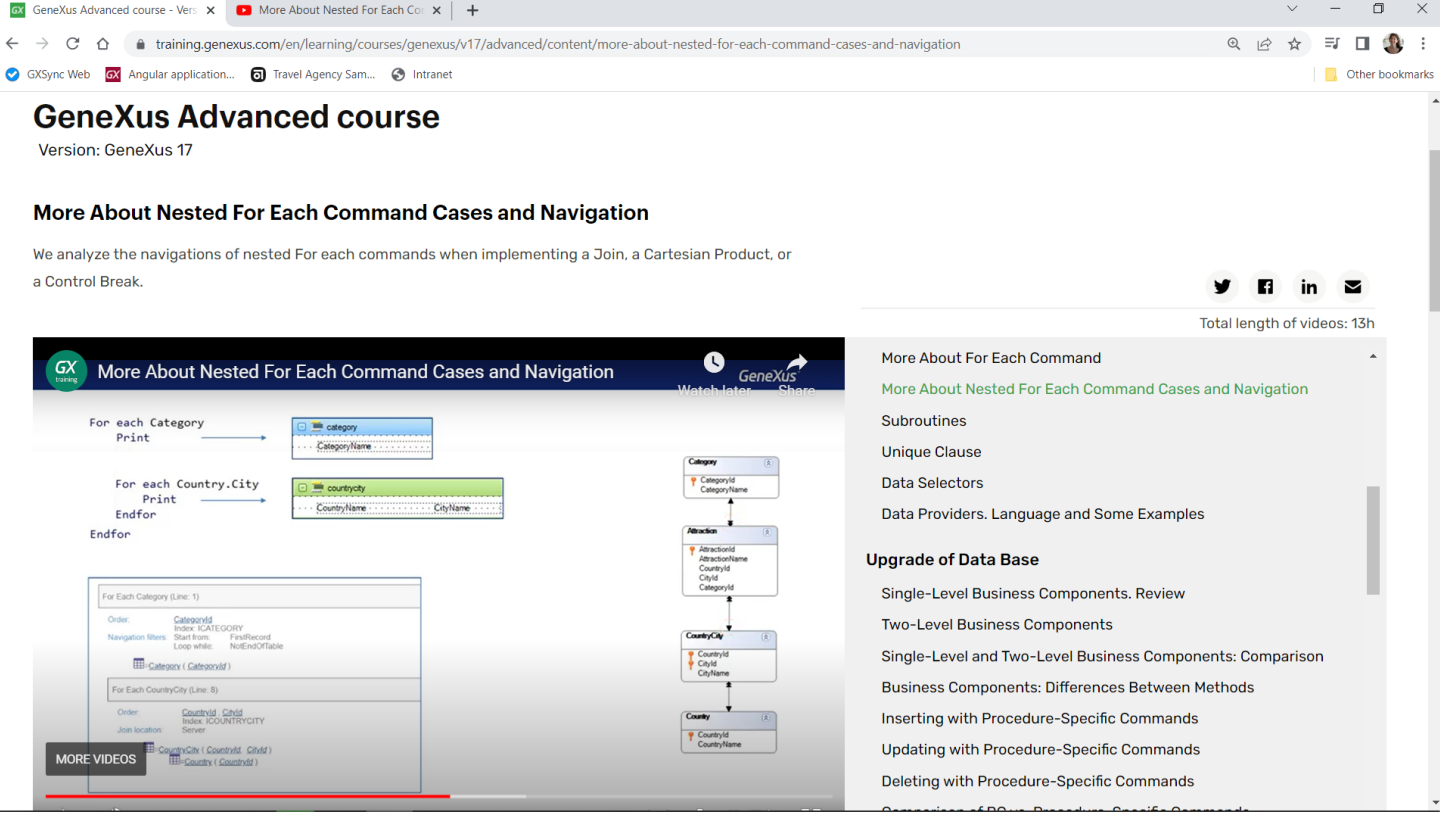
Lógica de consulta a la base de datos con GeneXus

Casos de For eachs andidados. Formalización

GeneXus™



En este video... habíamos analizado los tres tipos de navegaciones que GeneXus implementaba al encontrar for eachs anidados.

Volveremos a ellos para formalizarlos un poco más.

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

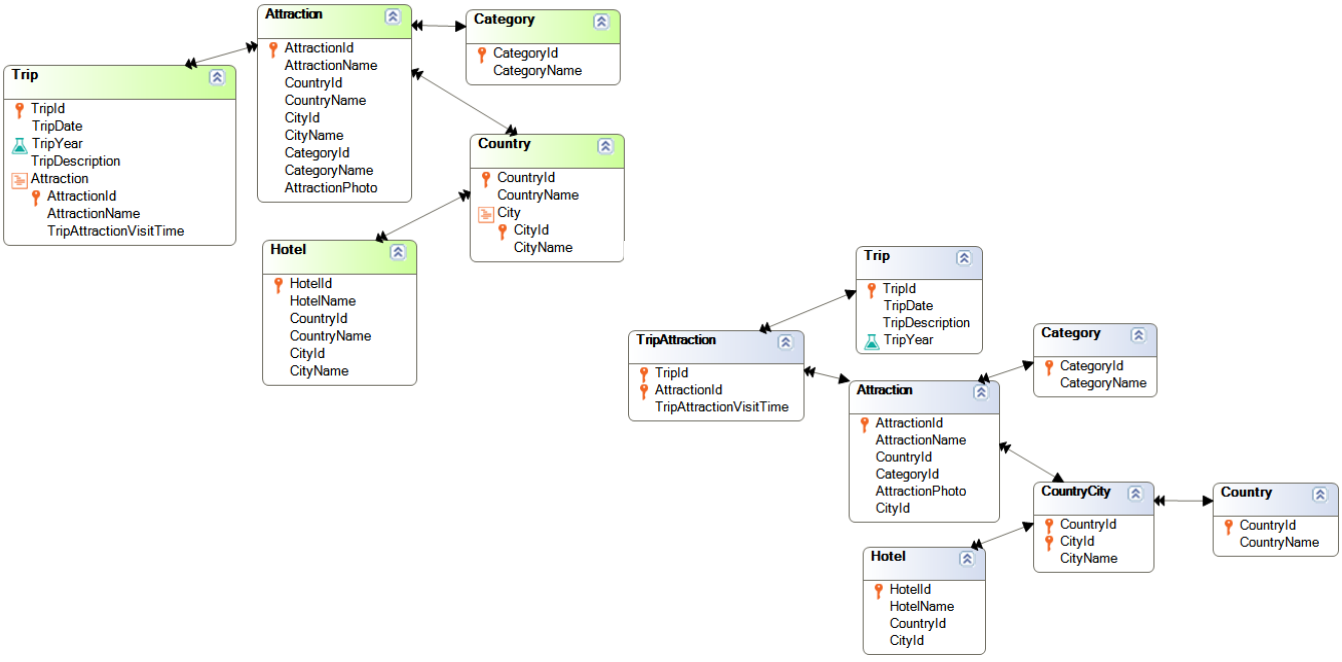
BaseTable$_1$ = BaseTable$_2$

Cuando dentro del cuerpo de un for each aparece otro querrá decir que para cada registro de una navegación sobre una tabla se quieren recorrer muchos registros en otra navegación de la misma o de otra tabla. Un caso particular es cuando se recupera un registro solo.

Primero se determinan las tablas base y luego se indaga sobre su relación, que dará lugar a uno de tres casos: Join, Producto Cartesiano y Corte de Control.

Sabemos de una primera gran diferencia: en los dos primeros casos se tratará de tablas base distintas, mientras que el último es el caso de la misma tabla base.

¿Qué diferencia el Join del Producto Cartesiano? La identificación de una relación 1 a N directa o indirecta. Formalicemos todos los casos.

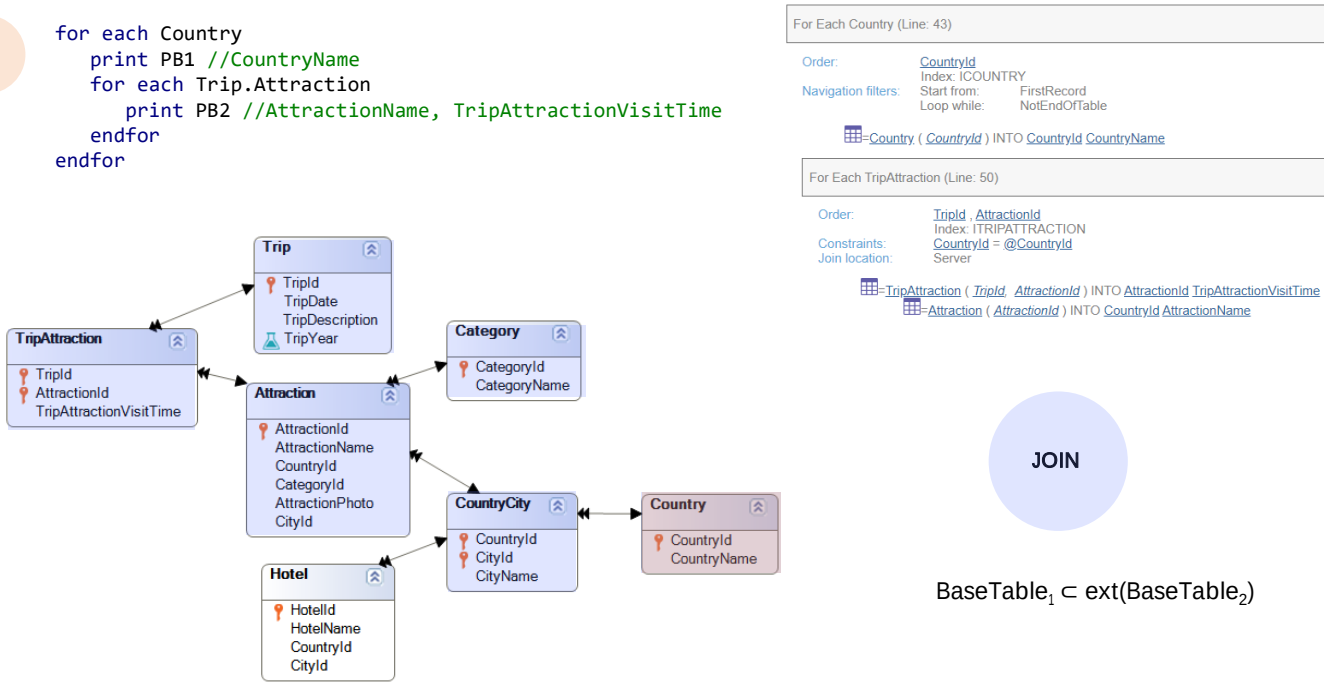


Supongamos que tenemos estas 5 transacciones para registrar las atracciones turísticas que pueden visitarse en excursiones, donde cada atracción corresponde a una categoría (como Museo o Monumento), y es de una ciudad de un país, y por otro lado se tienen hoteles de cada país y ciudad.

De estas transacciones obtenemos estas tablas con sus relaciones.

1

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



$BaseTable_1 \subset ext(BaseTable_2)$

Empecemos por el Join. ¿Cómo se resolverá este código?

Se está pidiendo imprimir para cada nombre de país los nombres de atracción con su tiempo de visita estipulado en el trip. ¿Hay relación entre la información?

Sabemos que un registro de TripAttraction corresponde a una única atracción, que corresponde a una única ciudad, que corresponde a un único país. Dicho de otro modo, desde la tabla TripAttraction llegamos a Country de modo único. Por eso podemos imprimir para el segundo for each tan solo los tripattractions que correspondan al país del for each principal, como vemos en el listado de navegación como constraint. Donde este arroba CountryId corrresponde al CountryId del registro del for each externo en el que estemos posicionados. Y este atributo CountryId es el de Attraction, a quien llegamos por TripAttraction. Es claro que se trata de un Join, con una relación 1 a N indirecta.

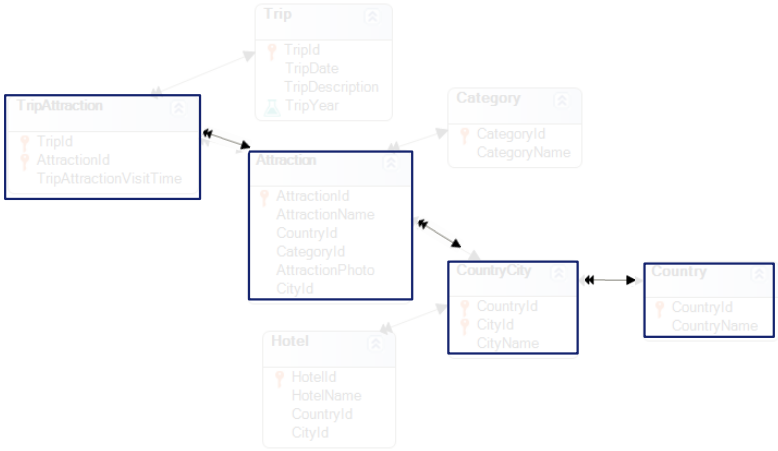
Es porque para un tripattraction siguiendo las claves foráneas encontramos un único CountryId, y, por tanto, para un CountryId podemos encontrar N tripattractions que haciendo ese recorrido den con él.

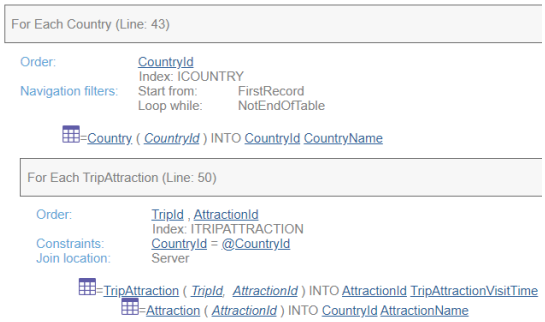
Si formalizamos este caso de relación 1 a N indirecta, estamos diciendo que la tabla base del for each principal está contenida en la tabla extendida de la del for each anidado.

Ese caso incluye al más simple de todos. Por ejemplo, pensemos que si la tabla base del segundo for each fuera CountryCity, esta fórmula se cumple.

1

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





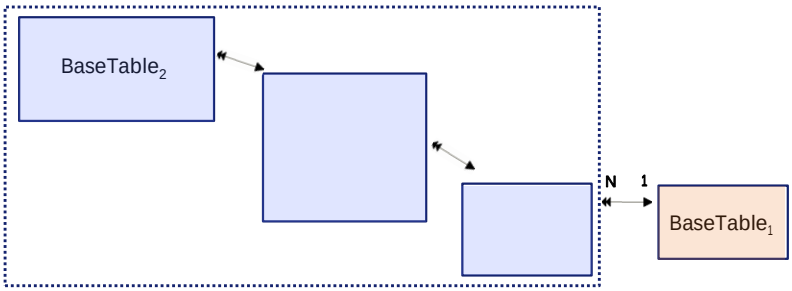
JOIN

$BaseTable_1 \subset ext(BaseTable_2)$

Aquí vemos mejor las relaciones entre las tablas involucradas en este caso.

1

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



JOIN

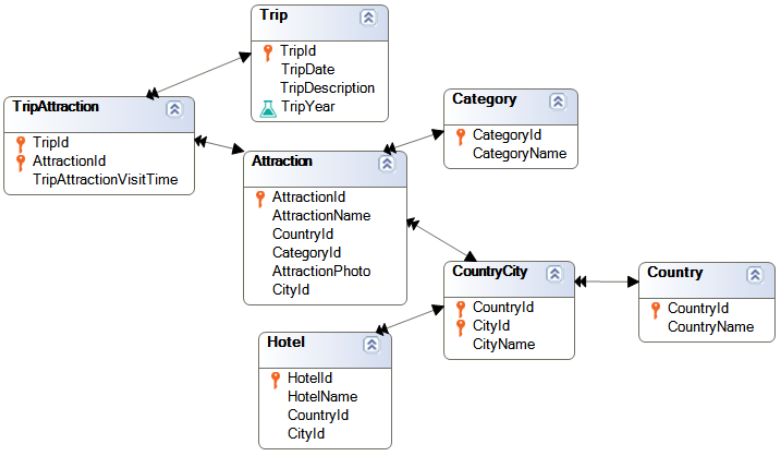
$BaseTable_1 \subset ext(BaseTable_2)$

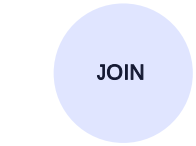
O sea, la tabla base del principal es esta, y la tabla base del anidado esta otra.

Se ve clara la relación 1 a N indirecta. Es indirecta a través de al tabla extendida del for each anidado.

2

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





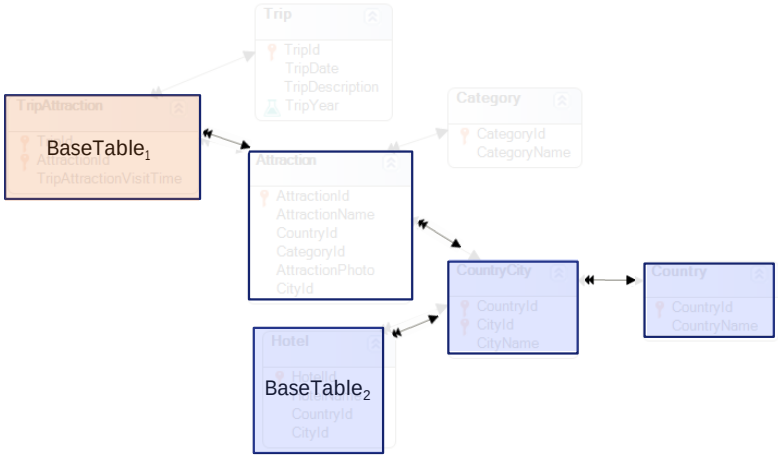
~~$BaseTable_1 = ext(BaseTable_2)$~~

Ahora veamos el otro tipo de relación 1 a N indirecta, esta vez no por la vía de la extendida del anidado, sino del principal.

Para cada tripattraction imprimimos nombre de atracción y tiempo de visita en ese trip, y luego los nombres de hoteles.

2

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



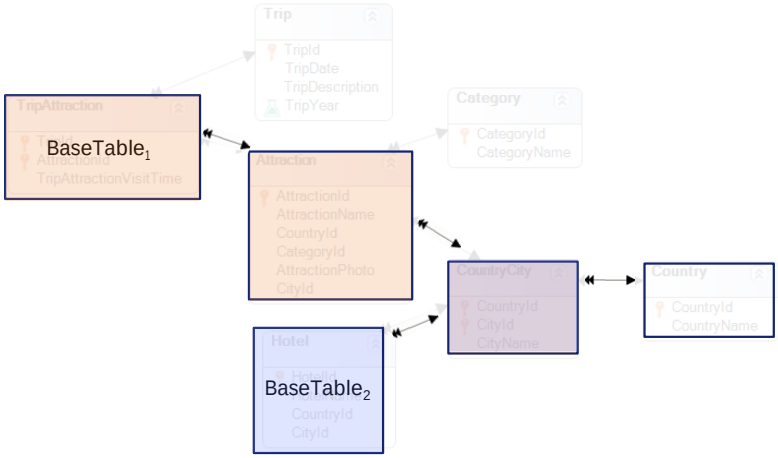
JOIN

~~$BaseTable_1 = ext(BaseTable_2)$~~

Aquí claramente no se cumple que la tabla base del for each principal esté incluida en la tabla extendida del for each anidado. No hay una relación 1 a N indirecta por esta vía. Sin embargo...

2

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

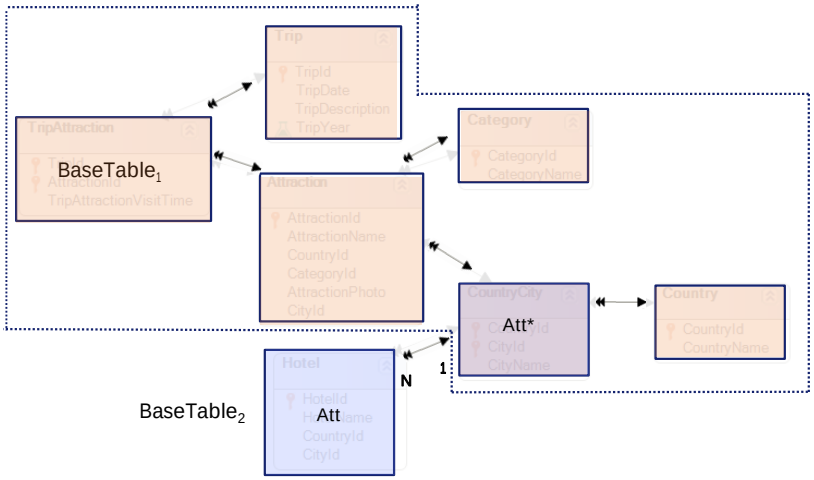


JOIN

...sabemos que para cada registro del **for each principal** llegamos a un registro de esta tabla, al que se llega también directamente desde la tabla base del anidado.

2

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



JOIN

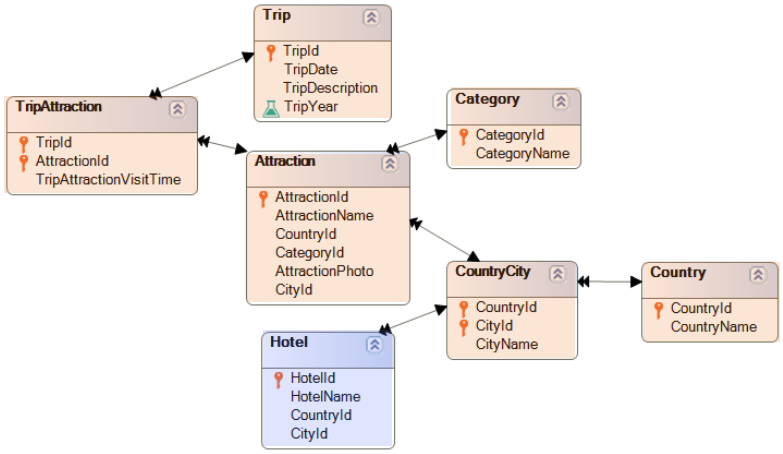
ext(BaseTable$_1$) ∩ BaseTable$_2$ ≠ ϕ

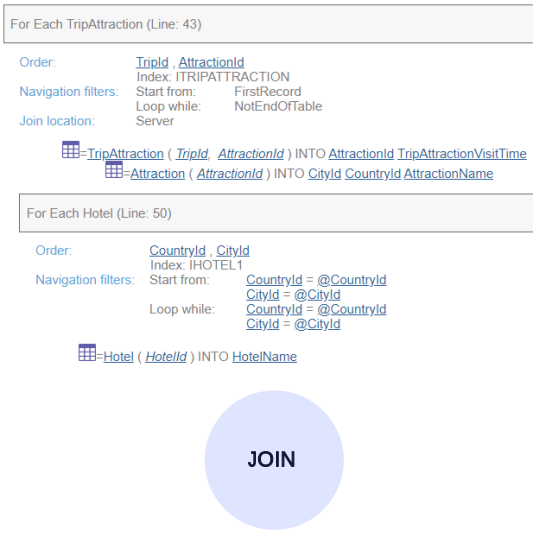
Es decir, en esta habrá una clave foránea a esta otra. En definitiva, compartirán ese atributo (debido a él se establece esta relación 1 a N entre las tablas).

En definitiva, la tabla extendida del for each principal tendrá algún atributo en común con la tabla base del anidado, y esos establecerán el join.

2

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





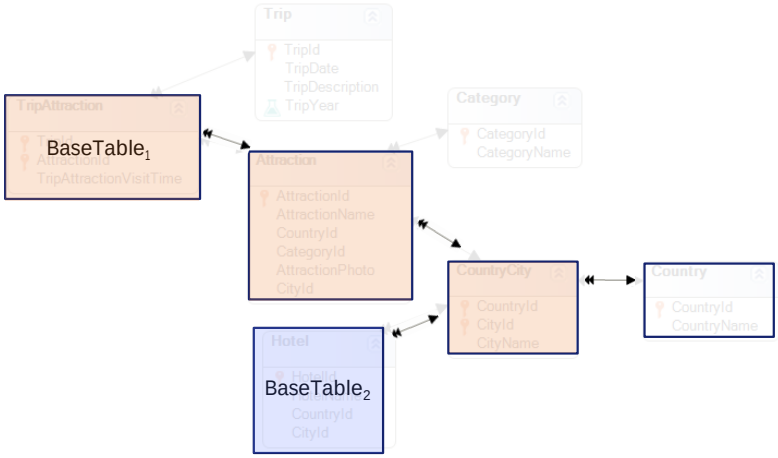
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

Aquí serán CountryId, CityId, clave foránea a CountryCity.

Lo vemos claramente en el listado de navegación. Se imprimirán solo los hoteles que correspondan a la misma ciudad de la atracción del trip.

2

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

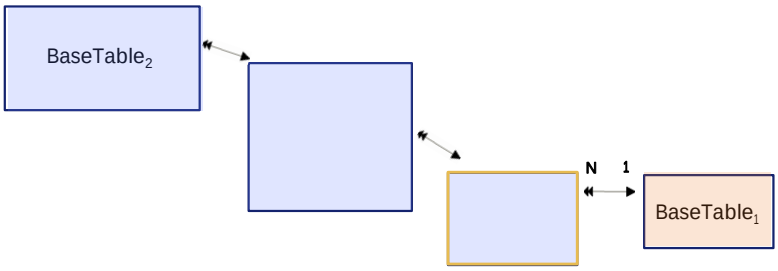


JOIN

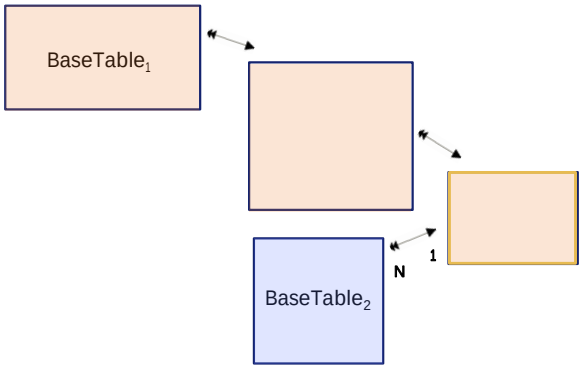
ext(BaseTable$_1$) ∩ BaseTable$_2$ ≠ ϕ

Otra vez, para verlo más claro, quedémonos solo con las tablas implicadas: aquí vemos la relación 1 a N indirecta.

1 $BaseTable_1 \subset ext(BaseTable_2)$

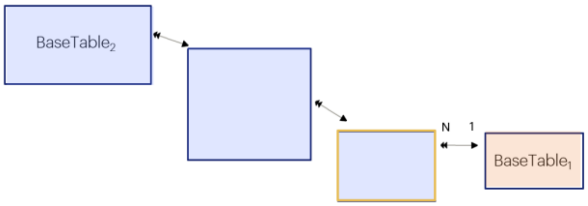


2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$



Si en el primer caso se recorría esta tabla base y para el anidado esta otra y por tanto la relación era 1 a N indirecta por vía de la extendida de la del anidado, en el segundo caso lo es por vía de la extendida de la tabla base del principal. Y es aquí que se establece la 1 a N con la tabla base del anidado.

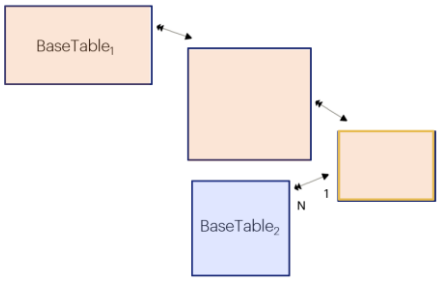
1 $BaseTable_1 \subset ext(BaseTable_2)$



JOIN

1-N relashionship

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



$BaseTable_1 \neq BaseTable_2$

Por tanto el caso de Join es el de tablas base distintas donde se encuentra una relación 1 a N directa o indirecta, de acuerdo a estas opciones: tabla base con extendida, o tabla extendida con tabla base.

No así tabla extendida con tabla extendida.

JOIN

1-N relashionship

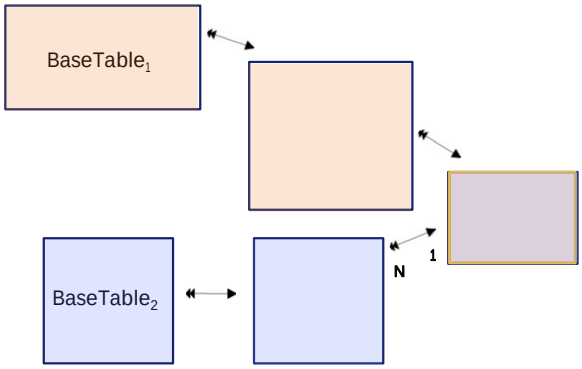
1 $BaseTable_1 \subset ext(BaseTable_2)$

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

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

NO 1-N relashionship



Es decir, si en lugar de ser esta la tabla base del anidado es esta otra, hay relación entre las tablas extendidas, claramente, pues ambas llegan a la misma tabla. Sin embargo aquí no se producirá un join sino un producto cartesiano.

¿Por qué, si en verdad por cada registro de la tabla base del for each principal podríamos quedarnos solo con los de la tabla base del anidado que correspondan al mismo registro de esta tabla a la que ambos llegan de modo único?

Es que cuanto más indirecta la relación, menos probable parece que el desarrollador esté buscando tomarla en cuenta, porque la relación cada vez parece más lejana, y si el desarrollador la estuviera buscando, siempre puede explicitarla.

JOIN

1-N relashionship

1 $BaseTable_1 \subset ext(BaseTable_2)$

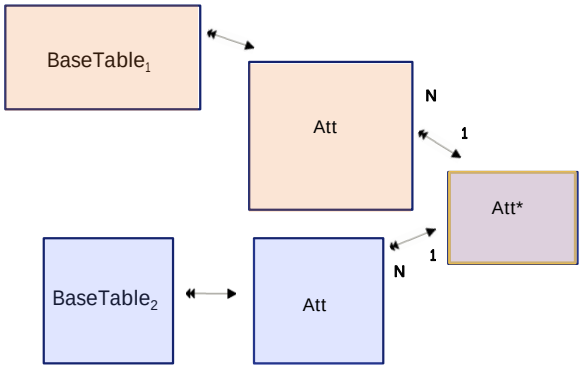
2 $ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

NO 1-N relationship

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



Por ejemplo asignándole a una variable el valor del atributo que se obtiene haciendo el camino de la tabla extendida del for each principal... es decir, el valor de este atributo que coincide con este.
Y en el for each anidado filtrando **explícitamente** los registros desde los cuales se llega a este otro atributo que se llama igual porque también es foreign key a la tabla común.

Aprovechemos a hacer esta aclaración: cuando hablamos de Join o Producto Cartesiano, de esa diferencia, nos referimos a las determinaciones implícitas de GeneXus, no a si finalmente termina filtrando o no la información del anidado. Observemos que en este caso se trata de un producto cartesiano en cuanto a que si no hubiéramos escrito ningún where GeneXus tampoco lo agregará implícitamente y se devolverán todos los registros del anidado. Pero en verdad en este caso no se devolverán todos los registros del anidado porque explicitamos un where, por lo que en verdad hará un join, pero no el Join implícito de GeneXus.

JOIN

1-N relashionship

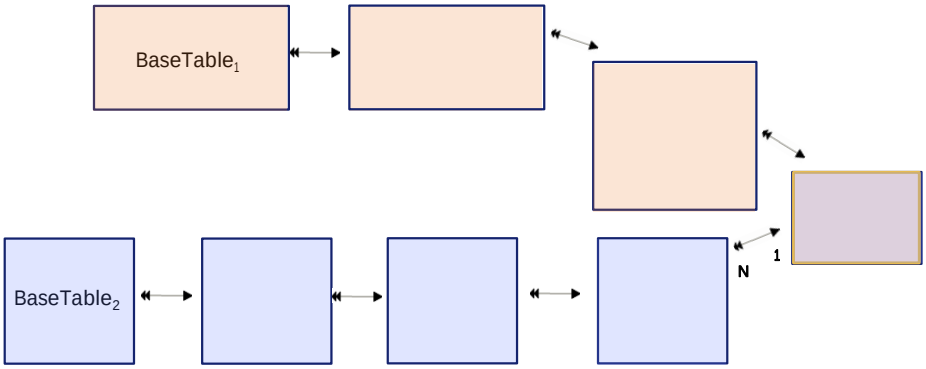
1 $BaseTable_1 \subset ext(BaseTable_2)$

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

$BaseTable_1 \neq BaseTable_2$

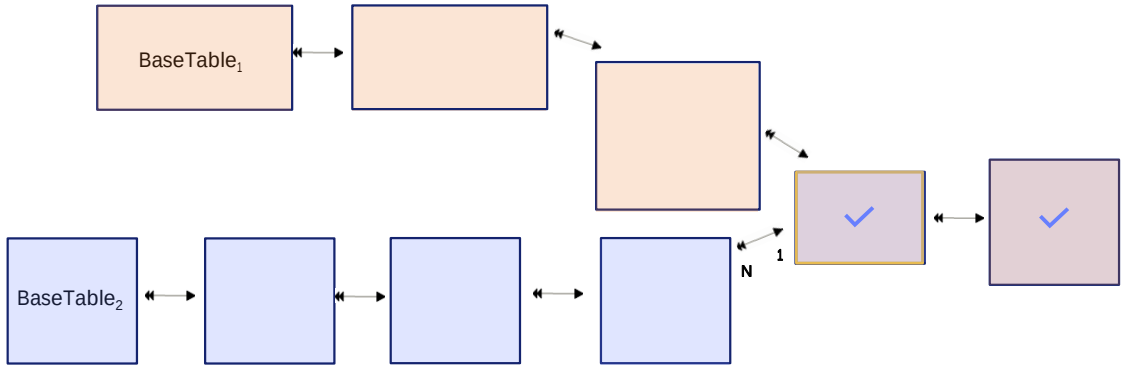
CARTESIAN PRODUCT

NO 1-N relashionship



De cuanto más lejos venga la relación menos probabilidad habrá de que el desarrollador la tenga en mente, la esté queriendo hacer valer implícitamente.

CARTESIAN PRODUCT




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

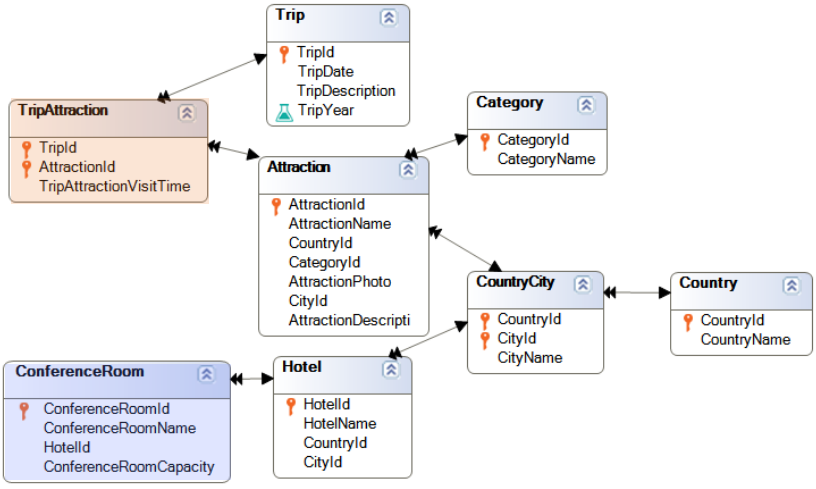
Una cuestión interesante de este caso es la siguiente. Agregamos una tabla más para que sea aún más claro.

Si hacemos la intersección de ambas tablas extendidas nos quedamos con las tablas en común. Ahora bien, si en el for each principal colocamos atributos de estas dos tablas, claramente los valores que se tomarán serán los que se obtengan partiendo de cada registro de la tabla base que satisfaga los filtros. Es decir, serán los que provengan de esta tabla extendida.

Pero ¿qué pasa si esos atributos se encuentran en el for each anidado? ¿Se toman los de la tabla extendida de este for each?
Si por un camino o por el otro llegáramos al mismo registro de esta tabla, no importaría en absoluto, porque darían el mismo valor por un camino o por el otro. Eso es lo que sucedería si se hiciera el join. Pero no hay join en este caso. Por lo que los valores de estos atributos que se obtengan por este camino no serán siempre iguales a los que se obtengan por este otro camino.

¿Entonces, qué camino se elegirá si en el for each anidado se colocan atributos de aquí o aquí? Será el del for each principal. Es que de hecho si no colocáramos transacción base para el for each anidado y dejamos que GeneXus la calcule, para hacerlo primero quitará todos los atributos del for each anidado que pertenezcan a la tabla extendida del principal. Y con los que le queden, con esos solos determinará la tabla base. Es decir, estos los quitaría, porque asume que a ellos se llega por el for each principal.

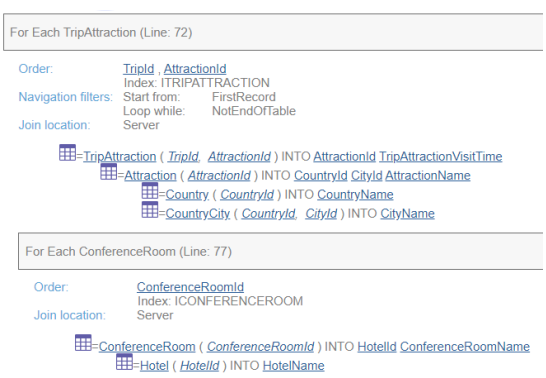
CARTESIAN PRODUCT

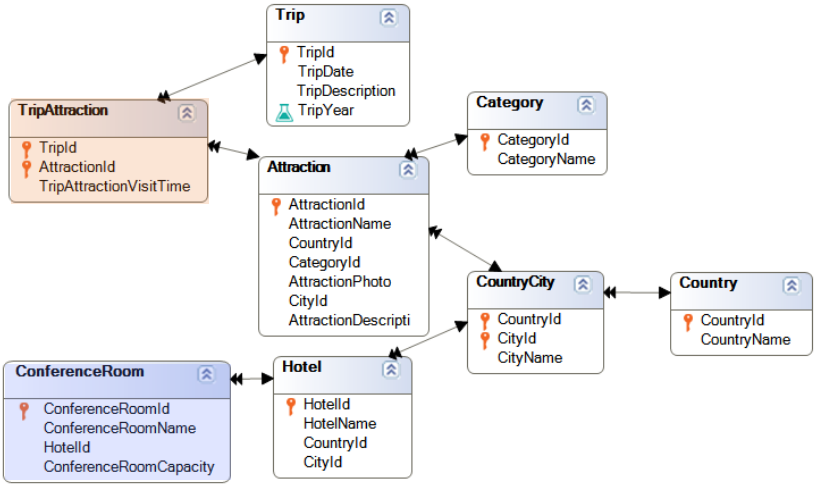



```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CountryName, CityName
   endfor
endfor
```

Por ejemplo, si agregamos una tabla ConferenceRoom con una relación N a 1 con Hotel, y especificamos estos for eachs anidados, donde estas son claramente las tablas base, vemos que en primer for each no se pide nada de las tablas en común. Pero además, desde la tabla base solamente sería necesario acceder a Attraction para obtener AttractionName.

Pero si observamos el for each anidado, están apareciendo además de un atributo de Hotel, CountryName de Country y CityName de CountryCity. Podríamos pensar que entonces va a utilizar los asociados a través de ConferenceRoom.
Sin embargo, si observamos el listado de navegación...






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

...vemos que no, que en el For each anidado solo accede a ConferenceRoom para obtener el ConferenceRoomName y el HotelId a través del cual accede en Hotel a este registro para recuperar HotelName. Y allí queda. (Observemos que no hay join)
¿De dónde recupera entonces los valores de CityName y CountryName?

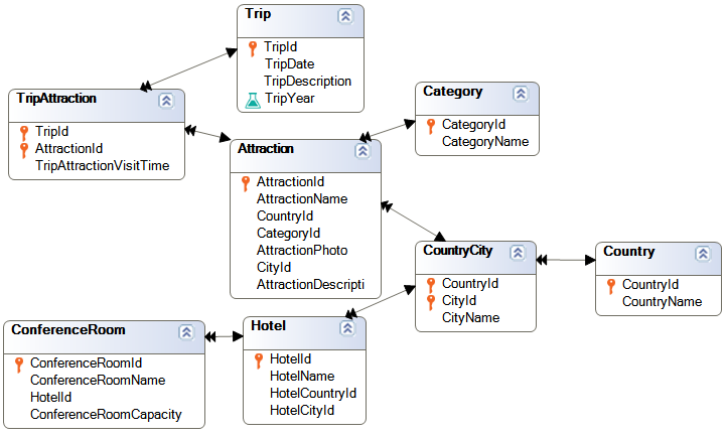
Del for each principal. Veamos que éste accede a Attraction para recuperar AttractionName pero también CountryId y CityId para poder acceder a estas dos tablas y recuperar respectivamente CountryName y CityName.

¿Y cómo haríamos si quisiéramos los valores del país y ciudad del Hotel de la ConferenceRoom?

Una primera idea puede ser a través de un subtipo.

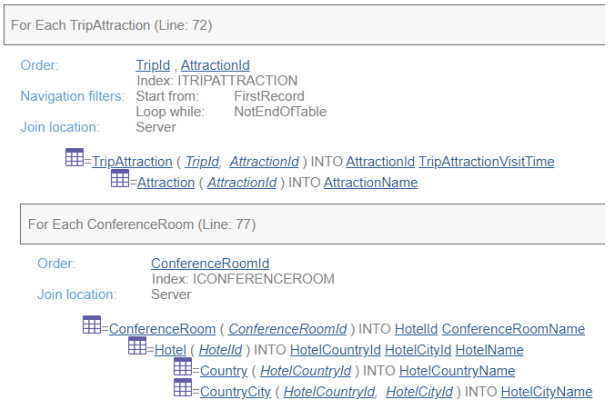
Subtype	Description	Supertype
HotelCountryCity		
HotelCountryId	Hotel Country Id	CountryId
HotelCityId	Hotel City Id	CityId
HotelCountryName	Hotel Country Name	CountryName
HotelCityName	Hotel City Name	CityName

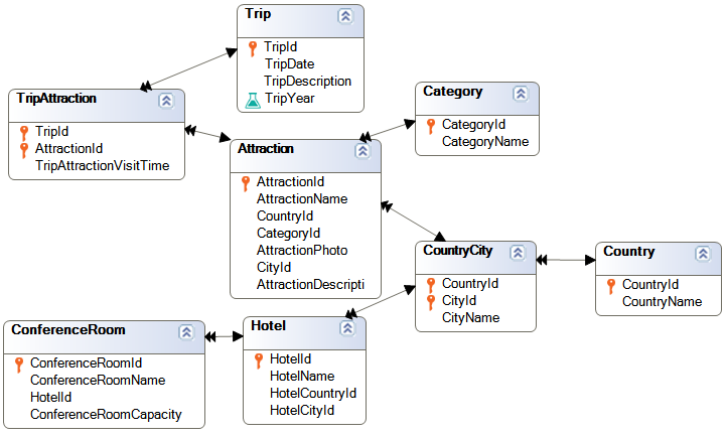
Name
Hotel
HotelId
HotelName
HotelCountryId
HotelCountryName
HotelCityId
HotelCityName




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

Por ejemplo, si definimos este grupo de subtipos y lo utilizamos en Hotel en lugar de los supertipos... Vemos que ahora la tabla tiene a los subtipos, y bastará con reemplazar en el for each anidado los supertipos por los subtipos correspondientes.

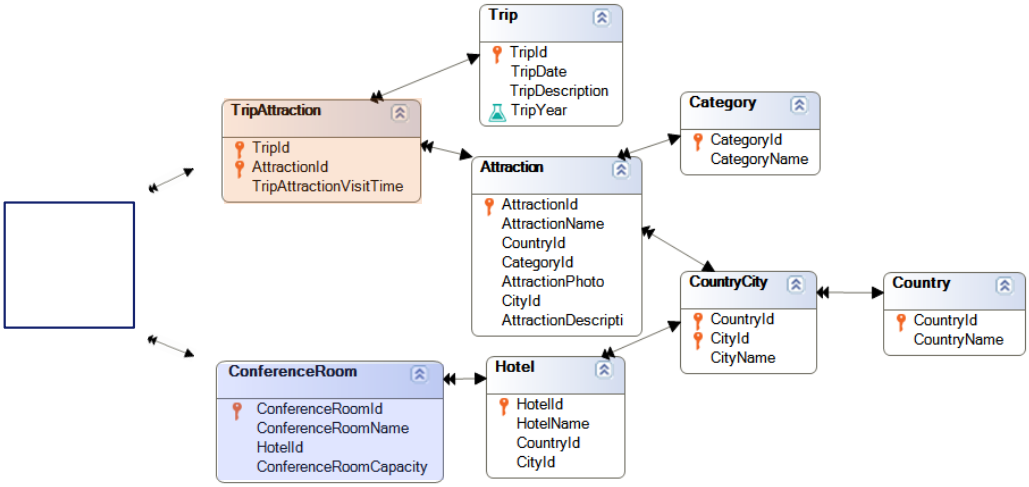




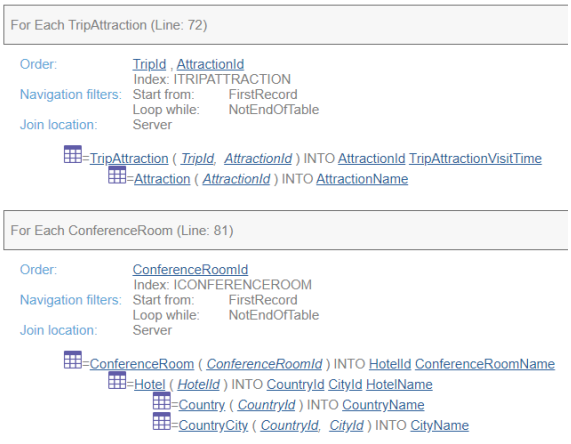

```
for each Trip.Attraction
   print PB1 //AttractionName, TripAttractionVisitTime
   for each ConferenceRoom
      print PB2 //ConferenceRoomName, HotelName, CountryName, CityName  HotelCountryName, HotelCityName
   endfor
endfor
```

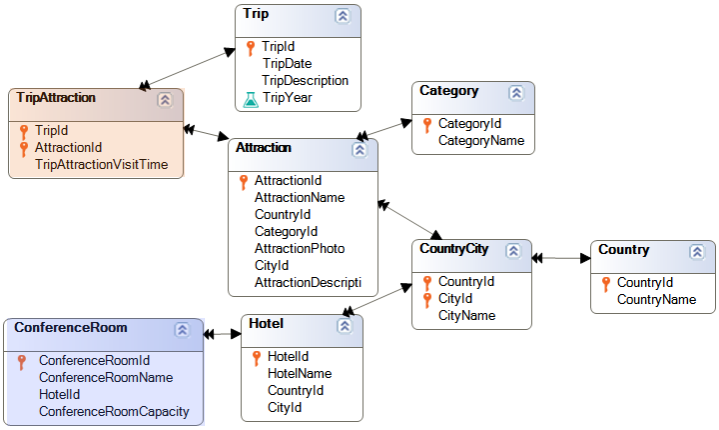
Aquí vemos el listado de navegación informando lo que buscábamos.

Pero no parece muy buena idea colocar un subtipo solo porque en un caso de for eachs anidados queremos desambiguar. Observemos que aquí no hay una ambigüedad en el modelo.



Distinto sería el caso si existiera esta tabla que introduce dos caminos para llegar a estas otras.






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

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

La forma más inteligente de resolver este problema, entonces será no modificar para anda el modelo y escribir el segundo for each en una subrutina. Y ahora el listado de navegación nos indica exactamente lo que queremos.

Aquí se está accediendo a CountryName y a CityName. Desde ConferenceRoom, y no desde TripAttraction.

JOIN

1-N relashionship

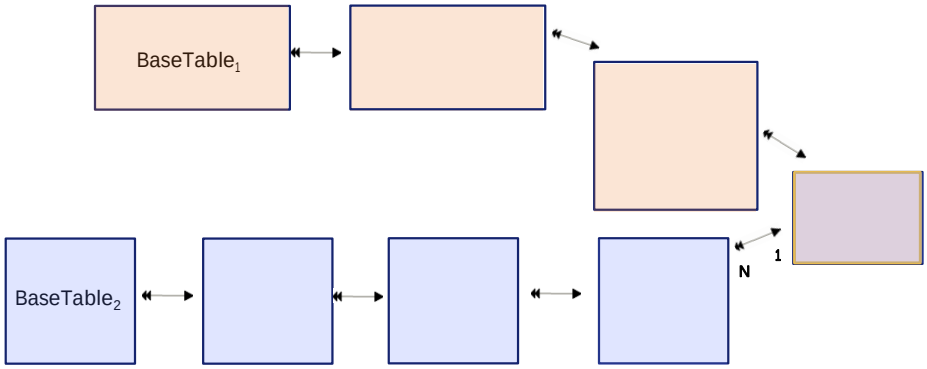
1 $BaseTable_1 \subset ext(BaseTable_2)$

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

$BaseTable_1 \neq BaseTable_2$

CARTESIAN PRODUCT

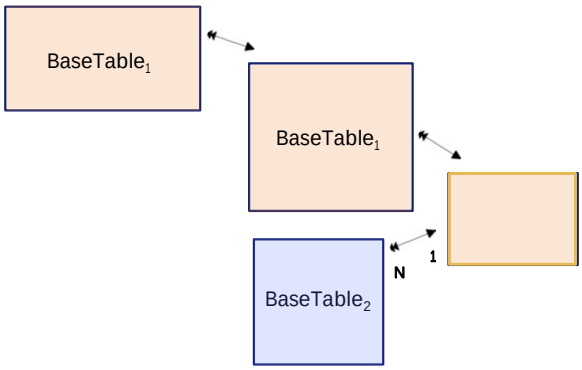
NO 1-N relashionship



Antes de continuar, volvamos a enfocarnos en el segundo caso de Join...

1 $BaseTable_1 \subset ext(BaseTable_2)$

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

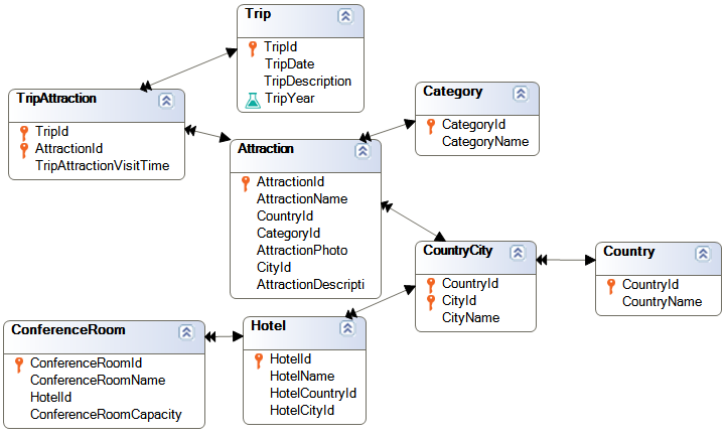


... que habíamos visto, para problematizar un instante qué pasa si la relación se da a través de subtipos. De hecho veamos el ejemplo más simple de todos.

¿Qué sucederá si esta relación no se produce utilizando los supertipos, sino subtipos?

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

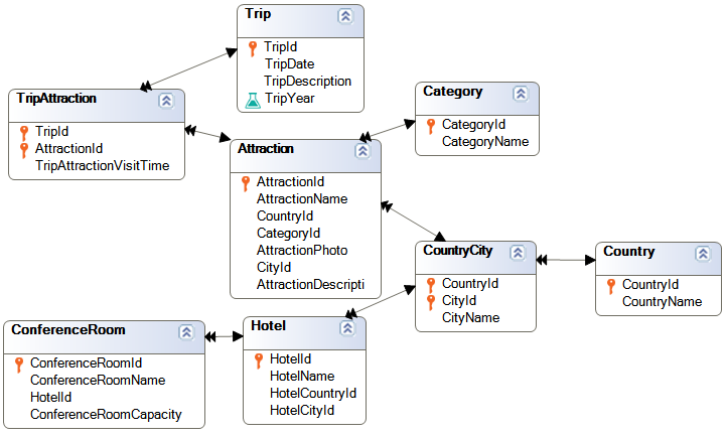
Name
Hotel
HotelId
HotelName
HotelCountryId
HotelCountryName
HotelCityId
HotelCityName




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

En el ejemplo que veíamos, si en lugar de CountryId y CityId, en Hotel colocamos estos subtipos.

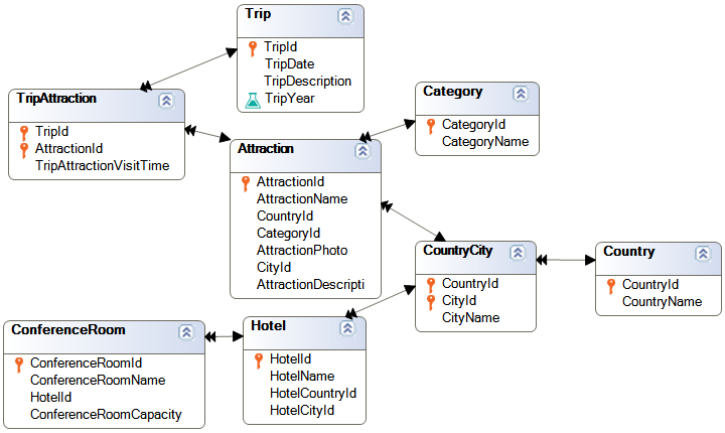
Si nuestros for eachs anidados se escriben así: es decir primero se recorre la tabla Attraction y para cada una se recorre la tabla Hotel, entonces GeneXus encuentra relación...




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

.. y realiza el join. Vemos que en el primer for each recupera los valores de CountryId y CityId, para luego en el for each anidado filtrar los registros de Hotel para los que los subtipos coincidan con éstos.

Sin embargo...




```
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
```

... si se escriben invertidos los For eachs, es decir, en el externo se recorren los hoteles y en el anidado las atracciones, no se hará un join sino un producto cartesiano.

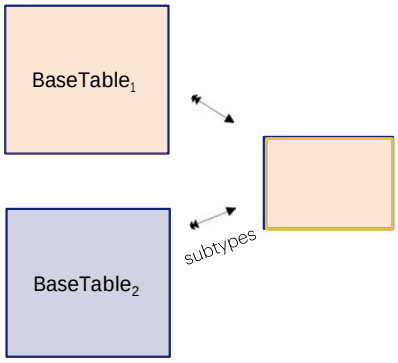
La regla es: se hace join entre supertipo y subtipo, pero no a la inversa. Es decir, como en hotel no tenemos a CountryId y CityId sino a subtipos de estos, es decir, a casos particulares, no es claro para GeneXus que el desarrollador quiera que se pase de lo particular a lo general y entonces se listen solo las atracciones con los mismos valores para los supertipos.

JOIN

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

CARTESIAN PRODUCT

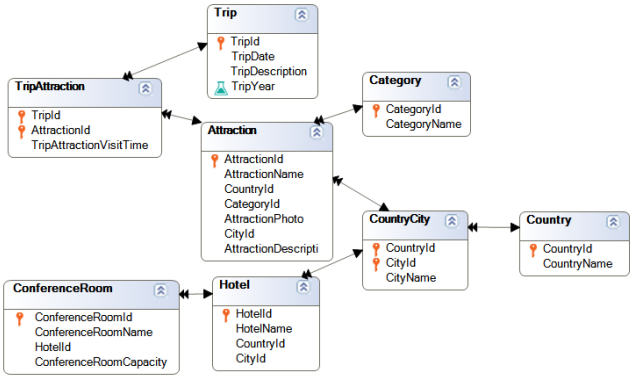
```
for each
   //SUBTYPE
   for each
      //SUPERTYPE
   endfor
endfor
```



Aquí lo vemos 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, (volvamos al caso sin subtipos) antes de pasar al caso de la misma tabla base, agreguemos que tanto en el caso de Join como en el de Producto Cartesiano pueden utilizarse en el for each anidado atributos de la tabla extendida del principal que no estén en la extendida del anidado. En los dos casos mostrados en el for each anidado se pide imprimir CategoryName que no está en al extendida de su for each. Pero sí está en la extendida del for each padre.

Por tanto para cada tripattraction se imprimen los valores de estos dos atributos, y se obtiene el valor de CategoryName, que será utilizado en el for each anidado como valor dado. Para el caso del Join también se obtienen los valores de CountryId y CityId para poder hacer el join, justamente.

Luego, en el primer caso se recorren todos los Hoteles del mismo país y ciudad (allí se utilizan esos valores recuperados de CountryId y CityId) y para cada uno se imprime el nombre de hotel y el nombre de la categoría que se había obtenido en el primer for each.

Ya para el segundo caso se recorren todas las conferencerooms (sin filtros porque no hay join) y se imprime su nombre, el nombre de su hotel y el valor de CategoryName de la atracción del for each padre.

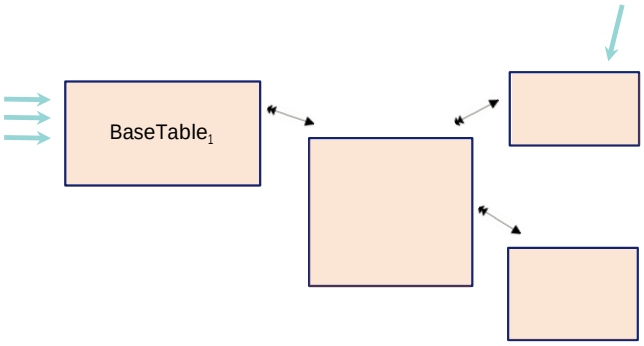
Recordemos que para determinar la tabla base del for each anidado (y es así también para resolver la navegación) se retiran primero los atributos que estando en el anidado ya formaran parte de la extendida del principal.

Es por esta razón.

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

CONTROL BREAK

$BaseTable_1 = BaseTable_2$



Bien, ahora pasemos a analizar el corte de control. Sabemos que sucede cuando la tabla base de cada for each es la misma y solo tiene sentido cuando queremos procesar información agrupada. Puede ser agrupada por cualquier atributo o conjunto de atributos de la tabla extendida.

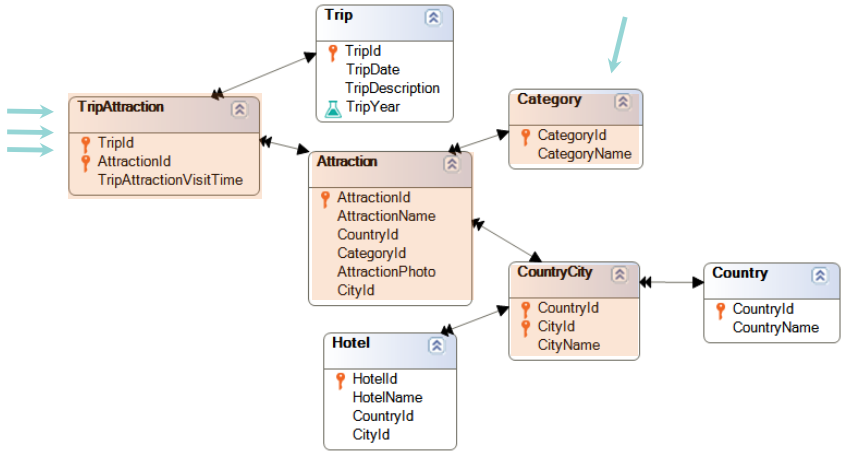
Por ejemplo, agrupamos de acuerdo al valor de un atributo de esta tabla, que deberá aparecer en la claúsula order, y procesamos todos los registros asociados de esta tabla (y la extendida) que tengan el mismo valor para ese atributo.

Así, por ejemplo...

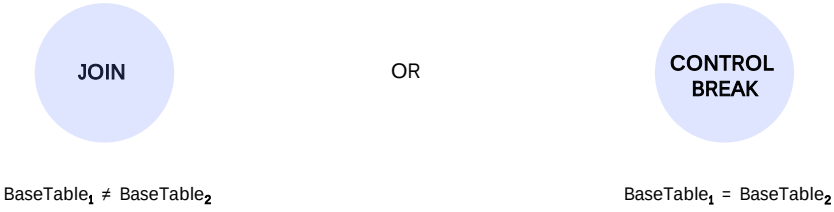
```
for each Trip.Attraction order CategoryName
   print PB1 //CategoryName
   for each Trip.Attraction
      print PB2 //AttractionName, TripAttracionVisitTime, CityName
   endfor
endfor
```

CONTROL BREAK

$BaseTable_1 = BaseTable_2$



Agrupamos las tripattractions por categoría y listamos para cada grupo el nombre de categoría y recorremos las tripattractions de esa categoría, imprimiendo el nombre de atracción, el tiempo de visita y el nombre de la ciudad de la atracción, de cada una de las tripattractions con la misma categoría.

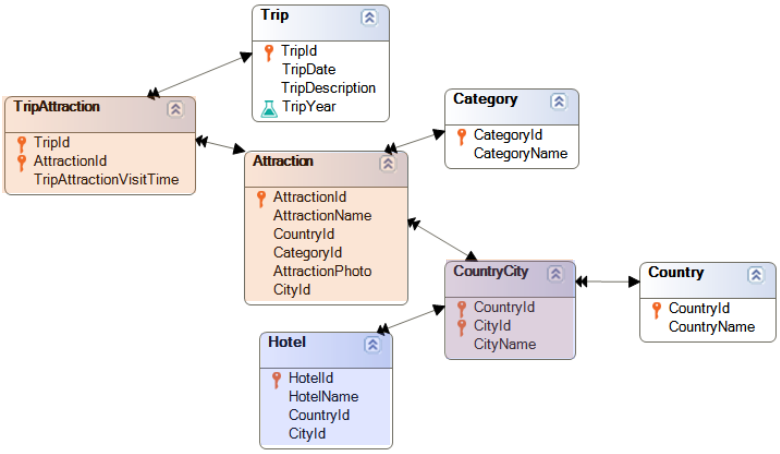


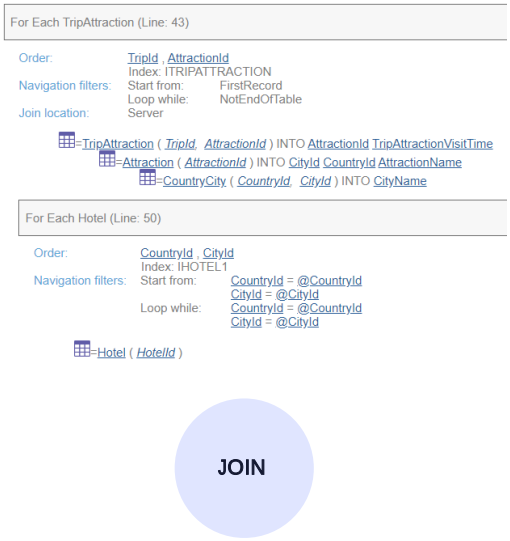
Hasta ahora no nos preocupamos en la determinación de la tabla base de cada for each porque la dábamos por dada, al utilizarse transacción base. Pero cuando dejamos esta tarea en manos de GeneXus lo que era un Join podrá pasar a ser un Corte de Control.

Lo veremos volviendo al último caso de Join que habíamos analizado.

2

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





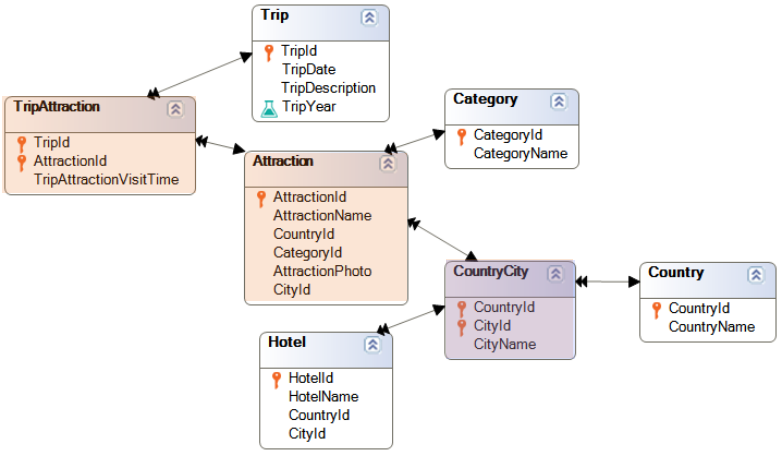
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

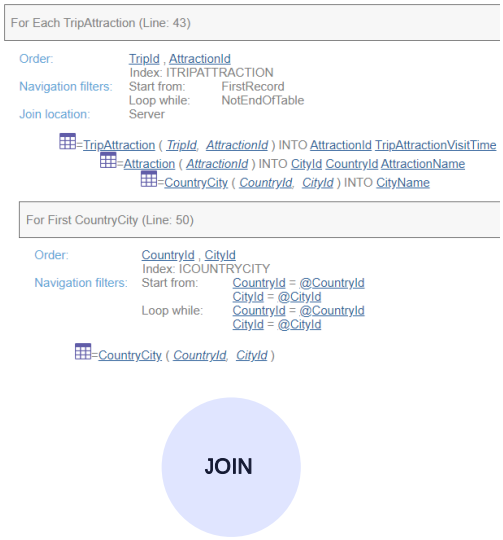
Con una mínima diferencia: estamos imprimiendo en el for each anidado, en lugar de los nombres de los hoteles con la misma ciudad de la atracción del trip, los nombres de la ciudad de cada uno de esos hoteles.

Ahora veamos que si en vez de Hotel hubiéramos especificado como transacción base Country.City, entonces esta pasa a ser la tabla base del anidado, y se tratará de un caso particular de este, pero donde el segundo for each solamente devolverá un registro.

2

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





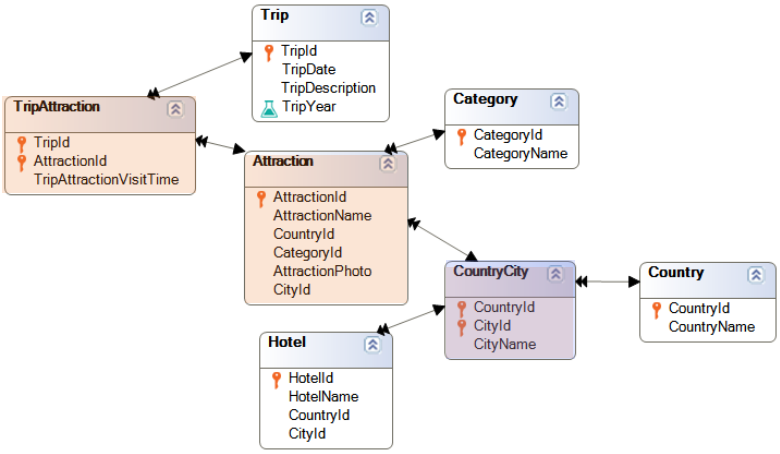
$ext(BaseTable_1) \cap BaseTable_2 \neq \phi$

Lo vemos claramente en el listado de navegación que indica **For First** en lugar de For Each. Porque, claro, CountryId, CityId son la clave primaria de la tabla.

Es como si no se hubiera especificado el for each y el printblock se hubiera impreso directamente, porque CityName está en la tabla extendida de TripAttraction.

2

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



$BaseTable_1 = BaseTable_2$

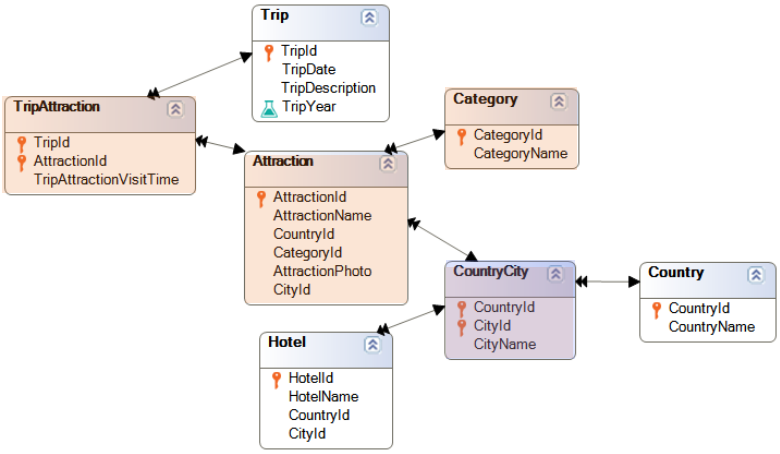
Es por ello que si no colocáramos transacción base y dejáramos a GeneXus determinar por sí solo la tabla base, elegirá TripAttraction, es decir, entenderá que se quiso implementar un corte de control, porque asumiendo que el desarrollador no escribe for eachs de más, ninguna otra cosa tendrá sentido.

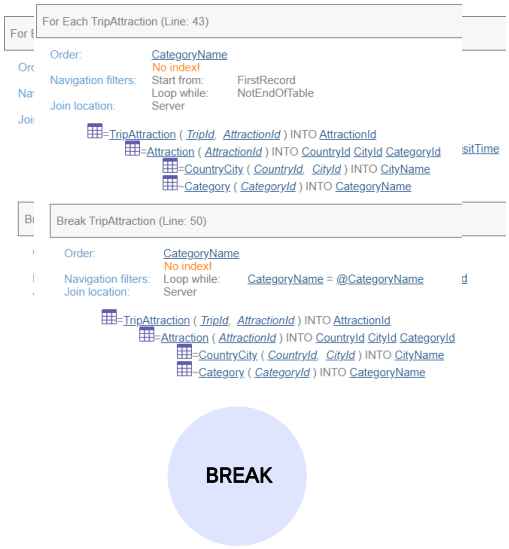
Así veremos el listado de navegación.

Sin embargo, esto tampoco tendrá sentido, porque será un corte de control por la clave primaria, o sea, se trabajará en el for each anidado con el mismo registro del for each principal cada vez.

2

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

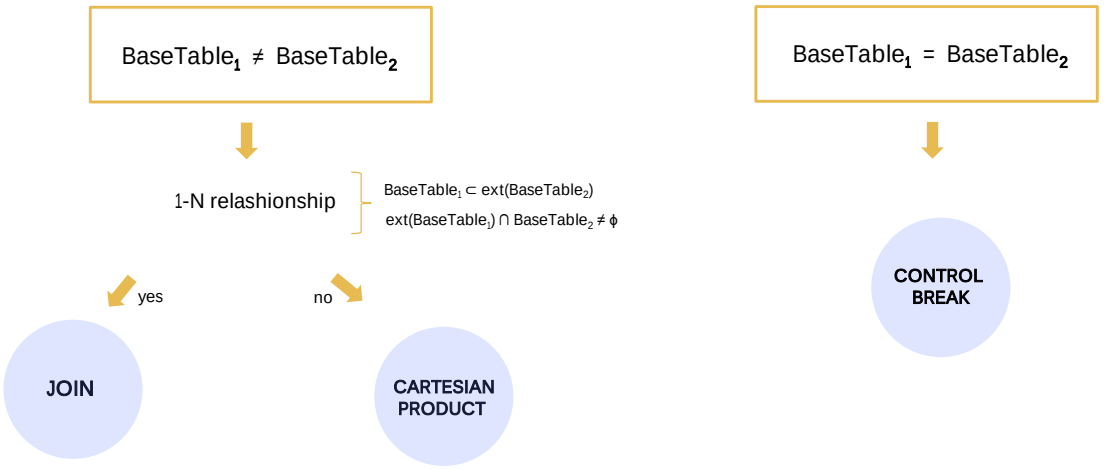




BREAK

$BaseTable_1 = BaseTable_2$

Faltaría especificar una cláusula order para cortar por algún atributo o conjunto de atributos cuyos valores puedan repetirse. Por ejemplo, CategoryName. Y así tenemos el mismo ejemplo que vimos antes de corte de control.



Resumiendo: si las tablas base son distintas, GeneXus buscará si encuentra una relación 1 a N de alguno de estos dos tipos. En caso de encontrarla, implementará un join implícito. En caso de no encontrarla no implementará ningún join, y a ese caso le llamamos producto cartesiano. Pero es importante hacer la salvedad de que llamarle producto cartesiano no significa que lo sea efectivamente. Si el desarrollador agrega explícitamente un filtro claramente se traerán los registros filtrados, por lo que habrá una suerte de join, pero no será el join automático. Decimos que el caso es producto cartesiano, entonces, solo desde el punto de vista de los filtros automáticos que determina GeneXus: es decir, en este caso, ninguno.

El corte de control es claro.

Con esto damos por concluido el análisis formal de los tres tipos de navegaciones posibles cuando tenemos for eachs anidados.

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