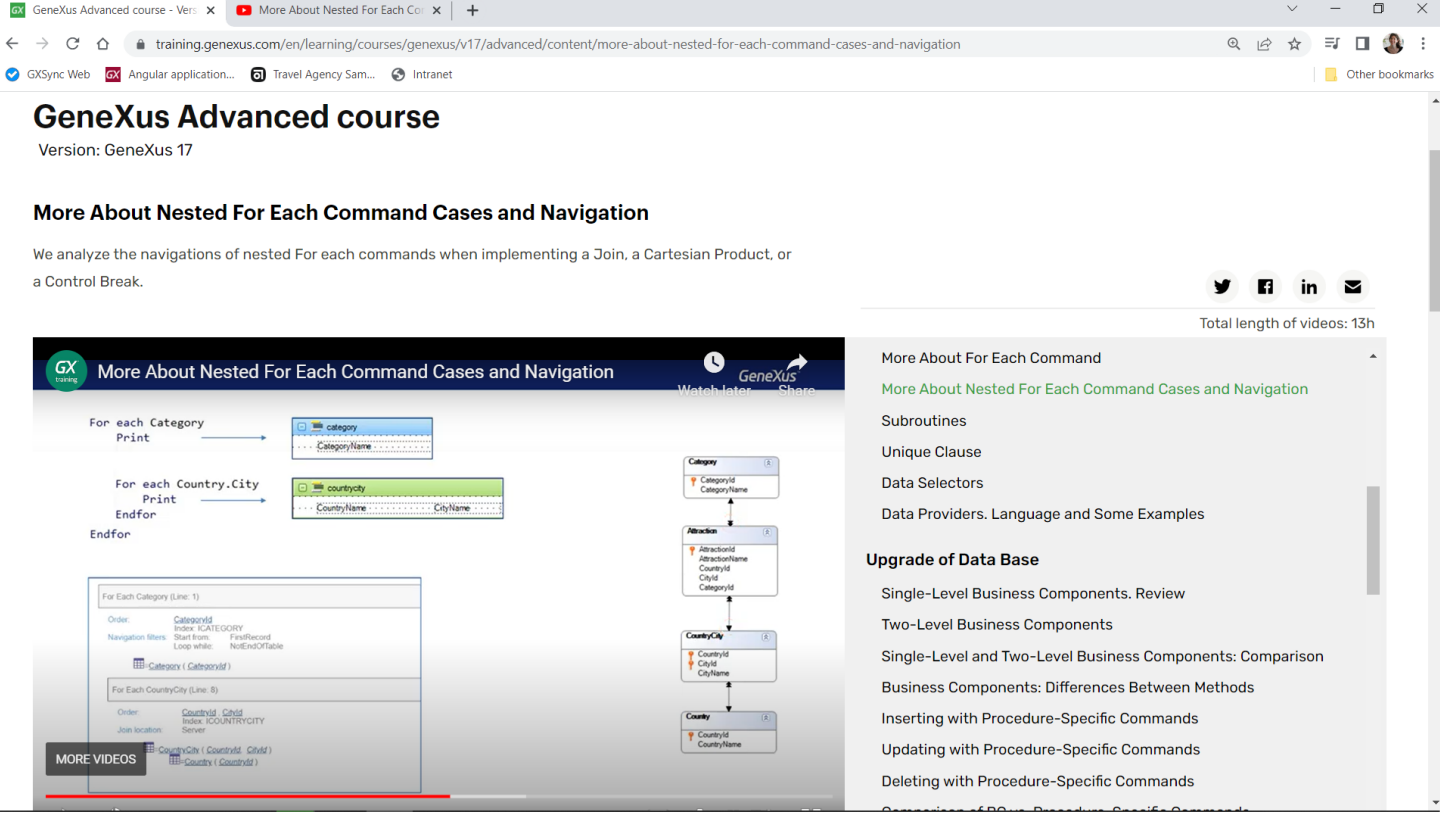
Lógica de consulta à base de dados com GeneXus

Casos de For eachs aninhados. Formalização

GeneXus™



Neste vídeo... analisamos os três tipos de navegações que GeneXus implementava ao encontrar for eachs aninhados.

Voltaremos a eles para formalizá-los um pouco mais.

BaseTable$_1$

```
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
```

BaseTable$_2$

```
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
```

JOIN

BaseTable$_1$ ≠ BaseTable$_2$

1-N relashionship

CARTESIAN PRODUCT

BaseTable$_1$ ≠ BaseTable$_2$

NO 1-N relashionship

CONTROL BREAK

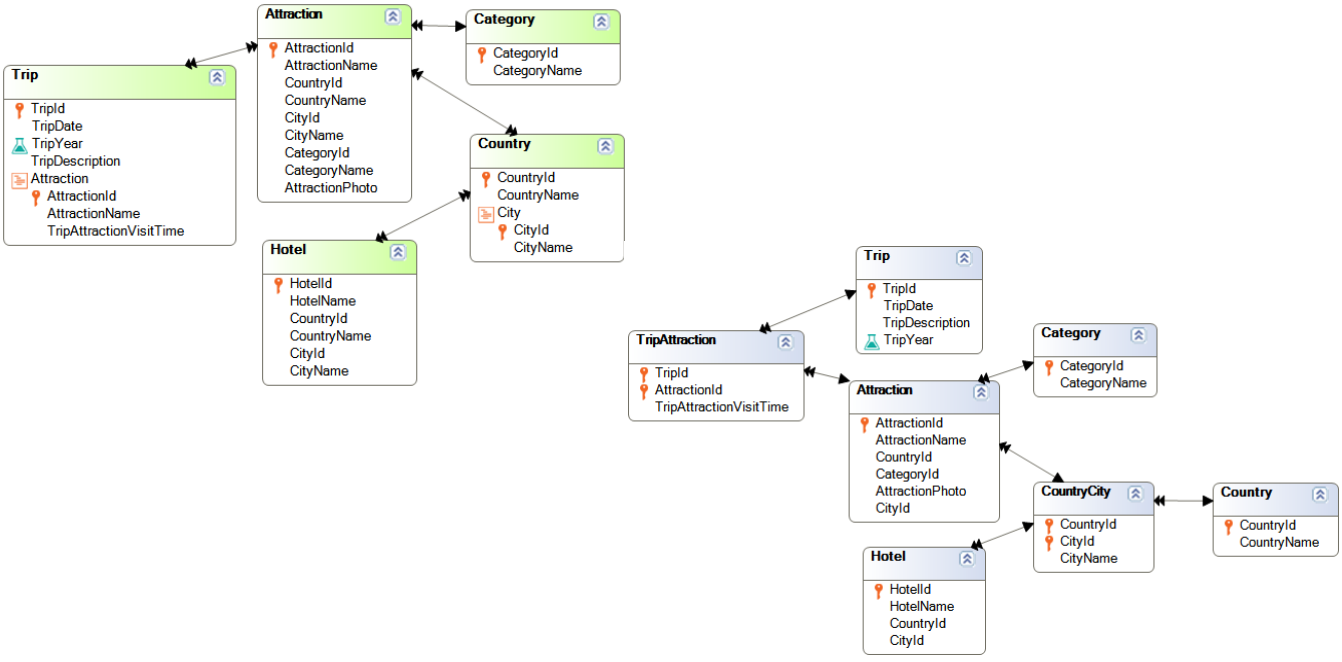
BaseTable$_1$ = BaseTable$_2$

Quando dentro do corpo de um for each aparece outro, quer dizer que para cada registro de uma navegação sobre uma tabela deseja-se percorrer muitos registros em outra navegação da mesma ou de outra tabela. Um caso particular é quando é recuperado apenas um registro.

Primeiro, determinam-se as tabelas base e, em seguida, investiga-se sua relação, o que dará origem a um dos três casos: Join, Produto Cartesiano e Corte de Controle.

Sabemos de uma primeira grande diferença: nos dois primeiros casos, se tratará de tabelas base diferentes, enquanto o último é o caso da mesma tabela base.

O que diferencia o Join do Produto Cartesiano? A identificação de uma relação 1 a N direta ou indireta. Vamos formalizar todos os casos.

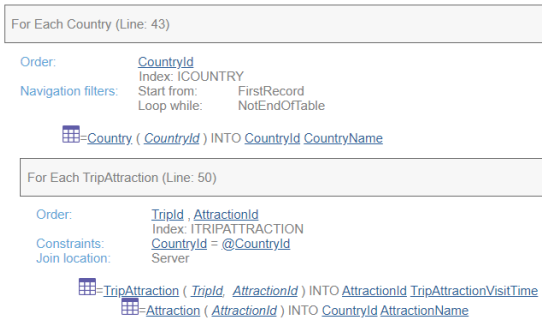


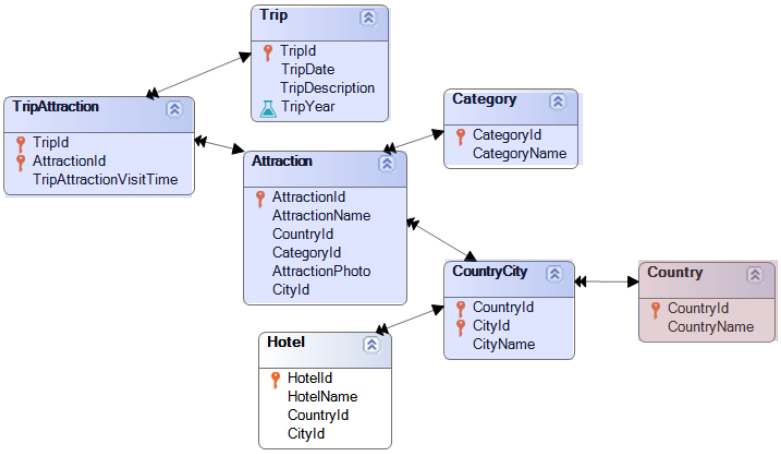
Suponhamos que temos estas 5 transações para registrar as atrações turísticas que podem ser visitadas em excursões, onde cada atração corresponde a uma categoria (como Museu ou Monumento), e é de uma cidade de um país, e por outro lado existem hotéis de cada país e cidade.

A partir destas transações, obtemos estas tabelas com suas relações.

1

```
for each Country
   print PB1 //CountryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttractionVisitTime
   endfor
endfor
```





$BaseTable_1 \subset ext(BaseTable_2)$

Vamos começar pelo Join. Como será resolvido este código?

Solicita-se imprimir para cada nome de país os nomes de atração com seu tempo de visita estipulado na trip. Existe relação entre as informações?

Sabemos que um registro de TripAttraction corresponde a uma única atração, que corresponde a uma única cidade, que corresponde a um único país. Dito de outra forma, a partir da tabela TripAttraction chegamos a Country de forma única. Por isso podemos imprimir para o segundo for each somente as tripattractions que correspondem ao país do for each principal, como vemos na lista de navegação como constraint. Onde este arroba CountryId corresponde ao CountryId do registro do for each externo em que estamos posicionados. E este atributo CountryId é o de Attraction, ao qual chegamos por TripAttraction. Está claro que se trata de um Join, com uma relação 1 a N indireta.

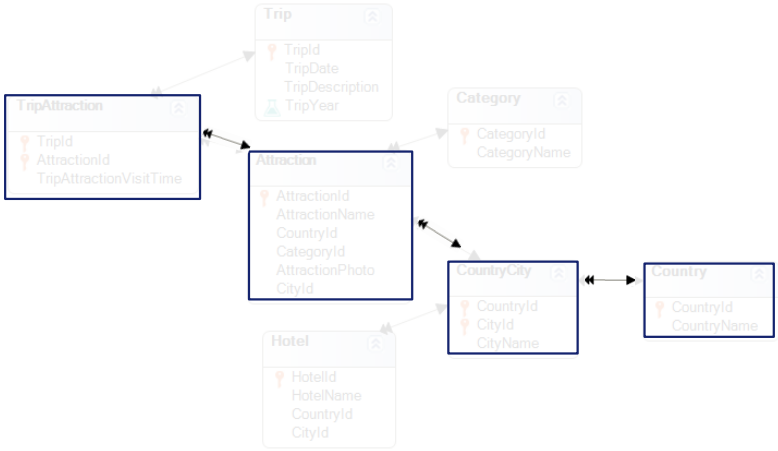
É porque para uma tripattraction seguindo as chaves estrangeiras encontramos um único CountryId, e, portanto, para um CountryId podemos encontrar N tripattractions que, fazendo este percurso, cheguem até ele.

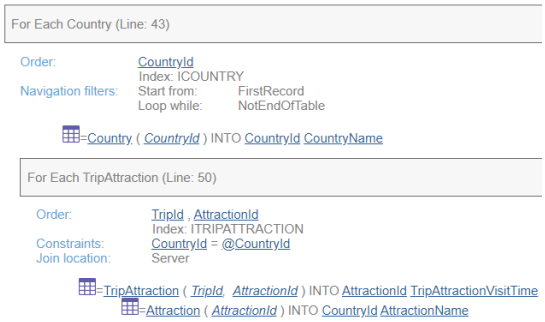
Se formalizarmos este caso de relação 1 a N indireta, estamos dizendo que a tabela base do for each principal está contida na tabela estendida daquela do for each aninhado.

Esse caso inclui o mais simples de todos. Por exemplo, vamos pensar que se a tabela base do segundo for each fosse CountryCity, esta fórmula é cumprida.

1

```
for each Country
   print PB1 //CountryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttractionVisitTime
   endfor
endfor
```





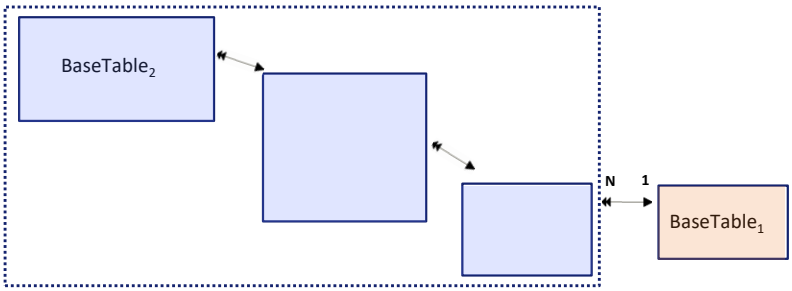
JOIN

$BaseTable_1 \subset ext(BaseTable_2)$

Aquí vemos mejor las relaciones entre las tablas envolvidas neste caso.

1

```
for each Country
   print PB1 //CountryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttractionVisitTime
   endfor
endfor
```



JOIN

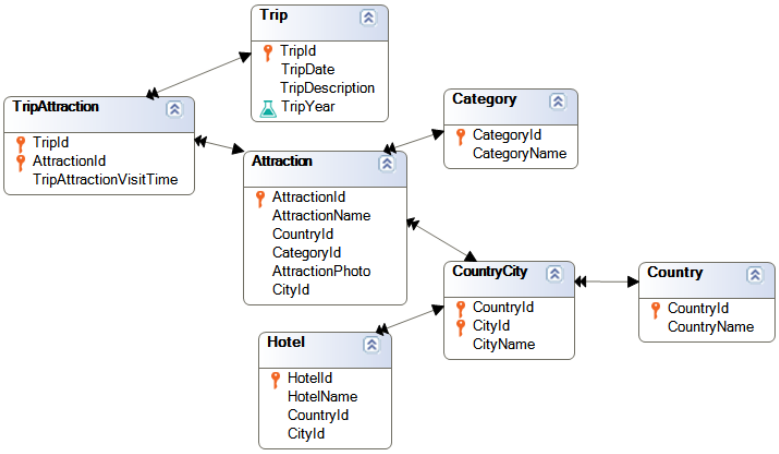
$BaseTable_1 \subset ext(BaseTable_2)$

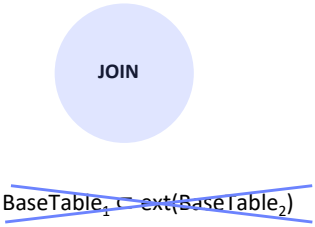
Ou seja, a tabela base do principal é esta, e a tabela base do aninhado esta outra.

Se vê clara a relação 1 a N indireta. É indireta através da tabela estendida do for each aninhado.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```



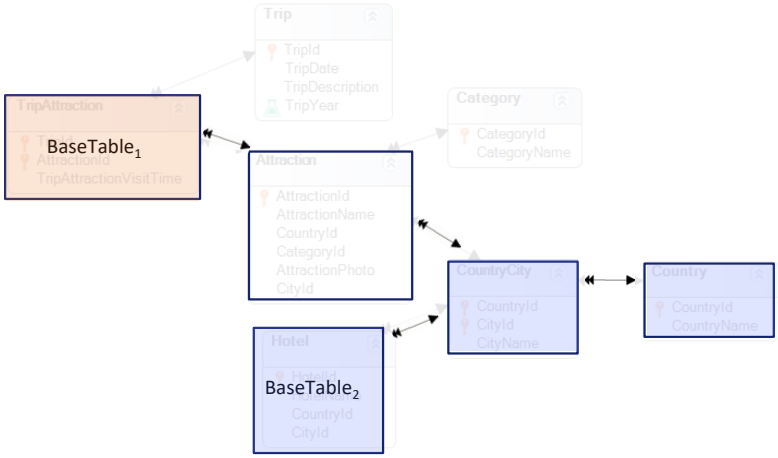


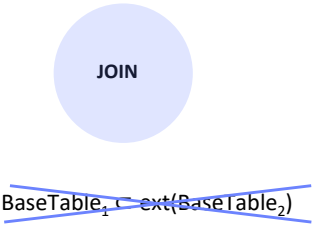
Agora vejamos o outro tipo de relação 1 a N indireta, desta vez não por meio da estendida do aninhado, mas do principal.

Para cada tripattraction, imprimimos nome da atração e tempo de visita nessa trip e, em seguida, os nomes de hotéis.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```

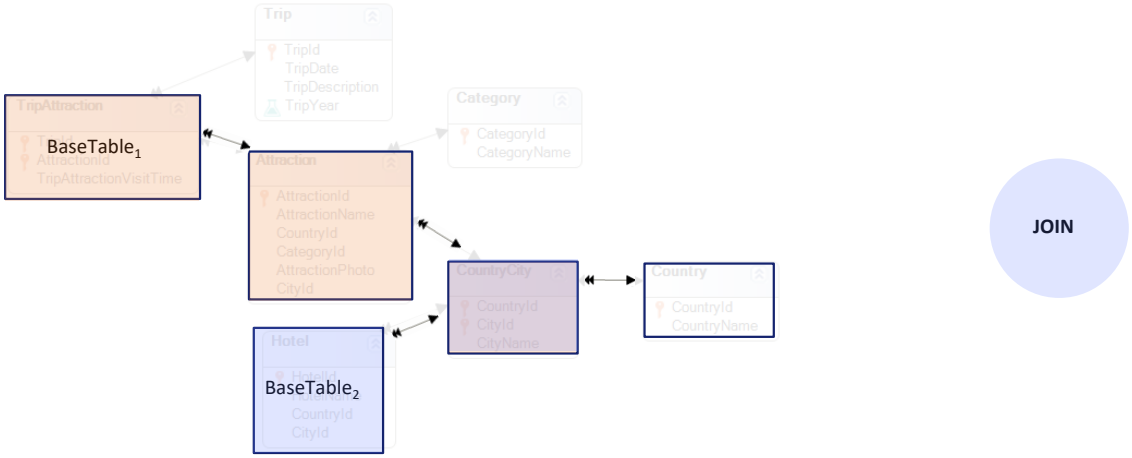




Aqui claramente não é verdade que a tabela base do for each principal esteja incluída na tabela estendida do for each aninhado. Não há relação 1 a N indireta através desta via. Porém…

2

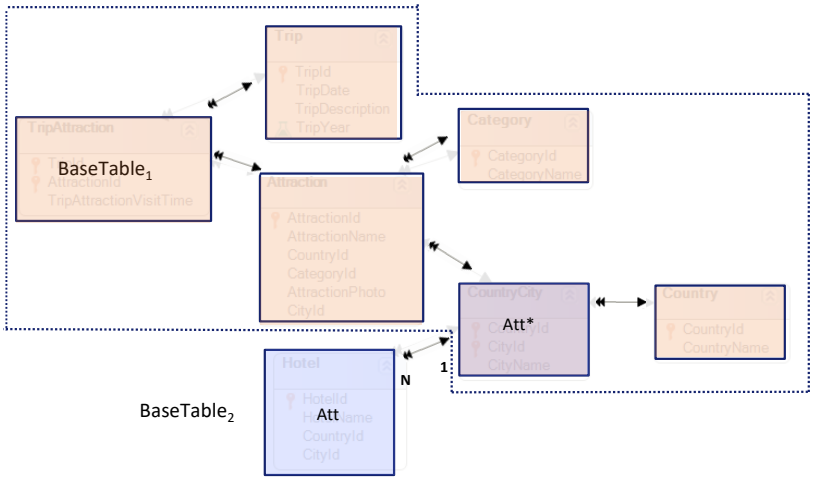
```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```



…sabemos que para cada registro do for each principal chegamos a un registro desta tabela, ao qual se chega também diretamente a partir da tabela base do aninhado.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```



JOIN

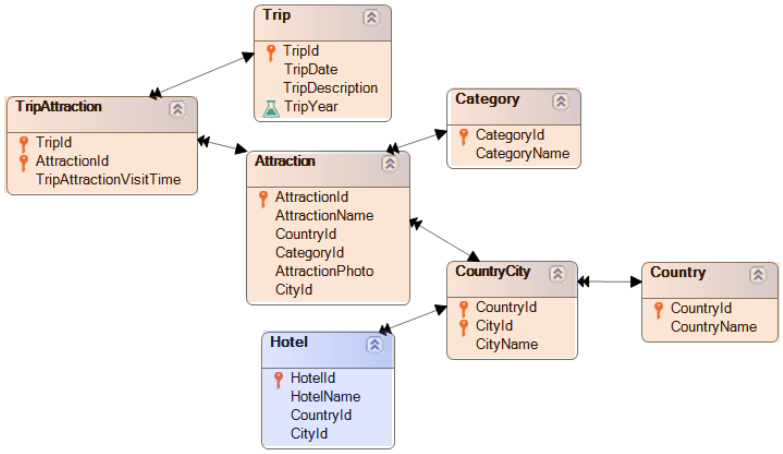
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

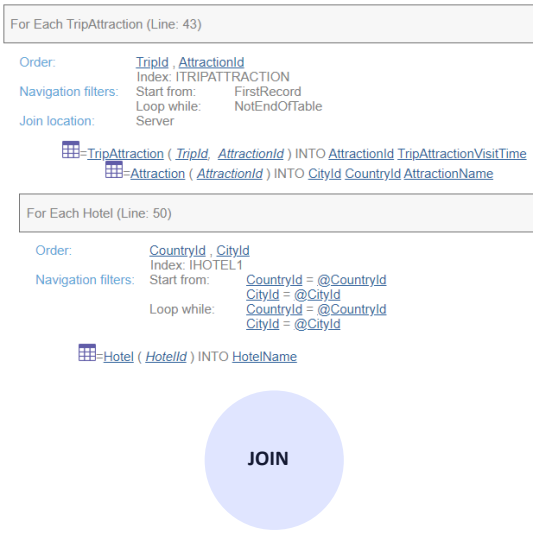
Ou seja, nesta haverá uma chave estrangeira para esta outra. Resumindo, vão compartilhar esse atributo (devido a ele, é estabelecida esta relação 1 a N entre as tabelas).

Em resumo, a tabela estendida do for each principal terá algum atributo em comum com a tabela base do aninhado, e estes estabelecerão o join.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```





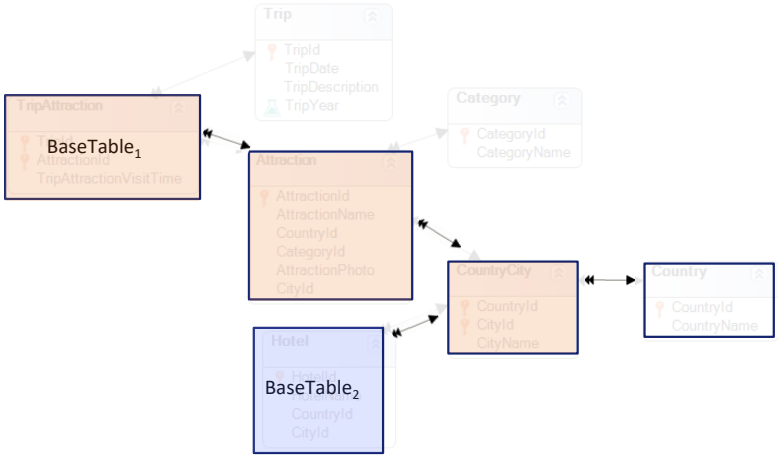
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

Aqui serão CountryId, CityId, chave estrangeira para CountryCity.

Vemos isso claramente na lista de navegação. Serão impressos apenas os hotéis que correspondem à mesma cidade da atração da trip.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```

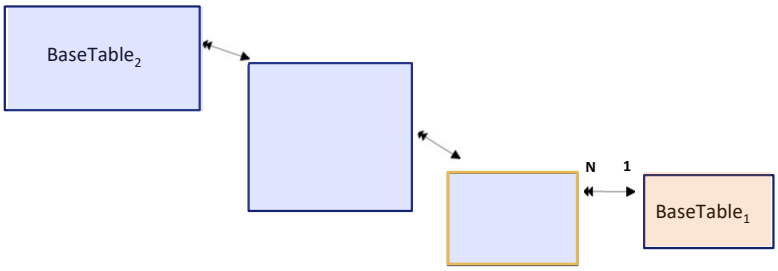


JOIN

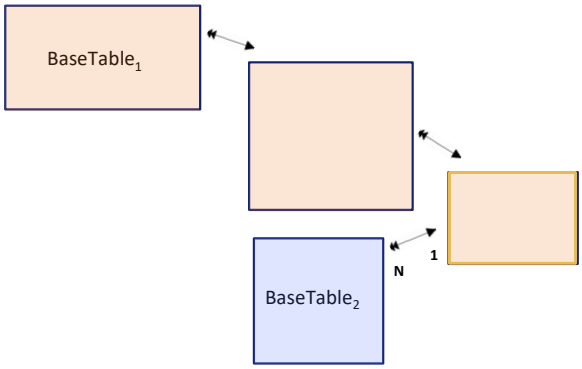
ext($BaseTable_1$) ∩ $BaseTable_2$ ≠ ϕ

Novamente, para ver mais claramente, vamos ficar apenas com as tabelas envolvidas: aqui vemos a relação 1 a N indireta.

1 $BaseTable_1 \subset ext(BaseTable_2)$



2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$



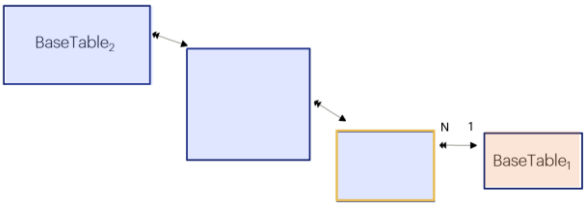
Se no primeiro caso percorria-se esta tabela base e para o aninhado esta outra e, portanto, a relação era 1 a N indireta através da estendida da do aninhado, no segundo caso é através da estendida da tabela base do principal. E é aqui que se estabelece a 1 a N com a tabela base do aninhado.

JOIN

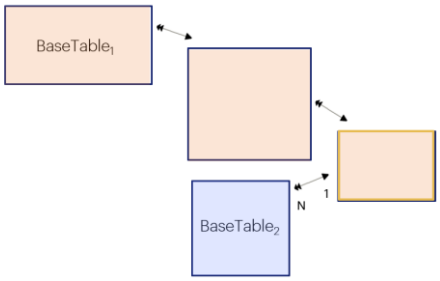
1-N relashionship

$BaseTable_1 \neq BaseTable_2$

1 $BaseTable_1 \subset ext(BaseTable_2)$



2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$



Portanto, o caso de Join é o de tabelas base diferentes onde se encontra uma relação 1 a N direta ou indireta, de acordo com estas opções: tabela base com estendida, ou tabela estendida com tabela base.

Não assim tabela estendida com tabela estendida.

JOIN

1-N relashionship

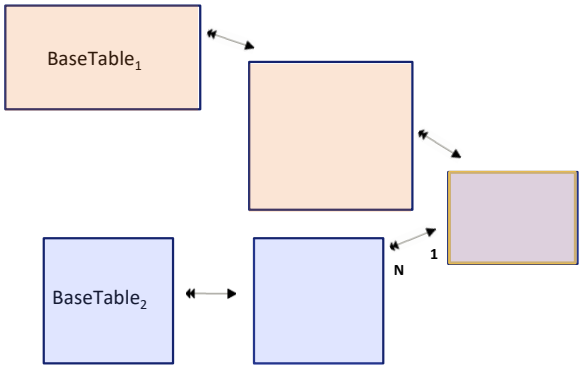
1 $BaseTable_1 \subset ext(BaseTable_2)$

2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

NO 1-N relashionship



Ou seja, se em vez de ser esta a tabela base do aninhado for esta outra, há relação entre as tabelas estendidas, claramente, pois ambas chegam à mesma tabela. No entanto, aqui não será produzido um join, mas um produto cartesiano.

Por que, se na verdade para cada registro da tabela base do for each principal poderíamos ficar apenas com aqueles da tabela base do aninhado que correspondem ao mesmo registro desta tabela a que ambos chegam de forma única?

É que quanto mais indireta a relação, menos provável parece que o desenvolvedor esteja procurando levá-la em consideração, porque a relação cada vez parece mais distante, e se o desenvolvedor estivesse procurando por ela, sempre pode torná-la explícita.

JOIN

1-N relashionship

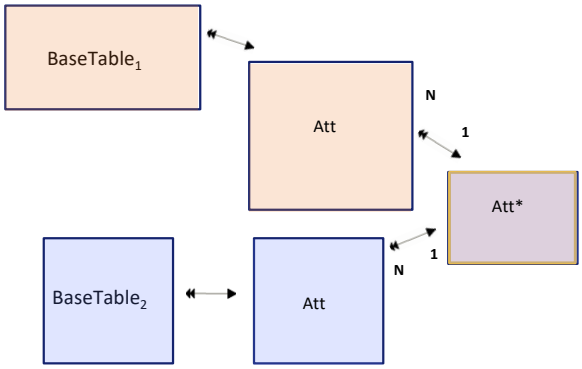
1 $BaseTable_1 \subset ext(BaseTable_2)$

2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

```
for each
   ...
   &var = Att
   for each
      where Att = &var
         ...
   endfor
endfor
```



Por exemplo, atribuindo a uma variável o valor do atributo que se obtém fazendo o caminho da tabela estendida do for each principal... ou seja, o valor deste atributo que coincide com este.
E no for each aninhado, filtrando **explicitamente** os registros a partir dos quais se chega a este outro atributo, que tem o mesmo nome porque também é foreign key para a tabela comum.

Aproveitemos para fazer este esclarecimento: quando falamos de Join ou Produto Cartesiano, dessa diferença, referimo-nos às determinações implícitas de GeneXus, não a se finalmente acaba filtrando ou não a informação do aninhado. Observemos que neste caso se trata de um produto cartesiano, pois se não tivéssemos escrito nenhum where, GeneXus também não o adicionará implicitamente e serão retornados todos os registros do aninhado. Mas, na verdade, neste caso não serão retornados todos os registros do aninhado porque explicitamos um where, portanto realmente fará um join, mas não o Join implícito de GeneXus.

JOIN

1-N relashionship

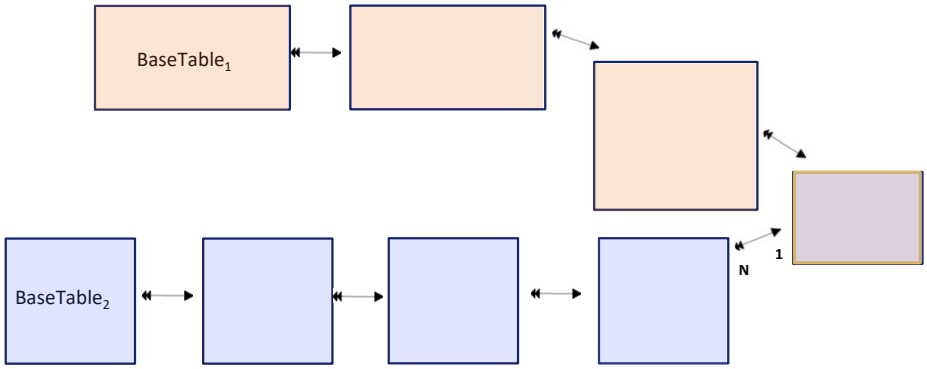
1 $BaseTable_1 \subset ext(BaseTable_2)$

2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

$BaseTable_1 \neq BaseTable_2$

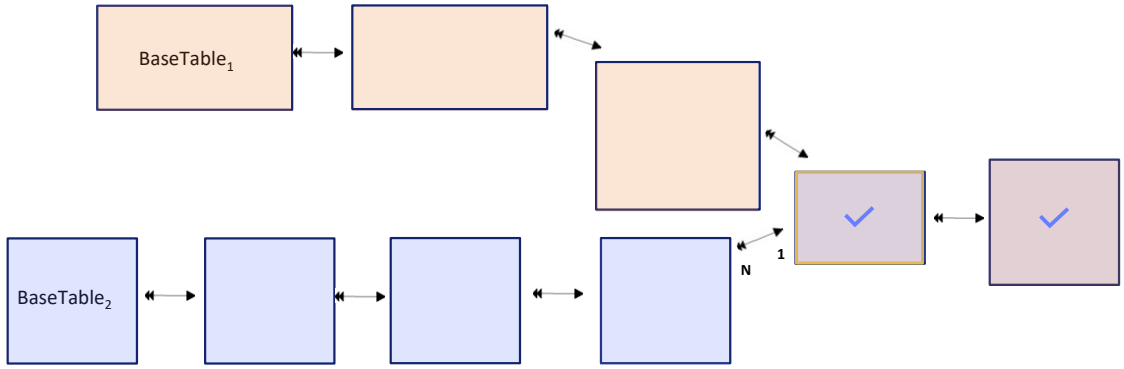
CARTESIAN PRODUCT

NO 1-N relashionship



De quanto mais distante venha a relação, menos probabilidade haverá de que o desenvolvedor a tenha em mente, a esteja querendo fazer valer implicitamente.

CARTESIAN PRODUCT




```
for each
   ...
  for each
      ...
   endfor
   ...
endfor
```

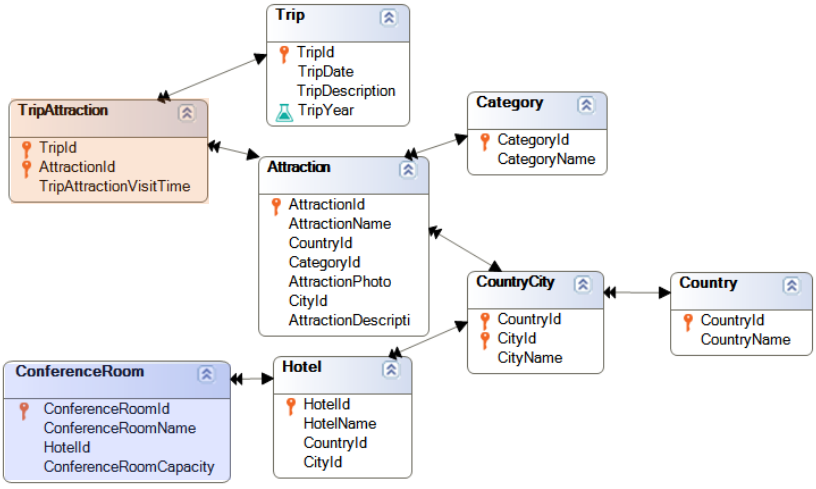
Uma questão interessante deste caso é a seguinte. Adicionamos mais uma tabela para deixar ainda mais claro.

Se fizermos a interseção de ambas as tabelas estendidas, ficamos com as tabelas em comum. Agora, se no for each principal colocamos atributos destas duas tabelas, claramente os valores que serão tomados serão aqueles obtidos partindo de cada registro da tabela base que satisfaça os filtros. Ou seja, serão aqueles que provenham desta tabela estendida.

Mas, o que acontece se esses atributos se encontram no for each aninhado? São tomados aqueles da tabela estendida deste for each?
Se por um caminho ou pelo outro chegássemos ao mesmo registro desta tabela, não importaria, pois dariam o mesmo valor por um caminho ou pelo outro. Isso é o que aconteceria se fosse feito o join. Mas não há join neste caso. Portanto, os valores destes atributos que se obtenham por este caminho não serão sempre iguais aos obtidos por este outro caminho.

Então, qual caminho será escolhido se no for each aninhado forem colocados atributos daqui ou daqui? Será o do for each principal. Ocorre que de fato, se não colocássemos transação base para o for each aninhado e deixamos que GeneXus a calcule, para fazer isso, primeiro removerá todos os atributos do for each aninhado que pertencem à tabela estendida do principal. E com os que ficam, só com esses determinará a tabela base. Ou seja, estes seriam removidos, pois assume que a eles se chega pelo for each principal.

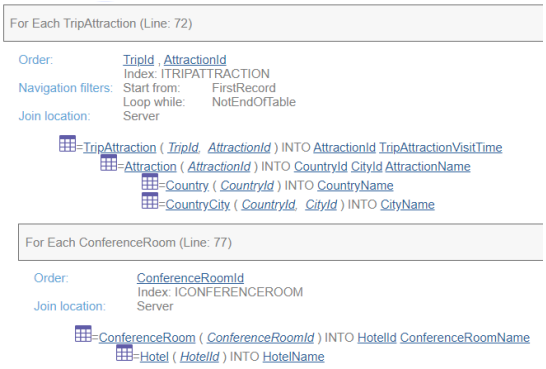
CARTESIAN PRODUCT

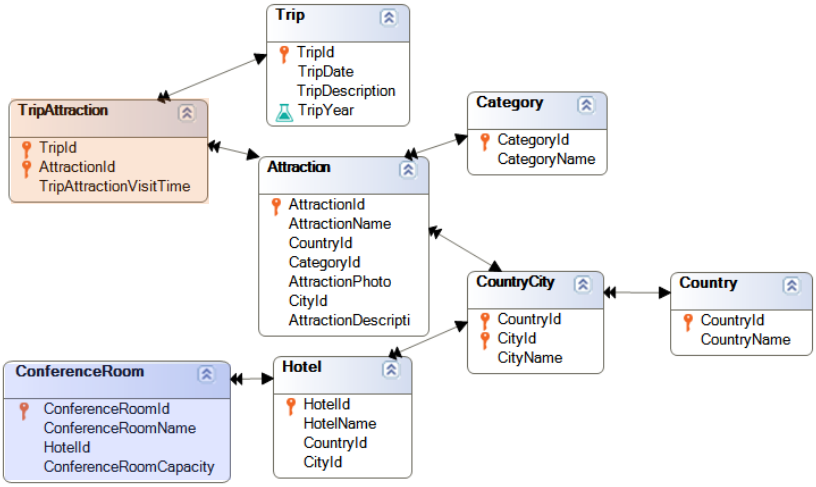



```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CountryName, CityName
   endfor
endfor
```

Por exemplo, se adicionamos uma tabela ConferenceRoom com uma relação N a 1 com Hotel e especificamos estes for eachs aninhados, onde estas são claramente as tabelas base, vemos que no primeiro for each não é solicitado nada das tabelas em comum. Mas além disso, a partir da tabela base só seria necessário acessar Attraction para obter AttractionName.

Mas se observamos o for each aninhado, estão aparecendo além de um atributo de Hotel, CountryName de Country e CityName de CountryCity. Poderíamos pensar que então vai utilizar os associados através de ConferenceRoom.
No entanto, se observamos a lista de navegação…






```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CountryName, CityName
   endfor
endfor
```

…vemos que não, que no For each aninhado só acessa ConferenceRoom para obter o ConferenceRoomName e o HotelId através do qual acessa em Hotel este registro para recuperar HotelName. E ali fica. (Observemos que não há join)
De onde recupera então os valores de CityName e CountryName?

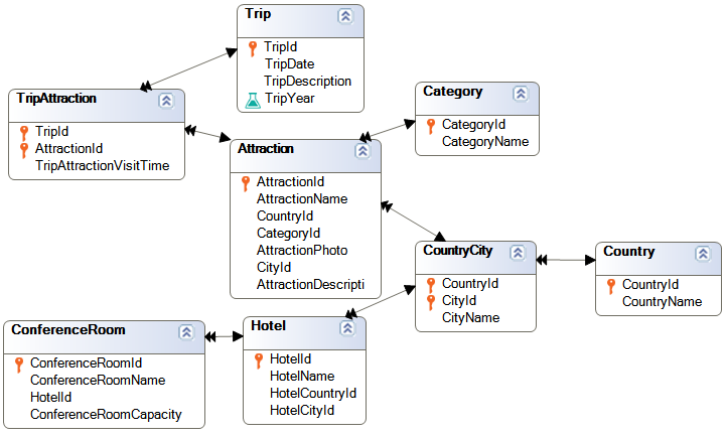
Do for each principal. Vejamos que ele acessa Attraction para recuperar AttractionName, mas também CountryId e CityId para poder acessar estas duas tabelas e recuperar respectivamente CountryName e CityName.

E como faríamos se quiséssemos os valores do país e cidade do Hotel da ConferenceRoom?

Uma primeira ideia pode ser através de um subtipo.

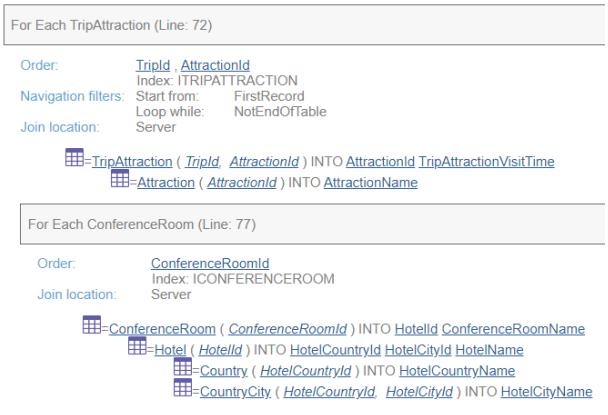
Subtype	Description	Supertype
HotelCountryCity		
HotelCountryId	Hotel Country Id	CountryId
HotelCityId	Hotel City Id	CityId
HotelCountryName	Hotel Country Name	CountryName
HotelCityName	Hotel City Name	CityName

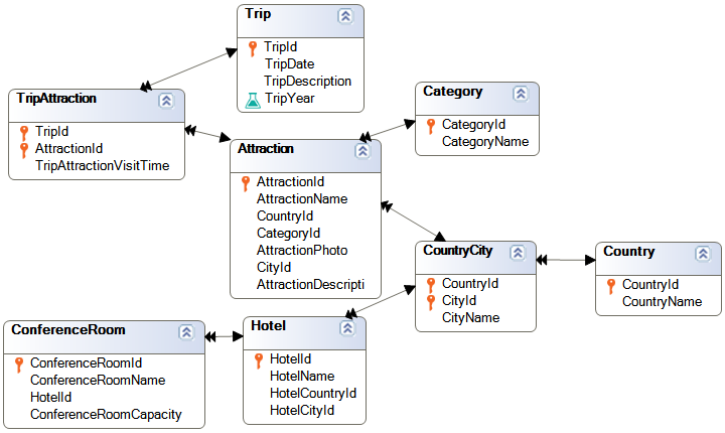
Name
Hotel
HotelId
HotelName
HotelCountryId
HotelCountryName
HotelCityId
HotelCityName




```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, ~~CountryName~~, ~~CityName~~  HotelCountryName, HotelCityName
   endfor
endfor
```

Por exemplo, se definimos este grupo de subtipos e o utilizamos em Hotel em vez dos supertipos… Vemos que agora a tabela tem os subtipos, e será suficiente substituir no for each aninhado os supertipos pelos subtipos correspondentes.

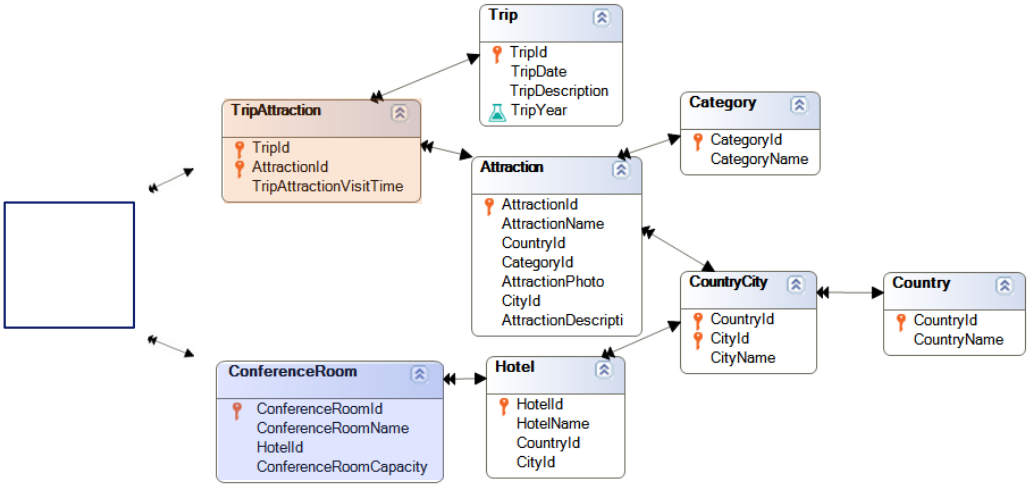




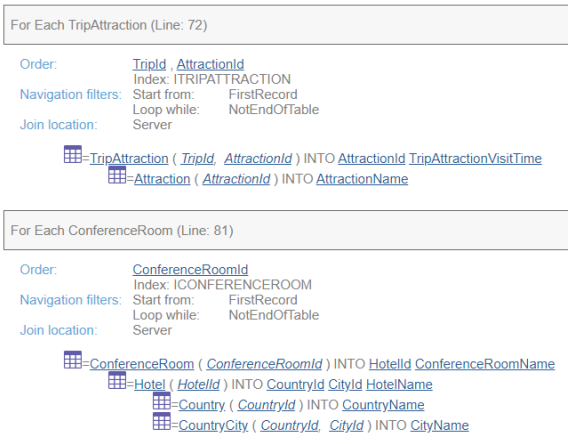

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, ~~CountryName~~, ~~CityName~~  HotelCountryName, HotelCityName
   endfor
endfor
```

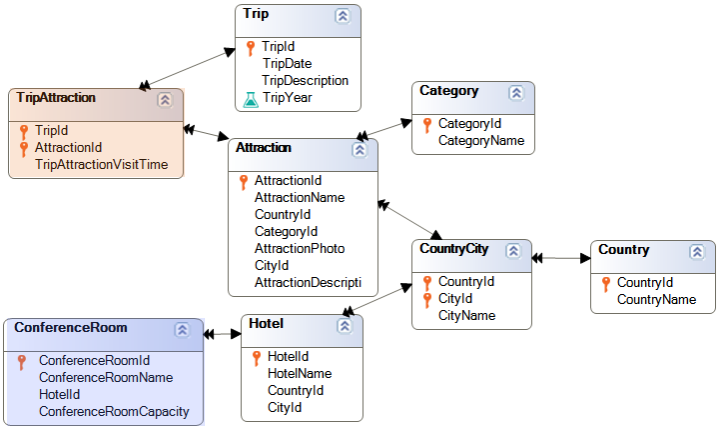
Aqui vemos a lista de navegação informando o que buscávamos.

Mas não parece uma boa ideia colocar um subtipo só porque em um caso de for eachs aninhados queremos desambiguar. Observemos que aqui não há uma ambiguidade no modelo.



Diferente seria o caso se existisse esta tabela que apresenta dois caminhos para chegar a estas outras.






```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   Do 'PrintRooms'
endfor

Sub 'PrintRooms'
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CountryName, CityName
   endfor
endsub
```

A maneira mais inteligente de resolver este problema, então, será não modificar em nada o modelo e escrever o segundo for each em uma sub-rotina. E agora a lista de navegação nos indica exatamente o que queremos.

Aqui está acessando CountryName e CityName. A partir de ConferenceRoom, e não a partir de TripAttraction.

JOIN

1-N relashionship

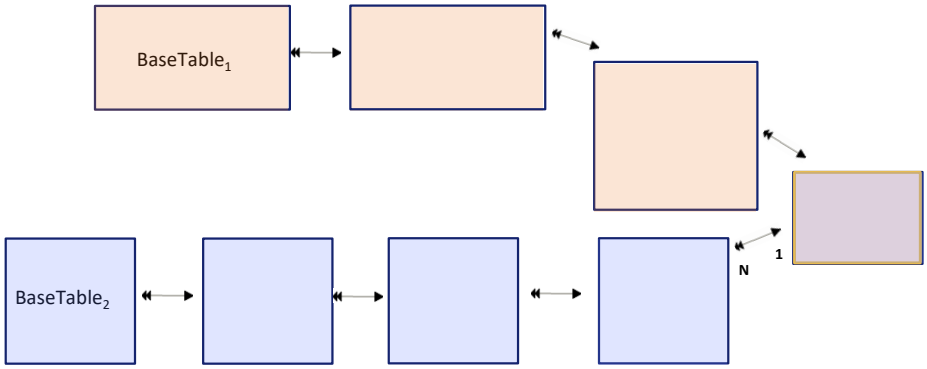
1 $BaseTable_1 \subset ext(BaseTable_2)$

2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

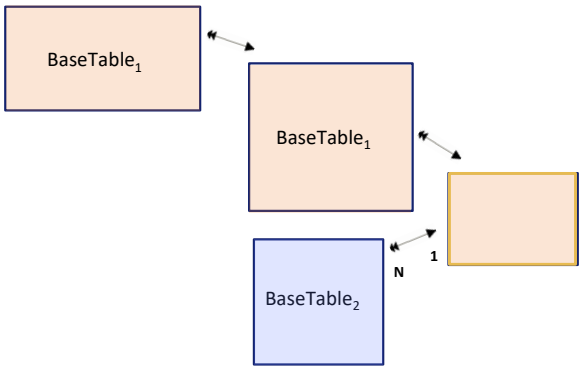
NO 1-N relashionship



Antes de continuar, vamos focar novamente no segundo caso de Join…

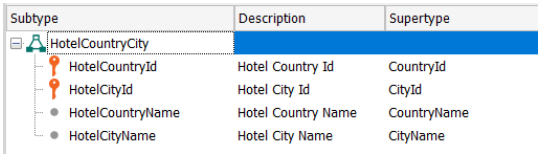
1 $BaseTable_1 \subset ext(BaseTable_2)$

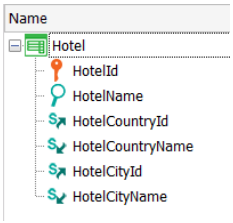
2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

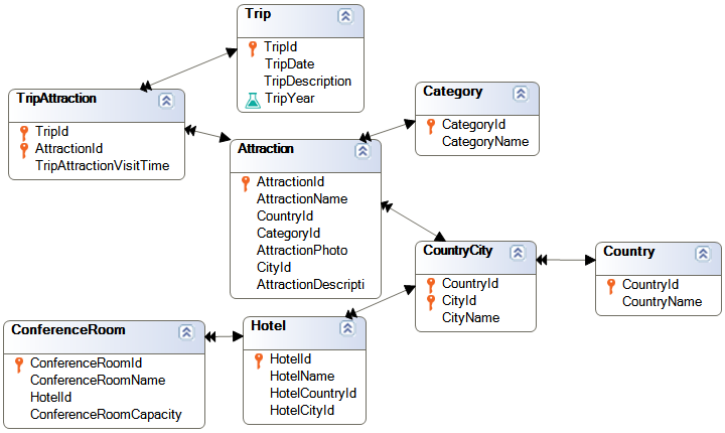


… que havíamos visto, para problematizar por um momento o que acontece se a relação se dá por meio de subtipos. De fato, vamos ver o exemplo mais simples de todos.

O que acontecerá se esta relação não for produzida utilizando os supertipos, mas subtipos?



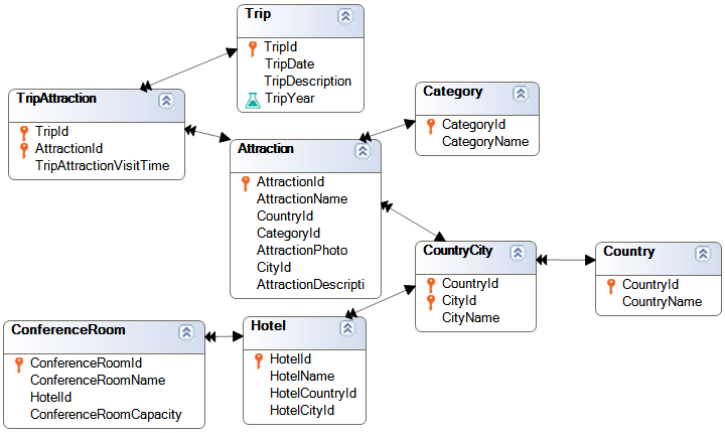





```
for each Attraction
   print PB1 //AttractionName
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```

No exemplo que vimos, se em vez de CountryId e CityId, em Hotel colocamos estes subtipos.

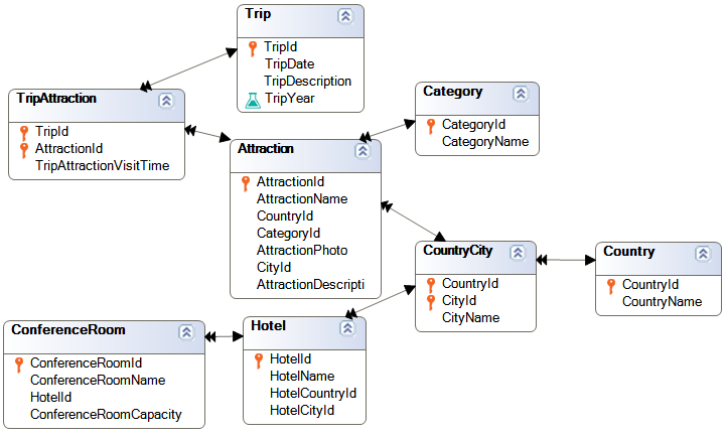
Se nossos for eachs aninhados são escritos assim: ou seja, primeiro percorre-se a tabela Attraction e para cada uma é percorrida a tabela Hotel, então GeneXus encontra relação…




```
for each Attraction
   print PB1 //AttractionName
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```

.. e realiza o join. Vemos que no primeiro for each recupera os valores de CountryId e CityId, para depois no for each aninhado filtrar os registros de Hotel para os quais os subtipos coincidam com estes.

Porém…




```
for each Attraction
   print PB1 //AttractionName
   for each Hotel
      print PB2 //HotelName
   endfor
endfor
```

```
for each Hotel
   PB2 //HotelName
   for each Attraction
      print print PB1 //AttractionName
   endfor
endfor
```

… se forem escritos invertidos os For eachs, ou seja, no externo são percorridos os hotéis e no aninhado as atrações, não será feito um join, mas um produto cartesiano.

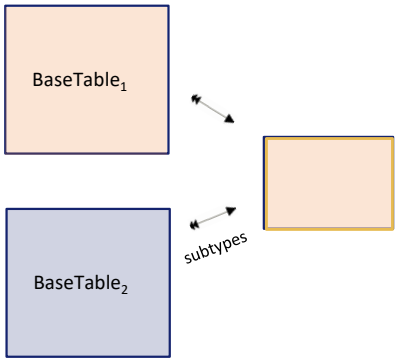
A regra é: é feito join entre supertipo e subtipo, mas não vice-versa. Ou seja, como em hotel não temos CountryId e CityId, mas subtipos destes, ou seja, para casos particulares, não fica claro para GeneXus que o desenvolvedor queira passar do particular ao geral e então sejam listadas apenas as atrações com os mesmos valores para os supertipos.

JOIN

```
for each
   //SUPERTYPE
   for each
      //SUBTYPE
   endfor
endfor
```

CARTESIAN PRODUCT

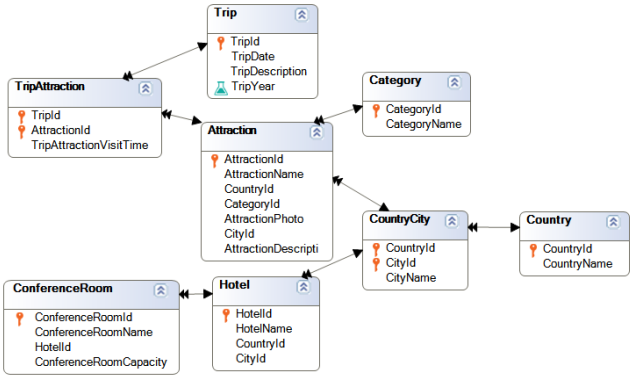
```
for each
   //SUBTYPE
   for each
      //SUPERTYPE
   endfor
endfor
```



Aqui vemos isso sintetizado.

JOIN

```
for each TripAttraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Hotel
      print PB2 //HotelName, CategoryName
   endfor
endfor
```



$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

```
for each TripAttraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CategoryName
   endfor
endfor
```

Por último, (vamos voltar ao caso sem subtipos) antes de passar para o caso da mesma tabela base, vamos acrescentar que tanto no caso de Join quanto no de Produto Cartesiano, podem ser utilizados no for each aninhado atributos da tabela estendida do principal que não estejam na estendida do aninhado. Nos dois casos mostrados, no for each aninhado, é solicitado imprimir CategoryName que não está na estendida de seu for each. Mas está na estendida do for each pai.

Portanto, para cada tripattraction são impressos os valores destes dois atributos, e é obtido o valor de CategoryName, que será utilizado no for each aninhado como valor dado. Para o caso do Join, também são obtidos os valores de CountryId e CityId para poder fazer o join, justamente.

Então, no primeiro caso, são percorridos todos os Hotéis do mesmo país e cidade(ali são utilizados esses valores recuperados de CountryId e CityId) e para cada um é impresso o nome de hotel e o nome da categoria que foram obtidos no primeiro for each.

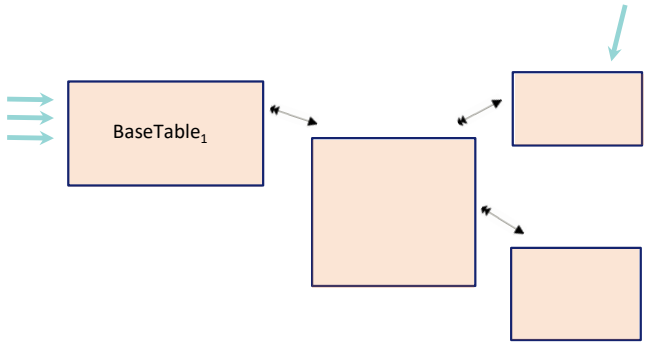
Já para o segundo caso, são percorridas todas as conferencerooms (sem filtros porque não há join) e é impresso seu nome, o nome de seu hotel e o valor de CategoryName da atração do for each pai.

Lembremos que para determinar a tabela base do for each aninhado (e é assim também para resolver a navegação) são removidos primeiro os atributos que, estando no aninhado, já fazem parte da estendida do principal. É por este motivo.

```
for each Trip.Attraction order CategoryName
   print PB1 //CategoryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttracionVisitTime, CityName
   endfor
endfor
```

CONTROL BREAK

$BaseTable_1 = BaseTable_2$



Bem, agora passemos a analisar o corte de controle. Sabemos o que acontece quando a tabela base de cada for each é a mesma e só faz sentido quando queremos processar informação agrupada. Pode ser agrupada por qualquer atributo ou conjunto de atributos da tabela estendida.

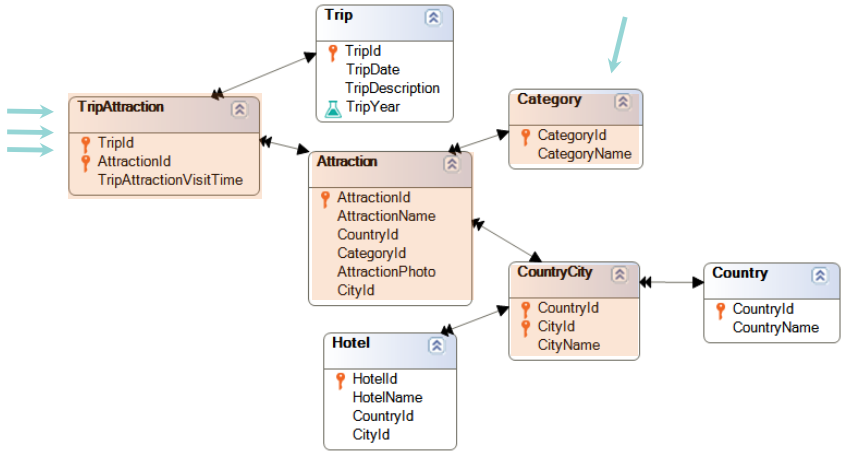
Por exemplo, agrupamos de acordo com o valor de um atributo desta tabela, que deverá aparecer na cláusula order, e processamos todos os registros associados desta tabela (e da estendida) que tenham o mesmo valor para esse atributo.

Assim, por exemplo…

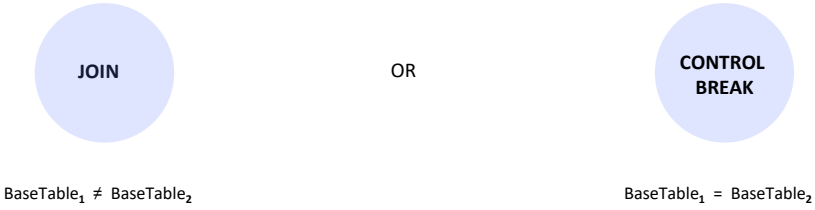
```
for each Trip.Attraction order CategoryName
   print PB1 //CategoryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttracionVisitTime, CityName
   endfor
endfor
```

CONTROL BREAK

$BaseTable_1 = BaseTable_2$



Agrupamos as tripattractions por categoria e listamos para cada grupo o nome de categoria e percorremos as tripattractions dessa categoria, imprimindo o nome da atração, o tempo de visita e nome da cidade da atração, de cada uma das tripattractions com a mesma categoria.

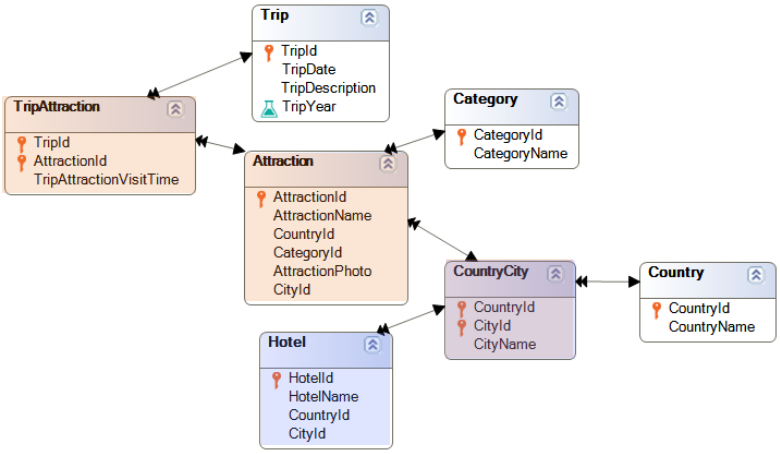


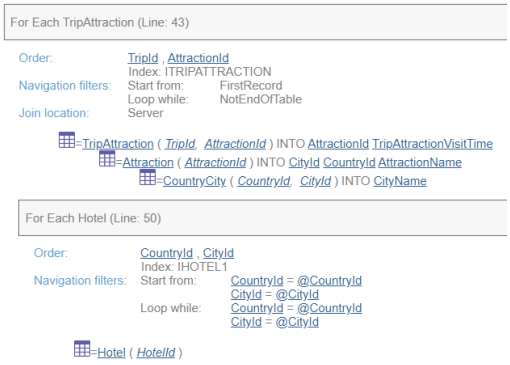
Até agora não nos preocupávamos com a determinação da tabela base de cada for each, pois a dávamos por determinada, ao utilizar transação base. Mas quando deixamos esta tarefa nas mãos do GeneXus, o que era um Join poderá se tornar um Corte de Controle.

Veremos isso voltando ao último caso de Join que analisamos.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ~~Hotel~~ Country.City
      print PB2 //CityName
   endfor
endfor
```





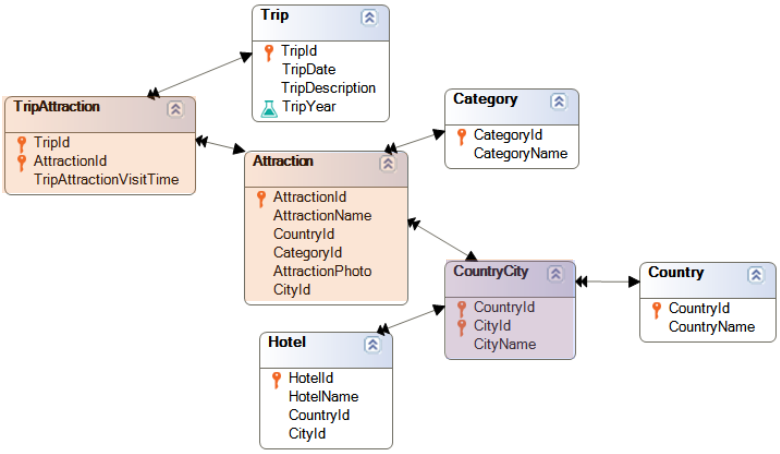
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

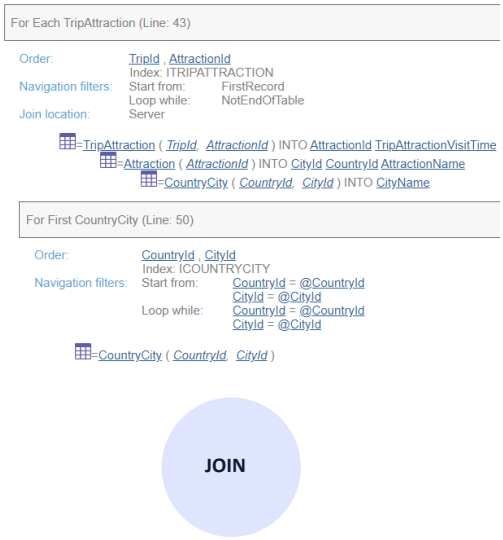
Com uma pequena diferença: estamos imprimindo no for each aninhado, em vez dos nomes dos hotéis com a mesma cidade da atração da trip, os nomes da cidade de cada um desses hotéis.

Agora vamos ver que, se em vez de Hotel tivéssemos especificado como transação base Country.City, então esta se torna a tabela base do aninhado, e será um caso particular deste, mas onde o segundo for each só retornará um registro.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   print PB2 //CityName
endfor
```





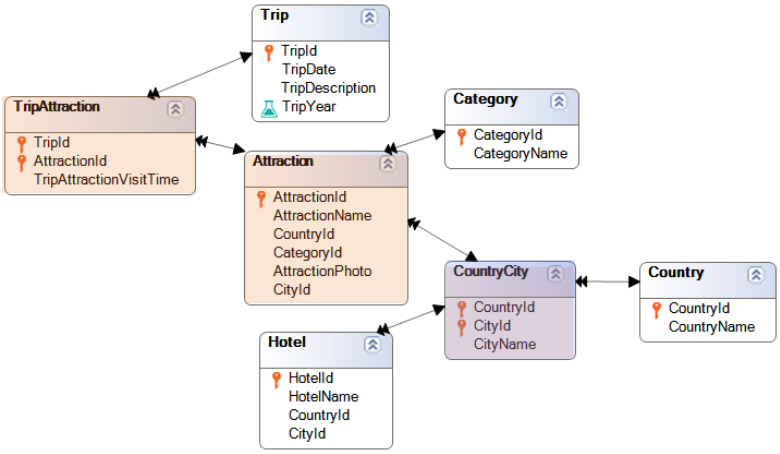
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

Vemos isso claramente na lista de navegação que indica **For First** em vez de For Each. Porque, claro, CountryId, CityId são a chave primária da tabela.

É como se não tivesse sido especificado o for each e o printblock tivesse sido impresso diretamente, porque CityName está na tabela estendida de TripAttraction.

2

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each Country.City
      print PB2 //CityName
   endfor
endfor
```



$BaseTable_1 = BaseTable_2$

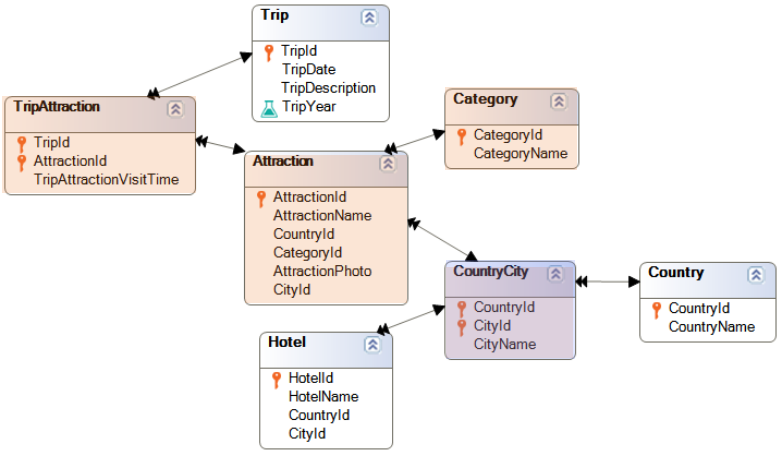
É por isso que se não colocássemos transação base e deixássemos para GeneXus determinar por si só a tabela base, escolherá TripAttraction, ou seja, entenderá que se pretendia implementar um corte de controle, pois supondo que o desenvolvedor não escreve for eachs, além disso, nenhuma outra coisa fará sentido.

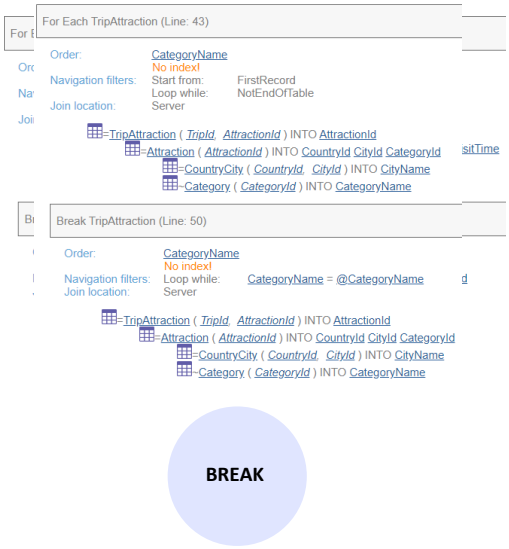
Assim veremos a lista de navegação.

Porém, isto também não fará sentido, pois será um corte de controle pela chave primária, ou seja, será trabalhado no for each aninhado com o mesmo registro do for each principal cada vez.

2

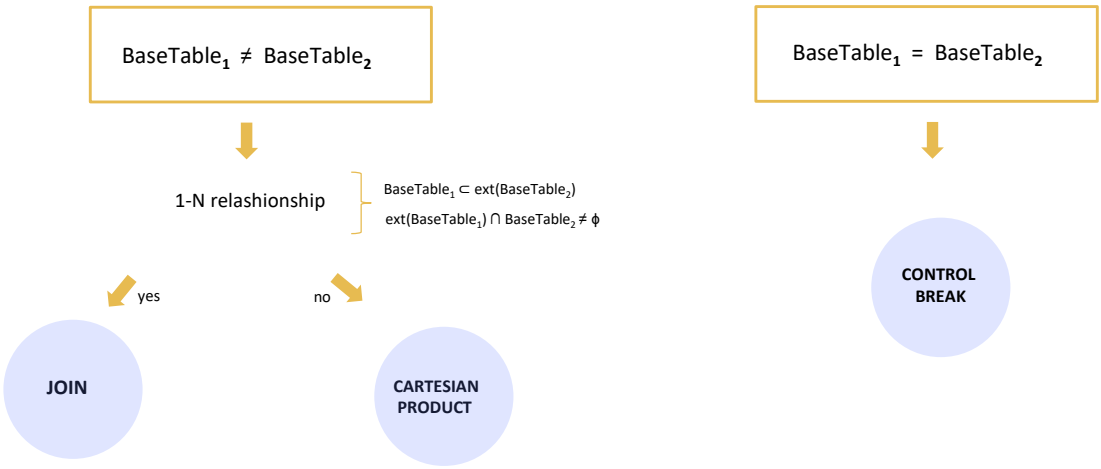
```
for each Trip.Attraction order CategoryName
   print PB1 //CategoryName
   for each
      print PB2 //CityName
   endfor
endfor
```





$BaseTable_1 = BaseTable_2$

Faltaria especificar uma cláusula order para cortar por algum atributo ou conjunto de atributos cujos valores possam ser repetidos. Por exemplo, CategoryName. E assim temos o mesmo exemplo que vimos antes do corte de controle.



Resumindo: se as tabelas base forem diferentes, GeneXus procurará se encontra uma relação 1 a N de algum destes dois tipos. No caso de encontrá-la, implementará um join implícito. No caso de não a encontrar, não implementará nenhum join, e este caso chamamos de produto cartesiano. Mas é importante fazer a ressalva de que chamá-lo de produto cartesiano não significa que o seja efetivamente. Se o desenvolvedor adiciona explicitamente um filtro, claramente serão trazidos os registros filtrados, então haverá um tipo de join, mas não será o join automático. Dizemos que o caso é produto cartesiano, portanto, apenas do ponto de vista dos filtros automáticos que determina GeneXus: ou seja, neste caso, nenhum.

O corte de controle é claro.

Com isto concluímos a análise formal dos três tipos de navegações possíveis quando temos for eachs aninhados.

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