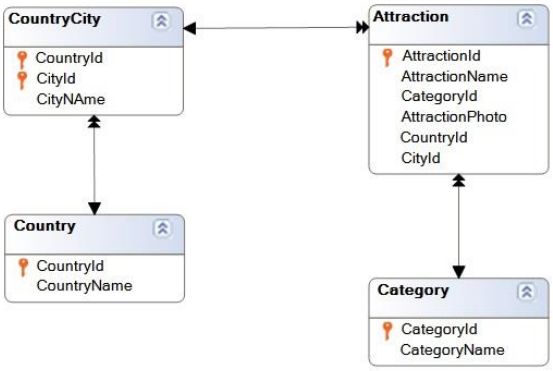
Mais sobre o comando For each

GeneXus™

Análise: Transação base




```
print Title

for each Attraction
     print Attractions
endfor
```

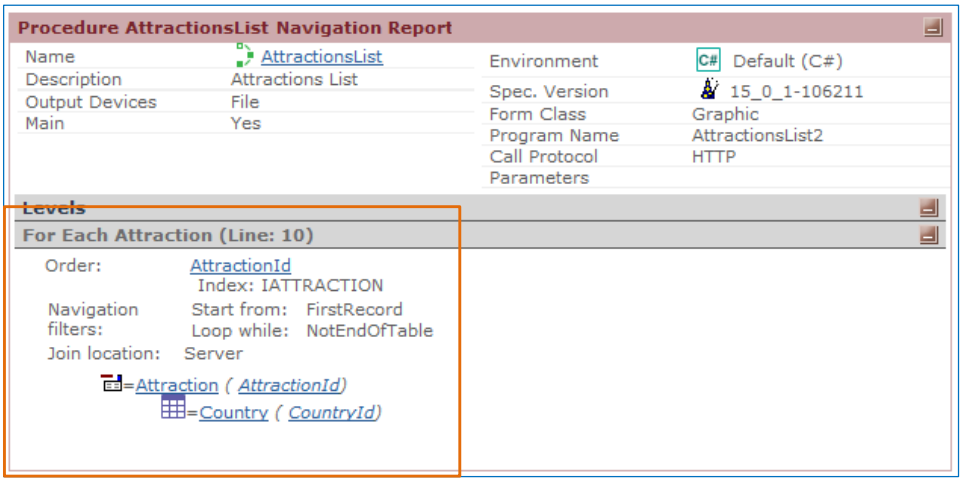
Lembremos que GeneXus determina a tabela base do for each levando em consideração o nome da transação que declaramos ao lado do for each, que corresponde ao nome da transação base, ou seja, aquela transação cuja tabela física associada queremos percorrer.

Mas, além disso, os atributos declarados dentro do for each, seja em printblocks, cláusulas where, order, etc, devem pertencer à tabela estendida da tabela base do for each

.

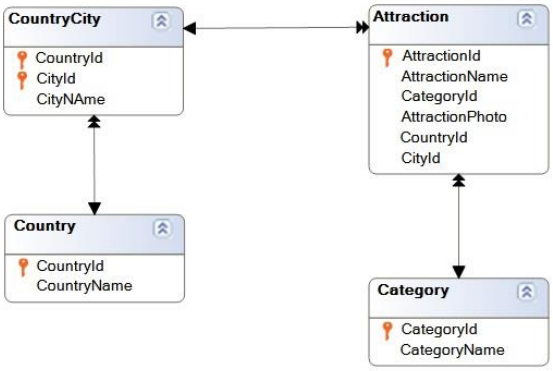
Neste exemplo que estamos vendo, a tabela base do for each será ATTRACTION, ou seja, a tabela que será percorrida e será acessada sua tabela estendida para recuperar os dados necessários.

Análise: Transação base



Se observamos a lista de navegação, vemos que nos informa claramente que a tabela base é ATTRACTION, que o percurso será ordenado pela chave primária da referida tabela, ou seja, por AttractionId, e que será percorrida toda a tabela, acessando também a tabela COUNTRY para recuperar o valor de CountryName, que corresponde ao país da atração.

Análise: Transação base



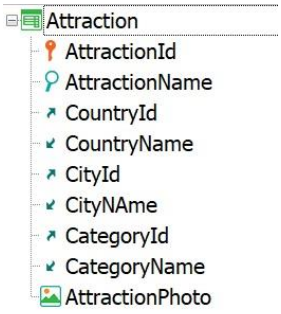
Não é obrigatório especificar a transação base de um For each.

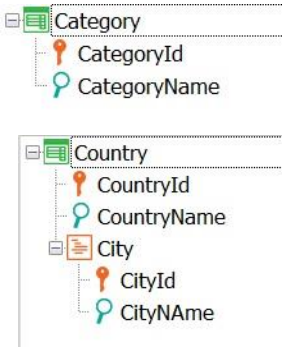
```
print Title

for each  ?
      print Attractions
endfor
```

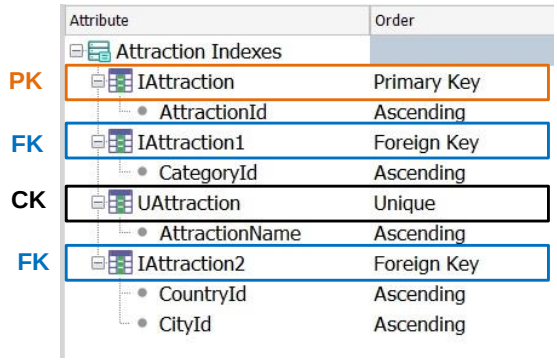
No entanto, é obrigatório especificar transação base para um for each?

A resposta é Não. GeneXus poderá calcular a tabela base do for each a partir dos atributos que participam do comando.

Índices e sua relação com as consultas do banco de dados







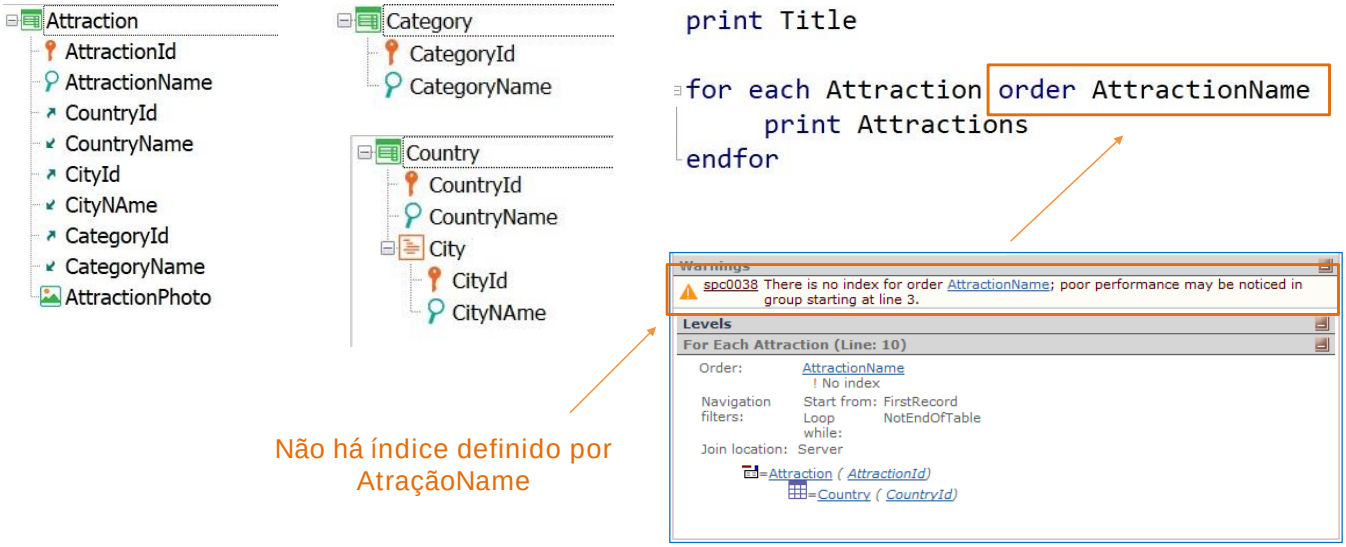
Passemos agora aos índices e sua relação com as consultas à base de dados.

Já sabemos que os **índices** são caminhos de acesso eficientes aos dados.

Se recordamos o que já vimos, em cada tabela, GeneXus cria um índice pelo atributo primário (seja uma chave simples ou composta) e um índice para cada chave estrangeira. Isto é feito para que sejam mais eficientes os controles de consistência dos dados entre as tabelas.

Vimos também que é possível definir índices, indicando se aceitam valores duplicados ou não. Se definimos um índice que não aceita valores duplicados, ou seja, um índice Unique, estamos dizendo para GeneXus que deve controlar automaticamente a unicidade de seu valor, e esse atributo, ou conjunto de atributos sobre os quais se define o índice, passa a ser uma chave candidata.

Índices e sua relação com as consultas do banco de dados

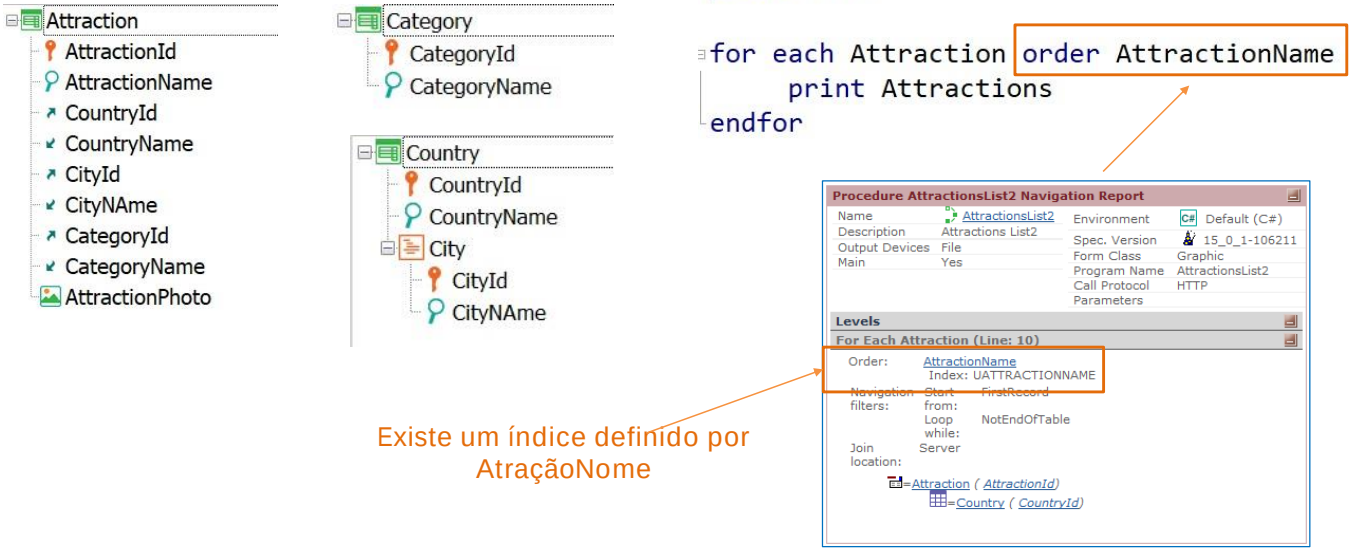


Vimos também que se adicionamos uma cláusula order, por exemplo, para ordenar pelo nome da atração, a lista de navegação dá um aviso, nos informando que na base de dados não existe um **índice** pelo atributo pelo qual necessitamos ordenar as informações, portanto poderíamos ter um baixo desempenho para esta consulta

.

Ocorre que ao indicar um atributo pelo qual ordenar, GeneXus tenta tornar a ordenação eficiente e, portanto, busca verificar se existe um índice por esse atributo. Mas como não o encontra, nos avisa.

Índices e sua relação com as consultas do banco de dados



Se necessitamos então obter os registros de ATTRACTION ordenados pelo atributo AttractionName, então estes registros terão que ser reordenados, pois por padrão estão ordenados pelo valor do atributo que é chave primária.

Quando se define uma consulta, se existe um índice físico criado na tabela pelo atributo a ordenar, GeneXus o utilizará. Mas, neste caso, a consulta precisa ser ordenada por um atributo secundário: AttractionName. E GeneXus nos avisa na lista de navegação, como vimos, que não existe um índice definido .

A existência do índice otimizaria a consulta. Mas a desvantagem de criar um índice é que, a partir daí, deve ser mantido. Ou seja, se os usuários vão adicionando, modificando e eliminando atrações na tabela ATTRACTION, este índice deve ser reorganizado.

Uma vez feito isto, ao pressionar F5, a base de dados deverá ser reorganizada, para criar este novo índice. Então, na lista de navegação veremos que GeneXus utilizará esse índice que acaba de ser criado.

Vale mencionar que assim como o criamos, a qualquer momento podemos eliminar um índice e, ao fazer F5 e reorganizar, voltaremos à situação da qual havíamos partido antes de criá-lo.

Índices e sua relação com as consultas do banco de dados: Exemplo


```
Parm (in:&NameFrom, in:&NameTo);

print Title

for each Attraction order AttractionName
    Where AttractionName >= &NameFrom
    Where AttractionName <= &NameTo

    print Attractions
endfor
```

Where AttractionName >= &NameFrom and AttractionName <= &NameTo

Vejamos este exemplo:

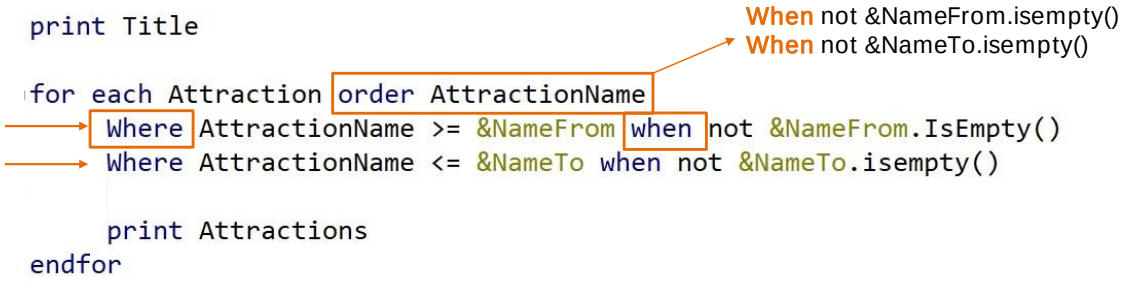
Suponha que nos interessa obter uma lista das atrações cujos nomes estejam alfabeticamente entre um par de valores recebidos por parâmetro. Por exemplo, entre as letras "B" e "N".

Para isso, especificamos as cláusulas where que estamos vendo. Ter várias cláusulas where é equivalente a ter apenas uma, onde as condições são conjugadas com o operador lógico "and". Ou seja, que serão considerados apenas os registros que atendam a todas as condições **ao mesmo tempo**.

Se vamos filtrar por AttractionName e temos um índice criado por esse atributo, será sempre conveniente **ordenar por AttractionName** para otimizar a consulta.

Observemos que se não especificamos cláusula order, GeneXus ordenará por chave primária, e deverá ser percorrida toda a tabela para saber se uma atração está dentro do intervalo especificado ou não.

Cláusula When



Vejamos agora a cláusula When que nos permitirá condicionar a aplicação das ordens e filtros.

Qual resultado será obtido para o for each que estamos vendo, se as variáveis &NameFrom e &NameTo estiverem vazias?
Se houvesse uma atração com o nome vazio, seria então a única retornada, pois seria a única que atenderia às condições. Caso contrário, nenhuma atração será listada.

É possível então considerar as ordenações e filtros, de modo que só se apliquem em determinadas circunstâncias? Por exemplo, para que só se aplique o primeiro where **quando** a variável &NameFrom não está vazia? E que só seja aplicado o segundo where **quando** a variável &NameTo não está vazia?

A resposta é sim. Conseguimos isso condicionando as cláusulas where com **when**. Só será aplicado cada where quando a condição do when for satisfeita.
Desta forma, em execução, ao deixarmos ambas as variáveis vazias, não será aplicado nenhum dos where, então serão listadas todas as atrações da tabela. Se a variável &NameFrom está vazia, mas &NameTo não está, não será aplicado o primeiro where, mas o segundo sim, de modo que serão listadas todas as atrações cujo nome seja menor ou igual a &NameTo.

Da mesma maneira, pode ser condicionada a aplicação ou não de uma order. De fato, pode ser especificada uma sucessão de ordens condicionadas, de modo que a primeira cuja condição seja satisfeita seja a escolhida.

Cláusula When none

```
print Title

for each Attraction order AttractionName
     Where AttractionName >= &NameFrom when not &NameFrom.IsEmpty()
     Where AttractionName <= &NameTo when not &NameTo.isempty()

     print Attractions
When none
     Print NoAttractions
endfor
```

Passemos agora para a **cláusula When none**

O que acontece quando nenhum dos registros da tabela base atende às condições indicadas?
Suponhamos que neste caso queremos imprimir na saída uma mensagem que o avisa... e que diga que não há registros associados.
Para isso vamos programar a cláusula **when none**.

Todos os comandos que sejam escritos entre o when none e o endfor serão executados sequencialmente e **no único caso em que não tenham sido encontrados registros da tabela base do for each que cumpram as condições indicadas.**

Neste exemplo que estamos vendo, decidimos imprimir uma mensagem, mas poderia ser escrita uma série de comandos, como por exemplo, outro for each.

Como a execução do que segue o **when none** implicará que não foi encontrado o que se procurava, se ali estiver escrito um for each, então não ficará aninhado ao do when none. Será como um for each independente.

Resumo...

For each *BaseTransaction*

order att_1, att_2, … , att_n [when *condition*]

order att_1, att_2, … , att_n [when *condition*]

where *condition* [when *condition*]

where *condition* [when *condition*]

main code

When none

.....

endfor

Resumindo isto que vimos...

Como já vimos, a **tabela base** de um For each é determinada a partir da transação base especificada; o restante dos atributos mencionados, tanto no corpo do For each (main code) quanto nas cláusulas Order e Where, deverão pertencer à tabela estendida dessa tabela base.

Os atributos mencionados no bloco When none não serão considerados .

Se observamos, deixamos em cinza tudo o que já havíamos visto antes. Aqui, adicionamos as cláusulas **when** e **when none**.
Mais adiante veremos que é possível adicionar mais cláusulas a este fundamental comando de acesso à base de dados

Estudo de caso

Customer
{
CustomerId*
CustomerName
}

Trip
{
TripId*
TripDescription
TouristGuideId
TouristGuideName
}

TouristGuide
{
TouristGuideId*
TouristGuideName
Phone
{
TouristGuidePhoneId*
TouristGuidePhoneNumber
}
}

Reservation
{
ReservationId*
ReservationDate
CustomerId
CustomerName
Trip
{
TripId*
TripDescription
}
}

Para finalizar, vamos analisar um estudo de caso:

Consideremos o seguinte desenho de transações
:
A transação Customer, a transação Trip que corresponde às excursões com um guia turístico responsável, a transação Tourist Guide com seu conjunto de telefones, e a transação Reservation para registrar, para cada cliente, o conjunto de excursões que tem reservada.

Necessitamos obter uma lista que mostre, para um determinado cliente, e a partir de uma determinada data, todas as excursões que tem reservadas e, para cada uma delas os telefones de contato do guia turístico responsável.

Estudo de caso

Trip
{
TripId*
TripDescription
TouristGuideI
TouristGuideName
}

TouristGuide
{
TouristGuideId*
TouristGuideName
Phone
{
TouristGuidePhoneId*
TouristGuidePhoneNumber
}
}

Reservation
{
ReservationId*
ReservationDate
CustomerId
CustomerName
Trip
{
TripId*
TripDescription
}
}

/out | **Rules *** | Conditions | Variables

```
Parm(in:&ReservationDate, in: CustomerId);


For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```

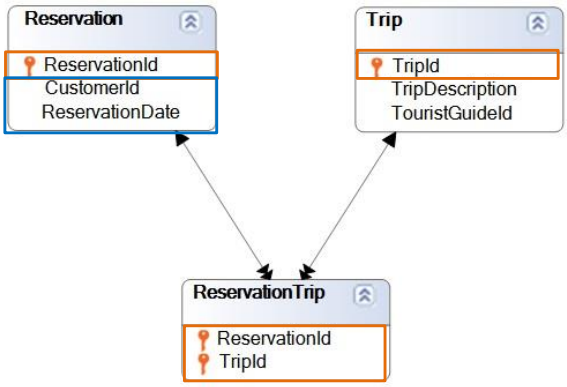
Para resolvê-lo, propomos o seguinte source
:
Vamos analisar se a informação listada é a que foi solicitada. Temos um par de for each aninhados. No primeiro, dizemos explicitamente que a tabela base será a correspondente ao nível Trip da transação Reservation, ou seja, a que se chama ReservationTrip.

Verificamos que dentro deste for each externo não existe nenhum atributo que não pertença à tabela estendida de RESERVATIONTRIP, pois em caso afirmativo, a lista de navegação lançará um aviso indicando que esse atributo não é acessível.

Então os atributos que devemos verificar são os que se encontram na cláusula Where e dentro do printblock de nome Trips, que neste caso são os atributos TripDescription, presente em TRIP, e ReservationDate, presente em RESERVATION.

Estudo de caso

```
/out | Rules * | Conditions | Variables

Parm(in:&ReservationDate, in: CustomerId);


For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```



Se observamos o diagrama de tabelas
:
vemos claramente que a partir de RESERVATIONTRIP acessamos um único registro de TRIP e um único registro de RESERVATION. GeneXus deverá, então, acessar essas duas tabelas cada vez que itere no for each.

Então, com quais registros da tabela base vai trabalhar o for each? Com aqueles que cumpram que, ao ir para a tabela RESERVATION para avaliar o valor de ReservationDate, este seja maior ou igual ao valor da variável &ReservationDate recebida por parâmetro.

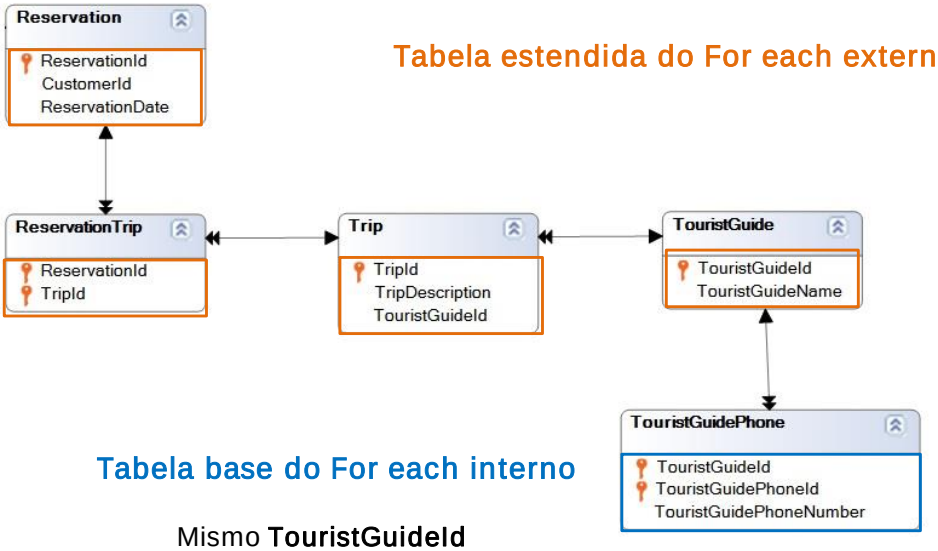
Mas também deverá cumprir que o valor de CustomerId corresponda ao valor recebido por parâmetro diretamente nesse atributo. Recordemos que "receber em atributo" é estabelecer que esse atributo estará instanciado. Ou seja, toda vez que esse atributo participe em algum lugar, o fará com o valor recebido quando foi invocado o objeto.

Como no for each que estamos analisando, já é acessada a tabela RESERVATION que o contém, então será aplicado um filtro automático por esse valor de CustomerId.

É importante fazer o seguinte esclarecimento: O fato de que o atributo recebido na regra parm pertença à tabela estendida do for each **NÃO FAZ** com que este seja aplicado automaticamente como filtro. Para que isso aconteça, o for each tem que estar acessando particularmente essa tabela da estendida para realizar alguma ação.

Estudo de caso

Relação indireta 1-N



Agora pensemos sobre o que acontece com o for each aninhado. Sua tabela base será claramente aquela associada ao nível Phone da transação TouristGuide. A próxima pergunta é: estabelece filtros implícitos para a informação que utilizará? Sabemos que sim, mostrará os telefones do guia de cada excursão

. Por quê? GeneXus busca se existe relação entre a tabela estendida do for each externo e a tabela base do for each aninhado
:
É uma forma de buscar uma relação de 1 a N, embora neste caso indireta. Se cada RESERVATIONTRIP tem um TouristGuideId, e na tabela a ser navegada também há um TouristGuideId, então GeneXus entende que pela relação entre as informações, se tratará do mesmo. E por isso realiza o join.

Nos próximos vídeos, continuaremos nos aprofundando a respeito do comando For each.

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