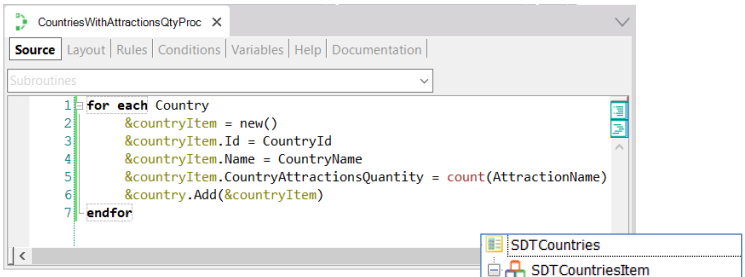
Lógica de consulta a la base de datos en GeneXus

For each: una mirada integradora

GeneXus™

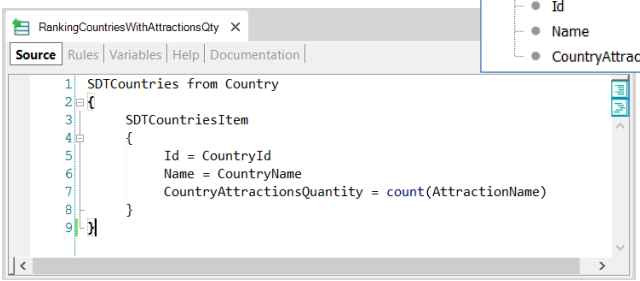
Aquí nos concentraremos en repasar la sintaxis del comando For each, pero con la idea de extender su alcance a otras formas de consulta.

For each



Navigation groups

DP Group



Grids

Es que el comando For each es dentro de GeneXus el paradigma de acceso a la base de datos. Esto quiere decir que en lo grueso su lógica va a valer para otras formas de acceso, como son los grupos de data providers o los grids con tabla base en paneles o web panels. Para referirnos genéricamente a cualquiera de estas formas a veces utilizamos la expresión "**grupos de navegación**".

Por supuesto habrá algunas diferencias; por ejemplo, el lenguaje de los Data Providers es declarativo y para devolver una salida estructurada, por lo que no se programa igual el cuerpo de un for each que un grupo de Data Provider. Pero las cláusulas que se le pueden aplicar son casi las mismas... aquí vemos la transacción 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

... pero también tenemos todas estas otras. La **blocking**, como veremos, solo aplica a For Eachs (y a for eachs que actualizan o eliminan).

Los Grids ofrecen en propiedades varias de estas cláusulas, y lo que correspondería al cuerpo del for each se escribe en el evento Load correspondiente (también Refresh si se trata de Panels).

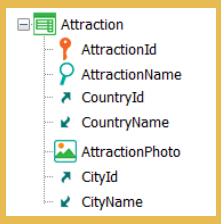
También, como repasaremos, aunque más restrictiva, tenemos una consulta a la base de datos cuando se ejecuta un Data Selector en una cláusula Where con el operador In.

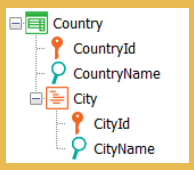
```
For each  BaseTrn1 , ... , BaseTrnn
        skip expression1 count expression2
        order  att1, att2, ... , attn [when condition]
        order att1, att2, ... , attn [when condition]
        order none [when condition]
        unique att1, att2, ... , attn
        using DataSelector ( parm1, 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
```

Repasemos, entonces, todas las partes del comando For each para luego profundizar en algunas.

For each $BaseTrn_1, \ldots, BaseTrn_n$

```
For each Attraction
    print info //CountryName
endfor
```



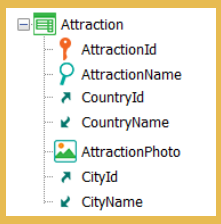


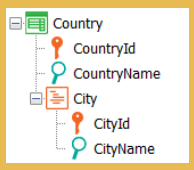
endfor

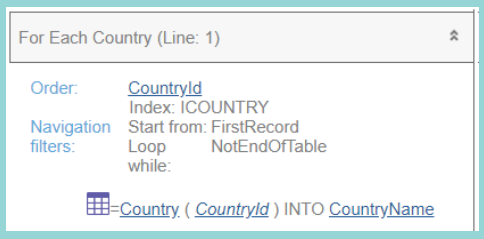
La transacción base es opcional y se utiliza para indicar cuál queremos que sea la tabla base del for each, es decir, la tabla que será accedida para devolver un conjunto de registros. En los cursos anteriores siempre la utilizamos, colocando un solo valor allí.
En este ejemplo estamos indicando que queremos recorrer todas las atracciones, y para cada una queremos imprimir el nombre de su país (para lo que deberá acceder también a la tabla Country y vemos que se indica la ubicación de ese join como teniendo lugar en el server, es decir, lo resolverá el DBMS).

For each $BaseTrn_1, \ldots, BaseTrn_n$

```
For each Attraction

    print info //CountryName

endfor
```





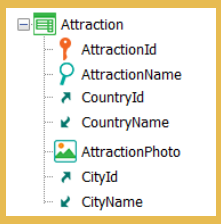


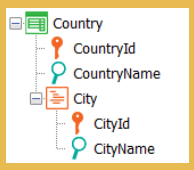
endfor

Cuando no se especifica transacción base GeneXus deberá determinar la tabla a navegar por sus propios medios, de acuerdo a los atributos indicados en los demás lados. En el ejemplo como el único atributo dentro del for each es CountryName elegirá tabla base Country y por tanto imprimirá todos los nombres de país de esa tabla y aquí vemos un caso donde declarar transacción base o no hacerlo cambia el comportamiento.

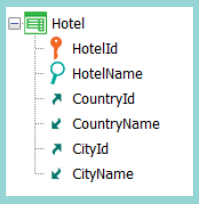
For each $BaseTrn_1, \ldots, BaseTrn_n$

```
For each Attraction
    print info //CountryName
endfor
```






```
For each Attraction, Hotel
    print info //AttractionName, HotelName
endfor
```



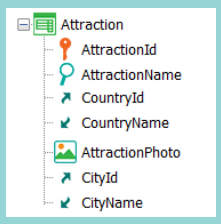
endfor

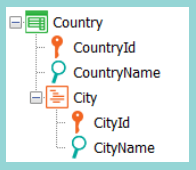
Un caso hasta ahora no mencionado es cuando especificamos más de una transacción base. Allí se realizará un join o un producto cartesiano. En este ejemplo en el que cada atracción tiene un país y ciudad y cada hotel también, se realizará un join.
Se mostrará una atracción con un hotel del mismo país y ciudad, luego la misma atracción con otro hotel del mismo país y ciudad, y así sucesivamente hasta agotar todos los hoteles del mismo país y ciudad de la atracción; y luego se pasa a la siguiente atracción y se repite lo mismo: se va listando repetida, pero variando cada vez el hotel (de los que son del mismo país y ciudad). Si no tuvieran esa relación, por ejemplo, si los hoteles no tuvieran país y ciudad entonces cada atracción saldría listada con cada hotel de la tabla, por tanto, se hará un producto cartesiano.

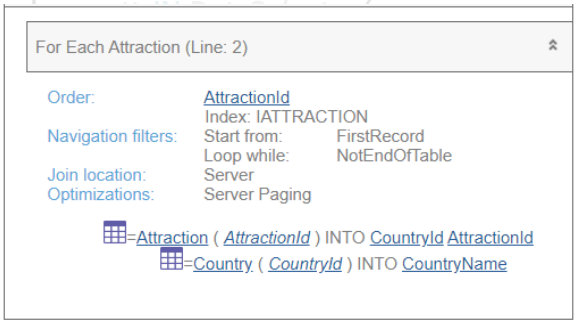
For each $BaseTrn_1$, ... , $BaseTrn_n$

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

```
For each Attraction
    skip 5 count 10

    print info //CountryName

endfor
```







Pasemos a la cláusula opcional skip en combinación con count. Permite saltearse los primeros n registros (siendo n el resultado de evaluar esta expresión numérica) y a partir de allí quedarse con los siguientes m registros (m el resultado de esta otra expresión).

En este ejemplo se saltean las primeras 5 atracciones de la consulta, para quedarse con las siguientes 10.

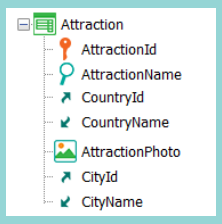
Esto se conoce como paginado de datos. El paginado básicamente consiste en dividir la información resultante de una consulta en bloques más pequeños, que, entre otras cosas, reducen la cantidad de datos enviados del servidor de base de datos al programa.

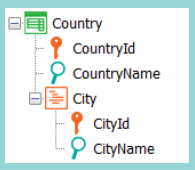
En el listado de navegación aparecerá una indicación de optimización, donde vemos que este paginado se realizará en el servidor, es decir, que lo hará el propio DBMS. Podemos quedar tranquilos.

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
```





Luego podíamos especificar una lista de cláusulas order condicionales con a lo sumo una incondicional al final, para ordenar la información a ser consultada y devuelta. Las condiciones nos permitían jugar también con las cláusulas where condicionales, de modo de poder ordenar por los mismos criterios de filtro, y lograr especificar, así, consultas más optimizadas.

En este ejemplo si la condición de la primera cláusula order se satisface, es decir, la variable &country no está vacía, se ordena por CountryId y la siguiente cláusula order no se considera en absoluto. Como en este caso la condición del where es la misma, el where se aplicará y el filtro de país estará, entonces, optimizado.

En cambio, si la condición de la primera cláusula order no se satisface, se investiga la segunda cláusula order, que como es incondicional aplicará. Como el where no aplicará en este caso, dado que la variable &country está vacía, se mostrarán todas las atracciones ordenadas por nombre de atracción (y seguramente desordenadas por país).

La cláusula order none, que no habíamos visto antes, se escribe cuando queremos que el orden a aplicar quede indefinido, lo que significa que dependerá de la plataforma e incluso puede variar de una ejecución a la siguiente.

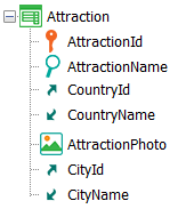
De todas maneras las cláusulas order que escribamos no determinan exactamente el plan de ejecución que elegirá el DBMS, porque se toman

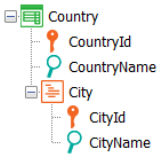
en cuenta también otras consideraciones, justamente para optimizar el acceso a la base de datos. Profundizaremos sobre este asunto en video aparte.

Sabemos también que la claúsula order tiene una fundamental relevancia cuando se quiere implementar un corte de control entre for eachs anidados. En ese caso no solo se utiliza para ordenar la información, sino también, y más importante, para establecer el criterio de corte. Allí la cláusula order no puede ser condicional.

For each $BaseTrn_1$, ... , $BaseTrn_n$

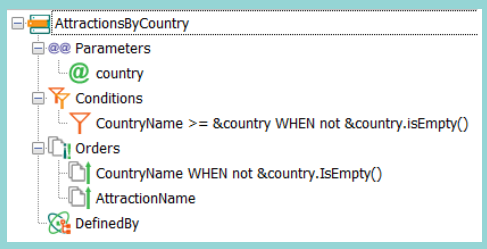
skip *expression*$_1$ **count** *expression*$_2$
order att_1, att_2, ... , att_n [**when** *condition*]
order att_1, att_2, ... , att_n [**when** *condition*]
order none [**when** *condition*]
unique att_1, att_2, ... , att_n
using *DataSelector* ($parm_1$, $parm_2$, ... , $parm_n$)






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

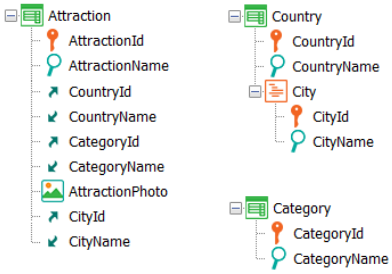
```
For each Attraction
    using AttractionsByCountry(&country)
            print info //AttractionName, CountryName
endfor
```



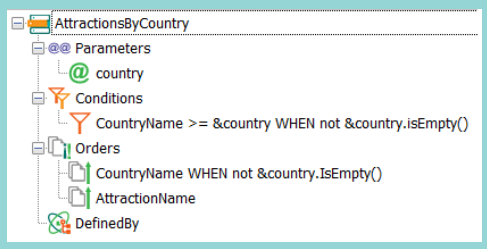
La cláusula using permitía incorporar más órdenes y filtros pero centralizados en un objeto Data Selector, que permite que le enviemos parámetros. De esta forma no teníamos que repetir esos mismos criterios de ordenamiento y de filtro explícitamente en cada consulta. A todos los efectos del For each, es como si sí se hubieran escrito explícitamente. No hay ninguna diferencia. De hecho en el listado de navegación no se podrá diferenciar una forma de la otra. Se verá exactamente lo mismo y no habrá ninguna indicación del 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$_n$*)



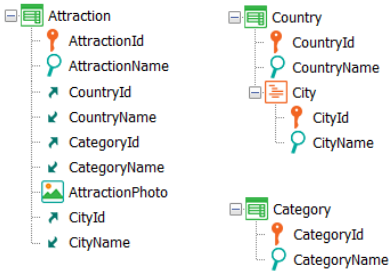

```
For each Attraction
    order CountryName when not &country.IsEmpty()
    order AttractionName
    where CountryName >= &country when not &country.IsEmpty()
    where CategoryName <> "Monument"
            print info //AttractionName, CountryName
endfor

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



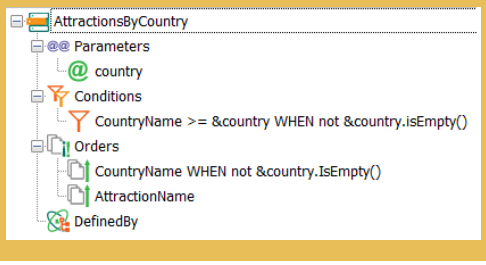
Esto significa que pueden convivir cláusulas order, where y using sin problema alguno. Para el programa generado no habrá ninguna diferencia entre este for each y este otro.

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
```



La diferencia aparece cuando se hace otro uso, esta vez especial, del Data Selector: cuando se lo utiliza como consulta ejecutable.

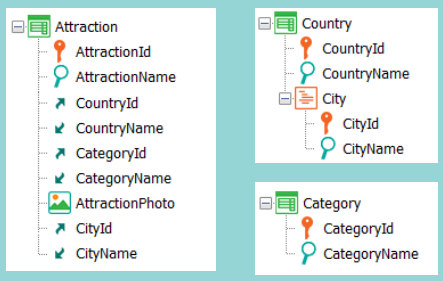
Esto sucede cuando se quiere filtrar indicando que solo queremos quedarnos con los registros de la tabla extendida del for each para los que el valor de un cierto atributo se encuentre entre los devueltos por esa consulta independiente.

En el ejemplo cuando queremos recorrer la tabla de categorías quedándonos únicamente con aquellas que se encuentren entre las categorías de las atracciones de la consulta independiente del Data Selector. La consulta del DataSelector, por los atributos implicados, claramente es de las atracciones de países de nombre posterior al del filtro (si no está vacío, si está vacío, todas). Debemos pensar al Data Selector en este caso como si fuera otro For each. Es por ello que los atributos que se encuentren dentro del Data Selector para este caso no tendrán ninguna implicancia para el for each en el que se está utilizando. Por eso la tabla base del for Each será Category, y la de la consulta del Data Selector Attraction.

For each $BaseTrn_1, \dots, BaseTrn_n$

skip $expression_1$ **count** $expression_2$
order $att_1, att_2, \dots, att_n$ [**when** *condition*]
order $att_1, att_2, \dots, att_n$ [**when** *condition*]
order none [**when** *condition*]
unique $att_1, att_2, \dots, att_n$
using *DataSelector* ($parm_1, parm_2, \dots, parm_n$)
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
```



También vimos la cláusula unique, para que si los atributos indicados se repiten para un conjunto de registros del for each, solo se tome uno para procesar en el cuerpo del for each, es decir, en el código principal.

Aquí estamos recorriendo la tabla de atracciones, agrupándolas por id de categoría y solo quedándonos con un registro del grupo, imprimiendo su nombre de categoría.

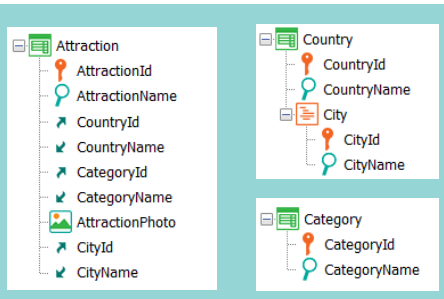
Vimos también que esta cláusula era sumamente útil cuando en el código del for each queríamos hacer una agregación sobre ese conjunto de registros repetidos para esos atributos unique. Algo que de otra forma no podíamos lograr (que el for each y la fórmula de agregación trabajaran sobre la misma tabla). En este ejemplo el for each recorre Attraction, la fórmula Count también, y al haber especificado la cláusula unique por CategoryId, la cuenta se realizará para el conjunto de registros con el mismo CategoryId en el 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

En el bloque de código principal es donde programamos qué queremos realizar con cada registro de la tabla base que haya pasado los filtros (por supuesto, si allí se utilizan atributos de la tabla extendida, se realiza automáticamente el acceso a esos registros y se trabaja con ellos).

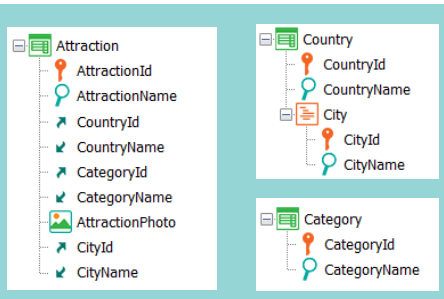
Entre todo lo que aquí puede programarse, está el escribir otro comando For each, anidado...

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

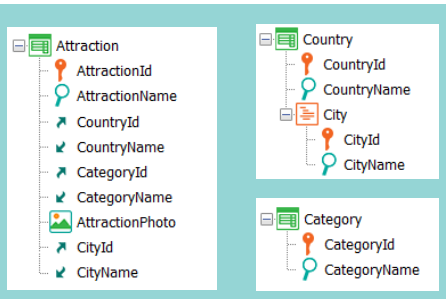
...una invocación a una subrutina o...

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

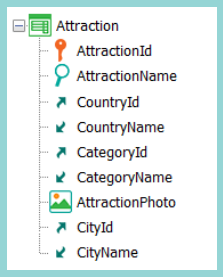
...a un procedimiento, que al ejecutarse retornará a lo que siga a la invocación.

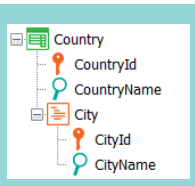
For each *BaseTrn$_1$, ... , BaseTrn$_n$*

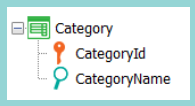
skip *expression$_1$* **count** *expression$_2$*

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

Si no hay ningún registro que pase los filtros, entonces pasará a ejecutarse el código especificado bajo la cláusula when none, si se especificó, claro.

En este ejemplo, será cuando no hay ninguna atracción de país de ese nombre.

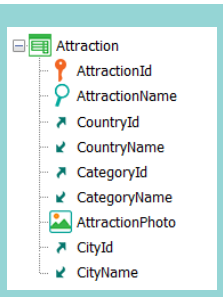
Como al ejecutarse esa cláusula se asume que no se consiguió hacer nada con los datos de la tabla extendida del for each, aquí no estamos posicionados en ningún registro, por lo que...

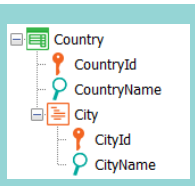
For each *BaseTrn$_1$, ... , BaseTrn$_n$*

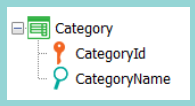
skip *expression$_1$* **count** *expression$_2$*

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

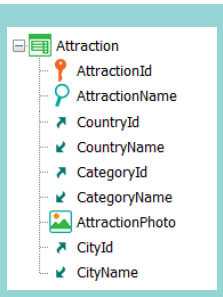
...si allí se nombran atributos, ¿de dónde se tomarán? Solo tiene sentido nombrar atributos si éstos están instanciados de antes (por ejemplo en la regla parm del objeto donde se encuentra este for each...

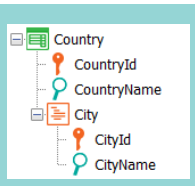
For each $BaseTrn_1$, ... , $BaseTrn_n$

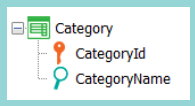
skip $expression_1$ **count** $expression_2$

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

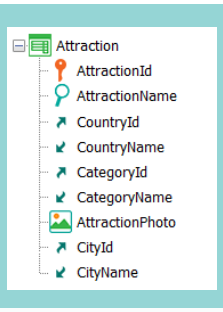
... o en otro for each al que éste esté anidado).

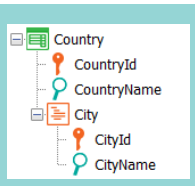
For each $BaseTrn_1$, ... , $BaseTrn_n$

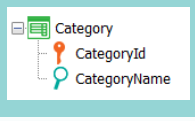
skip *expression*$_1$ **count** *expression*$_2$

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
```

Por supuesto aquí podremos escribir otra consulta (otro for each), utilizar una fórmula inline, etc. , pero es como si estuvieran escritas a continuación del For each.

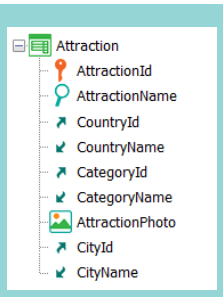
Procedure Object

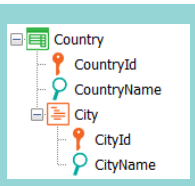
For each *BaseTrn$_1$, ... , BaseTrn$_n$*

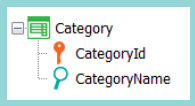
skip *expression$_1$* **count** *expression$_2$*

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

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







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

Si el for each se encuentra dentro de un procedimiento y solo en ese caso, en el código principal también se podrá asignar valor a un atributo o a varios, para así actualizar el registro de la tabla base y el o los de la extendida que correspondan.

Aquí se actualiza el atributo AttractionName del registro de la tabla base en el que nos encontramos en cada iteración, y el atributo CategoryName del registro asociado de la tabla Category, que es parte de la extendida.

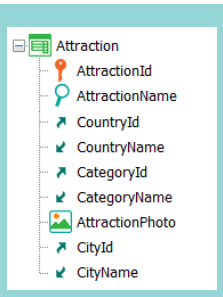
Procedure Object

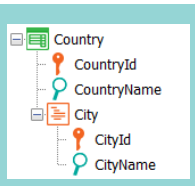
For each $BaseTrn_1$, ... , $BaseTrn_n$

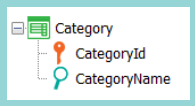
skip $expression_1$ **count** $expression_2$

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

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

        Delete

endfor
```







main_code

when duplicate

when_duplicate_code

when none

when_none_code

endfor

También podemos escribir un comando Delete para eliminar el registro en el que estamos posicionados (y solo elimina el registro de la tabla base, sin realizar chequeos de integridad referencial).

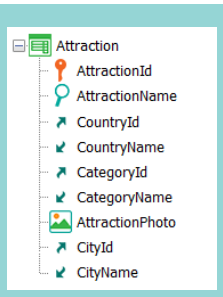
For each *BaseTrn$_1$, ... , BaseTrn$_n$*

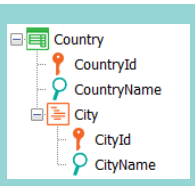
Procedure Object

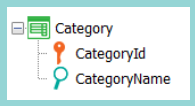
skip *expression$_1$* **count** *expression$_2$*

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

Por supuesto, también podemos escribir un comando new para insertar un registro en una tabla, pero esto queda por fuera del comportamiento del for each, que comanda tanto la actualización como la eliminación.

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

Procedure Object

Syntax

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

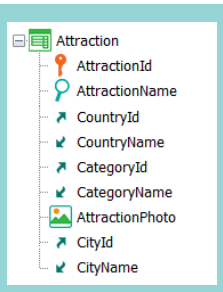
Para cualquiera de estos dos casos tenemos la cláusula blocking. Permite hacer las operaciones en un buffer y cuando se llegan a procesar los n registros indicados en la cláusula (en este ejemplo 10), recién ahí se realizan las operaciones sobre la base de datos. Es decir, se hace un único acceso a la base de datos para procesar de una sola vez los n registros que se están actualizando o eliminando, mejorando así la performance.

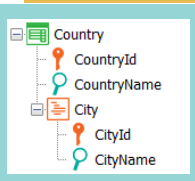
Como es de esperar, esta misma cláusula se encuentra también disponible en el comando New.

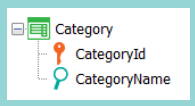
For each $BaseTrn_1$, ... , $BaseTrn_n$

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, la cláusula when duplicate se utiliza cuando se están queriendo actualizar atributos en el cuerpo del for each que son, en particular, clave candidata, por lo que no deben repetir su valor.

Si en el código principal se está modificando el valor de uno de esos atributos (para el registro en el que se encuentra posicionado el programa en esa iteración), asignándole un valor que ya existe para otro registro de la tabla, entonces no se permitirá realizar la actualización (recordemos que se utiliza el índice unique para realizar ese control).

Imaginemos para este caso que ya existe en la tabla Category un registro con CategoryName "France Attractions" y que existe un índice unique por CategoryName, entonces cuando para la atracción 1 que es de Francia se quiera actualizar el valor de su CategoryName, que era Museum, por "France Attractions", como se controlará la unicidad el chequeo fallará por clave duplicada. Es que ya existe un registro con ese valor, el 4.

Si el desarrollador programó cláusula when duplicate, allí es donde indica qué debe hacerse en ese caso. De no escribirse la cláusula no se hará nada con ese registro que se pretendía actualizar y se pasa a la siguiente iteración del For each, donde en este caso sucederá lo mismo cuando se quiera modificar el nombre de la categoría de la torre Eiffel, que es la 2.

Si sí se especifica cláusula when duplicate, entonces si bien se puede pensar que estamos aún posicionados en el registro que ocasionó el problema, en verdad ya no lo estamos para actualización, por lo que si

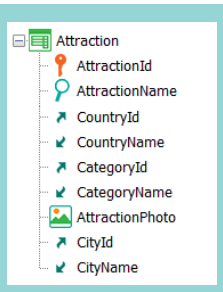
queremos actualizar el atributo u otro con otro valor, tenemos que escribir un for each para indicarlo.

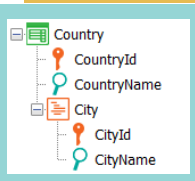
En este caso queremos que la primera actualización (que nunca fallará) sí se realice, entonces... para la primera atracción de Francia falla por duplicado...

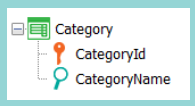
For each $BaseTrn_1, \ldots, BaseTrn_n$

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

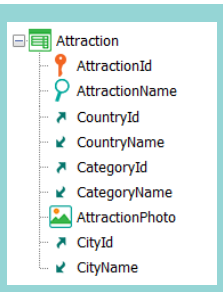
endfor

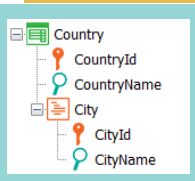
y ejecuta esta sección. Y luego para la segunda y última atracción de Francia: quiere actualizar y también falla...

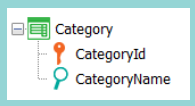
For each $BaseTrn_1, \ldots, BaseTrn_n$

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

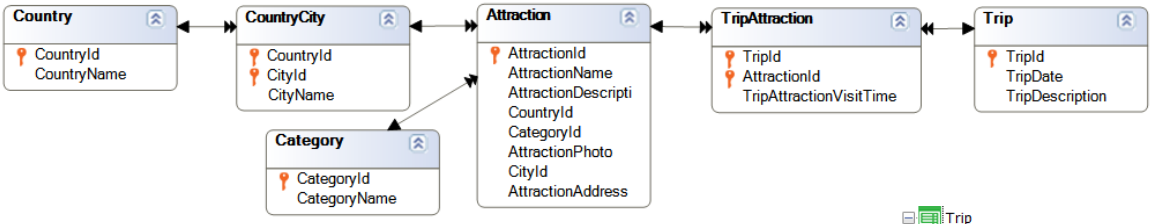
entonces ejecuta esta sección, y así nos quedará la tabla.

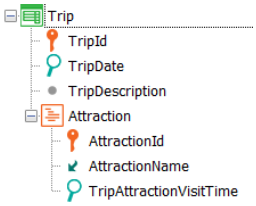
```
For each  BaseTrn1 , ... , BaseTrnn
    skip expression1 count expression2
    order  att1, att2, ... , attn [when condition]
    order att1, att2, ... , attn [when condition]
    order none [when condition]
    unique att1, att2, ... , attn
    using DataSelector ( parm1, parm2, ... , parmn )
    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
```

En definitiva, todas las cláusulas previas al código principal o cuerpo del for each se utilizan para filtrar los registros sobre los que ese código principal se ejecutará, para ordenarlos, para agruparlos en base a repetición de valores y procesarlos una vez, o para indicar que se accederán en bloque cuando se quieran actualizar y/o eliminar.

Luego, para cada uno de los registros que pasen esos filtros o el agrupamiento, y de forma ordenada de acuerdo al orden determinado, se ejecutará el código principal.

De los atributos que se utilicen en todas esas cláusulas así como dentro de ese código principal surge muchas veces la necesidad de acceder a registros de otras tablas de la tabla extendida, pero no a todos. Por asuntos de performance solo esos serán recuperados, lo que tiene algunas consecuencias.






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

      print info //AttractionName, TripAttracitionVisitTime

endfor
```

En este ejemplo se recorrerá la tabla TripAttraction, correspondiente al segundo nivel de la transacción Trip, pero se accederá a Attraction para ordenar por AttractionName y porque se pide imprimir su valor en la salida, y porque desde allí se accederá a CountryCity para poder acceder a Country para poder filtrar por CountryName.
Además se accederá a Trip para poder filtrar por TripDate.
Pero no es necesario en absoluto acceder a Category. Por tanto, si en la regla parm recibimos en el atributo CategoryName, contrario a lo que podríamos pensar, ese filtro no se aplicará. ¿Por qué? Porque dentro del for each no se accede a la tabla Category en absoluto. Entonces cuando, después de determinar todas las navegaciones del objeto donde se encuentra este for each se va a agregar el filtro implícito que proviene de la regla parm, inspeccionando cada una de las consultas, no se encuentra relación para este for each y no se agrega.

Procedure Object

```
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
```

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

Para el caso en el que nos encontramos en un procedimiento y en el código principal estamos queriendo actualizar atributos que son claves candidatas sucede los siguiente:
Supongamos que se está procesando el registro n que pasa los filtros, es decir, se está en la iteración n del for each, donde ya se ejecutó el código para los n-1 registros anteriores, y para ese registro o uno de sus relacionados por tabla extendida se está queriendo actualizar uno de sus atributos que es clave candidata dándole un valor repetido, entonces si no hay cláusula when duplicate no se realiza la actualización y se pasa al siguiente registro de la iteración, el n+1. Pero si sí se especificó cláusula when duplicate, se ejecuta. Luego se pasa a procesar el siguiente registro del for each, el n+1.

```
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
```

El código de la cláusula when none solo se ejecutará cuando no hubo iteración alguna del for each puesto que ningún registro pasó los filtros (o bien la tabla estaba vacía). Es decir, si se ejecuta este código es porque ninguno de estos se ejecutó para ningún registro.

```
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
```

Por lo que acabamos de ver los atributos que figuren en el código del when duplicate o del when none, así como los que aparezcan dentro del Data Selector cuando éste es ejecutado como consulta independiente, no son considerados en absoluto a lo hora de determinar la navegación del For each.

Esto importa especialmente para el caso en el que GeneXus debe determinar la tabla base del for each, debido a que no se colocó transacción base alguna.

Hasta aquí un compendio de lo esencial.

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