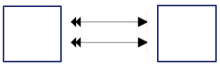
Más casos de uso de Subtipos

GeneXus™

MULTIPLE REFERENCES

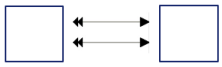
Direct



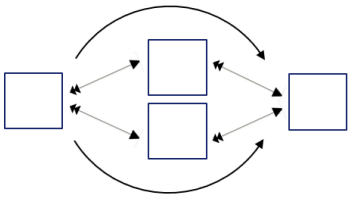
En videos anteriores hemos estudiado el caso de referencias múltiples de una tabla a otra relacionada directamente con ella...

MULTIPLE REFERENCES

Direct



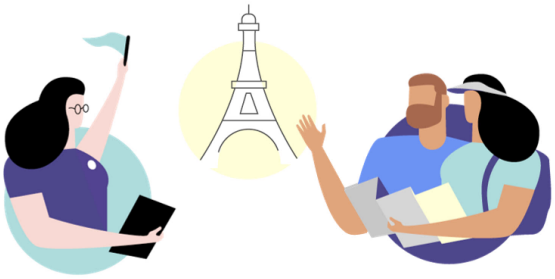
Indirect

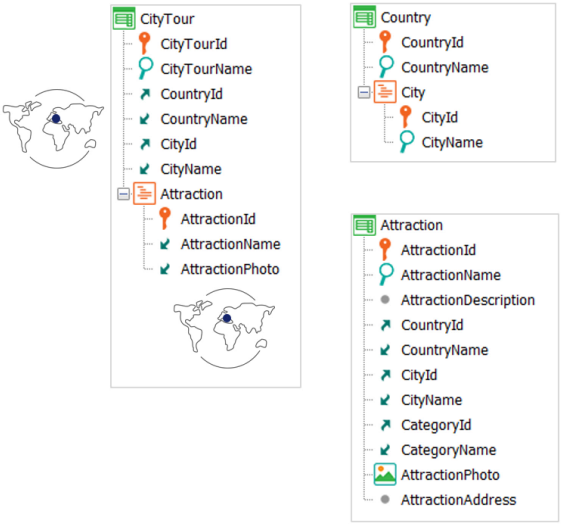


...y también el caso en que estas referencias están relacionadas de manera indirecta, ya que desde una tabla tenemos dos caminos para llegar a otra, por lo que muchas veces surge la necesidad de desambiguar utilizando subtipos.

En este video estudiaremos otro caso de referencia múltiple indirecta, sus problemas y posibles soluciones.

Indirect Multiple References

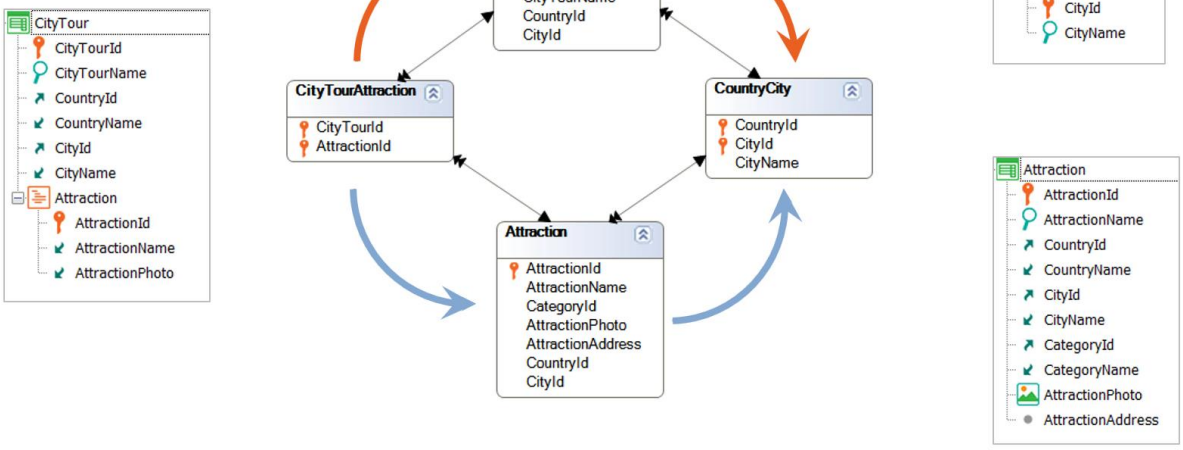




Supongamos que necesitamos registrar los tours que se ofrecen a los clientes de la agencia de viajes para visitar las diferentes atracciones turísticas de una ciudad determinada.

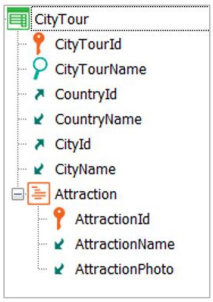
Para eso, crearemos la transacción CityTour, en donde en el primer nivel, además de registrar el nombre del tour, especificaremos su país y ciudad. El segundo nivel indicará las atracciones turísticas visitadas durante el tour.

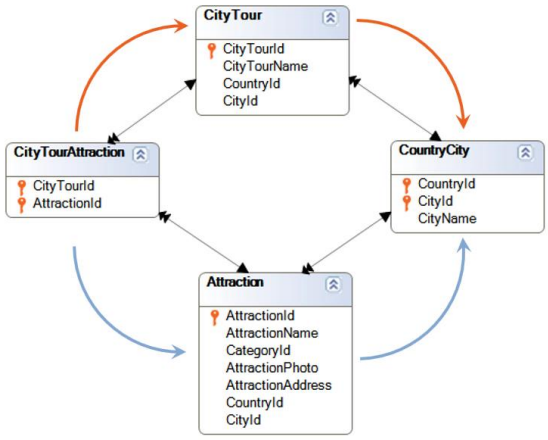
Observemos que cada atracción turística tiene definido un país y ciudad, por lo que, si no hacemos nada, el usuario podrá ingresar para un city tour una atracción que no se encuentre en el mismo país ni ciudad de los del city tour.



Es que, si miramos el diagrama de tablas, vemos claramente que tenemos dos caminos distintos para llegar desde el segundo nivel de CityTour a la tabla de ciudades. Es decir, en la tabla extendida de CityTourAttraction está la tabla de ciudades, CountryCity, pero llegamos a ella por dos caminos distintos, y no tenemos asegurado que partiendo de un registro de CityTourAttraction, la ciudad del city tour coincida con la ciudad de la atracción.

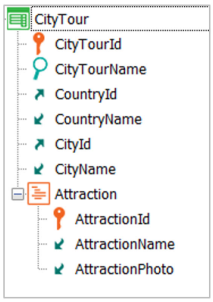
Para asegurarnos de que cuando se insertan o modifican city tours ambos caminos coincidan no es obligatorio el uso de subtipos.

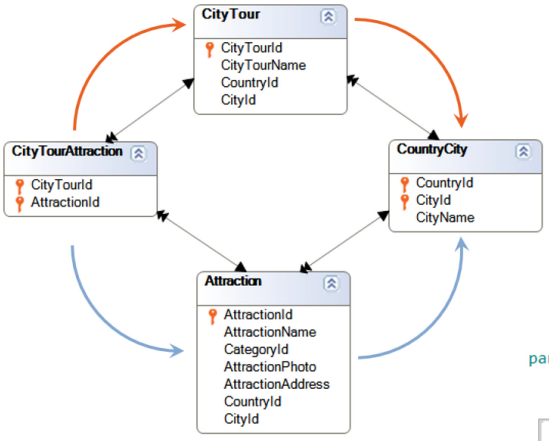




Podríamos, por ejemplo, invocar a un procedimiento en las reglas de la transacción CityTour al que le pasamos por parámetro los atributos CountryId, CityId y AttractionId, y que lo que hará será chequear que ese par CountryId, CityId, que es claramente el de CityTour coincida con el par que se encuentre al acceder al registro de Attraction de acuerdo a la atracción que se está queriendo agregar al city tour.

Aquí vemos que el procedimiento recibe en variables el id de país y de ciudad del CityTour, y en atributo el AttractionId de la línea que se está queriendo chequear. Y devolverá un booleano que indica si coinciden o no país y ciudad.






```
parm(in:&CityTourCountryId,in:&CityTourCityId,in:AttractionId,
       out:&IsOk);
```

Rules *

```
&isOk = CheckCountryCity(CountryId, CityId, AttractionId);

Error(Format("Please, choose attractions in the %1 city tour", CityName))
     if not &isOk;
```

CheckCountryCity *

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

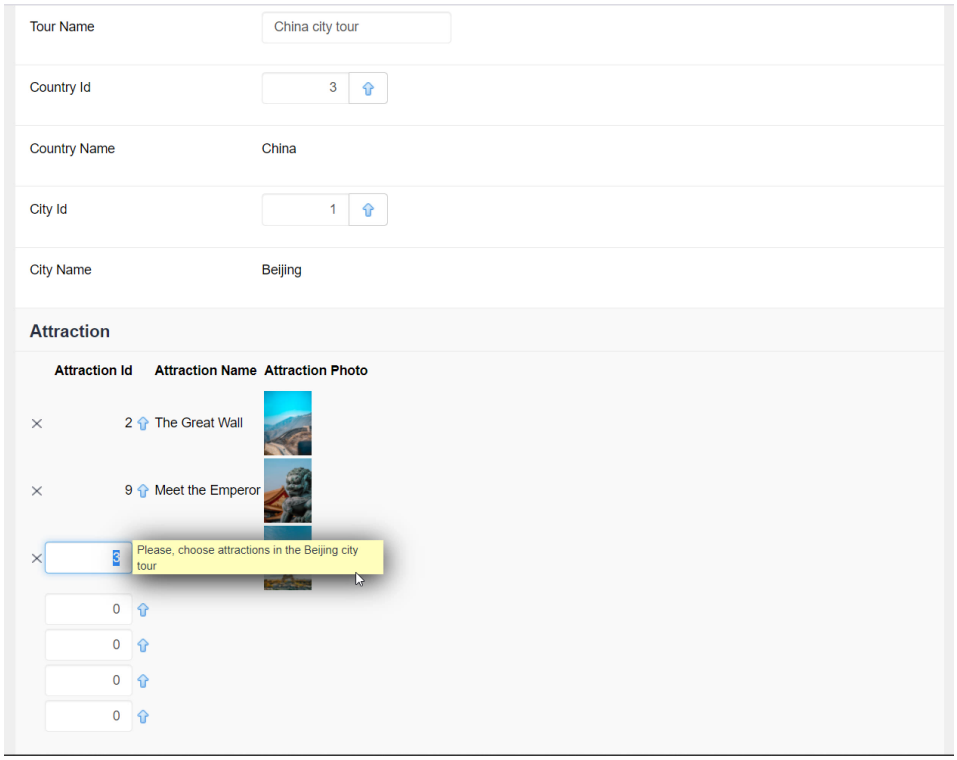
```
For each Attraction
      &AttractionCountryId = CountryId
      &AttractionCityId = CityId
endfor

&IsOk = (&CityTourCountryId = &AttractionCountryId) AND
        (&CityTourCityId = &AttractionCityId)
```

Entonces en el Source se accede a la tabla de la transacción Attraction y en dos variables nuevas se cargan el país y ciudad de **la** atracción (por el filtro automático).

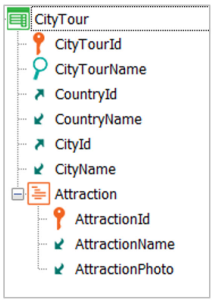
Y se devuelve en una variable booleana el valor True si el país recibido por parámetro coincide con el de la atracción y además la ciudad recibida por parámetro coincide con la de la atracción. En caso contrario, se devuelve False.

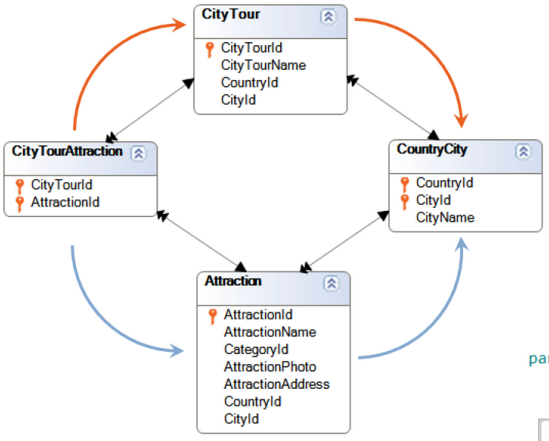
Y así es como en la transacción condicionamos la regla de error para que se dispare si el procedimiento devolvió False.



Aquí vemos esto en GeneXus.
Si ahora ejecutamos la transacción con un city tour que ya teníamos cargado y que recorre Beijing, con las dos atracciones que vemos de Beijing, y queremos agregar otra, que no es de Beijing, como la Torre Eiffel, vemos cómo se está disparando el error correctamente.

Y si en cambio agregmos una de Beijing, como la ciudad prohibida, grabamos sin problemas.






```
parm(in:&CityTourCountryId,in:&CityTourCityId,in:AttractionId,
       out:&IsOk);
```

Rules *

```
&isOk = CheckCountryCity(CountryId, CityId, AttractionId);

Error(Format("Please, choose attractions in the %1 city tour", CityName))
     if not &isOk;
```

CheckCountryCity *

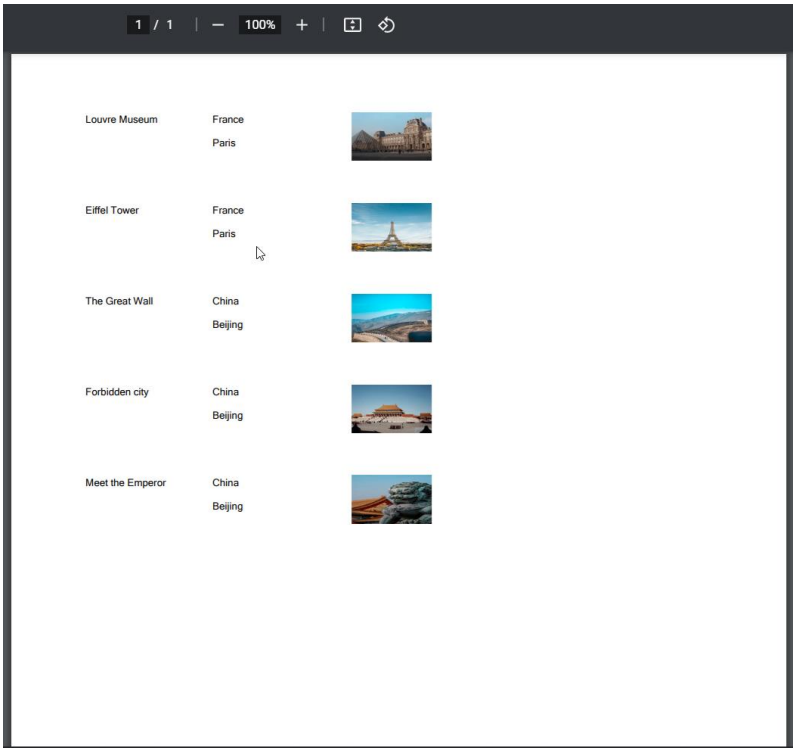
Source | Layout | Rules * | Conditions | Variables | Help | Documentation

```
For each Attraction
      &AttractionCountryId = CountryId
      &AttractionCityId = CityId
endfor

&IsOk = (&CityTourCountryId = &AttractionCountryId) AND
        (&CityTourCityId = &AttractionCityId)
```

De esta forma no tuvimos que utilizar subtipos para asegurar este chequeo a través de la transacción. Sin embargo, observemos que tuvimos que llamar a un procedimiento para poder acceder a los atributos CountryId y CityId de la atracción sin ambigüedad, cargándolos a mano en dos variables, para que no se confundan con los del CityTour.

Este problema lo vamos a tener cada vez que estemos haciendo algo con un registro de la tabla CityTourAttraction y necesitemos obtener el par país - ciudad. Porque la pregunta es, ¿cuáles de los dos? ¿El par de CityTour o el de Attraction?



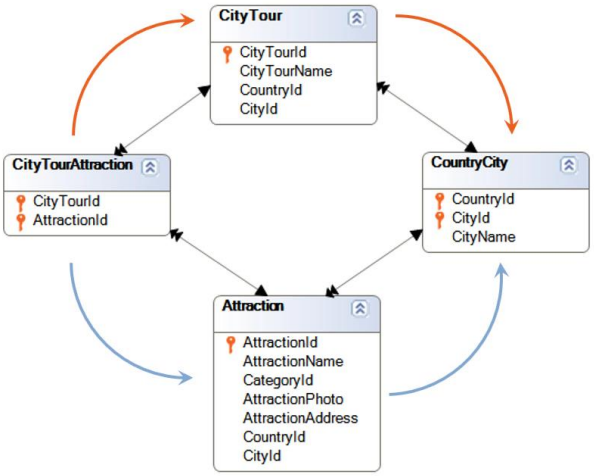
Por ejemplo, imaginemos que queremos recorrer la tabla del nivel Attraction de CityTour y mostrar el nombre de atracción, su país y ciudad y su foto. ¿Cómo sabemos si tomará los atributos CountryName y CityName a partir de los CountryId y CityId de la tabla Attraction o los tomará a partir de los de la tabla CityTour? Aquí hay una ambigüedad. Va a tomarlos de cualquiera de las dos. Para saber en concreto cuál elegió, analizamos el listado de navegación. En este caso el listado de navegación nos indica que desde CityTourAttraction va a acceder a CityTour, justamente para traer el id de país y de ciudad, y a Attraction para traer los demás atributos que queremos listar, que son la foto y el nombre. En definitiva, estamos viendo que no eligió mostrar el país y ciudad de la atracción, sino el país y ciudad del CityTour.

En este caso no importa esa ambigüedad, porque estamos chequeando cada vez que se ingresa una atracción que esos valores coincidan. Y por eso en el listado parece que estuviéramos viendo país y ciudad de la atracción y no del city tour.

Sin embargo, tan pronto como violamos ese chequeo de datos, por ejemplo yendo a la transacción Attraction y cambiando la ciudad de la Torre Eiffel por Niza en lugar de París, vemos que el listado sigue mostrando París, porque es el del cityTour donde está la torre Eiffel, y no el de la propia atracción. Es que nos permitió violar nuestra regla porque está definida únicamente en la transacción CityTour, y el cambio lo hicimos en Attraction, e incluso abriendo la transacción no salta el error porque no estamos haciendo nada con la línea.

Por lo tanto, vemos claramente que necesitaríamos chequear en todos los lugares donde estos datos pueden modificarse.

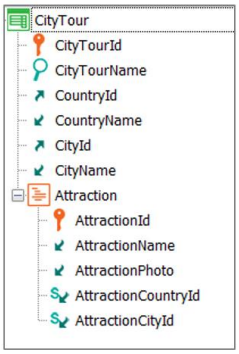
Louvre Museum	France Paris	
Eiffel Tower	France Paris	
The Great Wall	China Beijing	
Forbidden city	China Beijing	
Meet the Emperor	China Beijing	

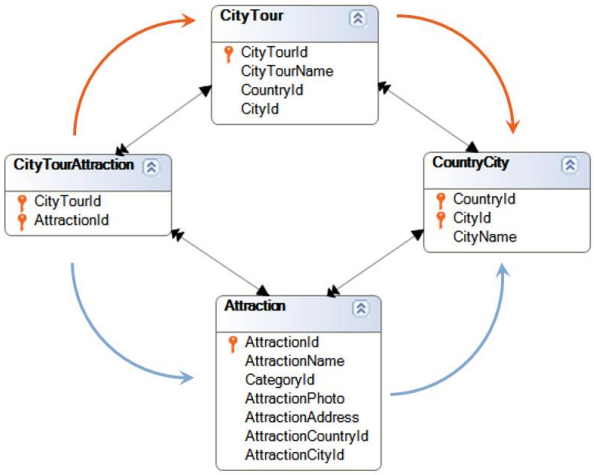


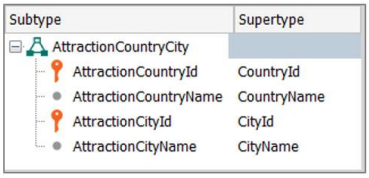
Si estamos seguros de que el chequeo se realizará en todos lados y por lo tanto los datos por ambos caminos siempre van a coincidir, entonces tal vez no nos importe el camino que elija para recuperarlos. Aunque a veces sí, por performance. En el ejemplo que vimos del listado de atracciones de los city tours, es más performante que solo tenga que acceder a Attraction para de allí ir a ContryCity y a Country, que el que tenga que ir a Attraction para recuperar su nombre y foto y a CityTour para de allí ir a CountryCity y Country.

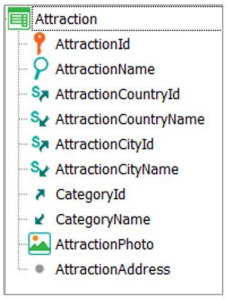
Si se nos hace necesario poder indicar en un momento dado uno de los dos caminos, porque no nos dan igual, allí aparecen los subtipos.

Analizaremos tres posibilidades, empezando por las dos más obvias, para terminar con la menos evidente.










```
Error(Format("Please, choose attractions in the %1 city tour", CityName))
     if CountryId <> AttractionCountryId or
        CityId <> AttractionCityId;
```

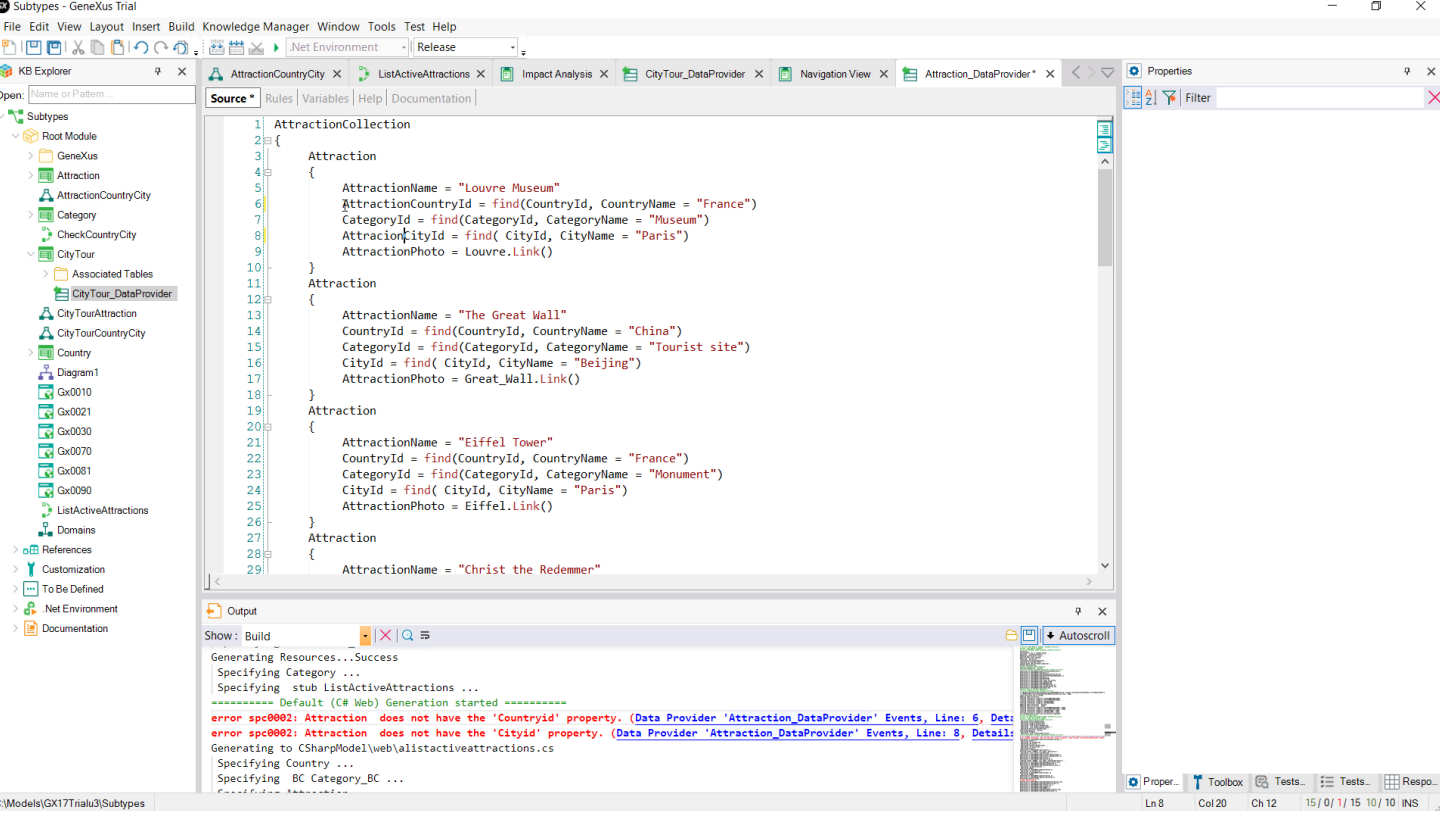
La primera será modificando el nombre de los atributos de país-ciudad en este camino, de modo de poder indentificarlo.

Así, definimos un grupo de subtipos para el país y ciudad de la atracción.

Observemos que este grupo tiene dos atributos primarios: AttractionCountryId y AttractionCityId, que corresponden a la llave primaria de la tabla CountryCity, por los supertipos que indicamos: {CountryId, CityId}.

Y que sustituimos los supertipos por estos subtipos en la transacción Attraction.

De este modo, vemos que el camino de abajo nos queda ahora identificable.
Podríamos incluso agregar a la transacción CityTour los atributos de país y ciudad inferidos a través de AttractionId, de manera de poder implementar el chequeo directamente a través de la regla error.



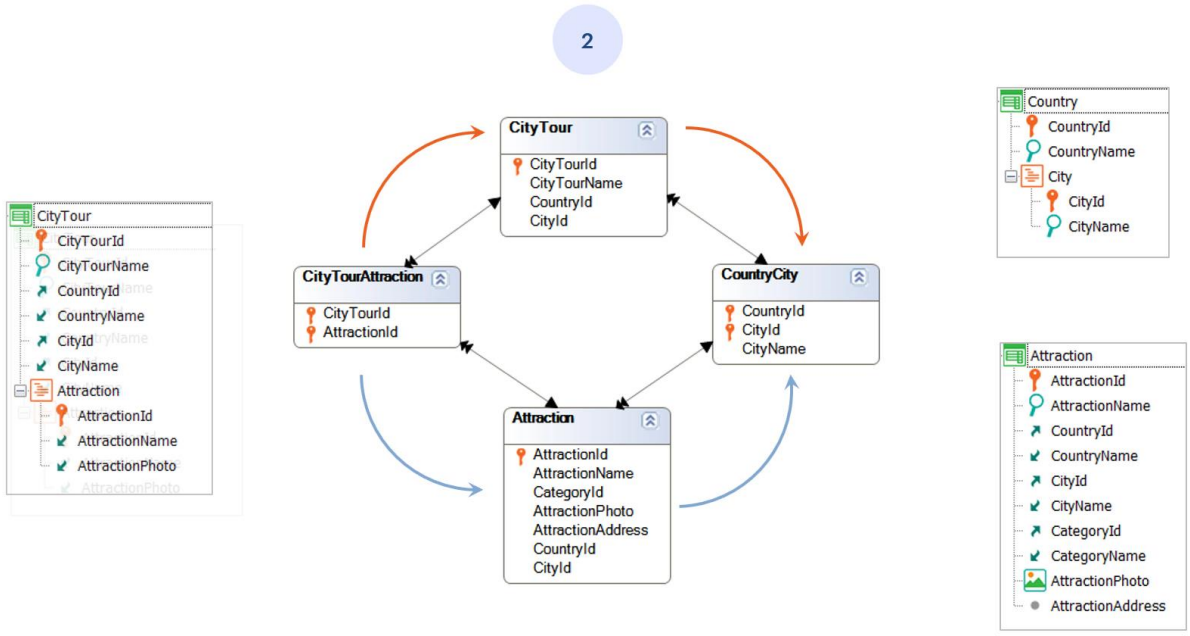
Observemos que habiendo hecho estos cambios, nos queda fácil desambiguar el listado que teníamos antes. Nos alcanza con cambiar aquí CountryName y CityName por los subtipos.

Sin embargo, como ya teníamos datos en las tablas, nos está indicando que debe reorganizar, en particular, la tabla Attraction. Debe colocar los nuevos atributos, los subtipos y eliminar los viejos, lo supertipos. Y parece que va a traspasar correctamente los datos del atributo viejo, el supertipo, al nuevo, el subtipo.
Sin embargo, al pedir que reorganice, al terminar la especificación nos encontramos con errores. En particular, vemos que nos está indicando que en el DataProvider para poblar la tabla con atracciones, Data Provider que teníamos desde antes, se está utilizando el atributo CountryId que no está más en Attraction. Y lo mismo con CityId. Aquí vemos claramente una desventaja de esta solución. Tendremos que ir uno por uno a modificar los atributos viejos por los nuevos en todos los objetos que ya accedían desde antes a la tabla Attraction.

De hecho si ya veníamos realizando todo un desarrollo de la aplicación donde CityTour aún no estaba contemplada, todavía no se nos había planteado ningún problema de ambigüedad, por lo que es de suponer que ya debíamos tener muchos otros objetos trabajando sobre CountryId y CityId en Attraction.

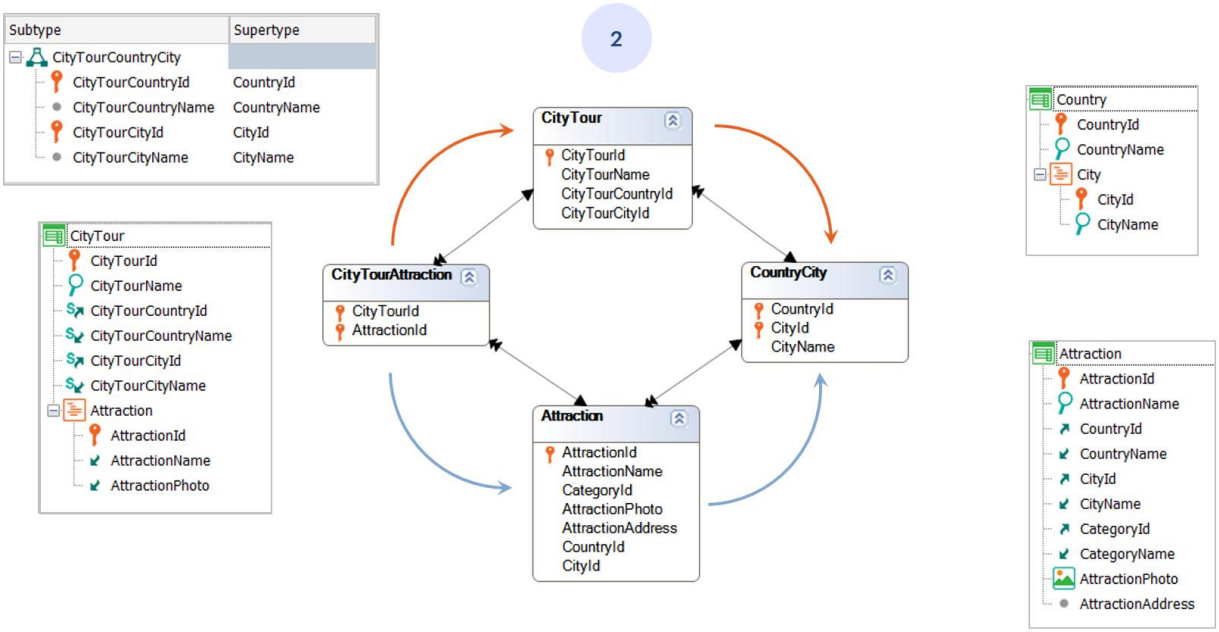
Si optamos por esta solución, tendremos que resolver todos esos escollos.

Dejando eso de lado, si ahora observamos el listado de navegación del listado que nos interesaba, ahora vemos que para obtener país y ciudad de cada registro de CityTourAttraction lo está haciendo a través de la tabla Attraction, como queríamos. Hemos desambiguado y elegido explícitamente el camino que queríamos.



Pensemos, ahora, en otra alternativa que nos complique menos en relación a objetos que ya existían.

Esta segunda solución desambigua modificando el nombre de los atributos de país-ciudad en este otro camino.

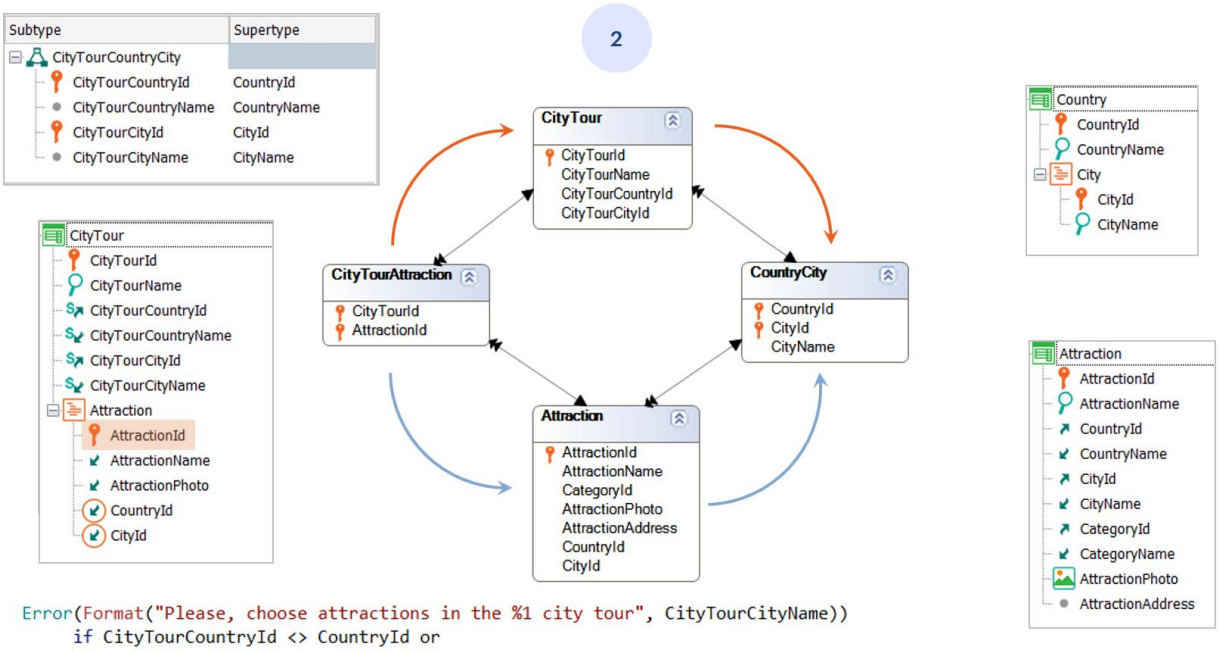


Para ello definimos el grupo de subtipos para country y city utilizándolos directamente en el cabezal de la transacción CityTour. Lo que produce este cambio en la tabla CityTour (de supertipos a subtipos).

La diferencia de esta solución con la anterior es que es menos probable que se haya construido primero la transacción plana, sin el segundo nivel –que es el que introduce los dos caminos hacia CountryCity-. Por lo que no es de esperar que existieran otros objetos navegando CityTour antes de que se nos ocurriera agregar el segundo nivel, y entonces tener que realizar el cambio de atributos por subtipos en el primero.

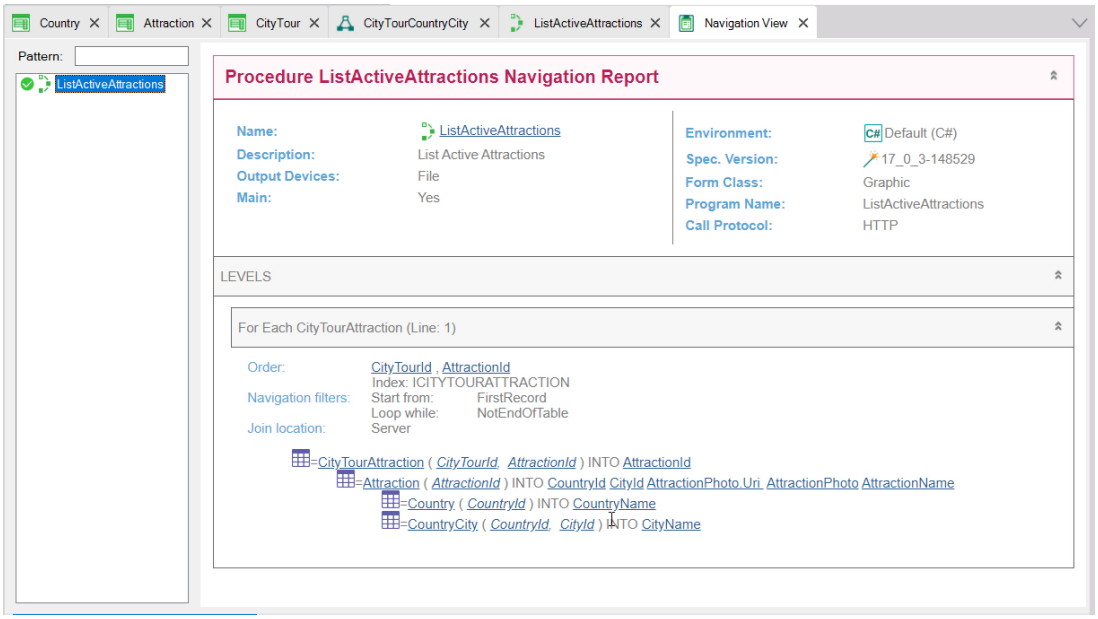
Es decir, es de esperar que las tablas CityTour y CityTourAttraction se creen a la vez, y no primero CityTour, se le carguen datos y recién después nos demos cuenta de que también vamos a necesitar un segundo nivel (que es el que genera los dos caminos y esta solución).

Con esta solución vemos que el camino de arriba nos queda ahora identificable, de manera que...

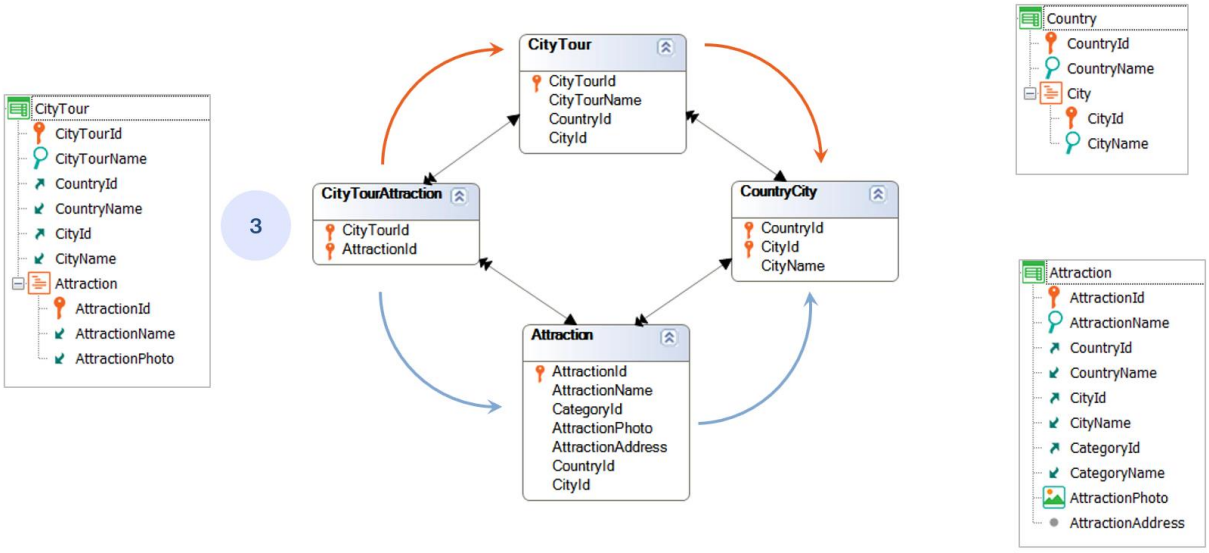



```
Error(Format("Please, choose attractions in the %1 city tour", CityTourCityName))
     if CityTourCountryId <> CountryId or
        CityTourCityId <> CityId;
```

...por ejemplo, ya podemos implementar el chequeo de que coincidan directamente a través de la regla error, sin necesidad del procedimiento.
Para lo que debemos agregar CountryId y CityId a la estructura, para poder utilizarlos en la regla. Observemos que se nos indica claramente que se están infiriendo de AttractionId.

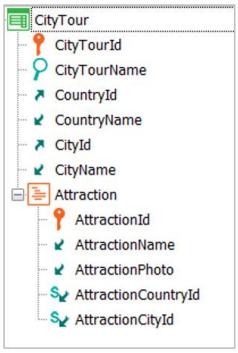


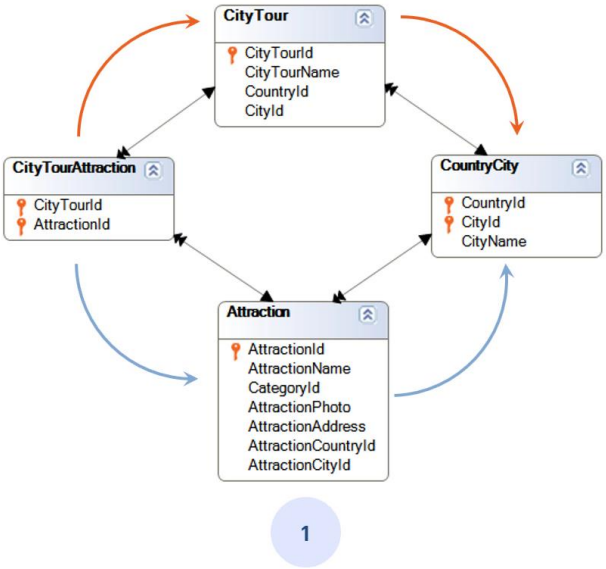
Si ahora tenemos, entonces, esta solución implementada en GeneXus, claramente si en el listado que estábamos analizando dejamos los supertipos CountryName y CityName, serán tomados de CountryId y CityId **de la tabla Attraction**. Y ya no por el otro camino, el de CityTour. Confirmémoslo con el listado de navegación. Nos muestra lo que esperábamos. Ya no hay ambigüedad.

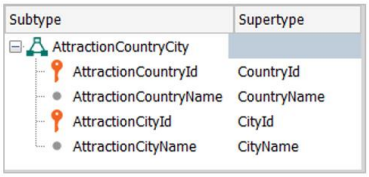


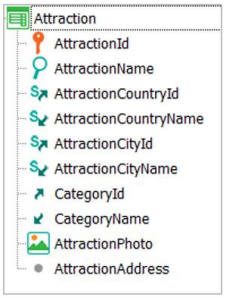
Llegamos, ahora, a la última alternativa con subtipos, que es la que resultará a priori menos intuitiva, pero que tiene una gran ventaja: provee la desambiguación en la propia tabla en la que la ambigüedad se produce. Podemos decir: la tabla que originalos dos caminos.

Es que si pensamos en las soluciones anteriores, estamos cambiando los nombres de los atributos de Country y de City en tablas que, miradas en su extendida, no tienen ningunaambigüedad.

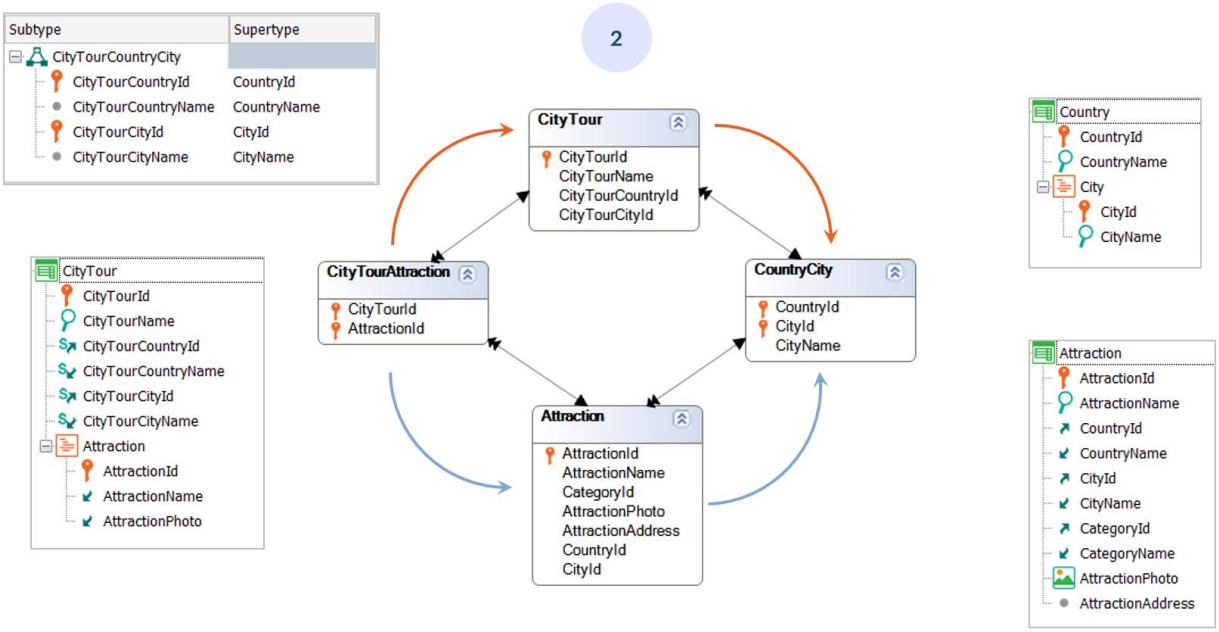






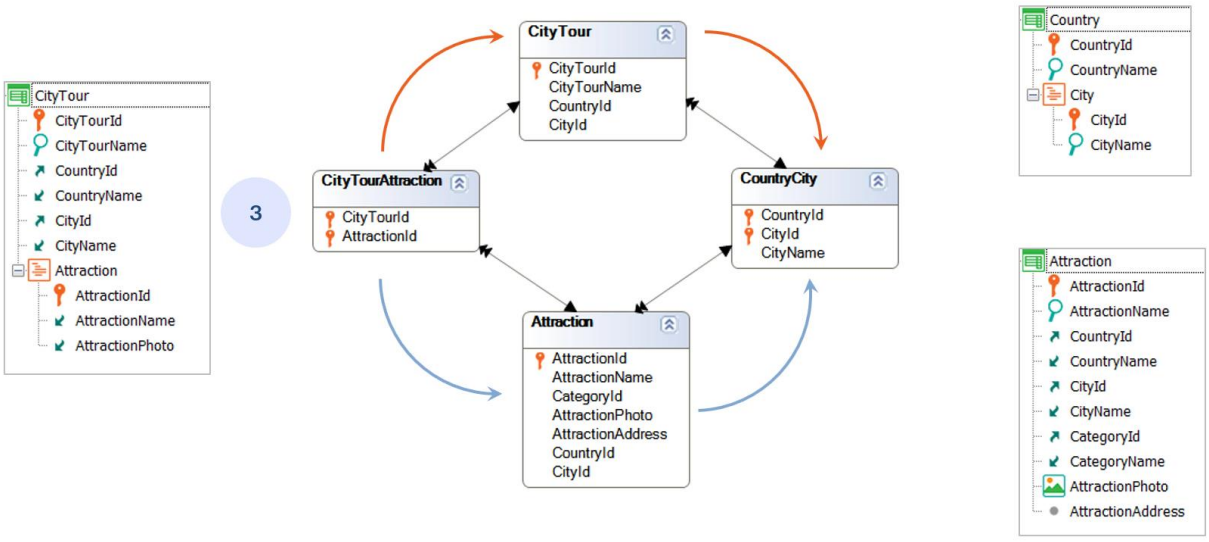


Así, si miramos la solución 1, y pensamos por ejemplo, que queremos desarrollar un Web Panel que muestre las atracciones turísticas con su información de país y ciudad, no se entiende por qué hay en esa tabla subtipos en lugar de los supertipos. Parados desde Attraction no hay ninguna necesidad de definirlos.



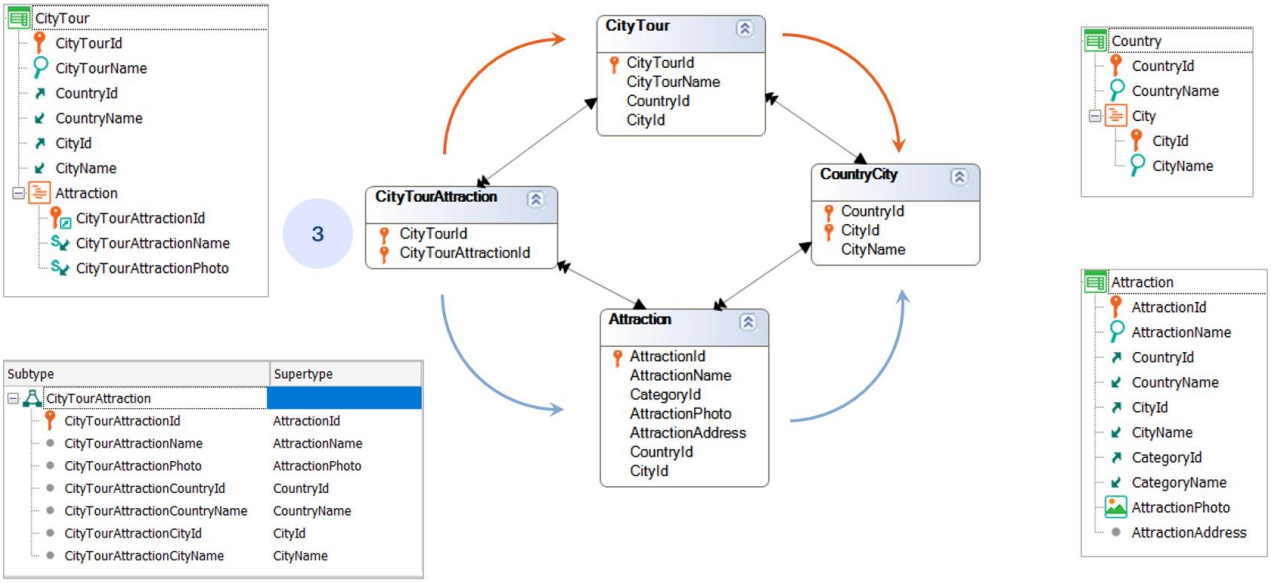
Lo mismo si miramos la solución 2 y nos paramos sobre CityTour.

Si quisiéramos listar los city tours con su país y ciudad, posicionados en esa tabla base, no se entiende, tampoco, por qué se están usando subtipos. Aunque en este caso, como la tabla CityTourAttraction proviene de un segundo nivel de la transacción que genera la tabla CityTour, están más estrechamente relacionadas y puede verse con más claridad la justificación de por qué esos subtipos.



Desambiguar en la propia tabla que ocasiona los dos caminos parece ventajoso. Al menos en el sentido en que es inherente a esta tabla la ambigüedad, por lo que allí nunca resultará injustificado que aparezca un subtipo.

Ahora, ¿cómo hacemos para desambiguar en la propia tabla que hace aparecer esos dos caminos? En el ejemplo, en la propia tabla CityTourAttraction.

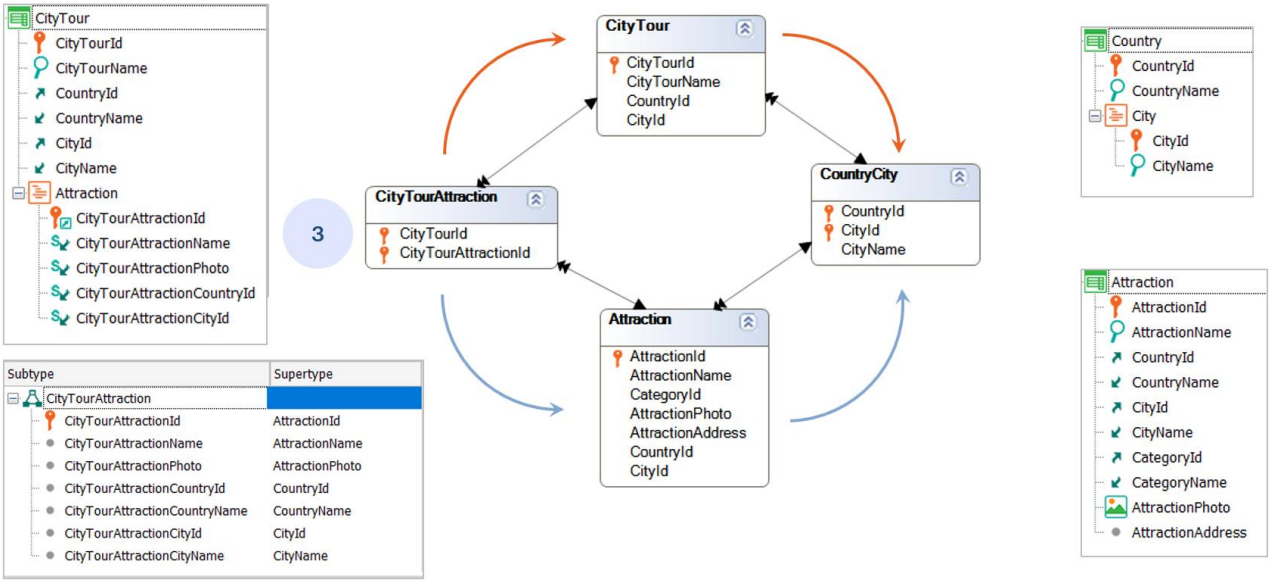


¿Y si definimos un grupo de subtipos que permitan modificar el nombre a AttractionId cuando aparezca como clave foránea, y todos los atributos que a partir de ella se infieren y nos interesan?

¿Y entonces en CityTour en lugar de utilizar el atributo AttractionId, utilizamos ese subtipo? Y, por supuesto, ahora debemos inferir nombre de atracción y foto también en subtipos de ese grupo.

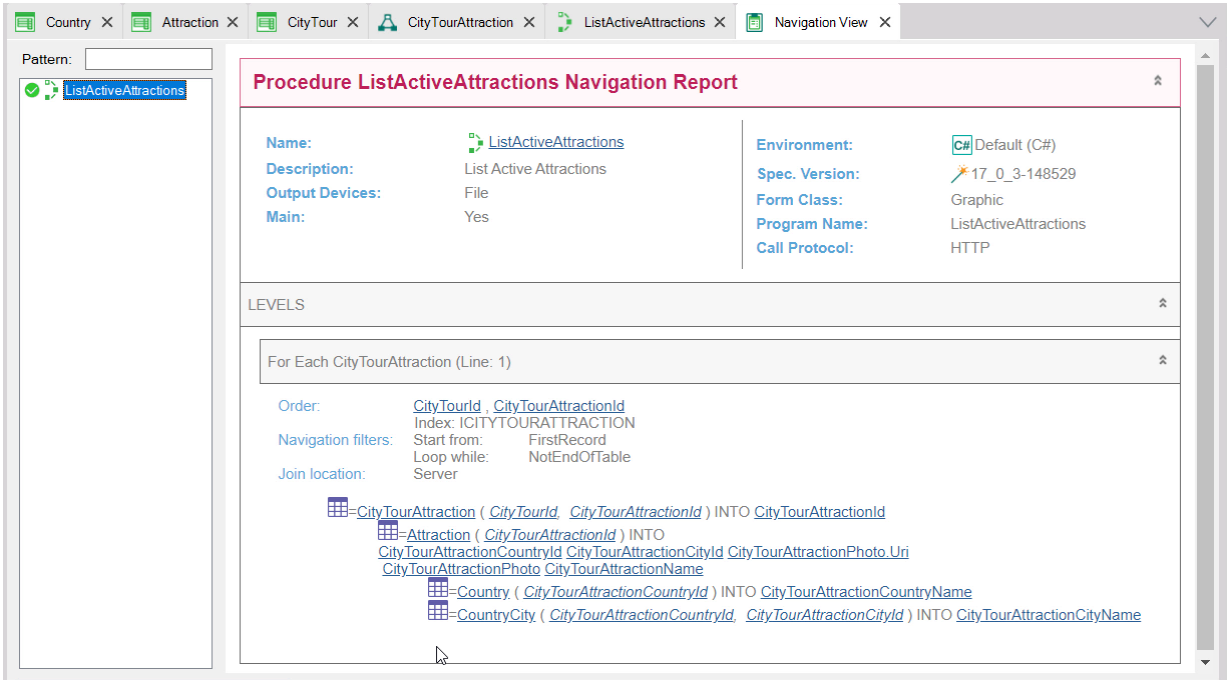
Al hacer esto el diagrama de tablas pasa a ser así. Observemos que las demás tablas no deben modificarse en absoluto, por lo que todos los objetos que estuvieran desde antes trabajando sobre esas otras tablas, lo seguirán haciendo sin ningún problema.

```
Error(Format("Please, choose attractions in the %1 city tour", CityName))
    if CountryId <> CityTourAttractionCountryId or
       CityId <> CityTourAttractionCityId;
```



Subtype	Supertype
CityTourAttraction	
CityTourAttractionId	AttractionId
CityTourAttractionName	AttractionName
CityTourAttractionPhoto	AttractionPhoto
CityTourAttractionCountryId	CountryId
CityTourAttractionCountryName	CountryName
CityTourAttractionCityId	CityId
CityTourAttractionCityName	CityName

Si ahora quisiéramos controlar que el país y ciudad del city tour sean idénticos al país y ciudad de la atracción, alcanza con agregar a la estructura de la transacción los dos atributos inferidos CityTourAttractionCountryId y CityTourAttractionCityId, y escribir la regla de error tal como la vemos.

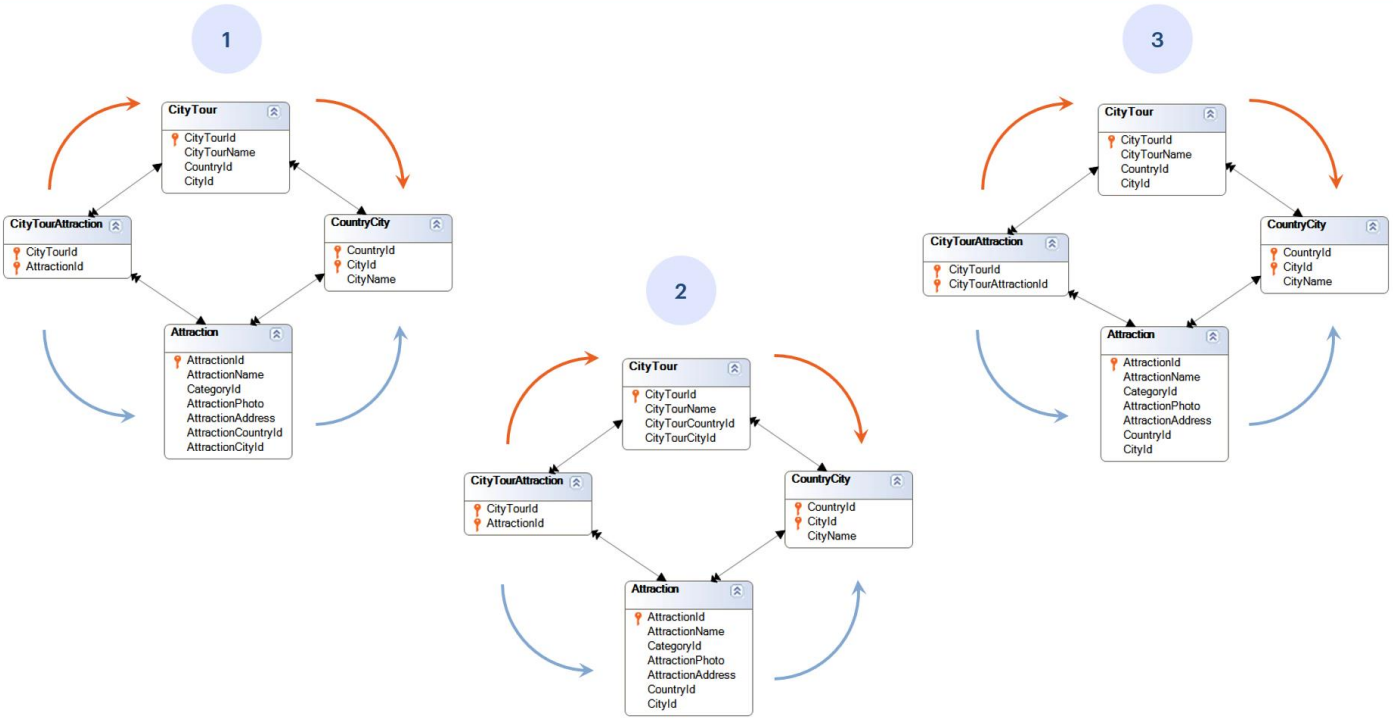


Con esta solución, el listado que estábamos buscando desambiguar desde el principio nos queda muy claro.
Recorremos con un for each el nivel Attraction de la transacción CityTour. Y aquí tenemos que modificar los atributos, que son todos inferidos de la atracción, pero que ahora no es más AttractionId, sino su subtipo. Modificamos, entonces, todos los atributos por los subtipos de estos. En particular el nombre de país, y el nombre de ciudad. Pero cuando vamos a ver el listado de navegación, nos indica que esos dos atributos, justamente, no están siendo alcanzables.

¿Por qué? Porque si bien tenemos definidos estos dos subtipos dentro del grupo, no los hemos especificado en el nivel de la transacción que estamos recorriendo con el For each. Si ahora los agregamos, lo que no tendrá ningún efecto negativo puesto que son inferidos, y volvemos a pedir la navegación de nuestro listado, encontramos que ya no hay ningún problema.

Y de hecho, vemos cómo está recuperando la información correctamente, desde el subtipo en la tabla navegada, accediendo a la tabla Attraction y de ésta a CountryCity y Country.

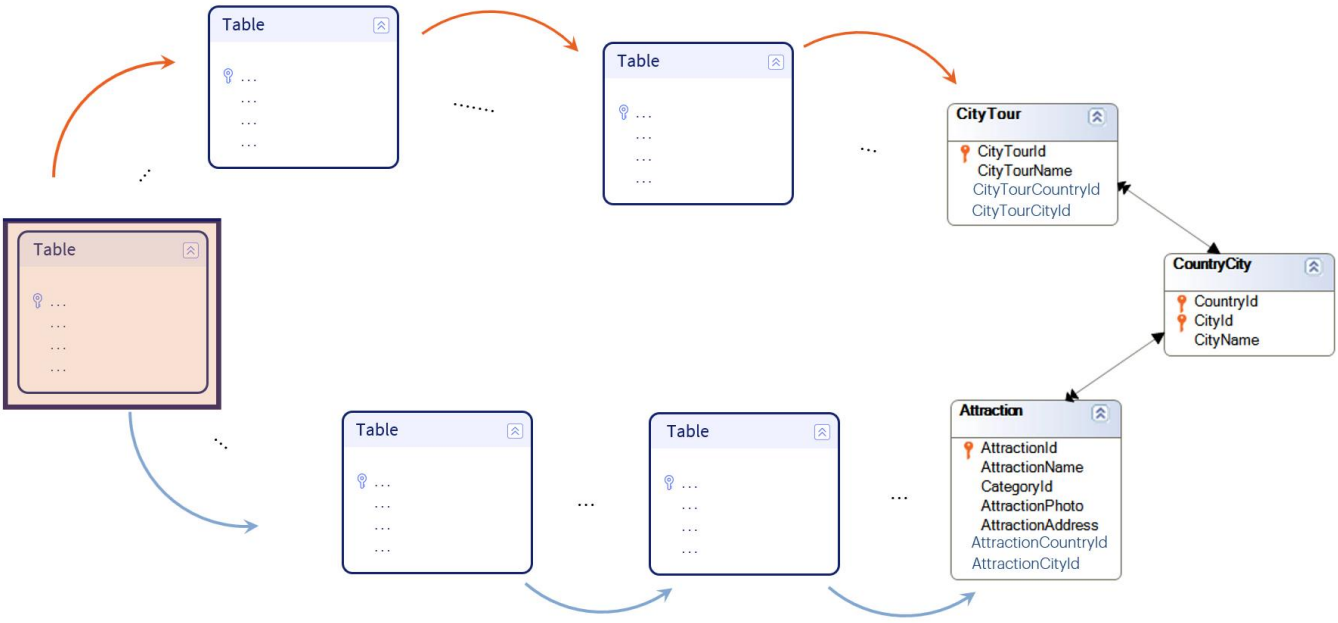
Pero esto de tener que agregar cada vez toda la información inferida que quiera utilizarse en cualquier navegación a la estructura de la transacción puede resultar algo molesto.



Para cerrar: toda solución tiene sus pros y sus contras y depende del caso particular al que se esté aplicando.

Por ejemplo, si todas las transacciones se crean juntas, entonces el problema de tener otros objetos que utilizaran de antes los atributos viejos desaparece, y las opciones 1 y 2 ya no resultan problemáticas en ese sentido. Sí en el de no justificar por sí mismas la aparición de subtipos.

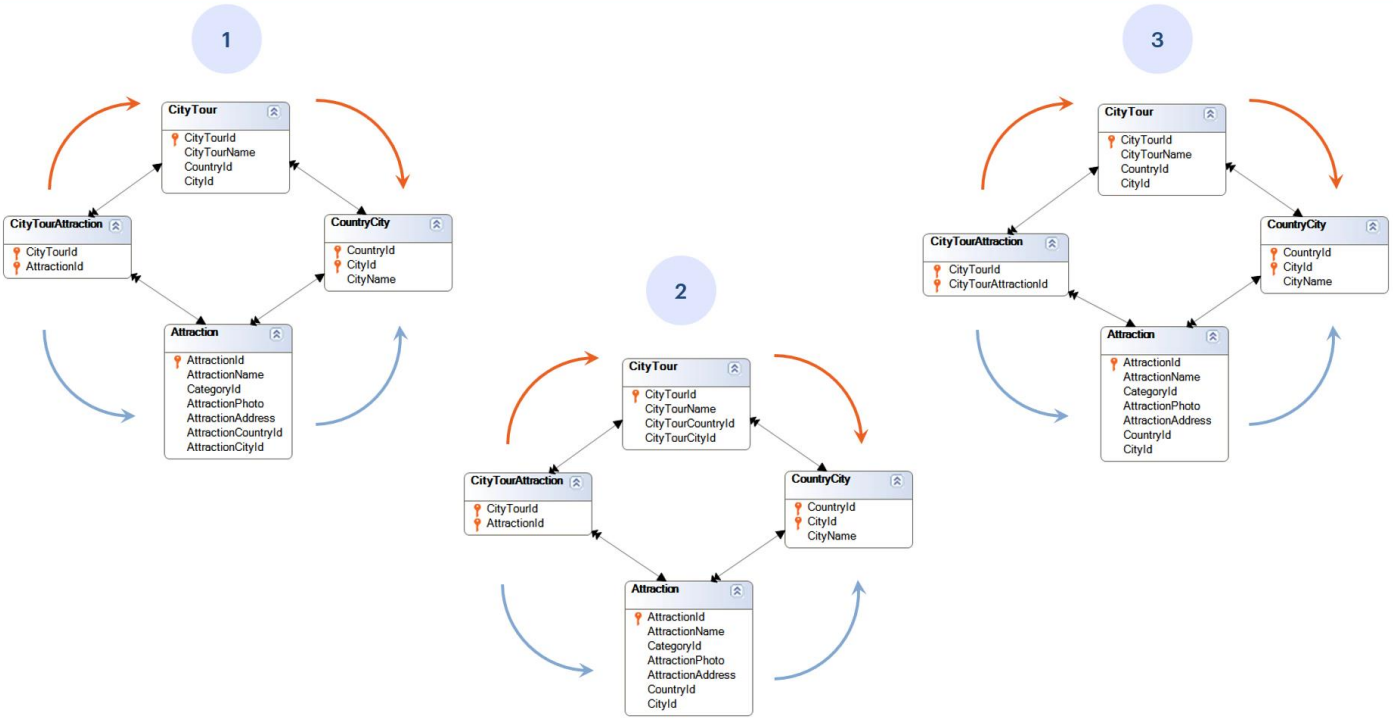
En este caso no parece muy problemático, porque, para la solución 1, parados en Attraction alcanza con mirar la tabla directamente superordinada para encontrar el por qué. Y para la solución 2, parados en CityTour, lo mismo.



Pero, si en la solución 1 estamos parados en Attraction y allí definimos subtipos para CountryId y CityId y la tabla que produce la ambigüedad de caminos estuviera muy lejos, allí la cosa se torna un poco más confusa.

O si, para la solución 2, estamos parados en CityTour, lo mismo. Ya no es tan fácil entender qué hacen allí esos subtipos.

En cambio si se construyen subtipos en la tabla de la ambigüedad, definiendo una de las claves foráneas como subtipo, allí es clarísimo. Alcanza con observar su tabla extendida para encontrar la tabla a la que se llega por varios caminos.

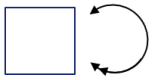


Es nuestra tercera solución. La desventaja de esta solución es que en cualquier objeto donde debamos utilizar atributos que se obtienen a partir del subtipo clave foránea, debemos agregarlos, claro, al grupo de subtipos, donde serán marcados como inferidos, pero además, debemos agregarlos a la estructura de la transacción.

Y, por supuesto, no queda claro a simple vista que la ambigüedad viene dada por Country y City. Si no analizamos la tabla extendida no sabemos por cuáles de los atributos del grupo de subtipos es que este se definió. Podría haber sido por CategoryId, por ejemplo.

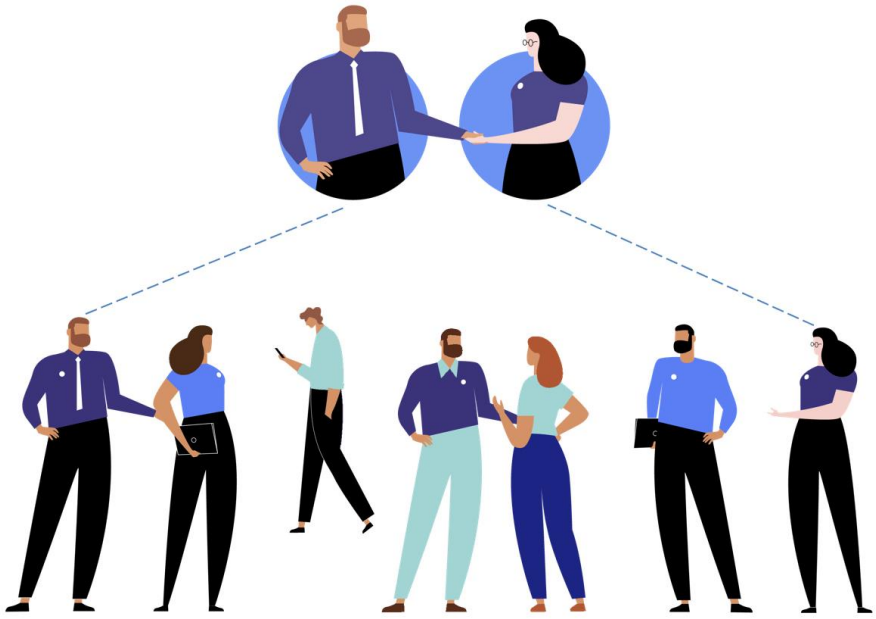
Aquí presentamos estas tres soluciones. El desarrollador elegirá la que entienda que más le conviene de acuerdo a su realidad.

RECURSIVE SUBTYPES



Veamos ahora otro caso de uso de subtipos, al que llamamos subtipos recursivos, en donde tenemos una entidad que debe auto referenciarse.

Recursive Subtypes

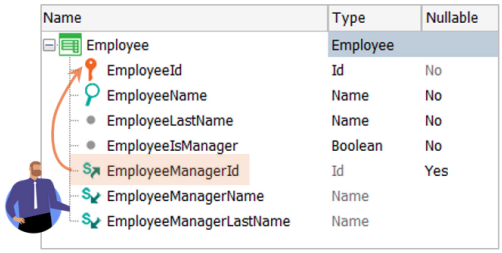


Para estudiar este caso, consideremos que estamos representando la información de los empleados de la agencia de viajes. Cada empleado puede ser, a su vez, gerente de otro u otros empleados.

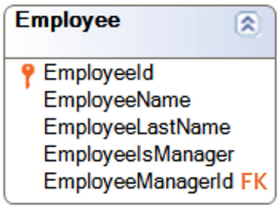
Recursive Subtypes

De todos los empleados que tienen un jefe, se necesita indicar quién es ese jefe.
Dicho jefe es, en particular, un empleado. Por lo tanto, se establece una relación en la tabla de empleados con ella misma.

Recursive Subtypes



Name	Type	Nullable
Employee	Employee	
EmployeeId	Id	No
EmployeeName	Name	No
EmployeeLastName	Name	No
EmployeeIsManager	Boolean	No
EmployeeManagerId	Id	Yes
EmployeeManagerName	Name	
EmployeeManagerLastName	Name	

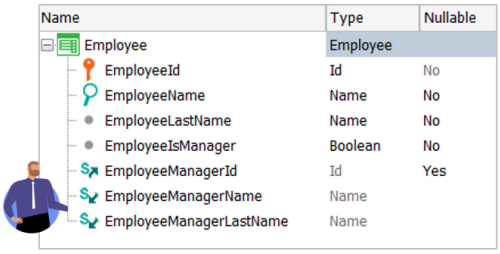


Subtype	Description	Supertype
EmployeeManager		
EmployeeManagerId	Employee Manager Id	EmployeeId
EmployeeManagerName	Employee Manager Name	EmployeeName
EmployeeManagerLastName	Employee Manager Last Name	EmployeeLastName

Para resolver esto, debemos crear un grupo de subtipos que represente la información del jefe del empleado.

El atributo EmployeeManagerId será, a todos los efectos, tomado como un EmployeeId, y por este motivo, conformará una llave foránea a la propia tabla Employee.

Recursive Subtypes



Por lo que, a la hora de ingresar la información de un empleado a través de la transacción, cuando el usuario elija un valor para el campo EmployeeManagerId, GeneXus controlará la integridad referencial, es decir, controlará que exista un registro en la tabla de empleados con ese valor para el atributo EmployeeId.

Lo invitamos a pensar situaciones reales en las que sea necesario utilizar los distintos casos de uso de subtipos que hemos visto y a realizar la implementación en GeneXus.

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