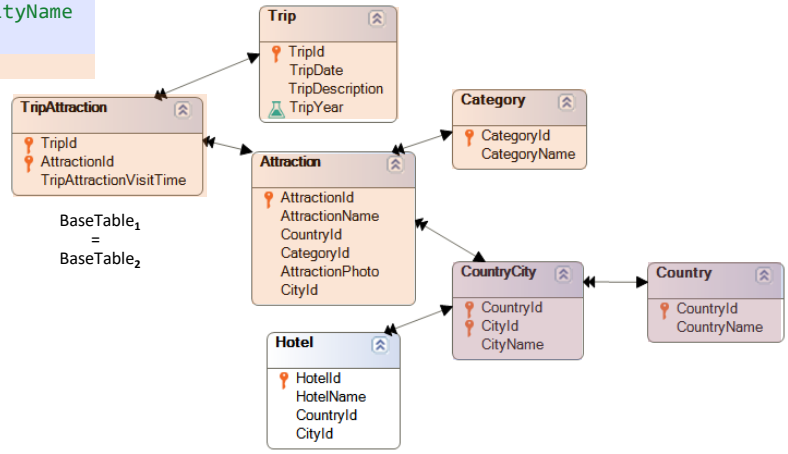


(suppose, these ones)… and finding the minimum extended table that contains them, regardless of the main For each.

However, it will not be exactly that way. In fact, from all the attributes extracted from the places already known, the first thing to do is remove all those that are also part of the extended table of the main For each, to operate only with the ones remaining, in order to find the minimum extended table that contains them.

In this case, when you remove, from the set of attributes to be calculated, those that belong to the main's extended table, then you will have none left! Empty set. In such case, the base table of the nested For each will be the same as the one of the main For each, and it will then be a control break.

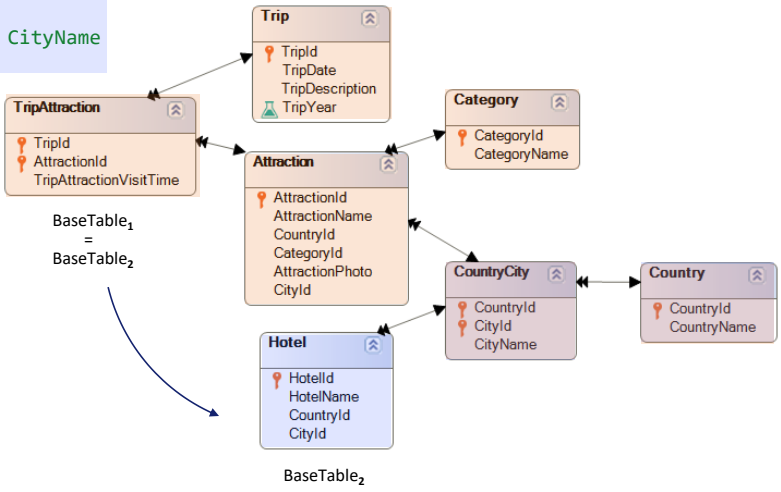
```
for each
   order CategoryName
   where TripDate > &today
   where AttractionName > 'N'
      print PB1 //CategoryName
      for each
         print PB2 //CountryName, CityName
      endfor
endfor
```



Here is a particular example. Whether Trip.Attraction is specified as base transaction for the main For each, or not, due to the attributes involved, the base table will be TripAttraction in any case. And for the nested For each, since the attributes used in it are from this table and from this other table, its base table would be this other one… but it so happens that it is located in the main's extended table, so GeneXus will select the same one, TripAttraction, as its base table.

And a control break will take place by CategoryName.

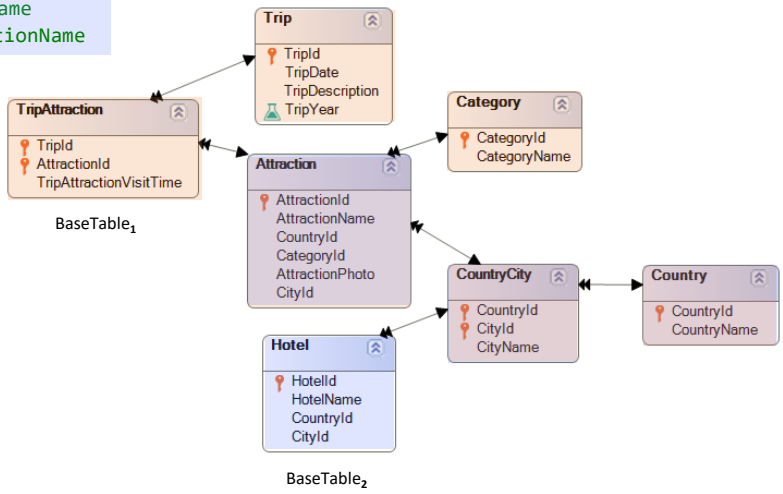
```
for each
   order CategoryName
   where TripDate > &today
   where AttractionName > 'N'
      print PB1 //CategoryName
      for each
         order HotelName
            print PB2 //CountryName, CityName
      endfor
endfor
```



If, for instance, you added an order by HotelName, then things would be different, and the base table of the nested For each will be Hotel.

Note that here, from these three attributes of the For each, you will have to remove this one and this one for the calculation of the base table, because they are already in the extended table of TripAttraction. Only HotelName remains for calculating the minimum extended table that contains it.

```
for each
   order CategoryName
   where TripDate > &today
   where AttractionName > 'N'
      print PB1 //CategoryName
      for each
         order HotelName
            print PB2 //CountryName, CityName
      endfor          // HotelName, AttractionName
endfor
```



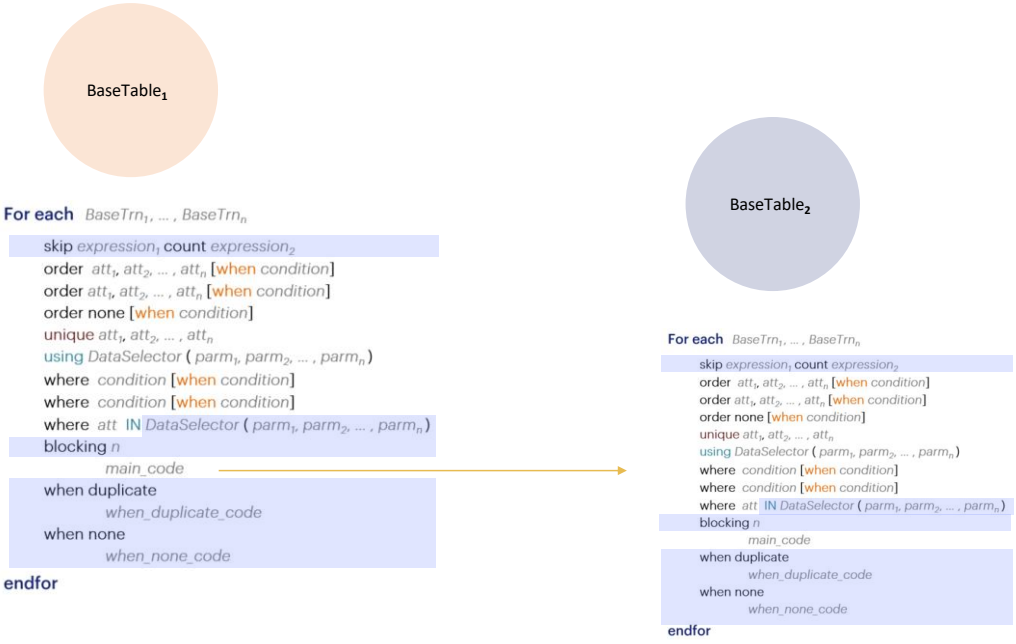
$BaseTable_1$

$BaseTable_2$

This is so because, in the nested For each you could possibly use attributes from the main's extended table, to do something with the **perfectly determined** values of those attributes.

For example, imagine that, in the print block of the nested For each, you add HotelName, which makes no difference, and also AttractionName. Neither CountryName, nor CityName or AttractionName will participate in determining the base table of the nested For each because they are in the main's extended table. It is clear that the base table will be Hotel.
What will be executed? For each TripAttraction going through the filters there will be **one** value of AttractionName, and **that** value will be the one shown for each record of the nested For each.

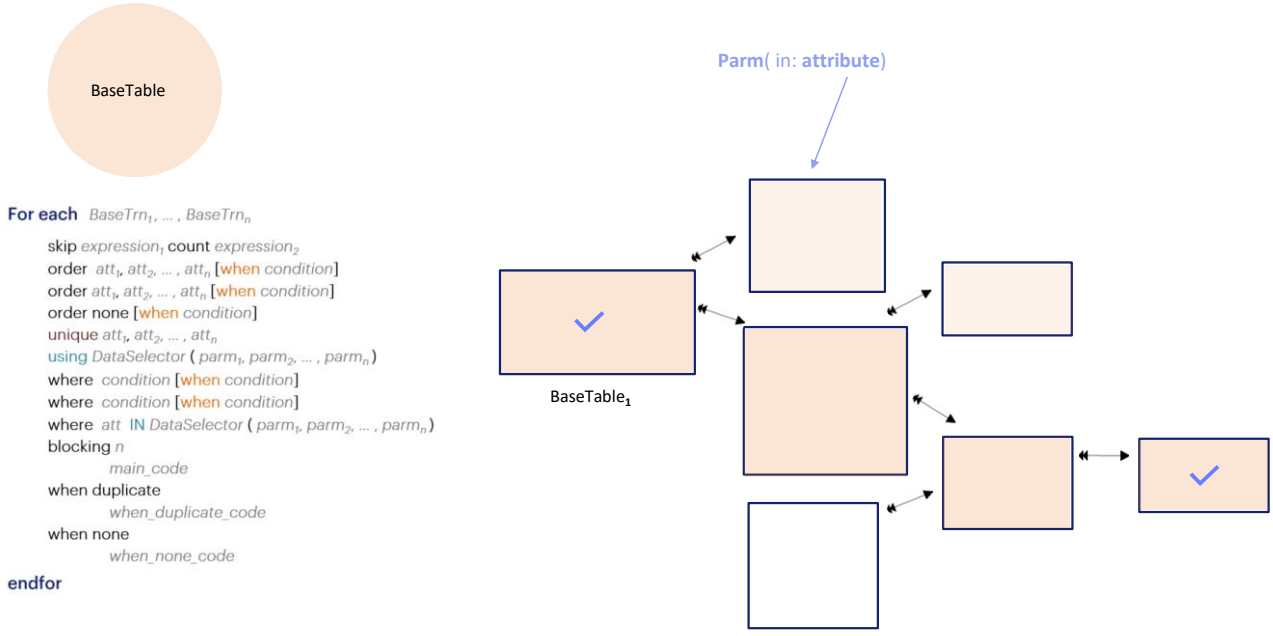
Likewise, for all hotels in the city of the trip attraction, that country and city will come up along with the name of the hotel and the name of **that** attraction; it will always be the same one for all these records.



It is important to bear in mind that the attributes in these places will not take part in determining the base table of the respective query.

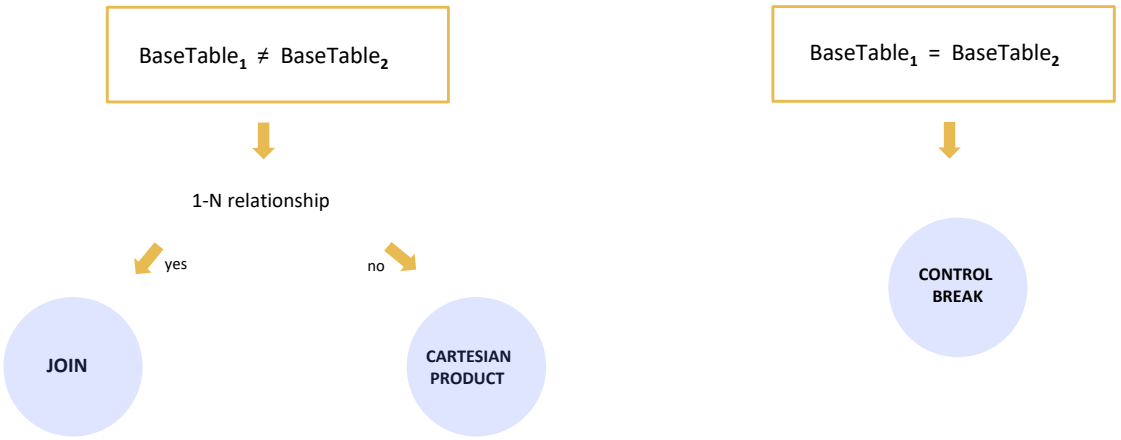
Of course, it is possible to specify base transaction for one and not for the other.

After, and only **after**, the base tables have been determined, GeneXus will solve the navigation, and more specifically the conditions that it will implicitly impose. For example…

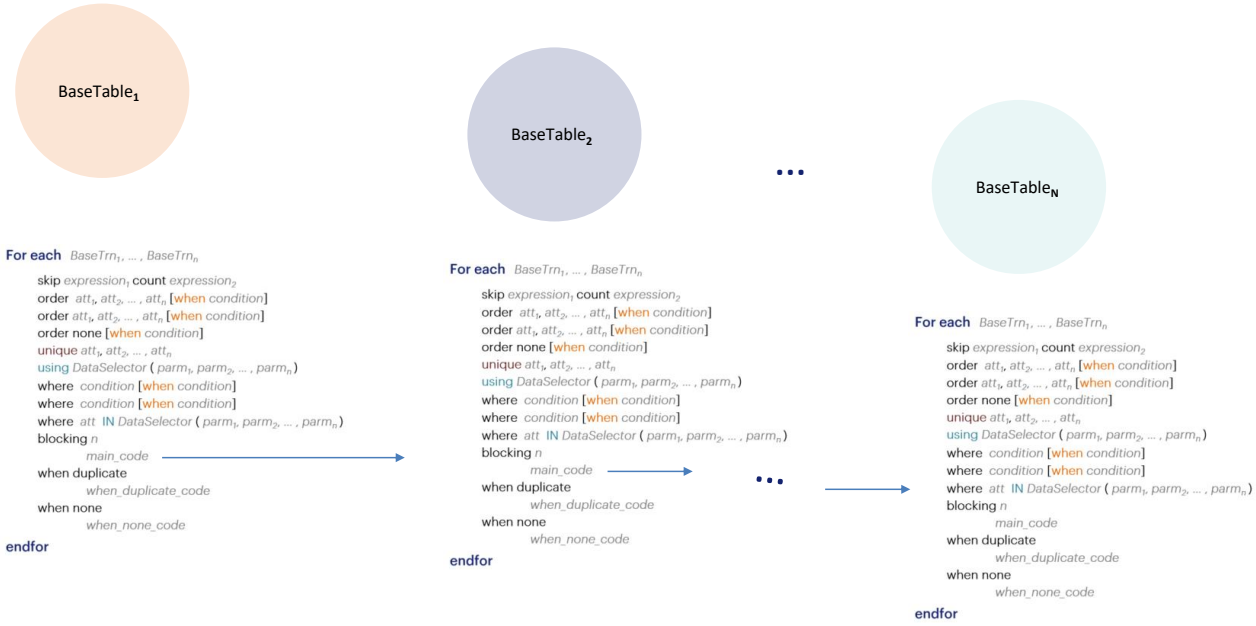


Remember that, if no access to a particular table of the extended table is required in the For each –for example, suppose that only attributes of these tables are used-, then the access will be to these tables of the extended table and not to all tables. These two will not be accessed.

Therefore, if, in the parm rule you receive –in an attribute of one of them– that automatic filter, based on equality, will not be applied to the For each. And that is because the implied conditions are placed AFTER the navigation has been defined.



The other clear example: the specific navigation is defined only after the base tables have been determined. This means that, first, you must know the base tables in order to determine the case. For example, finding if there is a (direct or indirect) 1 to N relation or not.



And what about determining the base table of a nested For each at N level of nesting?

It is logical that it may use attributes of the extended tables from its ancestors, and not only from the parent. In addition to parent-child inferences, there are also grandparent-child inferences. In fact, inferences take place with any ancestor.

And control breaks are only at the parent level. What might happen is that the parent level could also be control break with its own parent.

And this will be the end of this topic.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications