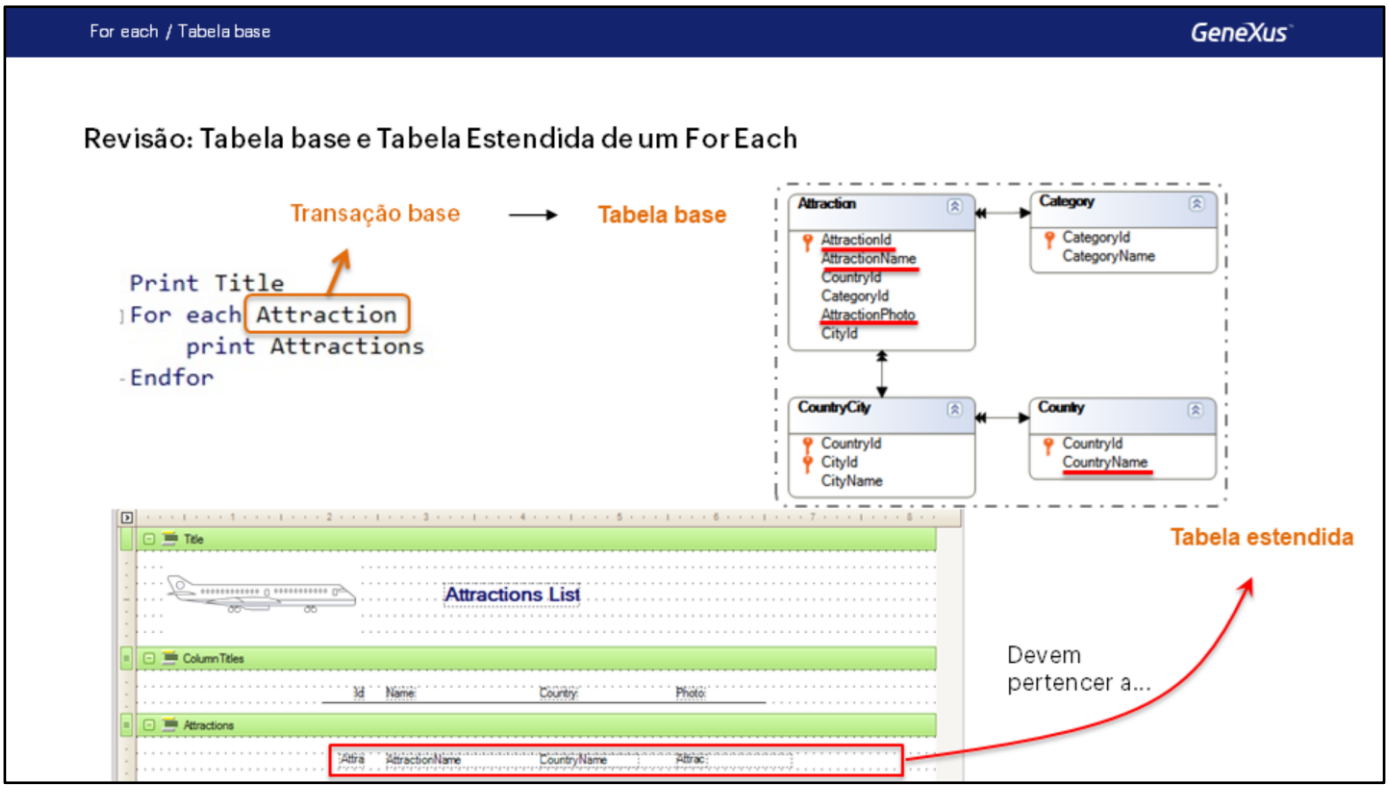
Mais sobre o comando For Each

GeneXus 16

Tabela base

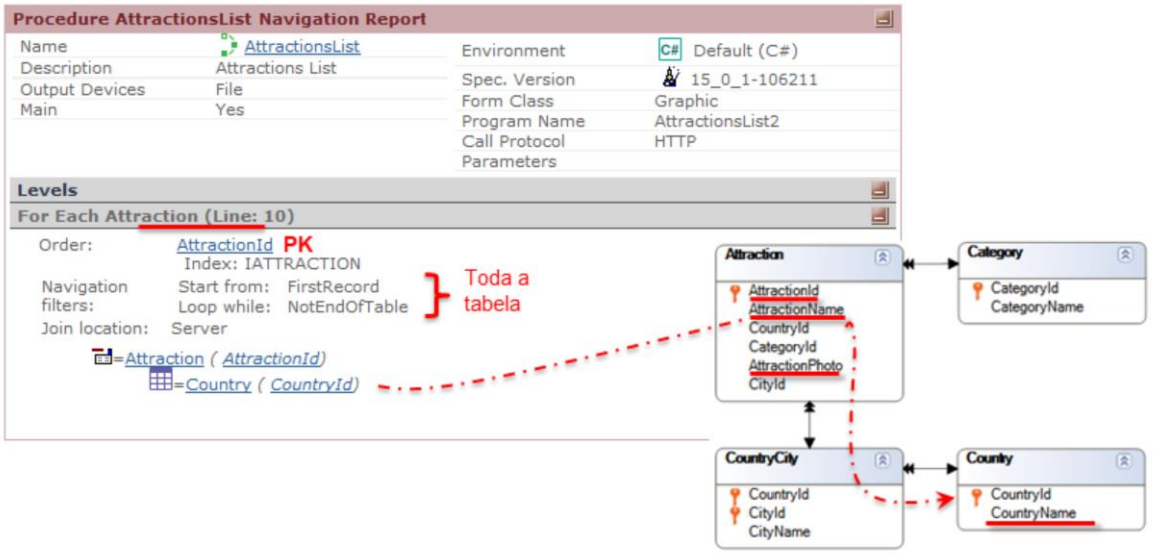


Recordemos que GeneXus determina a tabela base do for each levando em conta o nome da transação que declaramos ao lado do for each (que deve ser a transação cuja tabela física associada queremos recorrer).

Além do mais, os atributos declarados dentro do for each (printblocks, where, order, etc.), devem pertencer à tabela estendida da tabela base do for each.

No exemplo apresentado aqui, a tabela base do for each será ATTRACTION, ou seja, a tabela que será percorrida e será acessada sua tabela estendida para acesso aos dados solicitados.

Revisão: Navegação resultante



A lista de navegação informa-nos claramente que a tabela base é ATTRACTION e que esta será percorrida ordenada pela chave primária desta : AttractionId, e que toda a tabela será percorrida. E que a tabela COUNTRY também será acessada para recuperar CountryName que é o país da atração).

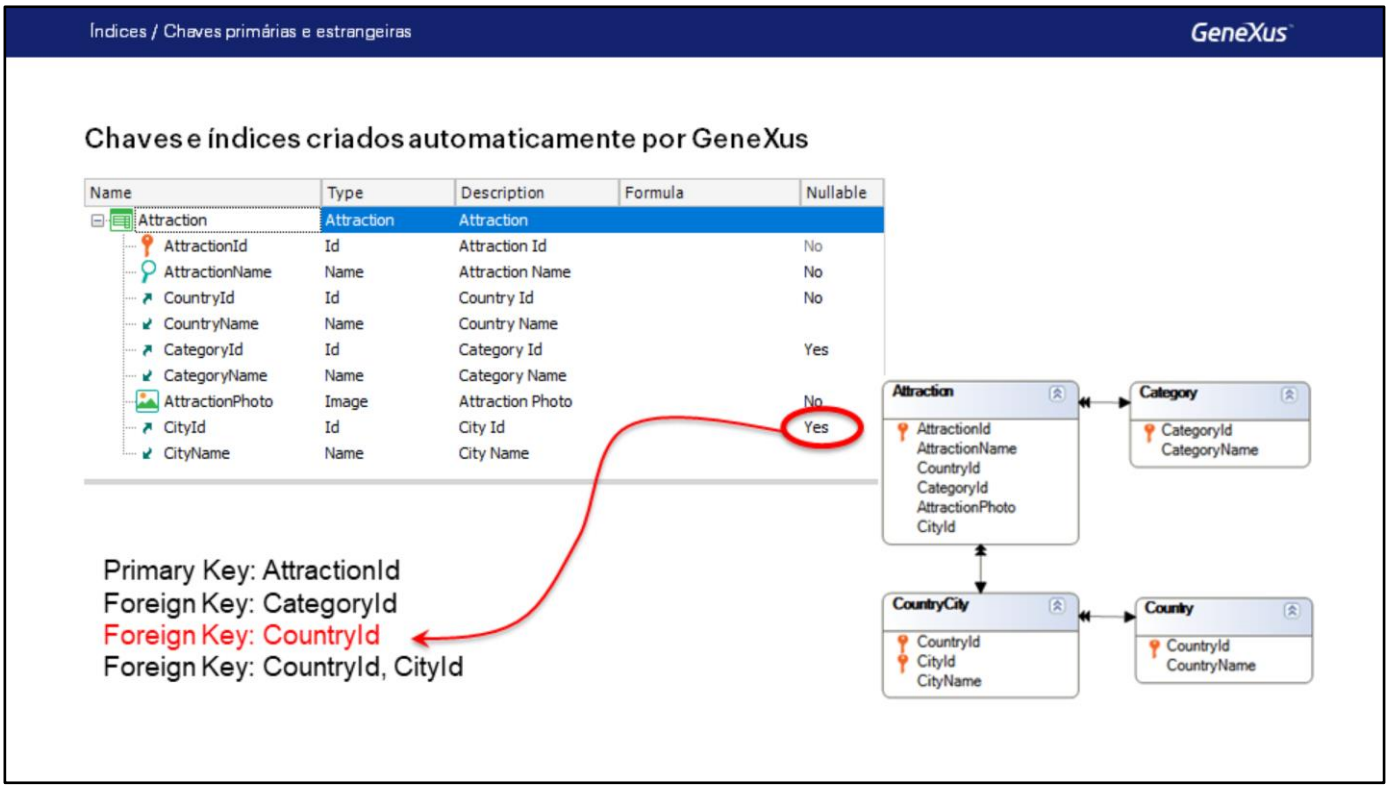
A definição da tabela base

- É obrigatório especificar a transação base para um for each?

```
print Title
print ColumnTitles
For each
    print Attractions
endfor
```

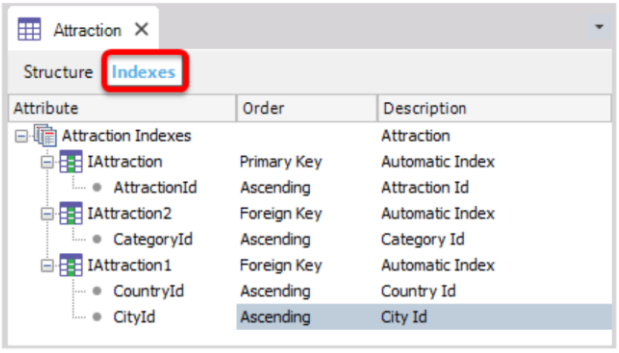
- Respuesta:
- Não. GeneXus poderá calcular a tabela base do for each a partir dos atributos que participem do comando. A forma pela qual é encontrada a tabela base não será abordada neste curso.

Índices e sua relação com as consultas ao BD
Cláusula order do for each



Se observamos a transação Attraction, podemos ver que GeneXus define a nível da tabela física associada quatro chaves: a primária e três estrangeiras.
A terceira, a chave por CountryId, que não é evidenciada no diagrama de tabelas é criado unicamente para os casos no qual o usuário deixe vazio o valor de CityId.
Visto que {CountryId, CityId} formam uma chave estrangeira composta, e que não teria sido possível que o usuário tenha deixado nulo o valor de CityId. Pois se não pode deixar de indicar valor da cidade, então seria desnecessário uma chave estrangeira por CountryId, dado que se existe um registro em CountryCity para esse país, é porque ao entrar com um valor este já devia existir na tabela Country.
A chave estrangeira por CountryId aparece, então somente porque foi setada a propriedade Nullable para CityId.

Chaves e índices criados automaticamente por GeneXus



Primary Key: AttractionId
Foreign Key: CategoryId
Foreign Key: CountryId
Foreign Key: CountryId, CityId

Para cada PK e FK GeneXus cria um índice na tabela associada.

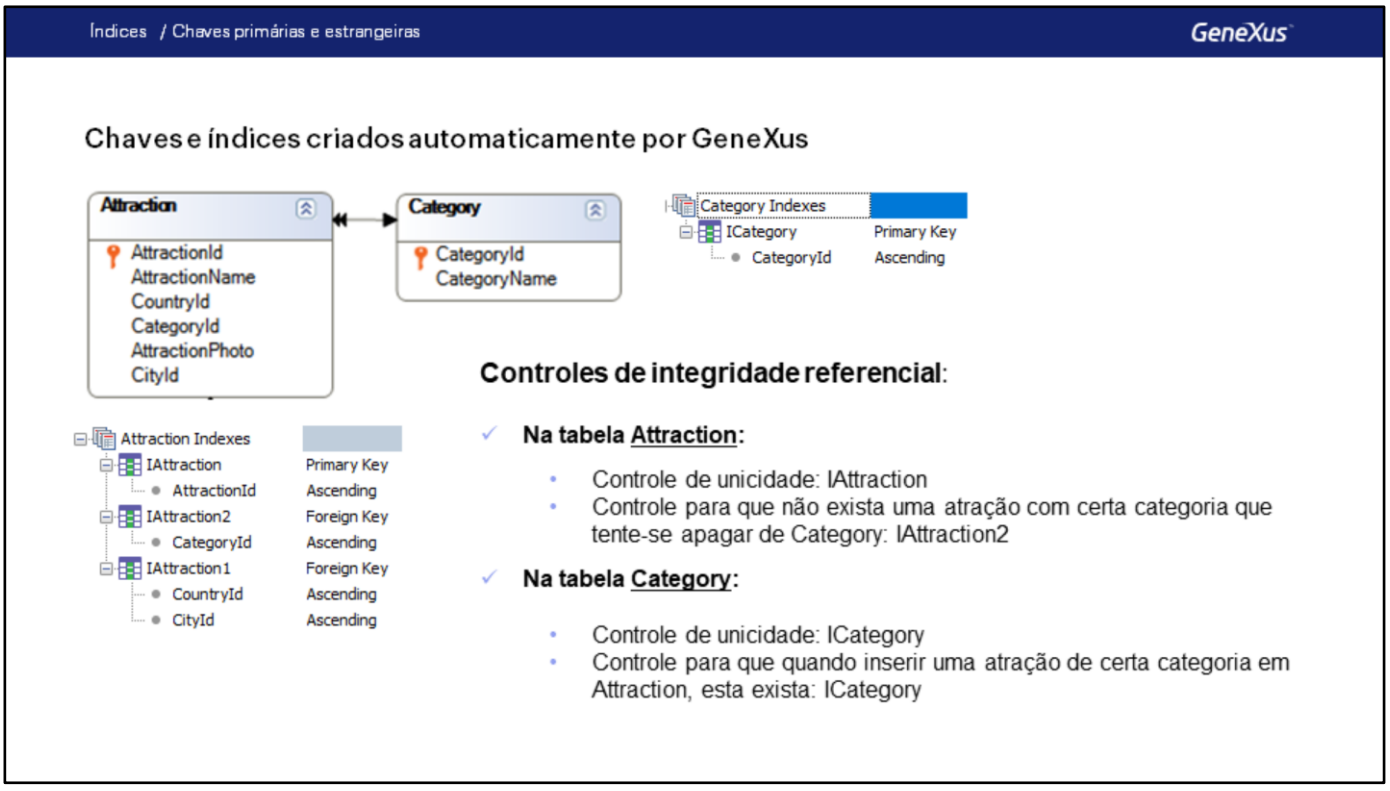
Exceção: um índice para {A, B} que já é na verdade um índice por {A}

Os **índices** são como vias de acesso eficiente aos dados. Podemos pensar por exemplo, em um libro de cozinha com muitas páginas que contém receitas, e que têm vários índices (índice alfabético, índice por tipos de comidas, etc.). De maneira análoga as tabelas que armazenam registros tem índices também.

GeneXus ao criar tabelas físicas cria para elas um índice para o atributo primário da tabela (ou seja, por sua chave primária seja simples ou composta) e um índice para cada chave estrangeira. Fazemos isso para que sejam mais eficientes os controles de consistência dos dados entre tabelas, como veremos na página seguinte.

Se editarmos a tabela Attraction no GeneXus é mostrada automaticamente a estrutura pela que é composta. Porém se vamos na aba Indexes, podemos ver os índices criados nesta tabela na base de dados.

Podemos ver foram criados três índices : um para a Primary Key, e dois para as chaves estrangeiras (Foreings Keys). Porquê não foi criado um índice para CountryId somente? Porque é desnecessário. Se temos um índice composto por CountryId, CityId, esse índice ja é em particular, um índice por CountryId.



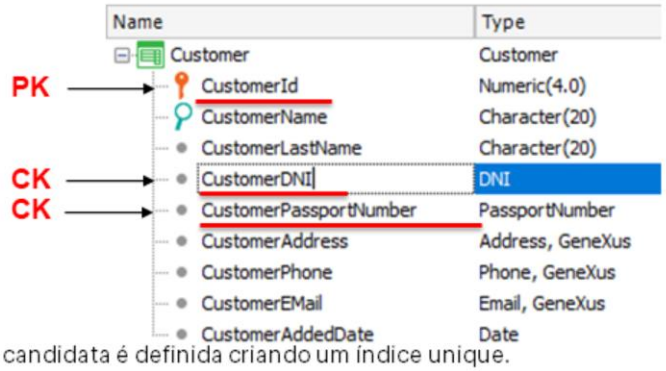
Este índices, como dizíamos, foram criados para tornar eficientes os controles de integridade referencial que GeneXus gerencia automaticamente nas transações.

Os **índices por chave primária** são criados nas tabelas para realizar um controle eficiente de duplicados, e também para tornar buscas eficientes quando queremos a partir de outra transação inserir ou modificar a chave estrangeira que refere a essa chave primária. No exemplo : quando em Attraction inserimos uma nova atração, e é preciso checar que exista uma categoria na tabela Category com esse valor de CategoryId. Ali é utilizado o índice pela PK de Category (ICategory).

Os **índices por chave estrangeira** são criados nas tabelas para que quando na transação que tenha esta chave primaria seja efetuado um processo de eliminação de registro, no nosso caso Category, possa-se saber de forma rápida e eficiente se existe algum registro relacionado. E neste caso impedir sua remoção. No nosso caso, se vamos eliminar uma categoria existente na transação Category, GeneXus deve saber, para assim permiti-lo, que não exista nenhuma atração com essa categoria. Então é usado para isso o índice IAttraction2 de Attraction.

Chaves candidatas e índices

- Para uma entidade podemos ter mais de um atributo ou conjunto de atributos que a identifiquem, ou seja, que seus valores não podem se repetir.



- Uma chave candidata é definida criando um índice unique.

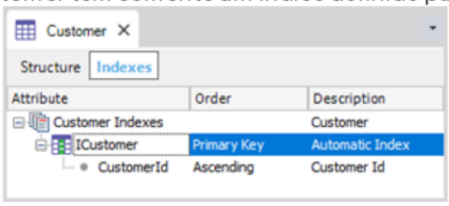
Como havíamos visto na aula sobre relações 1 a 1, para cada nível de cada transação é obrigatório definir o atributo ou conjunto de atributos que formam o identificador de nível. Esse identificador se traduzirá a nível de tabela física na chave ou chave primária da tabela. Com isso estamos dizendo que os valores deste atributo ou conjunto de atributos não poderão ser repetidos.

Porém em muitos casos existe mais de um atributo ou conjunto de atributos que devem cumprir essa condição. Por exemplo, para o cliente decidimos identificá-lo com um número interno de nosso sistema, porém também poderíamos ter como atributo secundário seu DNI (RG), documento nacional de identidade, expedido por seu país, ou inclusive seu número de passaporte, que também devem ser únicos. Como teremos que escolher que um dos três (CustomerId, CustomerDNI, CustomerPassportNumber) para identificar a entidade (no nosso caso escolhemos CustomerId), se não fizermos mais nada, os outros permanecerão como atributos secundários, podendo repetir.

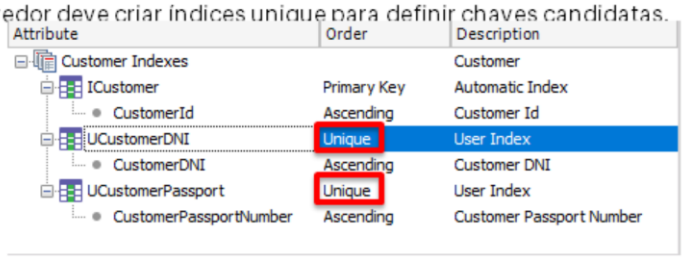
Como indicamos ao GeneXus que tanto CustomerDNI como CustomerPassportNumber são **chaves candidatas**, para que nos assegure que não sejam repetidas em clientes diferentes? Já tínhamos visto que era definindo um índice para cada chave candidata.

Chaves candidatas e índices

- A tabela Customer tem somente um índice definido para PK.



- O desenvolvedor deve criar índices unique para definir chaves candidatas.



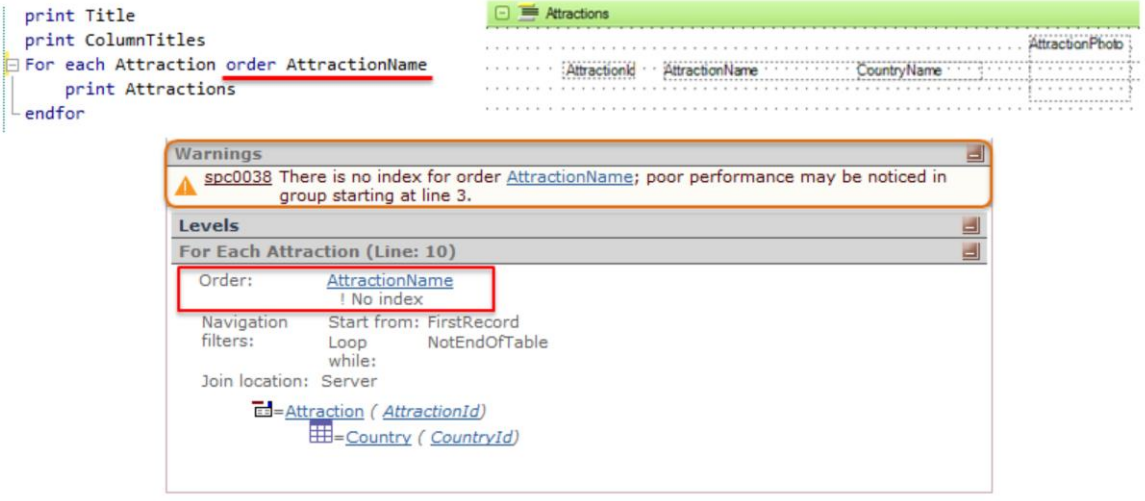
Se observamos os índices que automaticamente tinham sido definidos na tabela Customer, vemos que temos unicamente o índice por chave primária.

Devemos criar um índice para o atributo CustomerDNI, e indicar que será de tipo **Unique**. Ou seja, indicar que não pode ter valores repetidos.

E o mesmo para o atributo CustomerPassportNumber.

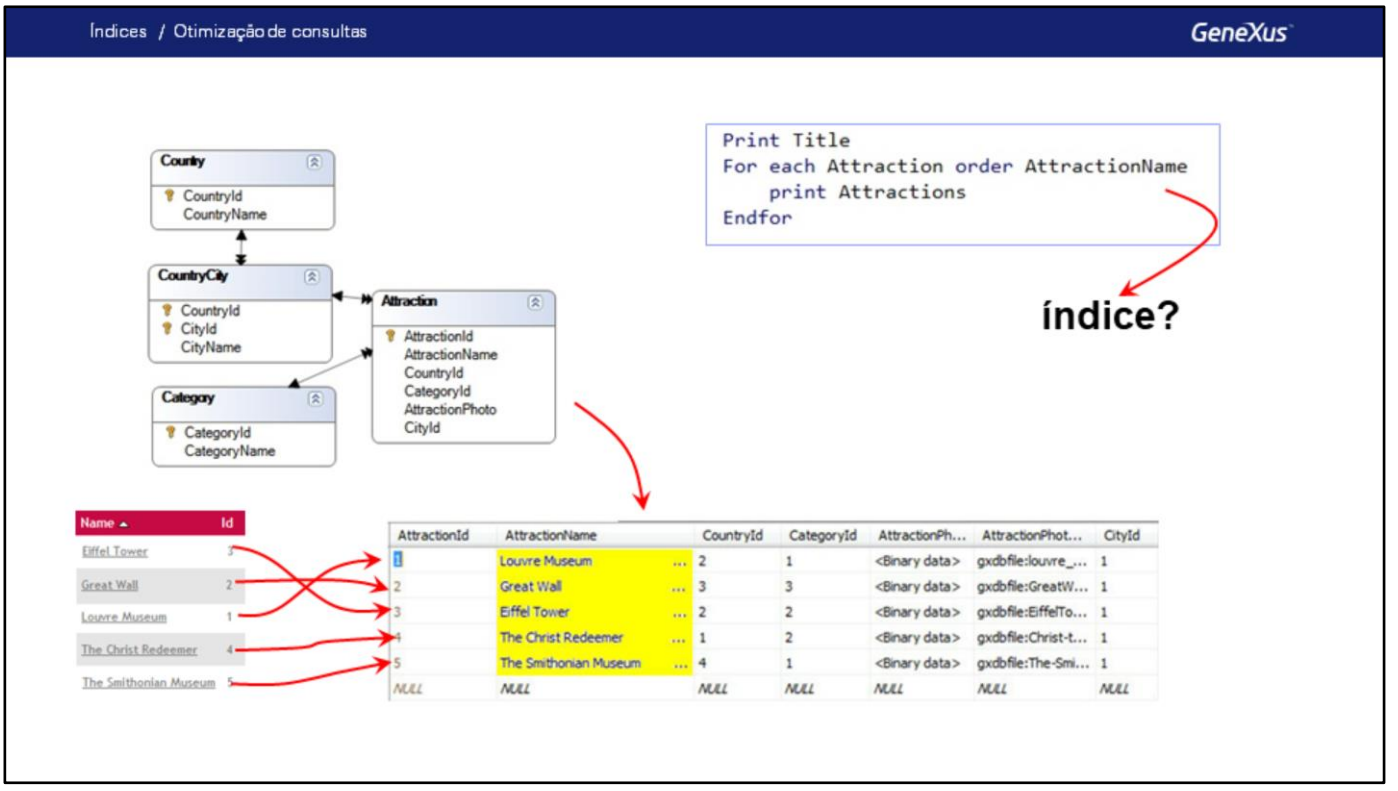
Desta maneira, GeneXus interpretará que deve utilizar cada índice unique que tenha sido definido na tabela para controlar a unicidade destes valores. Ou seja, se for ingressado um novo cliente e o usuário digitar um DNI que já existe para outro cliente, a transação disparará um erro informando sobre esta situação e não permitirá gravar o novo registro.

Outros índices que o desenvolvedor debe criar para otimizar consultas



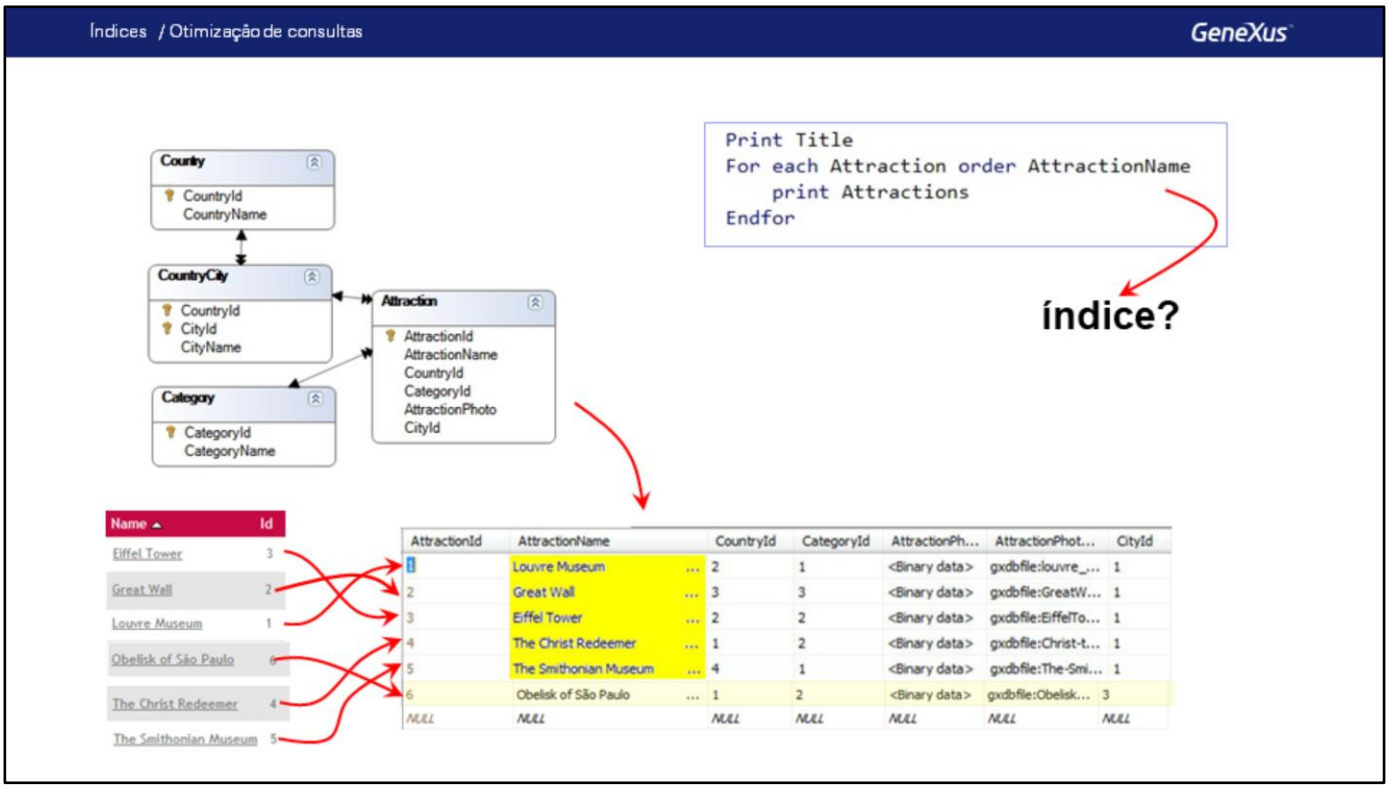
Já havíamos visto que se adicionarmos uma cláusula order para ordenar por nome de atração, a lista de navegação nos dá um aviso, informando-nos que na base de dados não existe um **índice** pelo atributo que necessitamos ordenar a informação e que podemos ter uma baixa performance para esta consulta.

É que ao selecionar um atributo para ordenar o GeneXus tenta que a ordenação seja eficiente e portanto tenta buscar se existe um índice para esse atributo. Como não o encontra, nos alerta.



Suponhamos que a tabela ATTRACTION tem os dados mostrados. Se necessitamos obter seus registros ordenados pelo atributo AttractionName, então os registro terão que ser reordenados já que por padrão estão ordenados pelo atributo que é a chave primária.

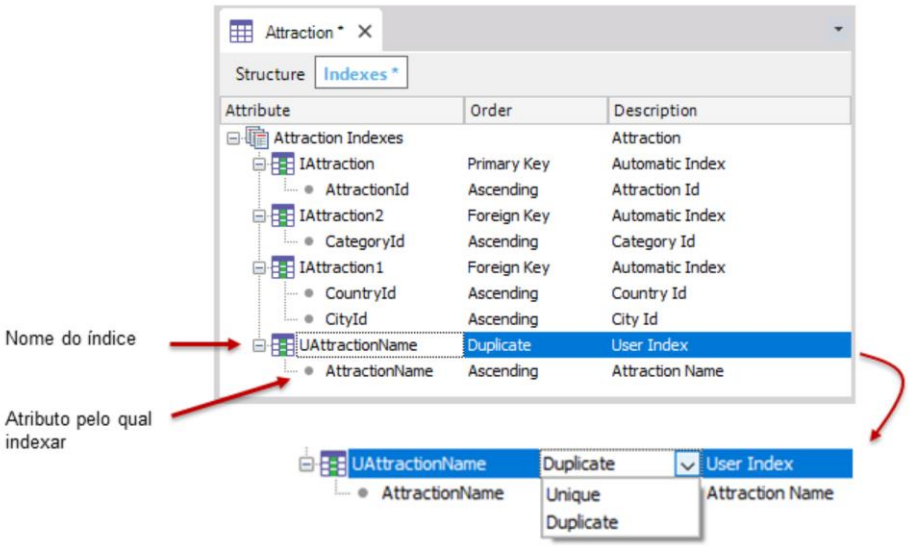
Quando uma consulta é feita, se há um índice físico criado na tabela para o atributo a ordenar, GeneXus o usará. Porém neste caso a consulta necessita ser ordenada por um atributo secundário: AttractionName. E GeneXus nos adverte na lista de navegação associado ao objeto que não há um índice definido.



A existência de índice otimizaria a consulta. Porém a desvantagem de criar um índice é que, a partir daí, deve ser mantido na base. Ou seja, se os usuários vão inserindo, modificando ou eliminando atrações na tabela ATTRACTION, debe-se reacomodar o índice (ou seja, os ponteiros do índice devem reajustar-se no BD de maneira a ter incluídas as novas atrações, de onde correspondam na ordem a ser mantida).

Criar um índice a partir do GeneXus para uma tabela da base de dados é simples e pode ser feito a qualquer momento. E assim como os criamos, podemos eliminá-los a qualquer momento.

Definição de índices de usuário (unique ou duplicate)



Definir um índice para uma tabela da base de dados é simples e pode ser feito a qualquer momento.

Como? Buscamos a tabela, abrimos e vamos na seção relacionada aos índices definidos.

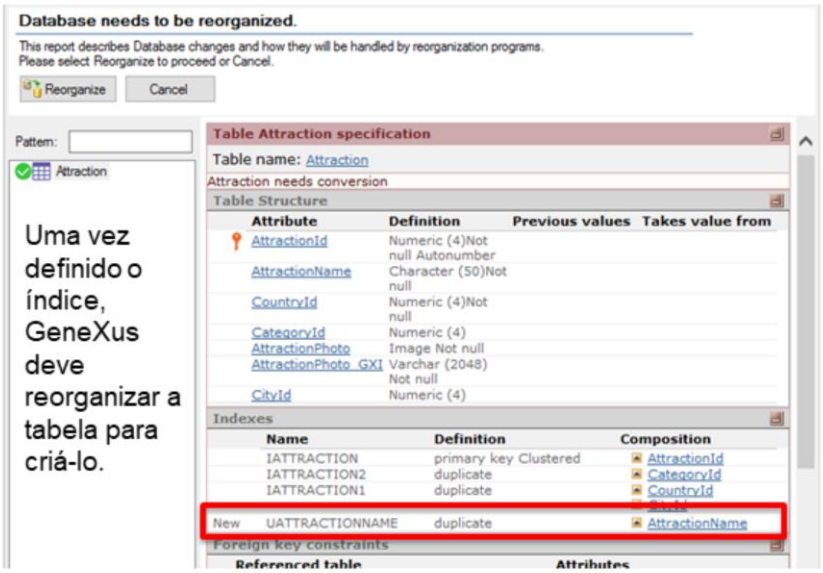
Os três primeiros que vemos no exemplo : que aparecem precedidos pelo prefixo "I", são os criados automaticamente por GeneXus a partir das chaves primária e estrangeiras.

Necessitamos criar um nosso, ou seja um índice de usuário. Para isso pressionamos enter, e já aparecerá por padrão o nome UAttraction. O modificamos a nosso gosto (agregando-lhe Name ao final, por exemplo). O prefixo "U" é por User.

Desejamos que este índice esteja composto pelo atributo AttractionName, ordenado no sentido ascendente.

Se for um requisito que os nomes das atrações não possam se repetir, podemos controla-lo indicando que o índice seja **Unique**, e não **Duplicate**, como já vimos. Se definimos para um índice que este seja Unique, **automaticamente** o BD controlará ao inserir uma atração (ou ao modificar seu nome), que não exista outra com o mesmo nome –utilizando este índice–. Em nosso exemplo os nomes podem se repetir (por exemplo pensemos que cada país geralmente tem um Obelisco), assim que para este índice por AttractionName, desejamos o valor: Duplicate.

Outros índices que o desenvolvedor debe criar para otimizar consultas



Uma vez feito isto , pressionar F5 afim de reorganizar a base de dados, para criar esse novo índice. Recordemos que a lista de navegação nos informava antes que não tínhamos índice para satisfazer a consulta, e vejamos o que dirá logo após a reorganização ...

Outros índices que o desenvolvedor deve criar para otimizar consultas

Nos informa que utilizará o índice que se acaba de ser criado.

Assim como o criamos, a qualquer momento podemos elimina-lo, e ao fazer o o F5 e reorganizar a base, voltaremos a situação que estávamos antes de tê-lo criado.

A decisão de se criar ou não o índice dependerá do DBMS que conta com a frequência com são executadas as consultas que devem ser ordenadas por AttractionName, e da frequência que os dados da tabela são atualizados.

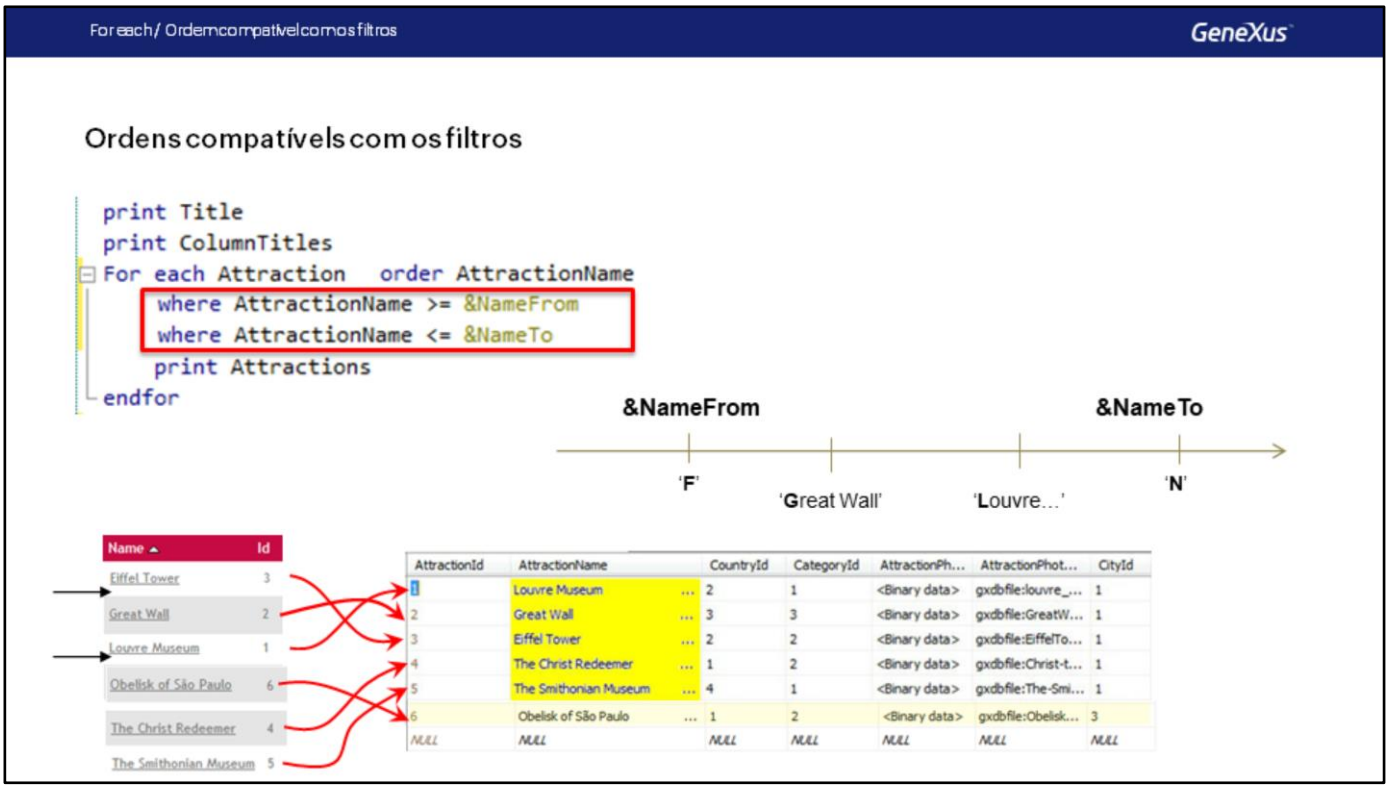
Ordem descendente

- Como fazemos para ordenar a lista de atrações pelo nome da atração, porém em ordem alfabética inversa?
- Colocar parênteses nos atributos que desejamos ordenar de forma descendente:

```
print Title
print ColumnTitles
For each Attraction order (AttractionName)
    print Attractions
endfor
```

Como fazemos para solicitar uma ordem descendente? Simplesmente inserindo os parêntesis no(s) atributo(s).

Ordens compatíveis com os filtros



Suponhamos que o que nos interessa é obter uma listagem das atrações cujos nomes estejam alfabéticamente entre um par de valores recebidos por parâmetro. Por exemplo, entre "F" e a letra "N".

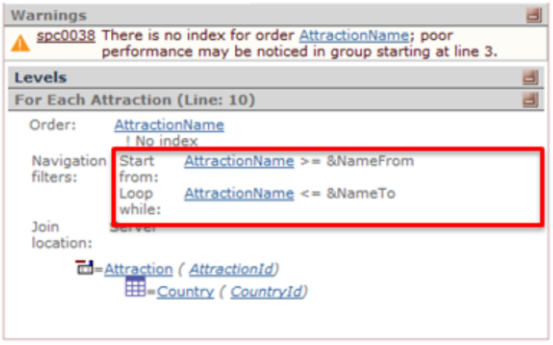
Para isso especificamos as cláusulas where vistas acima.

Ter varias cláusulas where é equivalente a ter somente uma, onde as condiciones se somam com operador lógico "and". Ou seja, são considerados somente os registros que cumpram com todas as condições **de uma vez**.

Se vamos filtrar por AttractionName, e temos um índice criado para esse atributo, é recomendado sempre **ordenar por AttractionName** para otimizar a consulta. Ao fazer isso,

Ordenar de forma compatível com os filtros

```
print Title
print ColumnTitles
For each Attraction order AttractionName
    where AttractionName >= &NameFrom
    where AttractionName <= &NameTo
    print Attractions
endfor
```



Observemos que ordenando pelo atributo que estamos filtrando usando menor ou igual e maior ou igual faz com que não seja percorrida toda a tabela. No caso de existir índice criado pelo desenvolvedor o GeneXus utiliza esse índice e a consulta estará otimizada.

No caso de não existir índice, e dependendo do DBMS, será criada temporariamente um índice que logo depois de utilizado será apagadoMas os gerentes geralmente têm estratégias de otimização que podem não exigir a criação desses índices temporários. Nós não vamos nos aprofundar nisso.

Ordenar de forma compatível com os filtros

```
print Title
print ColumnTitles
For each Attraction order AttractionName
    where AttractionName >= &NameFrom
    where AttractionName <= &NameTo
    print Attractions
endfor
```

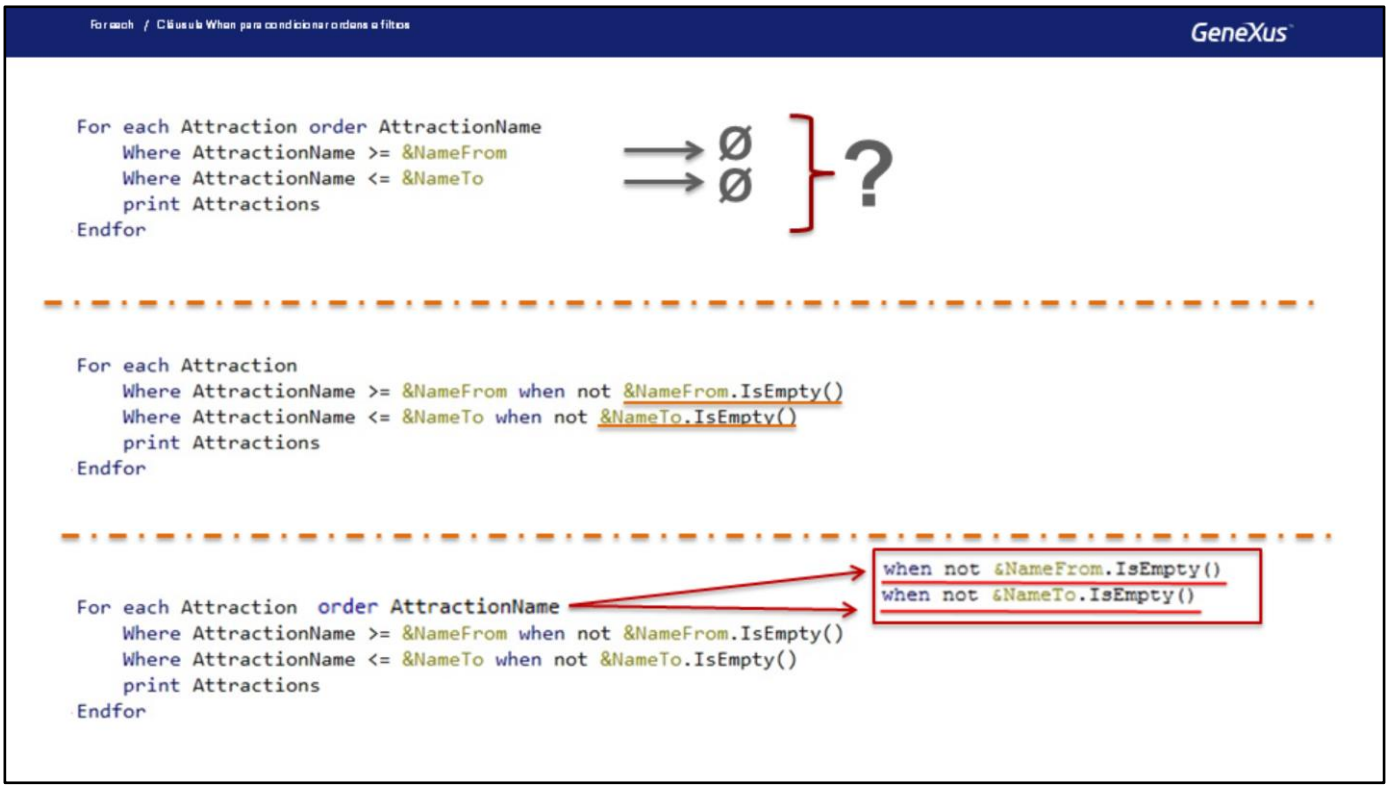
Consulta não otimizada!

Percorrerá toda a tabela...

...aplicando para cada registro estas restrições

Observemos que se não especificamos cláusula order, GeneXus ordenará por chave primária, e deverá percorrer toda a tabela para saber se uma atração está dentro dos critérios do where ou não.

Poder condicionar a execução das ordens e dos filtros.
Cláusulas When



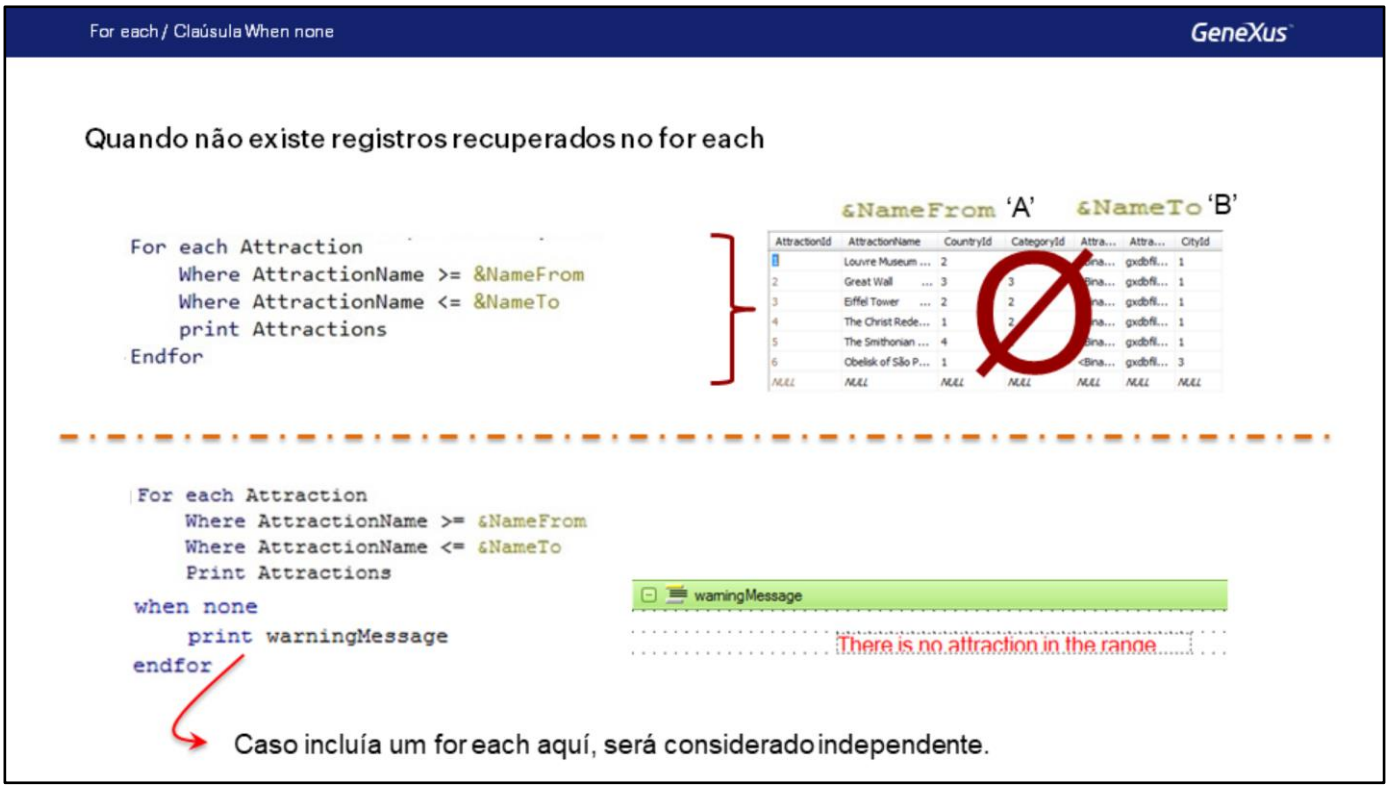
Que resultado o for each acima terá se as variáveis &NameFrom y &NameTo estão vazias? Caso exista uma atração com nome vazio, será a única a ser devolvida, pois será a única que atende as condições. Caso contrário, nenhuma atração será mostrada.

É possível condicionar as ordens e os filtros, para que somente sejam aplicados ante a determinadas circunstâncias? Por exemplo, que somente se aplique o primeiro where **quando** a variável &NameFrom não esteja vazia. E que somente seja aplicado o segundo where **quando** a variável &NameTo não esteja vazia. A resposta é sim. Conseguimos isso condicionando as cláusulas where com **when**, como vemos no segundo for each. Cada where somente será aplicado quando a condição do when seja satisfeita. Assim, na execução, quando desejemos ambas variáveis vazias, não será aplicado nenhum dos where, então todas as atrações da Tabela serão listadas. Se a variável &NameFrom está vazia porém &NameTo não, não será aplicado o primeiro where porém devido ao segundo, serão listadas todas as atrações cujo nome seja 'menor' ou igual a &NameTo.

Da mesma maneira pode-se condiciona a aplicação ou não de um order, como mostramos no terceiro for each. Na verdade podemos especificar uma sucessão de ordens condicionados, de maneira que o primeiro cuja condição se satisfaça seja o escolhido.

Veja mais sobre ordenação e filtros na wiki de GeneXus (ie: http://wiki.genexus.com/commwiki/servlet/wiki?6075,Order+clause).

Cláusula When none



O que acontece quando nenhum dos registros da tabela base cumpre com as condições?

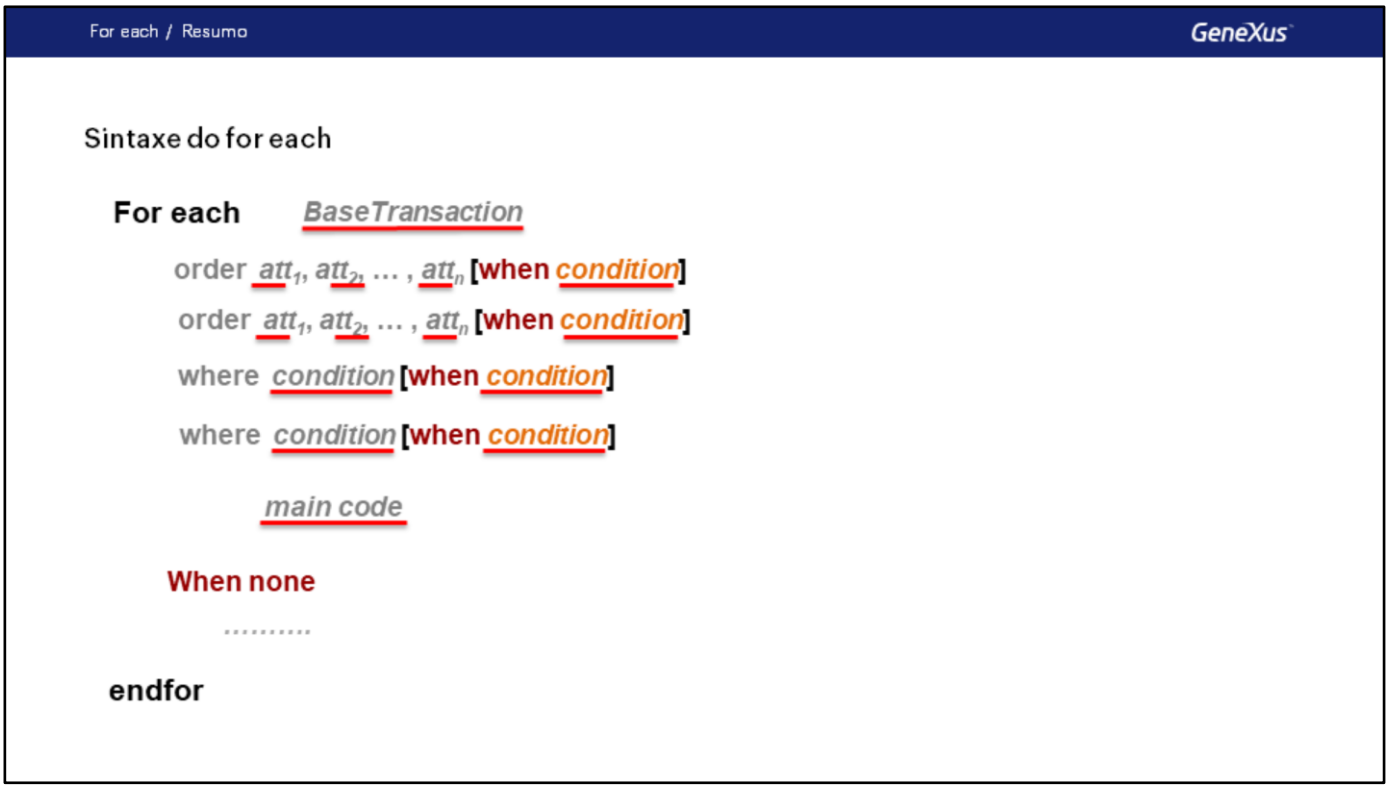
Suponhamos que queremos nesse caso imprimir ao sair uma mensagem que nos avise ... para isso programamos a cláusula **when none**.

Todos os comandos escritos entre o when none e o endfor são executados sequencialmente e **e somente caso não se tenha encontrado registros da tabela base do for each que cumpriram as condições.**

No nosso caso decidimos por imprimir uma mensagem, porém podemos escrever comandos, como outro for each, por exemplo.

Como a execução do que siga ao **when none** implicará que não foi encontrado o que se buscava, si ali for escrito um for each, este não se alinhará ao do when none. Será como um for each independente.

Resumo



Como já tínhamos visto, a tabela **base** de um For each é determinada a partir da transação base especificada; o resto dos atributos mencionados, tanto no corpo do For each (main code) como nas cláusulas Order e Where, deverão pertencer à tabela estendida dessa tabela base (por isso aparecem sublinhados na sintaxe que apresentamos acima).

Os atributos mencionados no bloco When none não são considerados.

Nós deixamos em cinza tudo o que já tínhamos visto antes.. Aqui agregamos as cláusulas **when** y **when none**.

Mais adiante veremos mais cláusulas agregadas a este fundamental comando de acesso á base de dados.

GeneXus™

The power of doing.

Vídeos	training.genexus.com
Documentação	wiki.genexus.com
Certificações	training.genexus.com/certifications