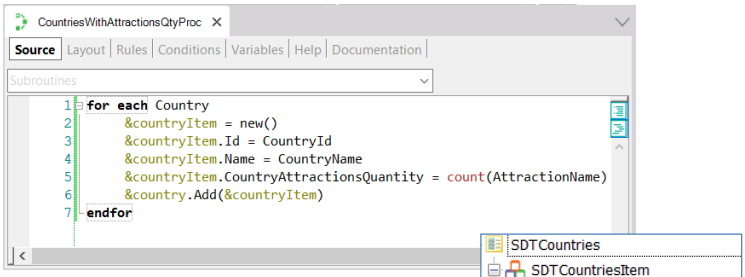
Lógica de consulta à base de dados em GeneXus

For each: uma visão integradora

GeneXus™

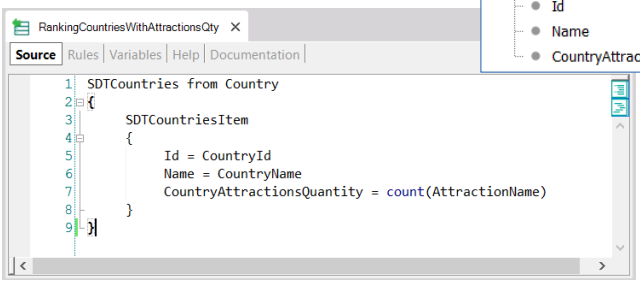
Aqui vamos nos concentrar em revisar a sintaxe do comando For each, mas com a ideia de estender seu escopo para outras formas de consulta.

For each



Navigation groups

DP Group



Grids

O fato é que o comando For each é dentro do GeneXus o paradigma de acesso à base de dados. Isso significa que basicamente sua lógica será válida para outras formas de acesso, como são os grupos de data providers ou os grids com tabela base em painéis ou web panels. Para nos referirmos genericamente a qualquer uma destas formas, às vezes utilizamos a expressão "**grupos de navegação**".

Claro que haverá algumas diferenças; por exemplo, a linguagem dos Data Providers é declarativa e para retornar uma saída estruturada, portanto, não é programado da mesma forma o corpo de um for each que um grupo de Data Provider. Mas as cláusulas que podem ser aplicadas são quase as mesmas… aqui vemos a transação base….

For each → *BaseTrn*
- **skip** *exp* **count** *exp*
- **order** *att…*
- **unique** *att…*
- **using** *DataSelector(parm…)*
- **where** *condition* when *condition*
- **blocking** *n*

Navigation groups

DP Group → *BaseTrn*
- **skip** *exp* **count** *exp*
- **order** *att…*
- **unique** *att…*
- **using** *DataSelector(parm…)*
- **where** *condition* when *condition*

Grids →
- **Base Trn** property
- **Order** property
- **Conditions** property
- **Unique** property
- **Data Selector** property

… mas também temos todas estas outras. A blocking, como veremos, só se aplica a For Eachs (e a for eachs que atualizam ou eliminam).

Os Grids oferecem em propriedades várias destas cláusulas, e o que corresponderia ao corpo do for each é escrito no evento Load correspondente (também Refresh no caso de Panels).

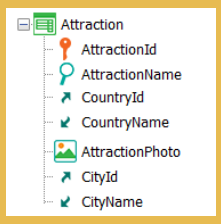
Também, como revisaremos, embora mais restritiva, temos uma consulta à base de dados ao executar um Data Selector em uma cláusula Where com o operador In.

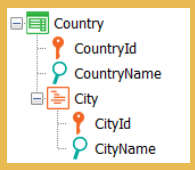
```
For each    BaseTrn1, … , BaseTrnn
        skip expression1 count expression2
        order  att1, att2, … , attn [when condition]
        order att1, att2, … , attn [when condition]
        order none [when condition]
        unique att1, att2, … , attn
        using DataSelector ( parm1, parm2, … , parmn )
        where  condition [when condition]
        where  condition [when condition]
        where  att  IN DataSelector ( parm1, parm2, … , parmn )
        blocking n
                main_code
        when duplicate
                when_duplicate_code
        when none
                when_none_code
endfor
```

Então, vamos revisar todas as partes do comando For each para, em seguida, aprofundar em algumas delas.

For each $BaseTrn_1, \dots, BaseTrn_n$

```
For each Attraction

    print info //CountryName

endfor
```

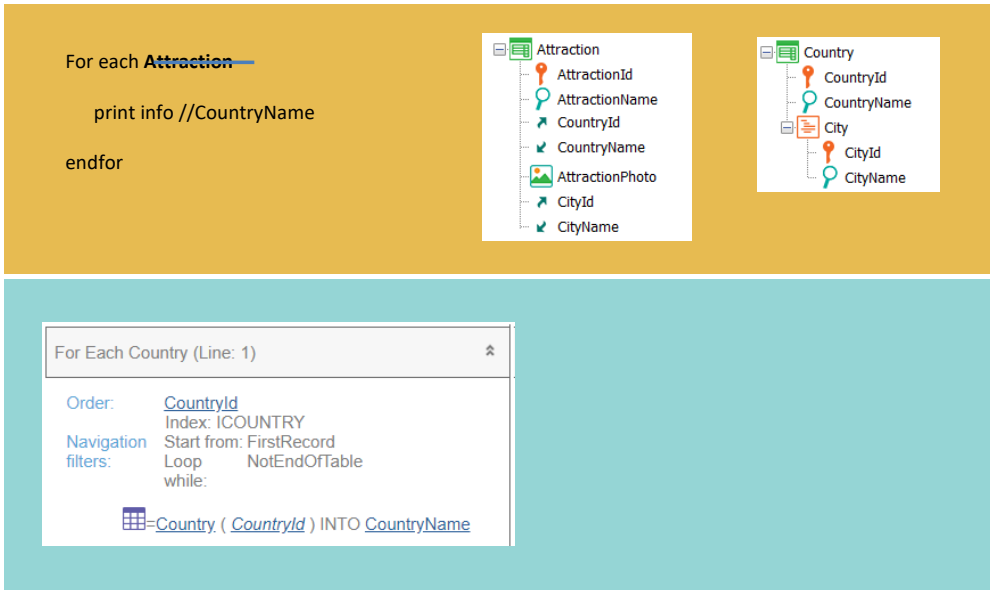




endfor

A transação base é opcional e é utilizada para indicar qual queremos que seja a tabela base do for each, ou seja, a tabela que será acessada para retornar um conjunto de registros. Nos cursos anteriores sempre a utilizamos, colocando um único valor ali.
Neste exemplo estamos indicando que queremos percorrer todas as atrações, e para cada uma queremos imprimir o nome de seu país (para o que deverá acessar também a tabela Country e vemos que é indicada a localização desse join como ocorrendo no server, ou seja, será resolvido pelo DBMS).

For each $BaseTrn_1, \ldots, BaseTrn_n$



endfor

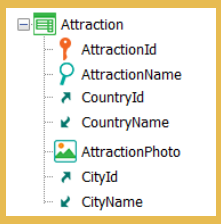
Quando não se especifica transação base, GeneXus deverá determinar a tabela a ser navegada por seus próprios meios, de acordo com os atributos indicados nos demais lados. No exemplo, como o único atributo dentro do for each é CountryName, escolherá a tabela base Country e, portanto, imprimirá todos os nomes de países dessa tabela e aqui vemos um caso onde declarar transação base ou não fazê-lo altera o comportamento.

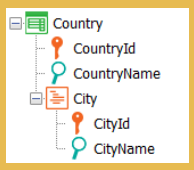
For each $BaseTrn_1, \dots, BaseTrn_n$

For each **Attraction**

print info //CountryName

endfor

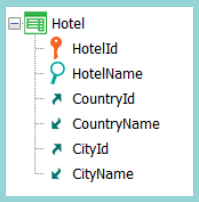




For each **Attraction, Hotel**

print info //AttractionName, HotelName

endfor



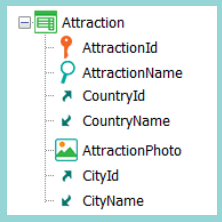
endfor

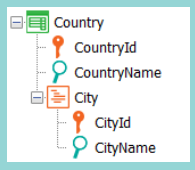
Um caso não mencionado até agora é quando especificamos mais de uma transação base. Ali, será realizado um join ou um produto cartesiano. Neste exemplo onde cada atração tem um país e cidade e cada hotel também, será realizado um join.
Será exibida uma atração com um hotel do mesmo país e cidade, depois a mesma atração com outro hotel do mesmo país e cidade, e assim sucessivamente até esgotar todos os hotéis do mesmo país e cidade da atração; e então vai para a próxima atração e se repete a mesma coisa: é listada repetida, mas variando cada vez o hotel (daqueles que são do mesmo país e cidade). Se não tivessem essa relação, por exemplo, se os hotéis não tivessem país e cidade, então cada atração seria listada com cada hotel da tabela, portanto, seria feito um produto cartesiano.

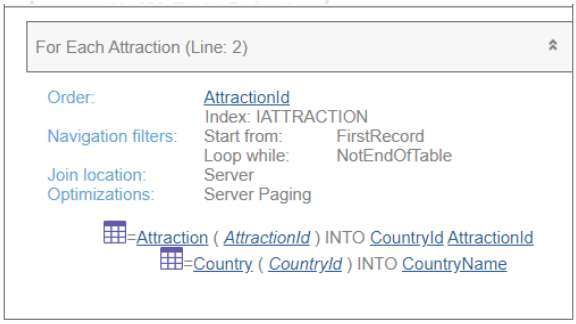
For each $BaseTrn_1, \ldots, BaseTrn_n$

skip $expression_1$ **count** $expression_2$

```
For each Attraction
    skip 5 count 10

    print info //CountryName

endfor
```







Vamos passar para a cláusula opcional skip em combinação com count. Permite pular os primeiros n registros (sendo n o resultado de avaliar esta expressão numérica) e a partir daí ficar com os próximos m registros (m o resultado desta outra expressão).

Neste exemplo, são ignoradas as primeiras 5 atrações da consulta, para ficar com as próximas 10.

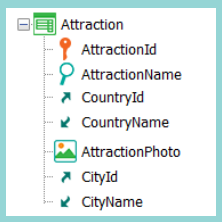
Isto é conhecido como paginação de dados. A paginação consiste basicamente em dividir a informação resultante de uma consulta em blocos menores, o que, entre outras coisas, reduz a quantidade de dados enviados do servidor de base de dados para o programa.

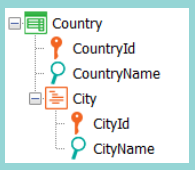
Na lista de navegação aparecerá uma indicação de otimização, onde vemos que esta paginação será realizada no servidor, ou seja, que será feita pelo próprio DBMS. Podemos ficar tranquilos.

For each $BaseTrn_1$, … , $BaseTrn_n$

skip *expression*$_1$ **count** *expression*$_2$
order att_1, att_2, … , att_n [**when** *condition*]
order att_1, att_2, … , att_n [**when** *condition*]
order none [**when** *condition*]
unique att_1, att_2, … , att_n
using *DataSelector* ($parm_1$, $parm_2$, … , $parm_n$)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* ($parm_1$, $parm_2$, … , $parm_n$)

```
For each Attraction
    order CountryId when not &country.IsEmpty()
    order AttractionName
    where CountryId >= &country when not &country.IsEmpty()

            print info //AttractionName, CountryName

endfor
```





Poderíamos então especificar uma lista de cláusulas order condicionais com no máximo uma incondicional no final, para ordenar a informação a ser consultada e retornada. As condições também nos permitiam jogar com as cláusulas where condicionais, de forma a poder ordenar pelos mesmos critérios de filtro e, conseguir especificar assim, consultas mais otimizadas.

Neste exemplo, se a condição da primeira cláusula order for satisfeita, ou seja, a variável &country não estiver vazia, será ordenada por CountryId e a cláusula order seguinte não será considerada. Como neste caso a condição da where é a mesma, a where será aplicada e o filtro de país estará, então, otimizado.

Em vez disso, se a condição da primeira cláusula order não for satisfeita, é investigada a segunda cláusula order, que por ser incondicional será aplicada. Como a where não será aplicada neste caso, uma vez que a variável &country está vazia, serão exibidas todas as atrações ordenadas por nome de atração (e seguramente desordenadas por país).

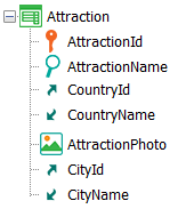
A cláusula order none, que não tínhamos visto antes, é escrita quando queremos que a ordem a ser aplicada fique indefinida, o que significa que dependerá da plataforma e pode até variar de uma execução para outra.

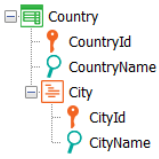
De qualquer forma, as cláusulas order que escrevemos não determinam exatamente o plano de execução que escolherá o DBMS, pois também são levadas em conta outras considerações, justamente para otimizar o acesso à base de dados. Vamos nos aprofundar neste assunto em um vídeo separado.

Sabemos também que a cláusula order tem uma relevância fundamental quando é desejado implementar um corte de controle entre for eachs aninhados. Neste caso, não é utilizado apenas para ordenar a informação, mas também, e mais importante, para estabelecer o critério de corte. Ali, a cláusula order não pode ser condicional.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

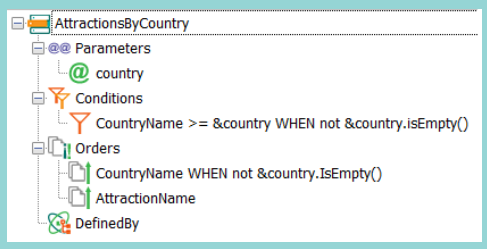
skip *expression$_1$* **count** *expression$_2$*
order *att$_1$, att$_2$, … , att$_n$* **[when** *condition***]**
order *att$_1$, att$_2$, … , att$_n$* **[when** *condition***]**
order none [when *condition***]**
unique *att$_1$, att$_2$, … , att$_n$*
using *DataSelector* **(** *parm$_1$, parm$_2$, … , parm$_n$* **)**






```
For each Attraction
    order CountryName when not &country.IsEmpty()
    order AttractionName
    where CountryName >= &country when not &country.IsEmpty()
            print info //AttractionName, CountryName
endfor
```

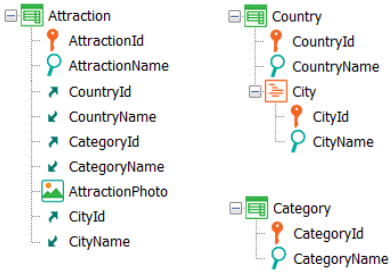
```
For each Attraction
    using AttractionsByCountry(&country)
            print info //AttractionName, CountryName
endfor
```



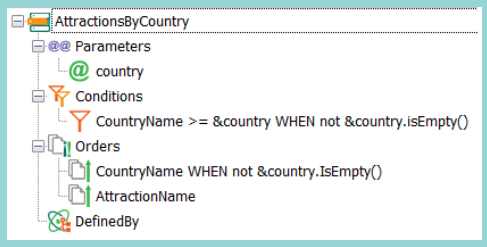
A cláusula using permitia incorporar mais ordens e filtros, mas centralizados em um objeto Data Selector, o que nos permite enviar parâmetros para ele. Desta forma, não precisaríamos repetir esses mesmos critérios de classificação e de filtro explicitamente em cada consulta. Para todos os efeitos do For each, é como se tivessem sido escritos explicitamente. Não há nenhuma diferença. De fato, na lista de navegação não será possível diferenciar uma forma da outra. Será visto exatamente o mesmo e não haverá nenhuma indicação do Data Selector.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$, att$_2$, … , att$_n$* **[when** *condition***]**
order *att$_1$, att$_2$, … , att$_n$* **[when** *condition***]**
order none [when *condition***]**
unique *att$_1$, att$_2$, … , att$_n$*
using *DataSelector* **(** *parm$_1$, parm$_2$, … , parm$_n$* **)**



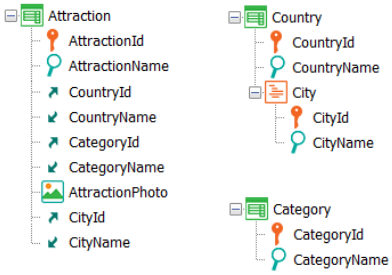

```
For each Attraction
    order CountryName when not &country.IsEmpty()
    order AttractionName
    where CountryName >= &country when not &country.IsEmpty()
    where CategoryName <> "Monument"
            print info //AttractionName, CountryName
endfor

For each Attraction
    using AttractionsByCountry(&country)
    where CategoryName <> "Monument"
            print info //AttractionName, CountryName
endfor
```



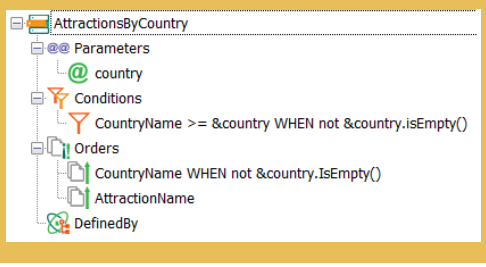
Isto significa que podem coexistir cláusulas order, where e using sem problema algum. Para o programa gerado não haverá nenhuma diferença entre este for each e este outro.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]
order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$, att$_2$, … , att$_n$*
using *DataSelector* (*parm$_1$, parm$_2$, … , parm$_n$*)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* (*parm$_1$, parm$_2$, … , parm$_n$*)
blocking *n*
main_code
when duplicate
when none
when_none_code
endfor




```
For each Category
    where CategoryId IN AttractionsByCountry(&country)
            print info //CategoryName
endfor
```



A diferença aparece quando é feito outro uso, desta vez especial, do Data Selector: quando ele é utilizado como consulta executável.

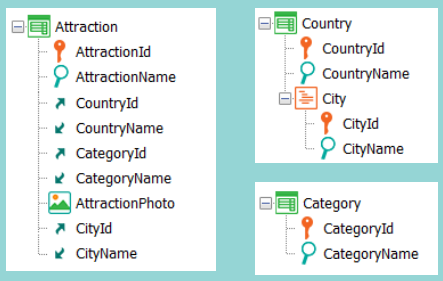
Isto acontece quando queremos filtrar indicando que queremos apenas ficar com os registros da tabela estendida do for each para os que o valor de um determinado atributo esteja entre os retornados por essa consulta independente.

No exemplo quando queremos percorrer a tabela de categorias, ficando unicamente com aquelas que estão entre as categorias das atrações da consulta independente do Data Selector. A consulta do DataSelector, pelos atributos envolvidos, claramente é das atrações de países com nome posterior ao do filtro (se não estiver vazio, se estiver vazio, todas). Devemos pensar no Data Selector neste caso como se fosse outro For each. É por isso que os atributos que se encontrem dentro do Data Selector para este caso não terão nenhuma implicação para o for each em que está sendo utilizado. Por isso a tabela base do for Each será Category, e a da consulta do Data Selector Attraction.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]
order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$, att$_2$, … , att$_n$*
using *DataSelector* (*parm$_1$, parm$_2$, … , parm$_n$*)
where *condition* [**when** *condition*]

```
For each Attraction
    unique CategoryId
        print info //CategoryName
endfor

For each Attraction
    unique CategoryId
        &qty = count( AttractionName )
        print info //CategoryName &qty
endfor
```



Também vimos a cláusula unique, de forma que se os atributos indicados se repetem para um conjunto de registros do for each, apenas um seja levado para processar no corpo do for each, ou seja, no código principal.

Aqui estamos percorrendo a tabela de atrações, agrupando-as por id de categoria e ficando apenas com um registro do grupo, imprimindo seu nome de categoria.

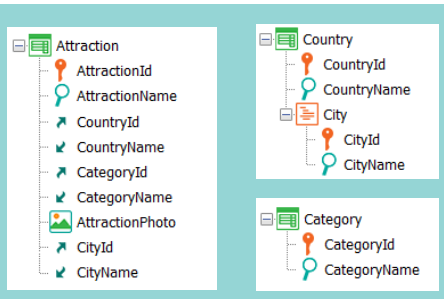
Vimos também que esta cláusula era extremamente útil quando, no código do for each, queríamos fazer uma agregação sobre esse conjunto de registros repetidos para esses atributos unique. Algo que, de outra forma, não poderíamos conseguir (que o for each e a fórmula de agregação trabalhassem sobre a mesma tabela). Neste exemplo, o for each percorre Attraction, a fórmula Count também, e tendo especificado a cláusula unique por CategoryId, a contagem será realizada para o conjunto de registros com o mesmo CategoryId em que se está posicionado cada vez.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Category
   …
      print info //CategoryName
      For each Attraction
         print info2 // AttractionName, CountryName, CityName
      endfor
endfor
```



main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

No bloco de código principal é onde programamos o que queremos realizar com cada registro da tabela base que passou pelos filtros (claro que, se ali são utilizados atributos da tabela estendida, é realizado automaticamente o acesso a esses registros e se trabalha com eles).

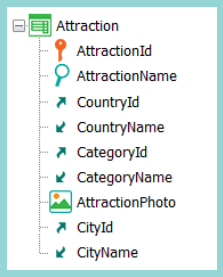
Entre tudo que pode ser programado aqui, está a escrita de outro comando For each, aninhado…

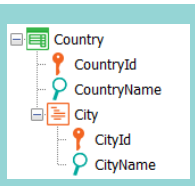
For each *BaseTrn$_1$, … , BaseTrn$_n$*

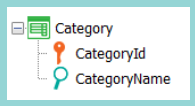
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Category
   …
      print info //CategoryName
      Do 'Something'
      print info3

endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

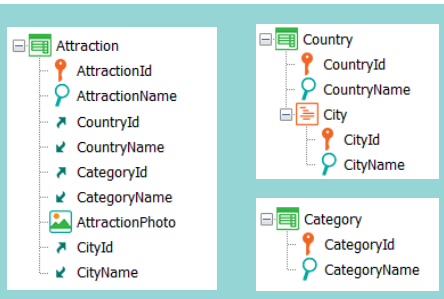
…uma invocação de uma sub-rotina ou…

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Category
   …
      print info //CategoryName
      MyProcedure( CategoryId )
      print info3

endfor
```



main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

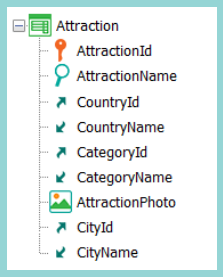
…para um procedimento, que quando executado retornará ao que segue a invocação.

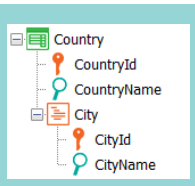
For each *BaseTrn$_1$, … , BaseTrn$_n$*

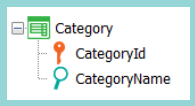
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
    where CountryName = “France”
        print info //AttractionName, CategoryName
    when none
        print infoNone      //”No attractions from France”
endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

Se não houver nenhum registro que passe pelos filtros, então passará a executar o código especificado na cláusula when none, se foi especificado, é claro.

Neste exemplo, será quando não houver atração de país com esse nome.

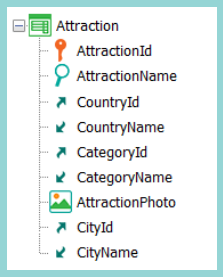
Como ao executar esta cláusula assume-se que não se conseguiu fazer nada com os dados da tabela estendida do for each, aqui não estamos posicionados em nenhum registro, então…

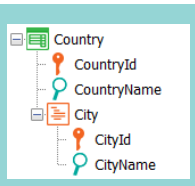
For each *BaseTrn$_1$, … , BaseTrn$_n$*

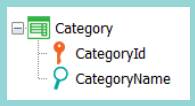
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
parm(in:CategoryName);

For each Attraction
    where CountryName = “France”
        print info //AttractionName, CategoryName
    when none
        print infoNone      //CategoryName
endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

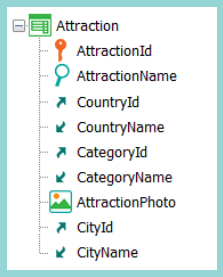
…se ali forem nomeados atributos, de onde eles serão obtidos? Só faz sentido nomear atributos se eles estão instanciados de antemão (por exemplo na regra parm do objeto onde se encontra este for each…

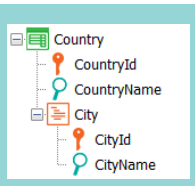
For each *BaseTrn$_1$, … , BaseTrn$_n$*

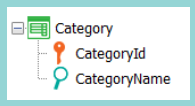
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Category
    For each Attraction
        where CountryName = "France"
            print info //AttractionName, CategoryName
        when none
            print infoNone      //CategoryName
    endfor
endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

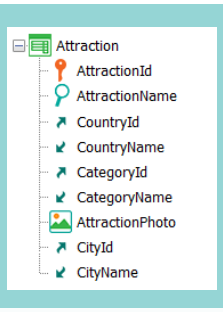
... ou em outro for each ao qual ele está aninhado).

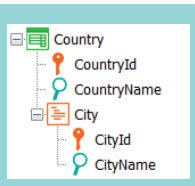
For each *BaseTrn$_1$, … , BaseTrn$_n$*

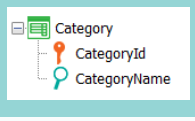
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
    where CountryName = “France”
        print info //AttractionName, CategoryName
    when none
        For each Country.City
            print info2 //CountryName, CityName
        endfor
endfor
```








```
For each Attraction
    where CountryName = “France”
        print info //AttractionName, CategoryName
endfor
For each Country.City
    print info2 //CountryName, CityName
endfor
```

Claro que aqui podemos escrever outra consulta (outro for each), utilizar uma fórmula inline, etc., mas é como se estivessem escritas após o For each.

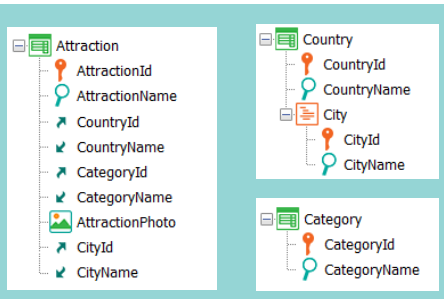
For each *BaseTrn$_1$, … , BaseTrn$_n$*

Procedure Object

skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
   where CountryName = “France”

      AttractionName = “France “ + AttractionName
      CategoryName = “France Attractions”

endfor
```



main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

Se o for each estiver dentro de um procedimento e somente nesse caso, no código principal também será possível atribuir valor a um atributo ou a vários, para assim atualizar o registro da tabela base e o ou os da estendida que correspondam.

Aqui é atualizado o atributo AttractionName do registro da tabela base em que estamos em cada iteração, e o atributo CategoryName do registro associado da tabela Category, que faz parte da estendida.

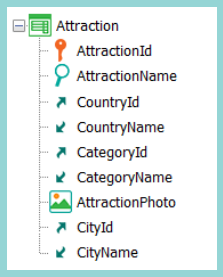
For each *BaseTrn$_1$, … , BaseTrn$_n$*

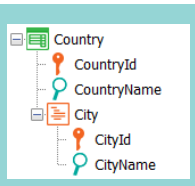
Procedure Object

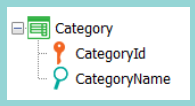
skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
    where CountryName = “France”

        Delete

endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

Também podemos escrever um comando Delete para remover o registro em que estamos posicionados (e remove apenas o registro da tabela base, sem realizar verificações de integridade referencial).

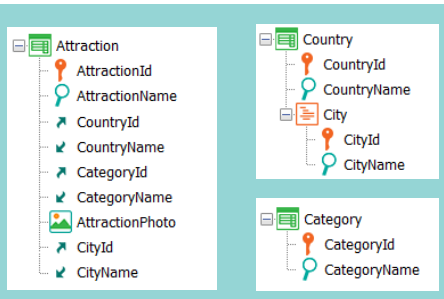
For each *BaseTrn$_1$, … , BaseTrn$_n$*

Procedure Object

skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
    where CountryName = “France”
        new
            CategoryName = “Famous Landmark”
        endnew
endfor
```



main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

É claro que também podemos escrever um comando new para inserir um registro em uma tabela, mas isso fica por fora do comportamento do for each, que comanda tanto a atualização quanto a exclusão.

Procedure Object

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*

order *att$_1$, att$_2$, … , att$_n$* [**when** *condition*]

```
For each Attraction
    where CountryName = "France"
    blocking 10
        AttractionName = "France " + AttractionName
        CategoryName = "France Attractions"
endfor
```

main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

Syntax

```
New
   [Defined by attributeList]
   [Blocking NumericExpression]
             BodyCode
[When duplicate
   { AnotherCode |
       For each
         {att = exp}
         …
       Endfor
     | AnotherCode } ]
 EndNew
```

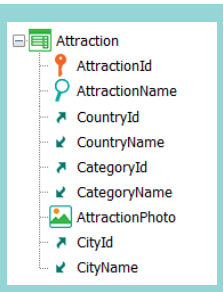
Para qualquer um destes dois casos, temos a cláusula blocking. Permite que as operações sejam realizadas em um buffer e quando forem processados os n registros indicados na cláusula (neste exemplo 10), somente então serão realizadas as operações sobre a base de dados. Ou seja, é feito um único acesso à base de dados para processar de uma só vez os n registros que estão sendo atualizados ou excluídos, melhorando assim o desempenho.

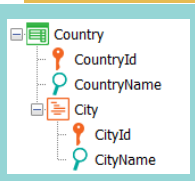
Como esperado, esta mesma cláusula também está disponível no comando New.

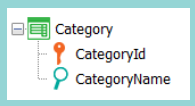
For each $BaseTrn_1, \dots, BaseTrn_n$

Procedure Object

```
For each Attraction
    where CountryName = "France"

        AttractionName = "France " + AttractionName
        CategoryName = "France Attractions"

    when duplicate
        For each
            AttractionName = "France " + AttractionName
        endfor
endfor
```







AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redemmer	1	2	2

CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site
4	France Attractions
5	Famous Landmark

endfor

Por último, a cláusula when duplicate é utilizada quando é desejado atualizar atributos no corpo do for each que são, em particular, chave candidata, portanto, não devem repetir seu valor.

Se no código principal estiver sendo modificado o valor de um desses atributos (para o registro em que está posicionado o programa naquela iteração), atribuindo-lhe um valor que já existe para outro registro da tabela, então não será permitido realizar a atualização (lembre-se que é utilizado o índice unique para realizar esse controle).

Imaginemos para este caso que já exista na tabela Category um registro com CategoryName "France Attractions" e que exista um índice unique por CategoryName, então quando para a atração 1, que é da França, deseja-se atualizar o valor de sua CategoryName, que era Museum , por "France Attractions", como será controlada a unicidade, a verificação falhará por chave duplicada. É que já existe um registro com esse valor, o 4.

Se o desenvolvedor programou cláusula when duplicate, é ali onde indica o que deve ser feito nesse caso. Se a cláusula não for escrita, nada será feito com aquele registro que se pretendia atualizar e se passa para a próxima iteração do For each, onde neste caso acontecerá a mesma coisa quando for desejado modificar o nome da categoria da torre Eiffel, que é a 2.

Se for especificada cláusula when duplicate, então, embora se possa pensar que ainda estamos posicionados no registro que causou o problema, na verdade não estamos mais para atualização, portanto, se queremos atualizar o atributo ou

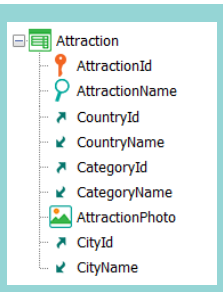
outro com outro valor, temos que escrever um for each para indicá-lo.

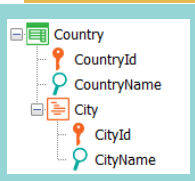
Neste caso queremos que a primeira atualização (que nunca falhará) seja realizada, então… para a primeira atração da França falha em duplicado…

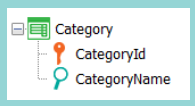
For each $BaseTrn_1, \dots, BaseTrn_n$

Procedure Object

```
For each Attraction
    where CountryName = "France"

        AttractionName = "France " + AttractionName
        CategoryName = "France Attractions"

    when duplicate
        For each
            AttractionName = "France " + AttractionName
        endfor
endfor
```







AttractionId	AttractionName	CountryId	CityId	CategoryId
1	France Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redemmer	1	2	2

CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site
4	France Attractions
5	Famous Landmark

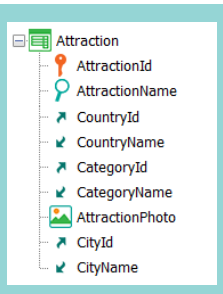
endfor

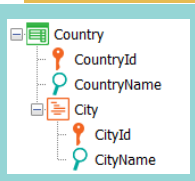
e executa esta seção. E então para a segunda e última atração da França: quer atualizar e também falha…

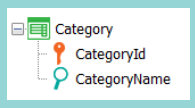
For each $BaseTrn_1, \dots, BaseTrn_n$

Procedure Object

```
For each Attraction
    where CountryName = "France"

        AttractionName = "France " + AttractionName
        CategoryName = "France Attractions"

    when duplicate
        For each
            AttractionName = "France " + AttractionName
        endfor
endfor
```







AttractionId	AttractionName	CountryId	CityId	CategoryId
1	France Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	France Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redemmer	1	2	2

CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site
4	France Attractions
5	Famous Landmark

endfor

então executa esta seção, e assim teremos a tabela.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

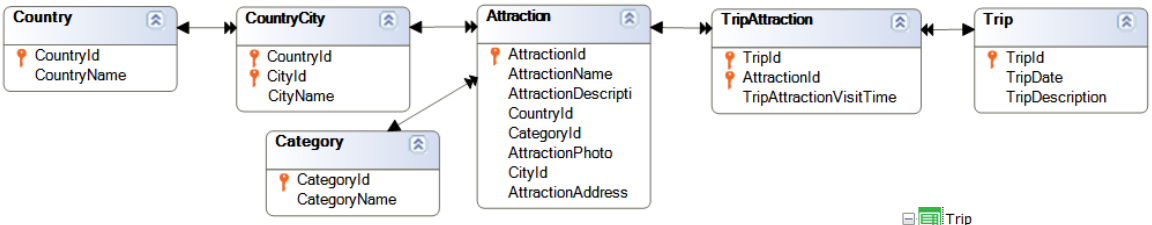
skip *expression$_1$* **count** *expression$_2$*
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$*, *att$_2$*, … , *att$_n$*
using *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
blocking *n*
main_code
when duplicate
when_duplicate_code
when none
when_none_code

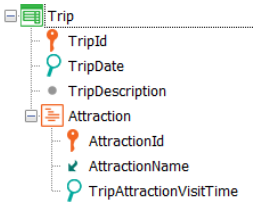
endfor

Em resumo, todas as cláusulas prévias ao código principal ou corpo do for each são utilizadas para filtrar os registros sobre os quais esse código principal será executado, para ordená-los, para agrupá-los com base na repetição de valores e processá-los uma vez, ou para indicar que serão acessados em bloco quando for desejado atualizar e/ou excluir.

Em seguida, para cada um dos registros que passem por esses filtros ou o agrupamento, e de forma ordenada de acordo com a ordem determinada, será executado o código principal.

Dos atributos que são utilizados em todas essas cláusulas, bem como dentro desse código principal, muitas vezes surge a necessidade de acessar registros de outras tabelas da tabela estendida, mas não a todos. Por motivos de desempenho, apenas esses serão recuperados, o que tem algumas consequências.






```
For each Trip.Attraction                    parm( in: CategoryName);
   order AttractionName
   where CountryName = “France”
   where TripDate >= &today

      print info //AttractionName, TripAttracitionVisitTime

endfor
```

Neste exemplo, será percorrida a tabela TripAttraction, correspondente ao segundo nível da transação Trip, mas será acessada Attraction para ordenar por AttractionName e porque é solicitado imprimir seu valor na saída, e porque a partir dali será acessada CountryCity para poder acessar Country para poder filtrar por CountryName.
Além disso, será acessada Trip para poder filtrar por TripDate.
Mas não é necessário acessar a Category. Portanto, se na regra parm recebemos no atributo CategoryName, ao contrário do que poderíamos pensar, esse filtro não será aplicado. Por quê? Porque dentro do for each não é acessada a tabela Category. Assim quando, após determinar todas as navegações do objeto onde se encontra este for each, será adicionado o filtro implícito que vem da regra parm, inspecionando cada uma das consultas, não se encontra relação para este for each e não é adicionado.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$*, *att$_2$*, … , *att$_n$*
using *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
blocking *n*
main_code
when duplicate
when_duplicate_code
when none
when_none_code

endfor

Procedure Object

```
For each Attraction
   where CountryName = "France"
      AttractionName = "France " + AttractionName
      CategoryName = "France Attractions"
   when duplicate
      For each
         AttractionName = "France " + AttractionName
      endfor
endfor
```

Para o caso em que nos encontramos em um procedimento e no código principal queremos atualizar atributos que são chaves candidatas, acontece o seguinte: Suponhamos que esteja sendo processado o registro n que passa pelos filtros, ou seja, está na iteração n do for each, onde já foi executado o código para os n-1 registros anteriores, e para esse registro ou um de seus relacionados por tabela estendida está querendo atualizar um de seus atributos, que é chave candidata, dando-lhe um valor repetido, então se não houver cláusula when duplicate, não é realizada a atualização e passa para o próximo registro da iteração, o n+1. Mas se foi especificada a cláusula when duplicate, é executada. Então passa a processar o próximo registro do for each, o n+1.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$*, *att$_2$*, … , *att$_n$*
using *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
blocking *n*
main_code
when duplicate
when_duplicate_code
when none
when_none_code

endfor

O código da cláusula when none só será executado quando não houve iteração alguma do for each, pois nenhum registro passou pelos filtros (ou a tabela estava vazia). Ou seja, se for executado este código, é porque nenhum destes foi executado para nenhum registro.

For each *BaseTrn$_1$, … , BaseTrn$_n$*

skip *expression$_1$* **count** *expression$_2$*
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order *att$_1$*, *att$_2$*, … , *att$_n$* [**when** *condition*]
order none [**when** *condition*]
unique *att$_1$*, *att$_2$*, … , *att$_n$*
using *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
where *condition* [**when** *condition*]
where *condition* [**when** *condition*]
where *att* **IN** *DataSelector* (*parm$_1$*, *parm$_2$*, … , *parm$_n$*)
blocking *n*
main_code
when duplicate
when_duplicate_code
when none
when_none_code

endfor

Pelo que acabamos de ver, os atributos que aparecem no código da when duplicate ou da when none, assim como os que aparecem dentro do Data Selector quando ele é executado como consulta independente, não são considerados na hora de determinar a navegação do For each.

Isto importa especialmente para o caso em que GeneXus deve determinar a tabela base do for each, já que não foi colocada transação base alguma.

Até aqui um resumo do essencial.

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