Mais casos de uso de Subtipos

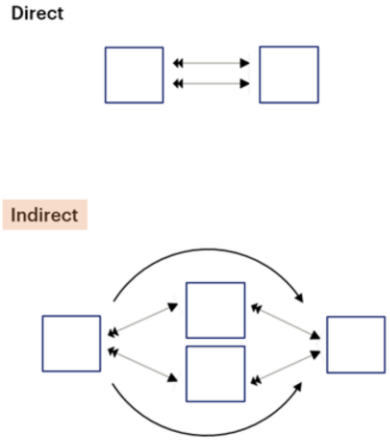
GeneXus™

MULTIPLE REFERENCES

Direct

Em vídeos anteriores, estudamos o caso de referências múltiplas de uma tabela a outra relacionada diretamente com ela...

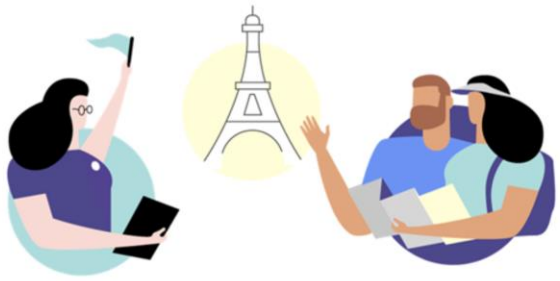
MULTIPLE REFERENCES

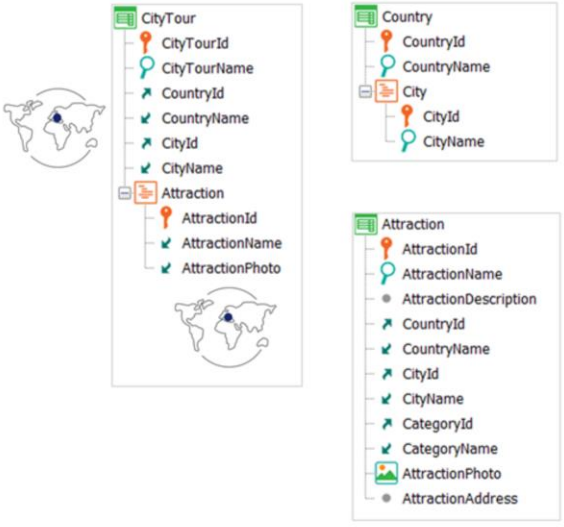


...e também o caso em que estas referências estão relacionadas de forma indireta, já que a partir de uma tabela temos dois caminhos para chegar a outra, portanto muitas vezes surge a necessidade de eliminar a ambiguidade usando subtipos.

Neste vídeo estudaremos outro caso de referência múltipla indireta, seus problemas e possíveis soluções.

Indirect Multiple References

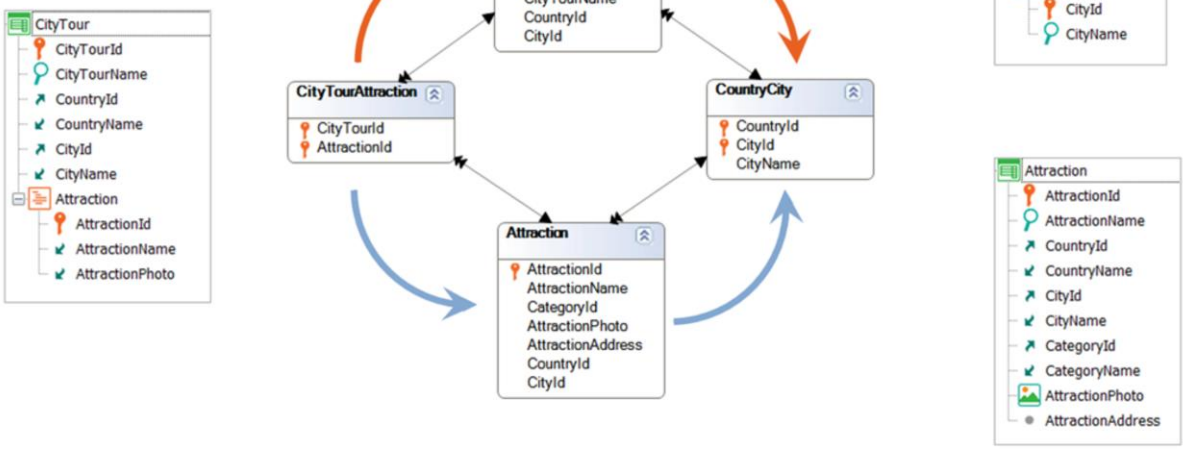




Suponhamos que precisamos registrar os tours oferecidos aos clientes da agência de viagens para visitar as diferentes atrações turísticas de uma determinada cidade.

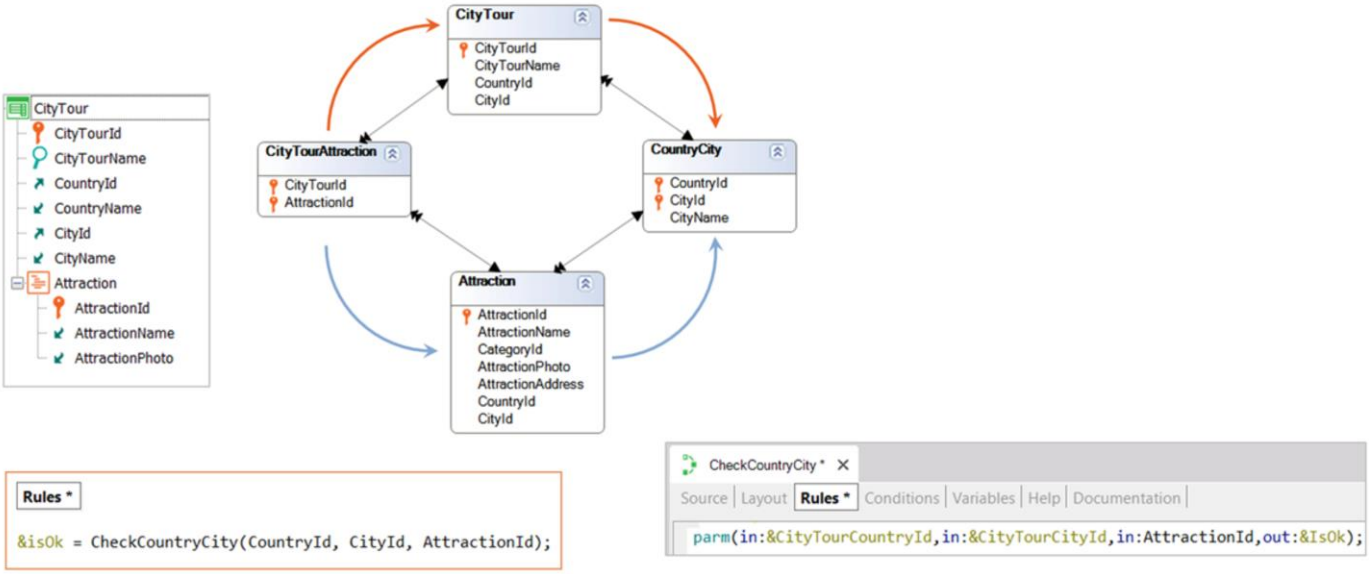
Para isso, criaremos a transação CityTour, onde no primeiro nível, além de registrar o nome do tour, especificaremos seu país e cidade. O segundo nível indicará as atrações turísticas visitadas durante o tour.

Observemos que cada atração turística tem definido um país e cidade, portanto, se não fizermos nada, o usuário poderá inserir para um city tour uma atração que não se encontre no mesmo país nem cidade que os do city tour.



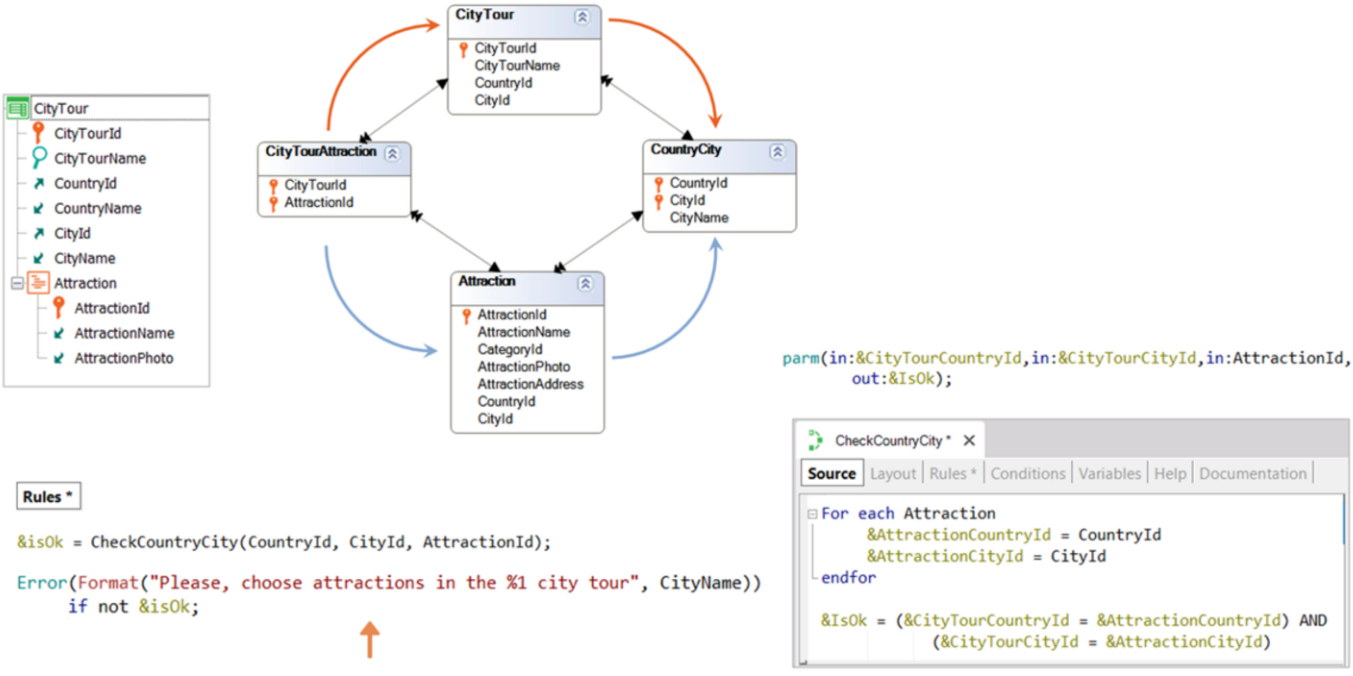
É que, se olhamos para o diagrama de tabelas, vemos claramente que temos dois caminhos diferentes para chegar do segundo nível de CityTour à tabela de cidades. Ou seja, na tabela estendida de CityTourAttraction está a tabela de cidades, CountryCity, mas chegamos a ela por dois caminhos diferentes, e não temos certeza de que partindo de um registro de CityTourAttraction, a cidade do city tour coincide com a cidade da atração.

Para garantir que quando são inseridos ou modificados city tours, ambos os caminhos coincidam, não é obrigatório o uso de subtipos.



Poderíamos, por exemplo, invocar um procedimento nas regras da transação CityTour para o qual passamos por parâmetro os atributos CountryId, CityId e AttractionId, e o que ele fará será verificar se esse par CountryId, CityId, que é claramente o de CityTour coincide com o par encontrado ao acessar o registro de Attraction de acordo com a atração que se deseja adicionar ao city tour.

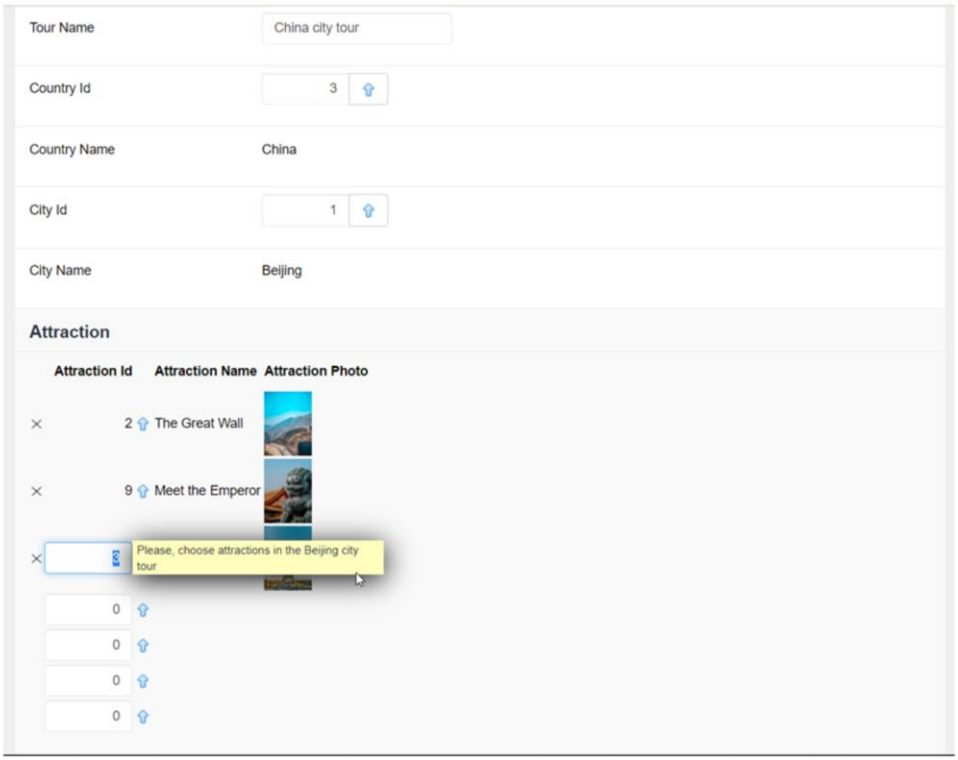
Aqui vemos que o procedimento recebe em variáveis o id de país e de cidade do CityTour, e em atributo o AttractionId da linha que se está querendo verificar. E retornará um booleano que indica se coincidem ou não, o país e cidade.



Então, no Source, é acessada a tabela da transação Attraction e em duas variáveis novas são carregados o país e cidade **da** atração (pelo filtro automático).

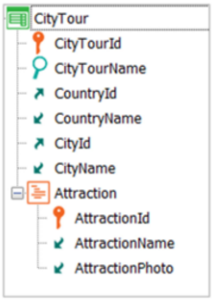
E é retornado em uma variável booleana o valor True se o país recebido por parâmetro coincide com o da atração e também a cidade recebida por parâmetro coincide com a da atração. Caso contrário, é retornado False.

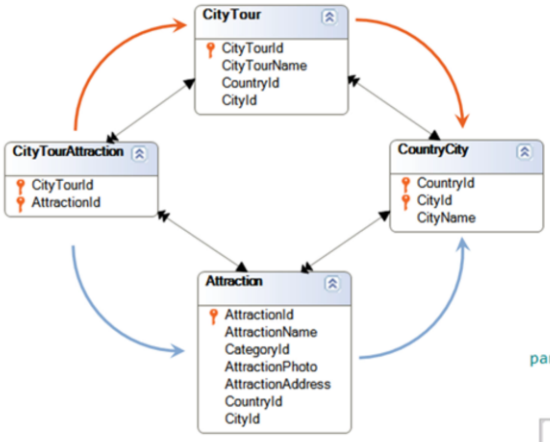
E é assim que na transação condicionamos a regra de erro para que se dispare se o procedimento retornou False.



Aqui vemos isto em GeneXus.
Se agora executamos a transação com um city tour que já carregamos e que percorre Beijing, com as duas atrações que vemos de Beijing, e queremos adicionar outra, que não é de Beijing, como a Torre Eiffel, vemos como está sendo disparado o erro corretamente.

E se em vez disso adicionamos uma de Beijing, como a cidade proibida, gravamos sem 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 *

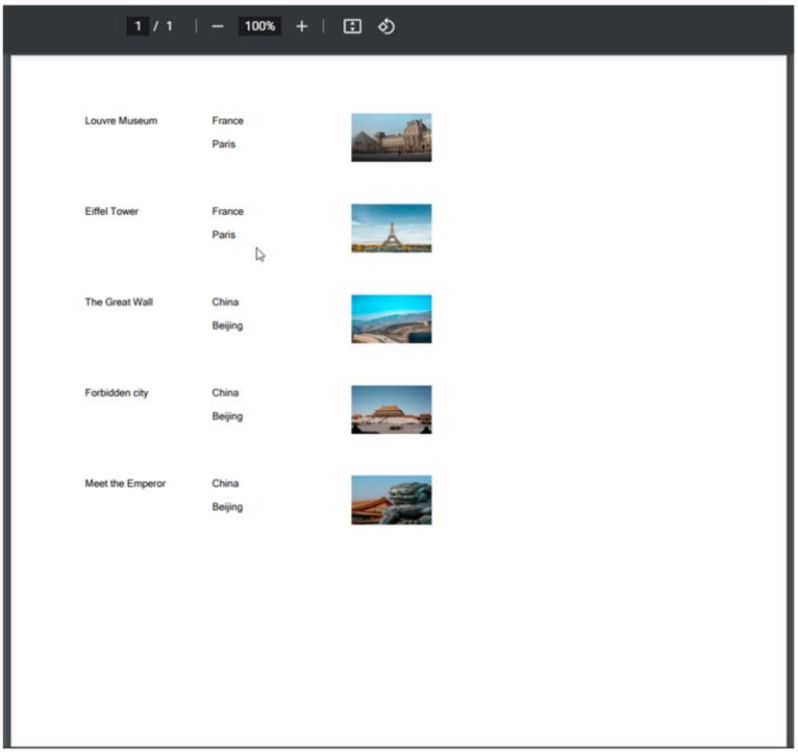
Source | Layout | Rules * | Conditions | Variables | Help | Documentation

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

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

Desta forma, não tivemos que utilizar subtipos para garantir esta verificação através da transação. Porém, observemos que tivemos que chamar um procedimento para poder acessar os atributos CountryId e CityId da atração sem ambiguidade, carregando-os manualmente em duas variáveis, para que não sejam confundidos com os do CityTour.

Teremos este problema toda vez que fizermos algo com um registro da tabela CityTourAttraction e precisarmos obter o par país - cidade. Porque a questão é, qual dos dois? O par de CityTour ou o de Attraction?

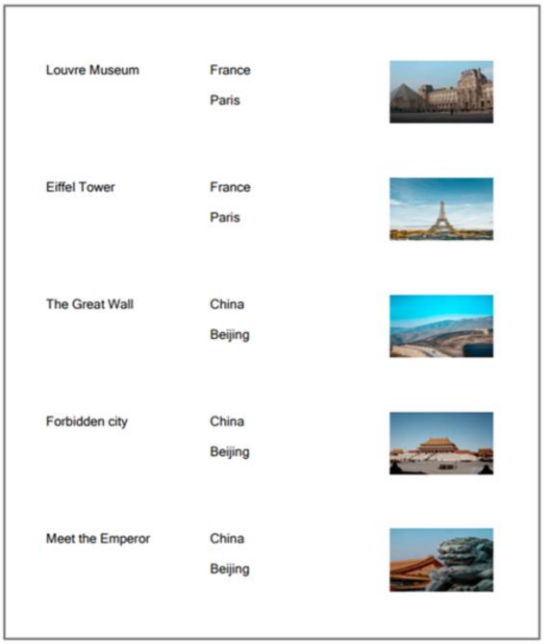


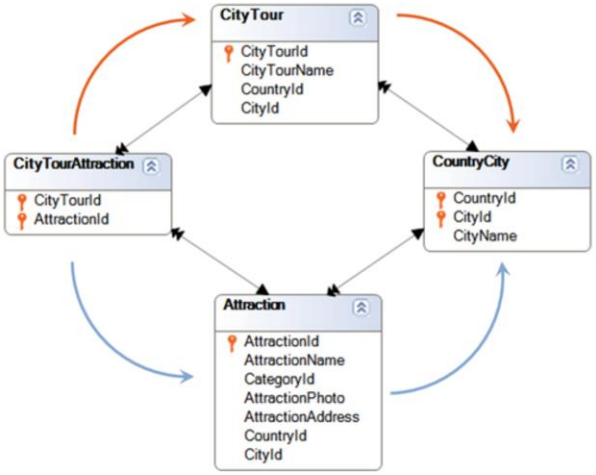
Por exemplo, imaginemos que queremos percorrer a tabela do nível Attraction de CityTour e mostrar o nome da atração, seu país e cidade, e sua foto. Como sabemos se pegará os atributos CountryName e CityName a partir dos CountryId e CityId da tabela Attraction ou os pegará a partir daqueles da tabela CityTour? Aqui existe uma ambiguidade. Irá pegá-los de qualquer uma das duas. Para saber especificamente qual escolheu, analisamos a lista de navegação. Neste caso, a lista de navegação nos indica que a partir de CityTourAttraction acessará CityTour, justamente para trazer o id de país e de cidade, e Attraction para trazer os demais atributos que queremos listar, que são a foto e o nome. Portanto, estamos vendo que não optou por mostrar o país e cidade da atração, mas sim o país e cidade do CityTour.

Neste caso não importa essa ambiguidade, porque estamos verificando cada vez que é inserida uma atração se esses valores coincidem. E por isso na lista parece que vemos país e cidade da atração e não do city tour.

No entanto, assim que violamos essa verificação de dados, por exemplo, indo para a transação Attraction e mudando a cidade da Torre Eiffel para Nice em vez de Paris, vemos que a lista continua mostrando Paris, porque é a do cityTour onde está a torre Eiffel, e não aquela da própria atração. É que nos permitiu violar nossa regra porque está definida apenas na transação CityTour, e fizemos a alteração em Attraction, e mesmo abrindo a transação não mostra o erro porque não estamos fazendo nada com a linha.

Portanto, vemos claramente que precisaríamos verificar em todos os lugares onde estes dados podem ser modificados.

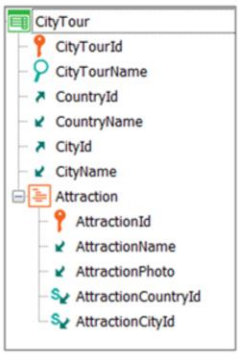


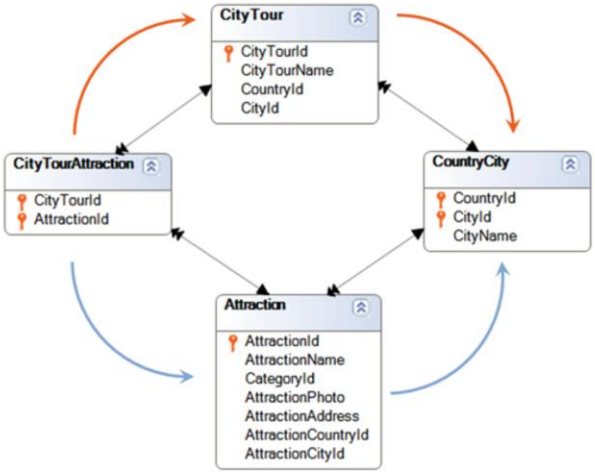


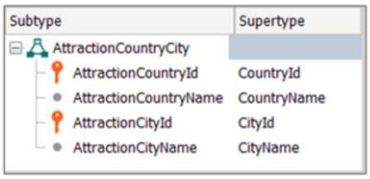
Se tivermos certeza de que a verificação será realizada em todos os lugares e, portanto, os dados em ambos os caminhos sempre coincidirão, então talvez não nos importe o caminho que escolha para recuperá-los. Embora às vezes sim, por desempenho. No exemplo que vimos da lista de atrações dos city tours, é mais performático que só tenha que acessar Attraction para dali ir a CountryCity e a Country, do que tenha que ir a Attraction para recuperar seu nome e foto, e a CityTour para de lá ir a CountryCity e Country.

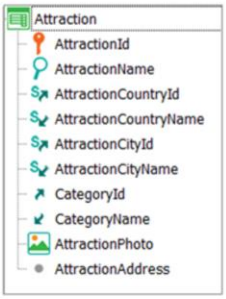
Se for necessário que possamos indicar, em um determinado momento, um dos dois caminhos, porque não nos resultam iguais, aí aparecem os subtipos.

Analisaremos três possibilidades, começando pelas duas mais óbvias, para terminar com a menos evidente.










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

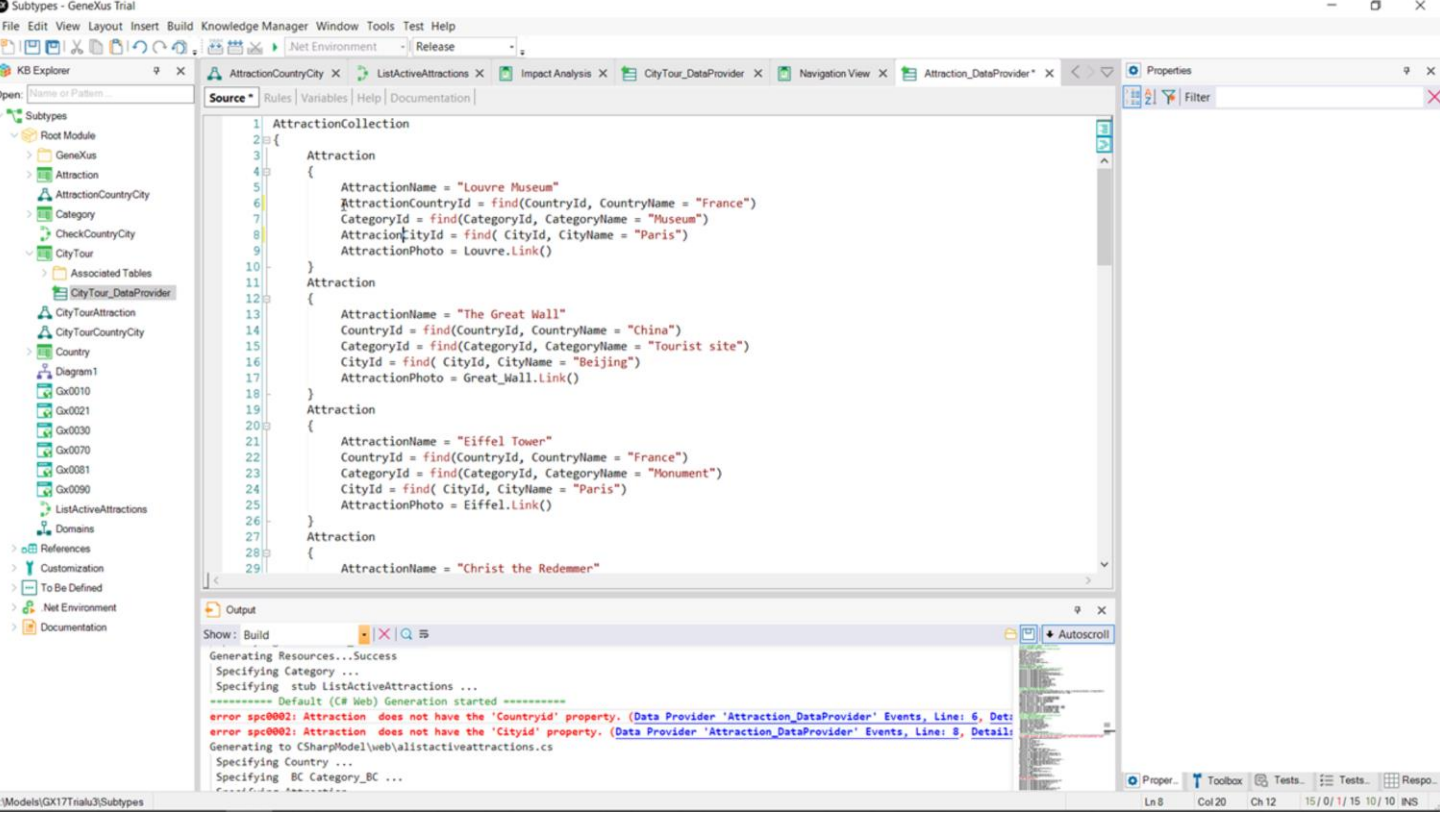
A primeira será modificando o nome dos atributos de país-cidade neste caminho, de forma a ser possível identificá-lo.

Assim, definimos um grupo de subtipos para o país e cidade da atração.

Observemos que este grupo tem dois atributos primários: AttractionCountryId e AttractionCityId, que correspondem à chave primária da tabela CountryCity, pelos supertipos que indicamos: {CountryId, CityId}.

E que substituímos os supertipos por estes subtipos na transação Attraction.

Desta forma, vemos que o caminho abaixo agora é identificável.
Poderíamos inclusive adicionar à transação CityTour os atributos de país e cidade inferidos através de AttractionId, de forma a ser possível implementar a verificação diretamente através da regra error.



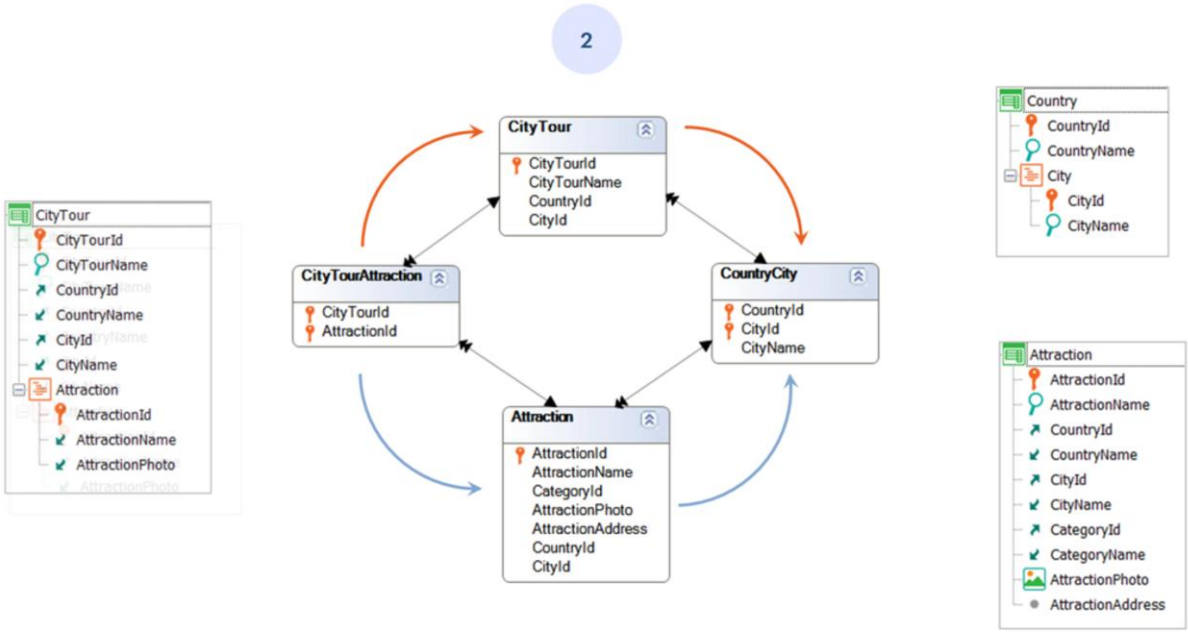
Observemos que, tendo feito estas alterações, fica fácil para nós eliminar a ambiguidade da lista que tínhamos antes. Basta alterarmos aqui CountryName e CityName para os subtipos.

Porém, como já tínhamos dados nas tabelas, está nos indicando que deve reorganizar, em particular, a tabela Attraction. Deve colocar os novos atributos, os subtipos e remover os antigos, os supertipos. E parece que vai transferir corretamente os dados do atributo antigo, o supertipo, para o novo, o subtipo.
Porém, ao pedir para reorganizar, ao finalizar a especificação encontramos erros. Em particular, vemos que está nos indicando que no DataProvider para preencher a tabela com atrações, Data Provider que tínhamos desde antes, está sendo utilizado o atributo CountryId, que não está mais em Attraction. E o mesmo com CityId. Aqui, vemos claramente uma desvantagem desta solução. Teremos que ir, um por um, para modificar os atributos antigos pelos novos em todos os objetos que já acessavam desde antes a tabela Attraction.

De fato, se já vínhamos realizando todo um desenvolvimento da aplicação onde CityTour ainda não estava contemplada, todavia não havia nos apresentado nenhum problema de ambiguidade, então é de se supor que já devíamos ter muitos outros objetos trabalhando sobre CountryId e CityId em Attraction.

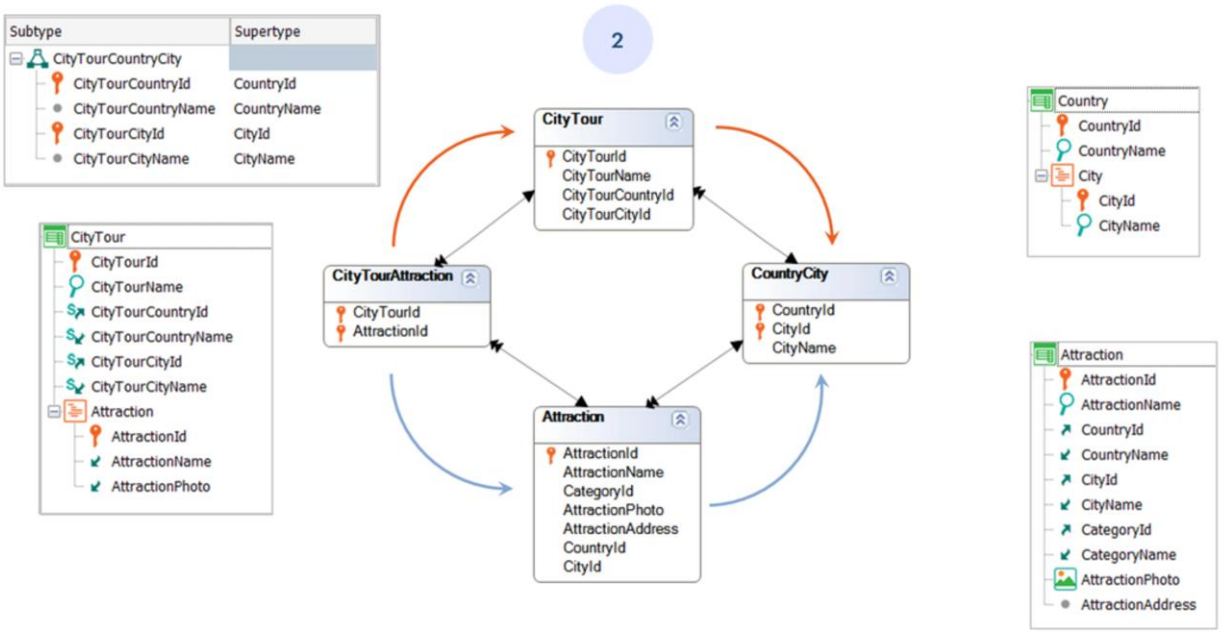
Se optamos por esta solução, teremos que resolver todos esses impasses.

Deixando isso de lado, se agora observamos a lista de navegação da lista que nos interessava, vemos agora que para obter país e cidade de cada registro de CityTourAttraction, o está fazendo através da tabela Attraction, como queríamos. Eliminamos a ambiguidade e escolhemos explicitamente o caminho que queríamos.



Pensemos, agora, em outra alternativa que nos complique menos em relação aos objetos que já existiam.

Esta segunda solução elimina a ambiguidade, modificando o nome dos atributos de país-cidade neste outro caminho.

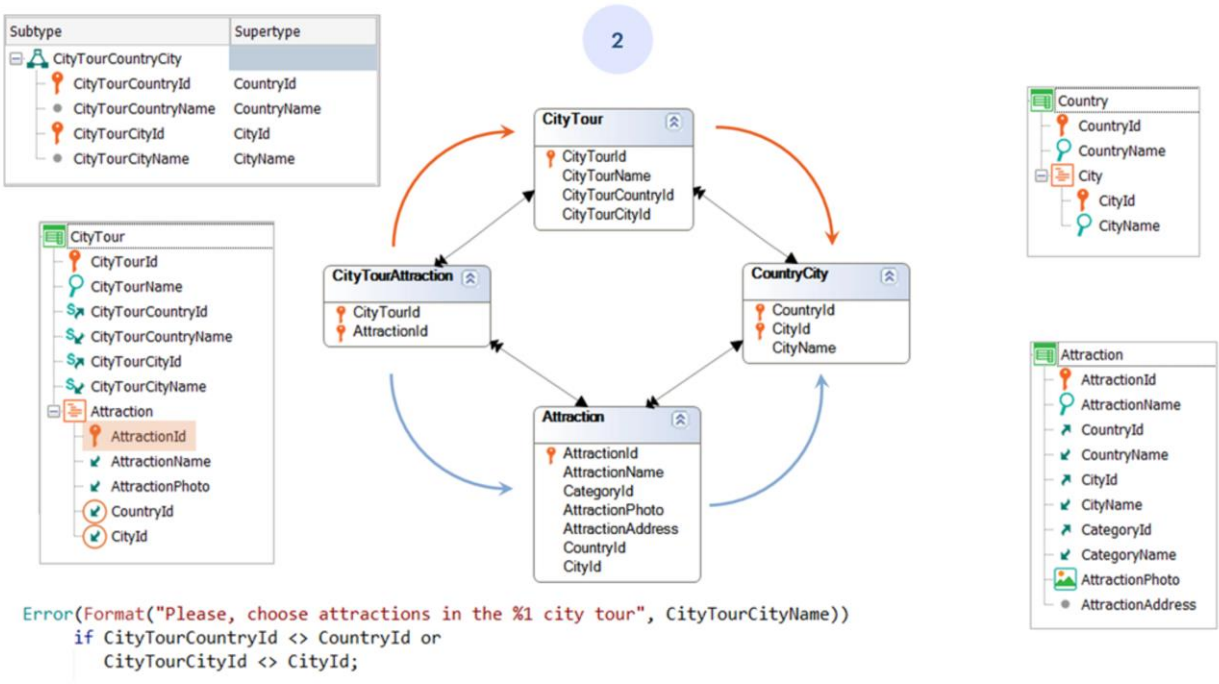


Para isso, definimos o grupo de subtipos para country e city utilizando-os diretamente no cabeçalho da transação CityTour. O que produz esta mudança na tabela CityTour (de supertipos para subtipos).

A diferença entre esta solução e a anterior é que é menos provável que tenha sido construída primeiro a transação plana, sem o segundo nível – que é aquele que introduz os dois caminhos para CountryCity-. Portanto, não é de se esperar que existissem outros objetos navegando CityTour antes de pensarmos em adicionar o segundo nível, e então ter que realizar a alteração de atributos por subtipos no primeiro.

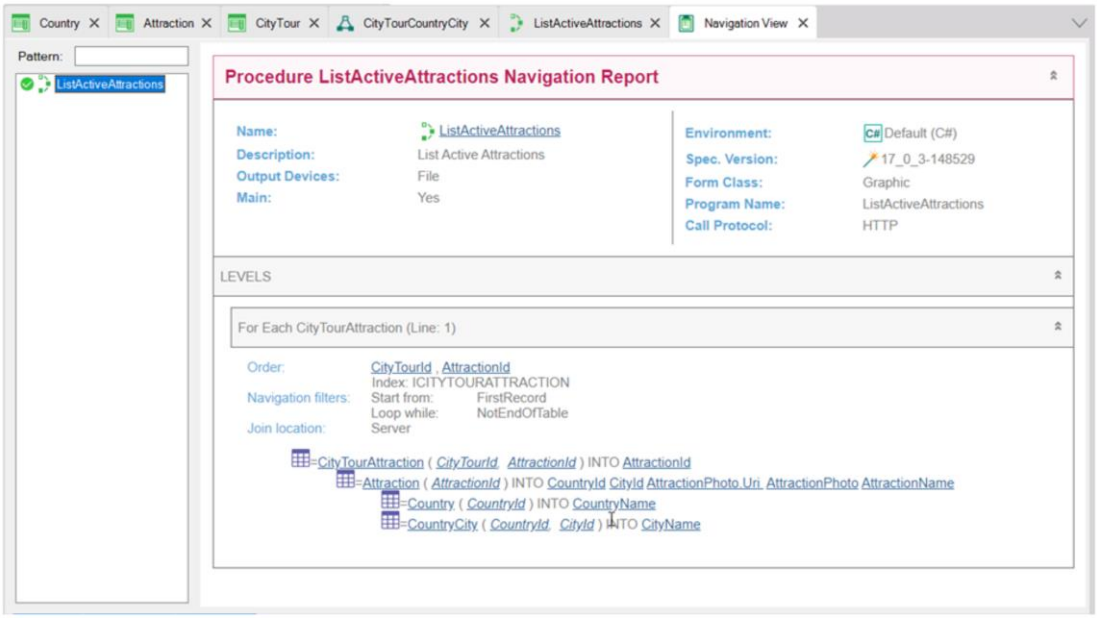
Ou seja, é de se esperar que as tabelas CityTour e CityTourAttraction sejam criadas ao mesmo tempo, e não primeiro CityTour, lhe sejam carregados dados e só depois percebemos que também vamos precisar de um segundo nível (que é aquele que gera os dois caminhos e esta solução).

Com esta solução, vemos que o caminho de cima agora é identificável, de forma que...

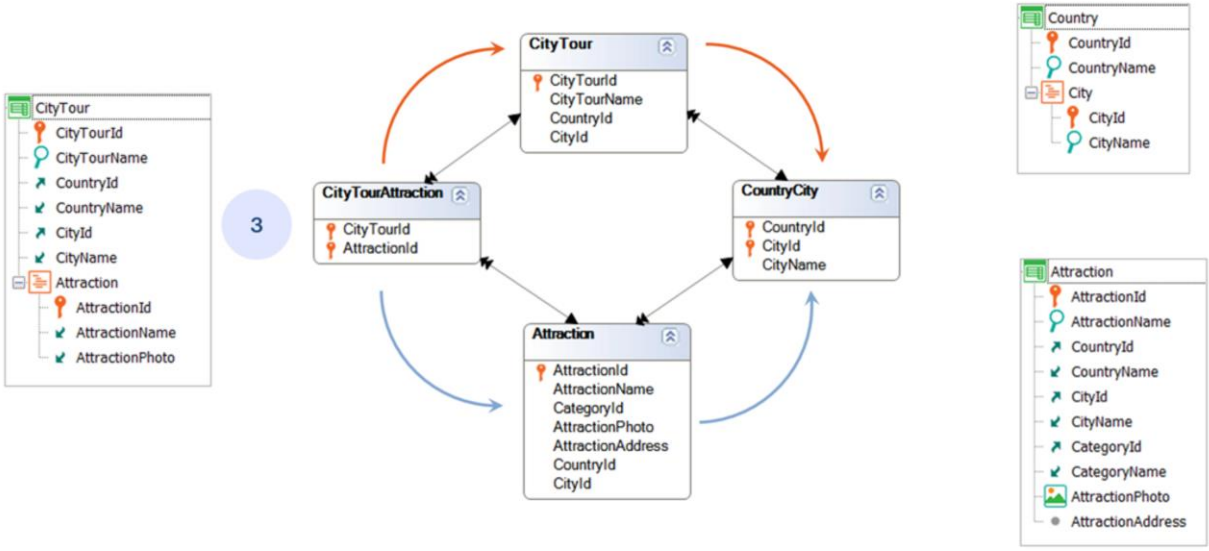



```
Error(Format("Please, choose attractions in the %1 city tour", CityTourCityName))
    if CityTourCountryId <> CountryId or
       CityTourCityId <> CityId;
```

...por exemplo, já podemos implementar a verificação de que coincidam diretamente através da regra error, sem necessidade do procedimento.
Para o que devemos adicionar CountryId e CityId à estrutura, para poder utilizá-los na regra. Observemos que nos indica claramente que estão sendo inferidos de AttractionId.

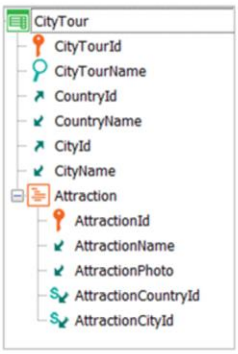


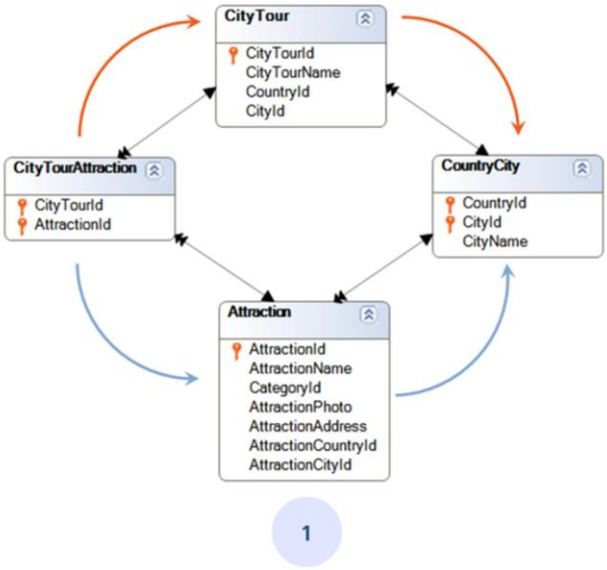
Se agora temos então esta solução implementada em GeneXus, claramente, se na lista que estávamos analisando deixamos os supertipos CountryName e CityName, serão retirados de CountryId e CityId **da tabela Attraction**. E não mais pelo outro caminho, o de CityTour. Vamos confirmar com a lista de navegação. Mostra-nos o que esperávamos. Não há mais ambiguidade.

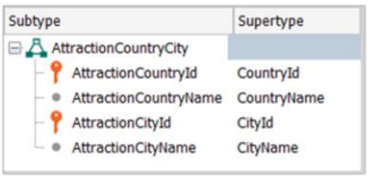


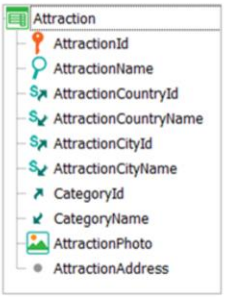
Chegamos agora à última alternativa com subtipos, que é aquela que será, a priori, menos intuitiva, mas que tem uma grande vantagem: fornece a eliminação da ambiguidade na própria tabela em que é produzida. Podemos dizer: a tabela que dá origem aos dois caminhos.

É que se pensamos nas soluções anteriores, estamos mudando os nomes dos atributos de Country e de City em tabelas que, vistas em suas estendidas, não têm nenhuma ambiguidade.

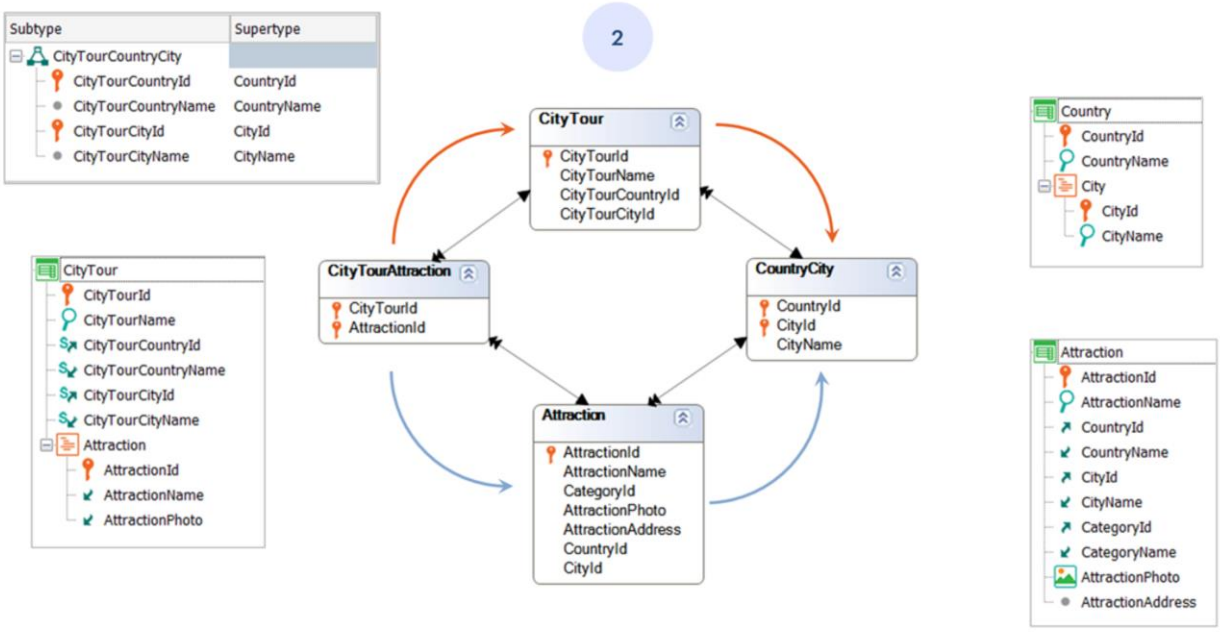






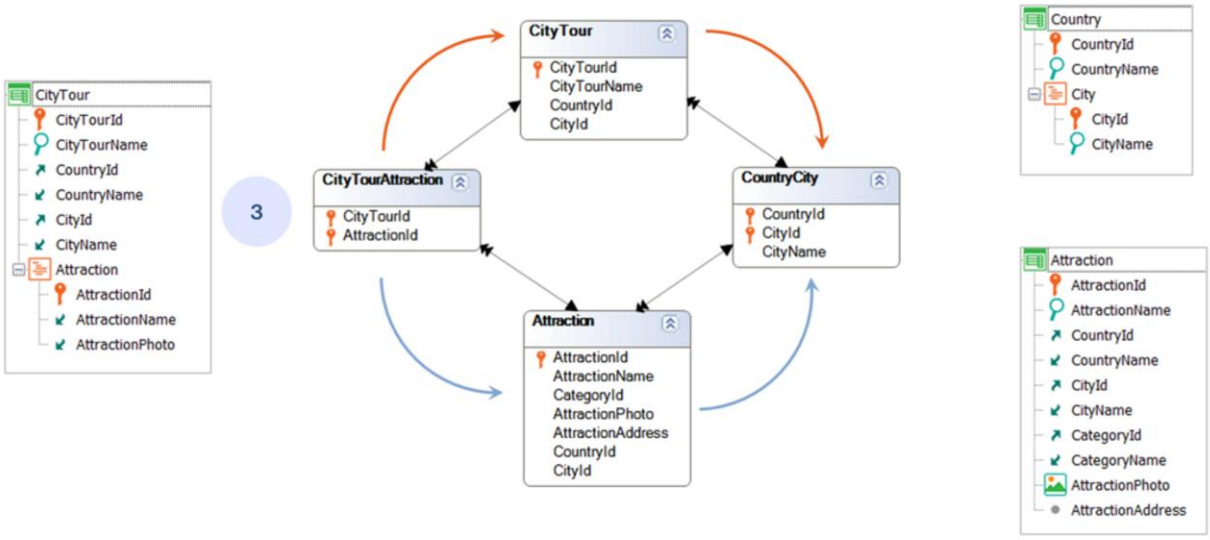


Assim, se olhamos para a solução 1, e pensamos, por exemplo, que queremos desenvolver um Web Panel que mostre as atrações turísticas com suas informações de país e cidade, não se compreende por que existem nessa tabela subtipos em vez dos supertipos. Parados a partir de Attraction não há nenhuma necessidade de defini-los.



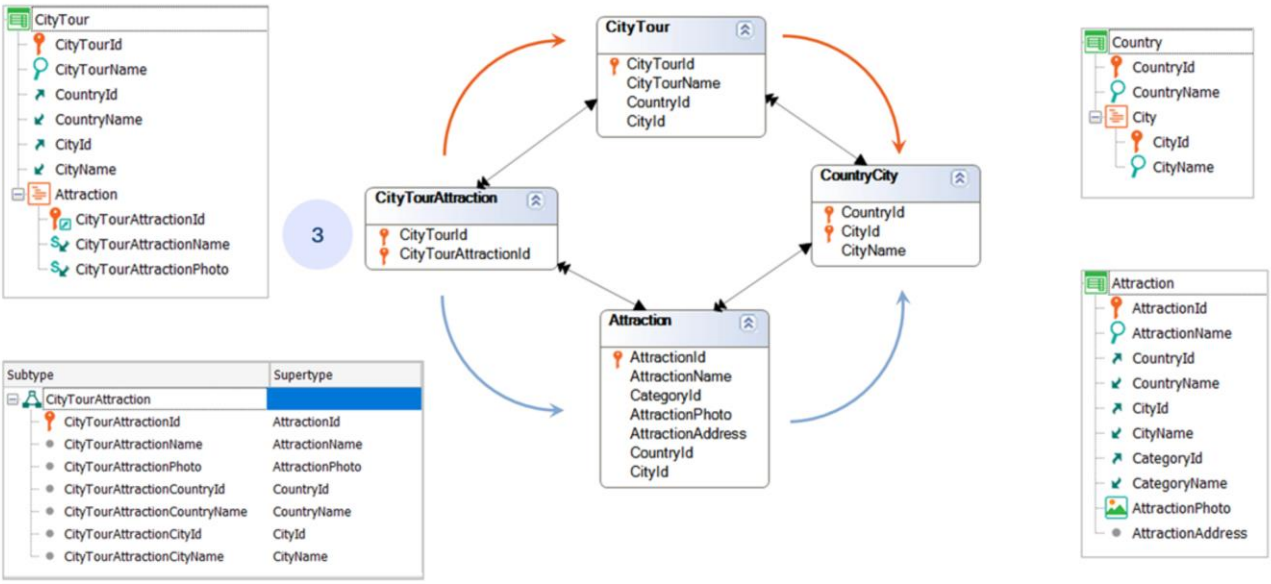
O mesmo se olhamos para a solução 2 e paramos sobre CityTour.

Se quiséssemos listar os city tours com seu país e cidade, posicionados nessa tabela base, também não se compreende por que estão sendo usados subtipos. Embora, neste caso, como a tabela CityTourAttraction provém de um segundo nível da transação que gera a tabela CityTour, estão mais intimamente relacionadas e pode ser vista mais claramente a justificativa de por que esses subtipos.



Eliminar a ambiguidade na própria tabela que causa os dois caminhos parece vantajoso. Pelo menos no sentido de que a ambiguidade é inerente a esta tabela, então ali nunca será injustificado que apareça um subtipo.

Agora, como fazemos para eliminar a ambiguidade na própria tabela que faz aparecerem esses dois caminhos? No exemplo, na própria tabela CityTourAttraction.

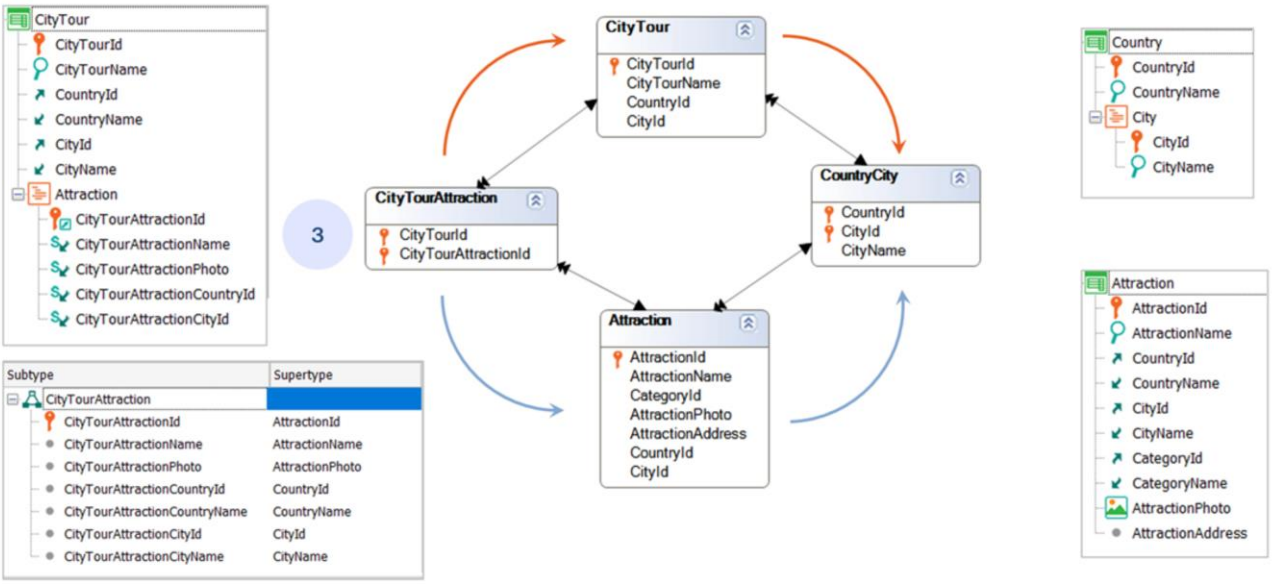


E se definimos um grupo de subtipos que permitam modificar o nome para AttractionId quando apareça como chave estrangeira, e todos os atributos que a partir dela são inferidos e nos interessam?

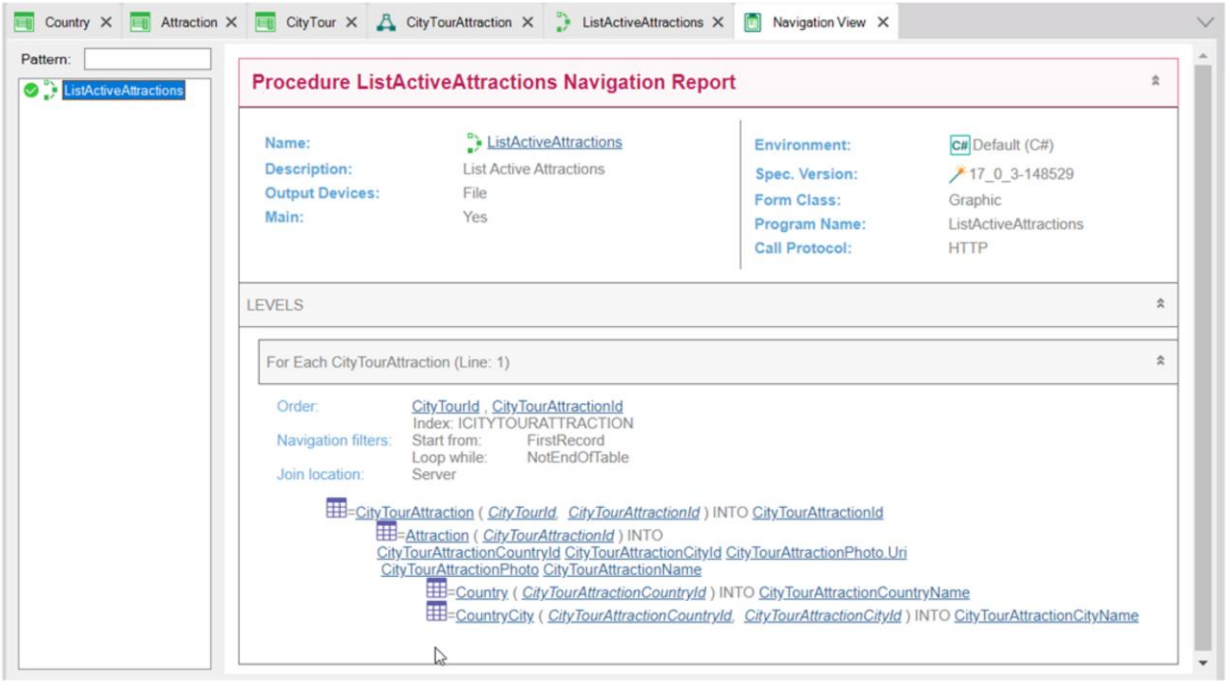
E então, em CityTour, em vez de utilizar o atributo AttractionId, utilizamos esse subtipo? E, é claro, agora devemos inferir nome de atração e foto também em subtipos desse grupo.

Ao fazer isto, o diagrama de tabelas passa a ser assim. Observemos que as outras tabelas não devem ser modificadas de forma alguma, portanto, todos os objetos que estavam trabalhando anteriormente sobre essas outras tabelas, continuarão a fazê-lo sem nenhum problema.

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



Se agora quiséssemos controlar se o país e cidade do city tour são idênticos ao país e cidade da atração, basta adicionar à estrutura da transação os dois atributos inferidos CityTourAttractionCountryId e CityTourAttractionCityId e escrever a regra de erro tal como a vemos.

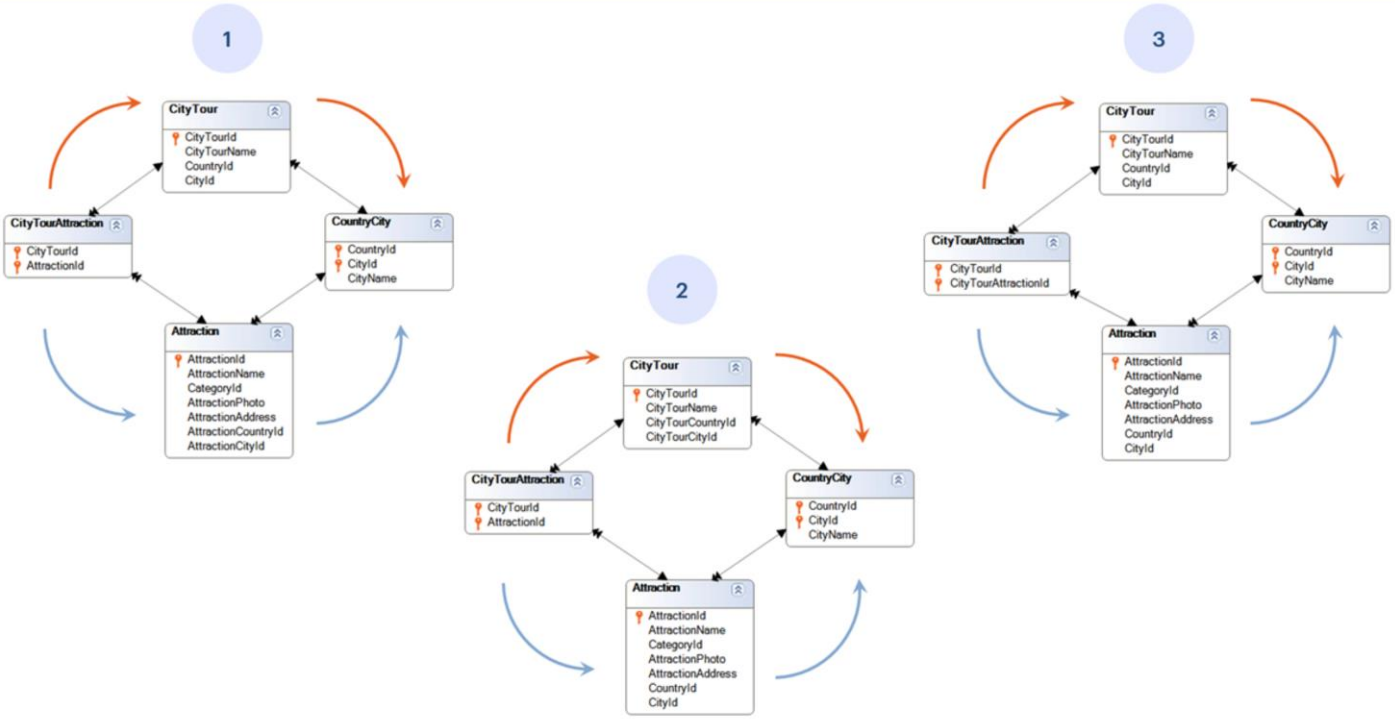


Com esta solução, a lista que estávamos buscando eliminar a ambiguidade desde o início fica muito clara para nós.
Percorremos com um for each o nível Attraction da transação CityTour. E aqui temos que modificar os atributos, que são todos inferidos da atração, mas que agora não é mais AttractionId, mas seu subtipo. Modificamos, então, todos os atributos para os subtipos destes. Em particular, o nome de país e o nome de cidade. Mas quando vamos ver a lista de navegação, nos indica que esses dois atributos, justamente, não estão sendo alcançáveis.

Por quê? Porque embora tenhamos definidos estes dois subtipos dentro do grupo, não os especificamos no nível da transação que estamos percorrendo com o For each. Se agora os adicionarmos, o que não terá nenhum efeito negativo uma vez que estão inferidos, e pedirmos novamente a navegação da nossa lista, descobrimos que já não há nenhum problema.

E de fato, vemos como está recuperando a informação corretamente, a partir do subtipo na tabela navegada, acessando a tabela Attraction e desta a CountryCity e Country.

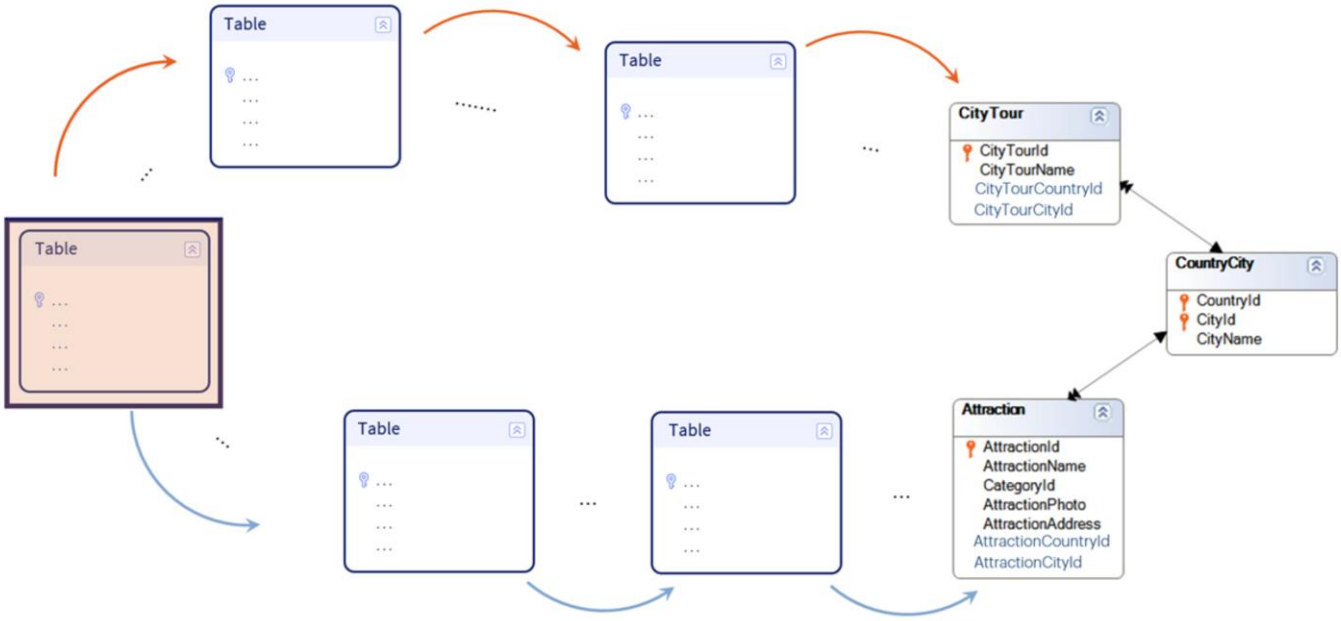
Mas isto, de ter que adicionar cada vez, todas as informações inferidas que deseja utilizar em qualquer navegação à estrutura da transação pode ser algo incômodo.



Para encerrar: toda solução tem seus prós e contras e depende do caso particular ao qual está sendo aplicada.

Por exemplo, se todas as transações são criadas juntas, então o problema de ter outros objetos que utilizaram anteriormente os atributos antigos desaparece e as opções 1 e 2 já não são problemáticas nesse sentido. Sim, em não justificar por si só o aparecimento de subtipos.

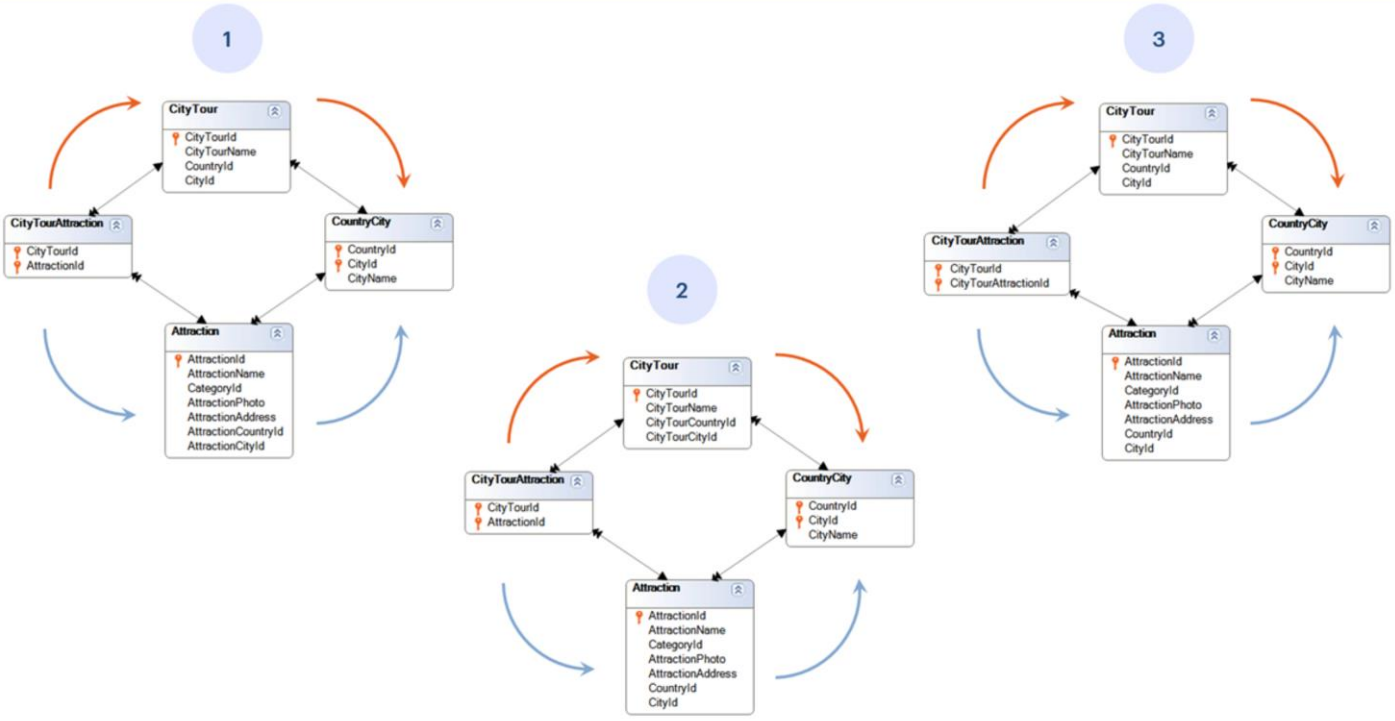
Neste caso, não parece muito problemático, pois, para a solução 1, parados em Attraction, basta olhar a tabela diretamente superordenada para descobrir o porquê. E para a solução 2, parados em CityTour, o mesmo.



Mas, se na solução 1 estamos parados em Attraction e ali definimos subtipos para CountryId e CityId e a tabela que produz a ambiguidade de caminhos está muito longe, aí as coisas se tornam um pouco mais confusas.

Ou se, para a solução 2, estamos parados em CityTour, o mesmo. Já não é tão fácil entender o que esses subtipos fazem ali.

Por outro lado, se os subtipos são construídos na tabela da ambiguidade, definindo uma das chaves estrangeiras como subtipo, ali é muito claro. Basta observar sua tabela estendida para encontrar a tabela que pode ser alcançada por vários caminhos.

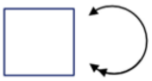


É nossa terceira solução. A desvantagem desta solução é que em qualquer objeto onde devemos utilizar atributos que são obtidos a partir do subtipo chave estrangeira, devemos adicioná-los, é claro, ao grupo de subtipos, onde serão marcados como inferidos, mas também, devemos adicioná-los à estrutura da transação.

E, obviamente, não fica claro à primeira vista que a ambiguidade é dada por Country e City. Se não analisamos a tabela estendida, não sabemos por quais dos atributos do grupo de subtipos foi definida. Poderia ter sido por CategoryId, por exemplo.

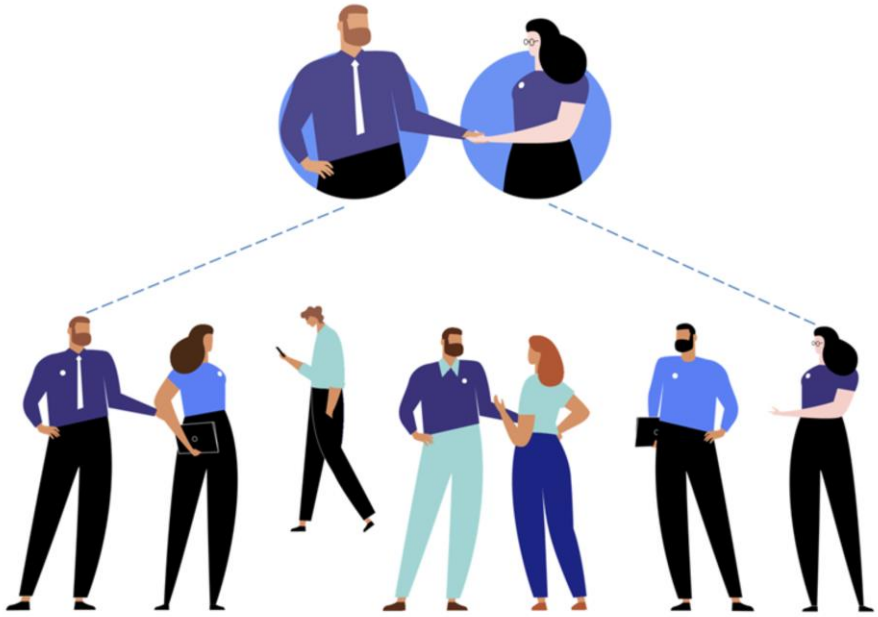
Aqui apresentamos estas três soluções. O desenvolvedor escolherá aquela que entenda que mais lhe convenha de acordo com sua realidade.

RECURSIVE SUBTYPES



Vejamos agora outro caso de uso de subtipos, que chamamos de subtipos recursivos, em que temos uma entidade que deve autorreferenciar-se.

Recursive Subtypes

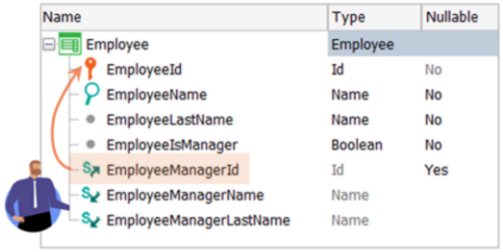


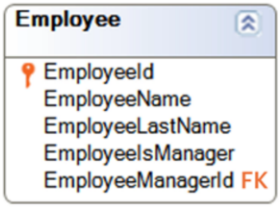
Para estudar este caso, consideremos que estamos representando as informações dos funcionários da agência de viagens. Cada funcionário pode ser, por sua vez, gerente de outro ou outros funcionários.

Recursive Subtypes

De todos os funcionários que têm um chefe, é necessário indicar quem é esse chefe.
O referido chefe é, em particular, um funcionário. Portanto, é estabelecida uma relação na tabela de funcionários com ela mesma.

Recursive Subtypes



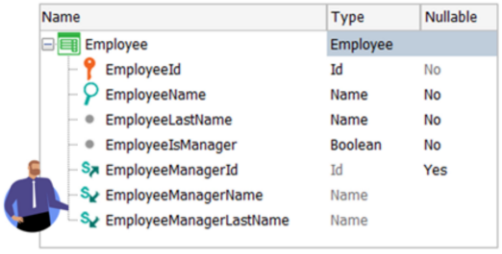


Subtype	Description	Supertype
EmployeeManager		
EmployeeManagerId	Employee Manager Id	EmployeeId
EmployeeManagerName	Employee Manager Name	EmployeeName
EmployeeManagerLastName	Employee Manager Last Name	EmployeeLastName

Para resolver isto, devemos criar um grupo de subtipos que represente as informações do chefe do funcionário.

O atributo EmployeeManagerId será, para todos os efeitos, tomado como um EmployeeId e, por este motivo, formará uma chave estrangeira para a própria tabela Employee.

Recursive Subtypes



Portanto, ao inserir a informação de um funcionário através da transação, quando o usuário escolher um valor para o campo EmployeeManagerId, GeneXus controlará a integridade referencial, ou seja, controlará que haja um registro na tabela de funcionários com esse valor para o atributo EmployeeId.

Convidamos você a pensar em situações reais nas quais seja necessário utilizar os diferentes casos de uso de subtipos que vimos e realizar a implementação em GeneXus.

GeneXus™

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