

# For each en profundidad

Cláusulas order y performance

*GeneXus™*

Vamos a concentrarnos en las cláusulas order y su relación con la optimización de la consulta.

```

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

```

Sabemos que para ordenar al información a ser consultada y devuelta, la sintaxis del for each nos permite especificar una lista de cláusulas order condicionales y una incondicional.

**For each**  $BaseTrn_1, \dots, BaseTrn_n$

```

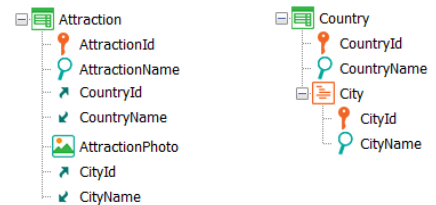
skip expression1 count expression2
order att1, att2, ... , attn [when condition]
order att1, att2, ... , attn [when condition]
order none [when condition]
unique att1, att2, ... , attn
using DataSelector ( parm1, parm2, ... , parmn)
where condition [when condition]
where condition [when condition]
where att IN DataSelector ( parm1, parm2, ... , parmn)
blocking n

```

```

main_code
for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor
when_none_code
endfor

```



Vayamos de lo más simple a lo más complejo: empecemos por analizar el caso de una única cláusula order sin condiciones, por ejemplo esta. Estamos pidiendo que se recorra la tabla de atracciones turísticas, filtrando las que correspondan a un país y ciudad determinados, y que las presente ordenadas por AttractionName, atributo secundario.

A la hora de especificar lo que queremos no debería importarnos si para obtener los datos requeridos primero los obtiene y luego los ordena, o si lo hace al revés o de alguna otra manera. Los desarrolladores especificamos lo que queremos y de resolverlo se encarga el especificador de GeneXus y, sobre todo, el DBMS.

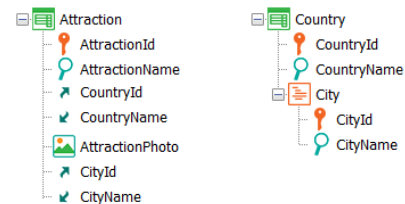
```

for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```



Net - SQL Server  
Java - Oracle



Warnings

**spc0038** There is no index for order **AttractionName**; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

Order: [AttractionName](#)  
No index!

Navigation Start from: FirstRecord

filters: Loop while: NotEndOfTable

Constraints: [CountryId](#) = &countryId  
[CityId](#) = &cityId

[Attraction](#) ( [AttractionId](#) ) INTO [CountryId](#) [CityId](#) [AttractionName](#)

| Attraction Indexes |             |
|--------------------|-------------|
| IAttraction        | Primary Key |
| • AttractionId     | Ascending   |
| IAttraction1       | Foreign Key |
| • CountryId        | Ascending   |
| • CityId           | Ascending   |

Cuando pedimos a GeneXus que especifique y genere el programa asociado al objeto que contiene el for each, lo hacemos para un environment determinado, es decir, en un lenguaje de programación determinado, como Net por ejemplo, y para una base de datos manejada por un DBMS determinado, como SQL Server, por ejemplo. O podría ser para un environment Java contra Oracle, o tantas otras alternativas.

Cuando el desarrollador escribe su For each lo hace en GeneXus, con cierta independencia de cuál será el environment final, justamente para que con el mismo código podamos obtener la aplicación en diferentes environments.

Esto significa que de la implementación concreta se encarga GeneXus que conoce las particularidades de cada environment. Sin embargo su conocimiento tiene un límite: conoce la estructura de la base de datos, pero no los datos, ni su distribución, cantidad, etcétera. Esta información la tiene el DBMS, que registra estadísticas, cachea datos y consultas en la historia de las consultas que se vienen realizando, arma planes de ejecución, mantiene índices, etcétera. Cuanto más evolucionado e inteligente el DBMS menos necesitará que GeneXus le indique con precisión cómo realizar la consulta, porque GeneXus nunca sabrá más que él.

Así, por ejemplo, si pedimos que se especifique el objeto que contiene este For each veremos este listado de navegación, que nos advierte que no existe un índice por el atributo por el que queremos mostrar ordenada

la consulta y que por lo tanto podríamos notar una baja performance. “Podríamos” no significa que efectivamente será así. ¿Por qué? Porque eso depende no solo de la cantidad de datos de la tabla, sino también del DBMS y sus estrategias. Lo que nos indica el listado de navegación es el peor escenario: en este deberá recorrerse toda la tabla ordenada por un atributo para el que posiblemente no exista índice (GeneXus no sabe si el DBMS lo creó por las suyas o no, lo cierto es que no tiene noticias de él) por lo que en el peor escenario que es el de arquitecturas centralizadas podría tener que crear temporalmente el índice para resolver la consulta por ese orden y así recorrer toda la tabla para evaluar registro a registro si cada uno satisface o no los filtros. Este sería el escenario de una base de datos con un manejador poco inteligente.

Pensemos, por ejemplo, que en este caso tal vez sea una mejor estrategia utilizar el índice por clave foránea {CountryId, CityId} que sabemos que existe porque GeneXus obliga a crearlo. Entonces se obtienen de forma óptima los registros que cumplen los filtros y recién allí, con ese resultado, se lo ordena por AttractionName.

Cuál sea la mejor estrategia dependerá en buena medida de la distribución de los datos. Si solo hay 3 atracciones de ese país y ciudad entre millones, esta parece una mejor estrategia porque el costo de ordenar 3 registros es insignificante. Sin embargo si hay millones de registros en la tabla siendo la inmensa mayoría de ese país y ciudad, entonces usar el índice por país y ciudad no reducirá demasiado la consulta de toda la tabla, y ordenar luego el resultado por AttractionName será casi lo mismo que de entrada ordenar por

AttractionName y después ir evaluando uno por uno si corresponde además al país/ciudad o no. GeneXus desconoce la distribución de datos como para tomar decisiones de este tipo. Además si esta misma consulta ya se realizó con anterioridad, el DBMS posiblemente guarde en caché el resultado y no tenga que volver a realizar la consulta de la misma manera. Esto, por supuesto, si los filtros no cambian y los datos tampoco.

En definitiva, entonces, el listado de navegación nos informa sobre la base del escenario más conservador, con el peor DBMS. Pero puede que el DBMS mejore muchísimo el escenario más pesimista y la consulta resulte optimizada.

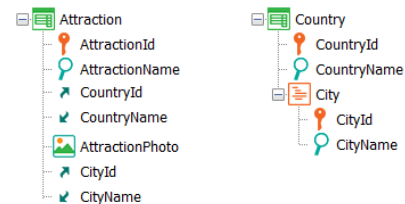
```

for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```



Net - SQL Server  
Java - Oracle



| LEVELS  |                           |
|---|---------------------------|
| For Each Attraction (Line: 24)                                    |                           |
| Order:  | AttractionName            |
| Index:  | UATTRACTION               |
| Navigation filters:   | Start from: FirstRecord   |
|   | Loop while: NotEndOfTable |
| Constraints:  | CountryId = &countryId    |
|   | CityId = &cityId          |
| =Attraction ( AttractionId ) INTO CityId CountryId AttractionName |                           |

| Attraction X       |             |                 |
|--------------------|-------------|-----------------|
| Structure Indexes  |             |                 |
| Attribute          | Order       | Description     |
| Attraction Indexes |             |                 |
| IAttraction        | Primary Key | Automatic Index |
| AttractionId       | Ascending   | Attraction Id   |
| IAttraction1       | Foreign Key | Automatic Index |
| CountryId          | Ascending   | Country Id      |
| CityId             | Ascending   | City Id         |
| IAttraction2       | Foreign Key | Automatic Index |
| CategoryId         | Ascending   | Category Id     |
| UAttraction        | Duplicate   | User Index      |
| AttractionName     | Ascending   | Attraction Name |

Podríamos vernos tentados a pensar que si sabemos que habrá millones de registros en la tabla Attraction lo mejor será ordenar desde GeneXus al DBMS la creación de índice de usuario por AttractionName. En ese caso se reorganizará la base de datos, y al especificarse nuevamente el objeto ahora GeneXus nos indicará que utilizará el nuevo índice para resolver la consulta y ya no se nos muestra la advertencia de performance.

Sin embargo esta puede ser una muy peor solución. Justamente, si el DBMS es inteligente no necesitará en absoluto que lo obliguemos a crear un índice. Él mismo lo hará si lo necesita. Tanto es así que ni siquiera en este caso en el que el propio GeneXus pidió la creación del índice se lo envía al DBMS cuando realiza la consulta desde environment con DBMS inteligente.

```
File Edit Selection View Go Run Terminal Help
listcountries.cs - Visual Studio Code
listcountries.cs X redAtt{ Untitled-1
C:\> Models > GX175tableForu9 > TravelAgency_forExpert > NetModel > Web > listcountries.cs
294 blic class listcountries__default : DataStoreHelperBase, IDataStoreHelper
295
296 public ICursor[] getCursors( )
297 {
298     cursorDefinitions();
299     return new Cursor[] {
300         new ForEachCursor(def[0])
301     };
302
303
304 private static CursorDef[] def;
305 private void cursorDefinitions( )
306 {
307     if ( def == null )
308     {
309         object[] prmpP000Q2;
310         prmpP000Q2 = new Object[4];
311         new ParDef("@AV14countryId", prmpP000Q2[0]);
312         new ParDef("@AV15cityId", prmpP000Q2[1]);
313     };
314     def= new CursorDef[] {
315         new CursorDef("P000Q2", "SELECT [CityId], [CountryId], [AttractionName], [AttractionId] FROM [Attraction] WHERE ([CountryId] = @AV14countryId) AND ([CityId] = @AV15cityId) ORDER BY [AttractionName] "
316     );
317 }
318
319
320 public void getResults( int cursor ,
321                         IFieldGetter rslt ,
322                         Object[] buf )
323 {
324     switch ( cursor )
325     {
326     case 0 :
327         ((short[]) buf[0])[0] = rslt.getShort(1);
328         ((short[]) buf[1])[0] = rslt.getShort(2);
329         ((string[]) buf[2])[0] = rslt.getString(3, 50);
330         ((short[]) buf[3])[0] = rslt.getShort(4);
331         return;
332     }
333 }
```

Alcanza con observar el fuente generado para environment Net contra SQL Server. En la sentencia Select no se está enviando ninguna información respecto al índice a ser utilizado. ¿Para qué se lo va a enviar si el DBMS conoce de su existencia? Si lo necesita lo va a utilizar. Y si no lo hace es porque va a utilizar una estrategia mejor.

Podemos suponer que distinto sería el caso si el DBMS fuera uno sin inteligencia.

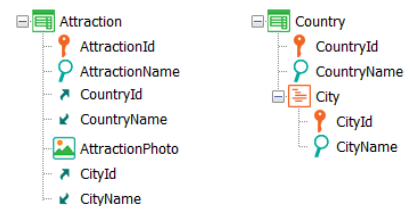
```

for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```



Net - SQL Server  
Java - Oracle



LEVELS

For Each Attraction (Line: 24)

Order: [AttractionName](#)  
Index: UATTRACTION

Navigation filters: Start from: FirstRecord  
Loop while: NotEndOfTable

Constraints: [CountryId](#) = &countryId  
[CityId](#) = &cityId

= [Attraction](#) ( [AttractionId](#) ) INTO [CityId](#) [CountryId](#) [AttractionName](#)

| Attraction X       |             |                 |
|--------------------|-------------|-----------------|
| Structure Indexes  |             |                 |
| Attribute          | Order       | Description     |
| Attraction Indexes |             |                 |
| IAtraction         | Primary Key | Automatic Index |
| • AttractionId     | Ascending   | Attraction Id   |
| IAtraction1        | Foreign Key | Automatic Index |
| • CountryId        | Ascending   | Country Id      |
| • CityId           | Ascending   | City Id         |
| IAtraction2        | Foreign Key | Automatic Index |
| • CategoryId       | Ascending   | Category Id     |
| UAttraction        | Duplicate   | User Index      |
| • AttractionName   | Ascending   | Attraction Name |

Debemos tener en cuenta que la creación de un índice es algo costoso, y que tiene implicancias no solo al momento de crearlo, sino después a la hora de mantenerlo, durante todo el ciclo de vida de la tabla. Cada vez que se actualiza la tabla se paga un pequeño precio por mantenerlo.

Es por eso que no parece una buena práctica la creación de índices de usuario, a menos que se los necesite para controlar unicidad, es decir, para definir claves candidatas, como índices unique.



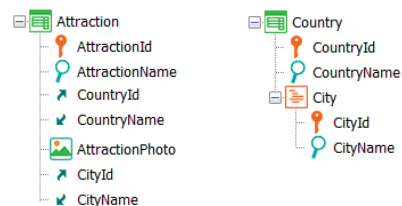
```

for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```



Net - SQL Server  
Java - Oracle



Warnings

**spc0038** There is no index for order AttractionName; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

Order: AttractionName  
No index!

Navigation Start from: FirstRecord

filters: Loop while: NotEndOfTable

Constraints: CountryId = &countryId  
CityId = &cityId

Attraction ( AttractionId ) INTO CountryId CityId AttractionName

Entonces, decíamos, el especificador de GeneXus hace lo mejor que puede con la información que tiene y con su inteligencia actual (es esperable que esa inteligencia se vaya incrementando conforme GeneXus evoluciona), y envía la consulta lo más optimizada que puede al DBMS sin indicarle obviedades, sabiendo que éste en el peor de los casos la tomará pero en general la mejorará.

En definitiva nunca podemos asegurar que lo que indica el listado de navegación será lo que finalmente hará el DBMS cuando éste es inteligente. Lo que sí sabemos es que será eso o algo mejor.


Veamos ejemplos de la inteligencia actual del especificador de GeneXus, más allá de lo que después termine haciendo el DBMS.

Sabemos que el orden en el que se pedirá que los registros resultantes sean retornados se determina en base a la especificación del desarrollador en la cláusula order, pero también a algoritmos internos de optimización.

```
for each Attraction
```

```
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```

| LEVELS                         |   |
|--------------------------------|---|
| For Each Attraction (Line: 11) |   |
| Order:                         | <u>CountryId</u> , <u>CityId</u><br>Index: IATTRACTION1   |
| Navigation                     | Start from: <u>CountryId</u> = &countryId   |
| filters:                       | <u>CityId</u> = &cityId<br>Loop while: <u>CountryId</u> = &countryId<br><u>CityId</u> = &cityId   |
|                                |  =Attraction ( <u>AttractionId</u> ) INTO <u>CityId</u> <u>CountryId</u> <u>AttractionName</u> |

Por ejemplo, si no nos importara cómo saldrán ordenados los nombres de atracción podríamos escribir el For each sin cláusula order. En general sabíamos que si hacíamos esto la consulta se ordenaba por clave primaria. Es decir que se enviaba al DBMS un Select con order AttractionId. Sin embargo en este caso sucederá algo distinto, como vemos en el listado de navegación. Sabemos que existe en la base de datos un índice por CountryId y CityId, los dos filtros por igualdad. Lo sabemos porque conforman una clave foránea. Claramente, entonces, será preferible que se utilice ese índice y se devuelva la consulta ordenada por esos valores. Es por eso que si observamos la consulta que envía el fuente al DBMS, encontramos este ORDER BY. Si nuestra intención al no especificar cláusula order era que salieran ordenados por clave primaria, en este caso tendremos que explicitarlo.

De igual manera, si ahora la consulta es esta otra, donde estamos pidiendo que las atracciones salgan ordenadas por identificador de ciudad, y solo estamos filtrando por identificador de país, veremos que el listado de navegación no está mostrando que pedirá ordenar por CityId, sino por el par, debido a que así optimiza la consulta, porque sabe de la existencia del índice compuesto (por ser clave foránea, justamente), sin dejar de cumplir con el resultado del ordenamiento pedido por el desarrollador.

En definitiva, con la cláusula order el desarrollador indica el orden en que desea que los registros sean devueltos, pero para ello el especificador podría alterar esa cláusula, suplementándola con información contextual (si existen condiciones implícitas o explícitas por igualdad y existe índice

que las contiene, además de los atributos del order) de manera tal de optimizar la consulta, aunque el DBMS será quien tenga al final del día la última palabra.

Lo importante es entender que los datos serán devueltos en el orden explicitado por el desarrollador, aunque para resolver la consulta se utilicen otros criterios.

```

for each Attraction
  order AttractionId
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor


```

```

"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"

```

[AttractionId]

| LEVELS                         |   |
|--------------------------------|---|
| For Each Attraction (Line: 11) |   |
| Order:                         | CountryId , CityId  |
|                                | Index: IATTRACTION1   |
| Navigation                     | Start from: CountryId = &countryId  |
| filters:                       | CityId = &cityId  |
|                                | Loop while: CountryId = &countryId  |
|                                | CityId = &cityId  |
|                                |  =Attraction ( AttractionId ) INTO CityId CountryId AttractionName |

Si nuestra intención al no especificar cláusula order era que salieran ordenados por clave primaria, en este caso tendremos que explicitarlo.


```
for each Attraction
```


```
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor
```

```
"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"
```

```
for each Attraction
```

```
  order CityId
  where CountryId = &countryId
  print attractionInfo //AttractionName
endfor
```

| LEVELS                         |   |
|--------------------------------|---|
| For Each Attraction (Line: 11) |   |
| Order:                         | CountryId , CityId  |
| Index:                         | IATTRACTION1  |
| Navigation                     | Start from: CountryId = &countryId  |
| filters:                       | CityId = &cityId  |
|                                | Loop while: CountryId = &countryId  |
|                                | CityId = &cityId  |
|                                |  =Attraction ( AttractionId ) INTO CityId CountryId AttractionName |

| LEVELS                         |   |
|--------------------------------|---|
| For Each Attraction (Line: 11) |   |
| Order:                         | CountryId , CityId  |
| Index:                         | IATTRACTION1  |
| Navigation                     | Start from: CountryId = &countryId  |
| filters:                       | Loop while: CountryId = &countryId  |
|                                |  =Attraction ( AttractionId ) INTO CountryId AttractionName CityId |

De igual manera, si ahora la consulta es esta otra, donde estamos pidiendo que las atracciones salgan ordenadas por identificador de ciudad, y solo estamos filtrando por identificador de país, veremos que el listado de navegación no está mostrando que pedirá ordenar por CityId, sino por el par, debido a que así optimiza la consulta, porque sabe de la existencia del índice compuesto (por ser clave foránea, justamente), sin dejar de cumplir con el resultado del ordenamiento pedido por el desarrollador.

En definitiva, con la cláusula order el desarrollador indica el orden en que desea que los registros sean devueltos, pero para ello el especificador podría alterar esa cláusula, suplementándola con información contextual (si existen condiciones implícitas o explícitas por igualdad y existe índice que las contiene, además de los atributos del order) de manera tal de optimizar la consulta, aunque el DBMS será quien tenga al final del día la última palabra.

Lo importante es entender que los datos serán devueltos en el orden explicitado por el desarrollador, aunque para resolver la consulta se utilicen otros criterios.

```

for each Attraction

  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```

```

"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"

```

```

for each Attraction

```

```


  where AttractionName = &attractionName
  print attractionInfo //AttractionName
endfor

```

```

"SELECT [AttractionName], [AttractionId]
FROM [Attraction] WHERE [AttractionName] =
@AV8AttractionName ORDER BY [AttractionId]"

```

| LEVELS                         |   | ⌵ |
|--------------------------------|---|---|
| For Each Attraction (Line: 11) |   | ⌵ |
| Order:                         | <u>CountryId</u> , <u>CityId</u>  |   |
|                                | Index: IATTRACTION1   |   |
| Navigation                     | Start from: <u>CountryId</u> = &countryId   |   |
| filters:                       | <u>CityId</u> = &cityId   |   |
|                                | Loop while: <u>CountryId</u> = &countryId   |   |
|                                | <u>CityId</u> = &cityId   |   |
|                                |  =Attraction ( <u>AttractionId</u> ) INTO <u>CityId</u> <u>CountryId</u> <u>AttractionName</u> |   |

| LEVELS                         |  | ⌵             |
|--------------------------------|--|---------------|
| For Each Attraction (Line: 24) |  | ⌵             |
| Order:                         | <u>AttractionId</u>  |               |
|                                | Index: IATTRACTION   |               |
| Navigation filters:            | Start from:  | FirstRecord   |
|                                | Loop while:  | NotEndOfTable |
| Constraints:                   | <u>AttractionName</u> = &AttractionName  |               |
|                                |  =Attraction ( <u>AttractionId</u> ) INTO <u>AttractionName</u> |               |

En este caso donde no se especificó order, como existe índice que permite optimizar esas condiciones, es el elegido, en lugar del índice por clave primaria.

Cuando no hay índice, entonces sí elige la clave primaria. Aquí vemos la sentencia SQL que construye GeneXus.

```

for each Attraction

  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo //AttractionName
endfor

```

```

"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE [CountryId] =
@AV14countryId and [CityId] = @AV15cityId
ORDER BY [CountryId], [CityId]"

```

```


for each Attraction
  order NONE
  where AttractionName = &attractionName
  print attractionInfo //AttractionName
endfor

```

```

"SELECT [AttractionName], [AttractionId]
FROM [Attraction] WHERE [AttractionName] =
@AV8AttractionName-ORDER BY [AttractionId]"

```

| LEVELS                         |   | ⌵ |
|--------------------------------|---|---|
| For Each Attraction (Line: 11) |   | ⌵ |
| Order:                         | <u>CountryId</u> , <u>CityId</u>  |   |
|                                | Index: IATTRACTION1   |   |
| Navigation                     | Start from: <u>CountryId</u> = &countryId   |   |
| filters:                       | <u>CityId</u> = &cityId   |   |
|                                | Loop while: <u>CountryId</u> = &countryId   |   |
|                                | <u>CityId</u> = &cityId   |   |
|                                |  =Attraction ( <u>AttractionId</u> ) INTO <u>CityId</u> <u>CountryId</u> <u>AttractionName</u> |   |

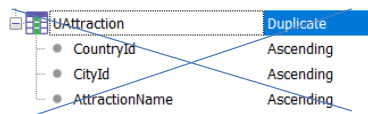
| LEVELS                         |   | ⌵ |
|--------------------------------|---|---|
| For Each Attraction (Line: 24) |   | ⌵ |
| Order:                         | NONE  |   |
| Navigation filters:            | Start from: FirstRecord   |   |
|                                | Loop while: NotEndOfTable   |   |
| Constraints:                   | <u>CountryId</u> = &countryId   |   |
|                                | <u>CityId</u> = &cityId   |   |
|                                |  =Attraction ( <u>AttractionId</u> ) INTO <u>CountryId</u> <u>CityId</u> <u>AttractionName</u> |   |

A menos que especifiquemos cláusula Order none, y en ese caso le delegamos al DBMS la elección del orden. No se agrega ORDER BY a la sentencia SQL armada por GeneXus.

```

for each Attraction
  order AttractionName
  where CountryId = &countryId
  where CityId = &cityId
  print attractionInfo
endfor

```



```

"SELECT [CityId], [CountryId],
[AttractionName], [AttractionId] FROM
[Attraction] WHERE ([CountryId] =
@AV14countryId) AND ([CityId] = @AV15cityId)
ORDER BY [AttractionName] "

```

LEVELS

For Each Attraction (Line: 11)

Order: CountryId , CityId , AttractionName

Index: UATTRACTION

Navigation filters: Start from: CountryId = &countryId

CityId = &cityId

Loop while: CountryId = &countryId

CityId = &cityId

Table Icon

=Attraction ( AttractionId ) INTO CountryId CountryId AttractionName

Warning Icon

spc0038 There is no index for order **AttractionName**; poor performance may be noticed in group starting at line 11.

LEVELS

For Each Attraction (Line: 11)

Order: AttractionName

No index!

Navigation filters: Start from: FirstRecord

Loop while: NotEndOfTable

Constraints: CountryId = &countryId

CityId = &cityId

Table Icon

=Attraction ( AttractionId ) INTO CountryId CityId AttractionName

Volvamos a este ejemplo. Si GeneXus pidió crear este índice de usuario compuesto entonces lo propondrá en el listado de navegación (aunque finalmente no lo envíe al SQL Server, porque éste ya sabe de su existencia y para qué decirle algo que ya sabe). La gracia es que nos hace saber que al menos esta optimización será realizada por el DBMS. Será así o mejor.

En cambio si el índice de usuario no existe, entonces el listado de navegación nos mostrará esto otro, aunque el fuente sea exactamente el mismo.

¿Entonces? Repitémoslo una vez más: el listado de navegación nos propone el peor escenario. Si el índice existe, sabemos que el peor escenario será bastante bueno. Eso si el índice se creó antes por alguna otra razón y ya que estamos lo aprovechamos.

Crear el índice solo para asegurarnos de que esta navegación esté optimizada no parece una buena idea si estamos utilizando un DBMS inteligente. Y si el DBMS no fuera inteligente pero la tabla tiene pocos registros, tampoco. En definitiva la sugerencia es crear índices solo tras verificar problemas de performance y evaluar pros y contras.



```

for each Attraction
  order CityName
  unique CountryId, CityId
  print relevantInfo //CityName
endfor

```

```

"SELECT DISTINCT T1.[CityId], T1.[CountryId],
T2.[CityName] FROM ([Attraction] T1 INNER JOIN
[CountryCity] T2 ON T2.[CountryId] = T1.[CountryId] AND
T2.[CityId] = T1.[CityId])
ORDER BY T2.[CityName] "

```

```

for each Attraction
  //order CityName
  unique CountryId, CityId
  print relevantInfo //CityName
endfor

```

```

"SELECT DISTINCT T2.[CityName], T1.[CityId],
T1.[CountryId] FROM ([Attraction] T1 INNER JOIN
[CountryCity] T2 ON T2.[CountryId] = T1.[CountryId] AND
T2.[CityId] = T1.[CityId])
ORDER BY T1.[CountryId], T1.[CityId] "

```

For Each Attraction (Line: 18)

Order: [CityName](#)  
 No index!  
 Unique: [CountryId](#) , [CityId](#)  
 Navigation Start from: FirstRecord  
 filters: Loop while: NotEndOfTable  
 Join location: Server

=Attraction ( [AttractionId](#) ) INTO [CityId](#) [CountryId](#)  
=CountryCity ( [CountryId](#) , [CityId](#) ) INTO [CityName](#)

For Each Attraction (Line: 18)

Order: [CountryId](#) , [CityId](#)  
 Index: IATTRACTION1  
 Unique: [CountryId](#) , [CityId](#)  
 Navigation Start from: FirstRecord  
 filters: Loop while: NotEndOfTable  
 Join location: Server

=Attraction ( [AttractionId](#) ) INTO [CityId](#) [CountryId](#)  
=CountryCity ( [CountryId](#) , [CityId](#) ) INTO [CityName](#)

Otro ejemplo que muestra cómo GeneXus intenta mejorar las cosas:

Si queremos obtener todos los nombres de ciudad para las que hay atracciones turísticas, utilizamos la cláusula unique por CountryId, CityId, para que de todas las atracciones que comparten país y ciudad solo se quede con una, para poder listar su nombre de ciudad en la salida. Si queremos que esa salida se muestre ordenada por nombre de ciudad, colocamos la cláusula order y vemos que en el listado de navegación el especificador escribe exactamente ese order, para el que no conoce ningún índice. Quedará en manos del DBMS la optimización de esta consulta.

En cambio, si no nos importa el orden en que se muestren en la salida esas ciudades, entonces veamos que al no escribirlo, el especificador está eligiendo ordenar por los atributos que estamos pidiendo que sean únicos, ya que tiene un índice por ellos.

```

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

```

Ahora repasemos las cláusulas order condicionales.

The screenshot displays the GeneXus IDE interface. On the left, a web panel is shown with three input fields: 'Category Id' with value '&CategoryId', 'Country Id' with value '&CountryId', and 'Attraction Name' with value '&AttractionName'. Below these fields is a button labeled 'Print Attractions'. To the right of the web panel is a tree view of the 'Attraction' entity, listing attributes: AttractionId, AttractionName, AttractionDescription, CountryId, CountryName, CategoryId, CategoryName, AttractionPhoto, CityId, CityName, and AttractionAddress.

At the top right, an event definition is shown:

```
Event 'Print Attractions'
  ListAttractions( &CategoryId, &CountryId, &AttractionName)
Endevent
```

An arrow points from the event definition to the 'Source' tab in the IDE. The 'Source' tab shows the following code:

```
Source *
Subroutines
1 print AttractionTitles
2
3 for each Attraction
4   order AttractionName
5   where CategoryId = &categoryId when not &categoryId.IsEmpty()
6   where CountryId = &countryId when not &countryId.IsEmpty()
7   where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
8     print attractionInfo
9 endfor
```

Queremos desde este Web Panel listar las atracciones turísticas pero permitiéndole al usuario filtrar las que son de una categoría determinada, como Tourist Site, de un país determinado, y cuyo nombre sea posterior a un valor dado. Así que al presionar el botón llamamos a este procedimiento, pasándole las tres variables.

Si el usuario no ingresa valor en una de las variables de filtro, no queremos que ese filtro se aplique, y por eso condicionamos las tres cláusulas Where.

Si quisiéramos que independientemente de los filtros que se apliquen las atracciones se muestren ordenadas por nombre de atracción, entonces escribiríamos una única cláusula order incondicional.

### Procedure ListAttractions Navigation Report

**Name:** ListAttractions  
**Description:** List Attractions  
**Output Devices:** File

**Environment:** Default (.NET)  
**Spec. Version:** 17\_0\_10-159906  
**Form Class:** Graphic  
**Program Name:** ListAttractions  
**Parameters:** in: &categoryId, in: &countryId, in: &AttractionName

**Warnings**

▲ spc0038 There is no index for order **AttractionName**: poor performance may be noticed in group starting at line 3.

**LEVELS**

For Each Attraction (Line: 4)

**Order:** AttractionName  
**No index!**

**Navigation filters:** Start from: FirstRecord  
 Loop while: NotEndOfTable

**Constraints:** CategoryId = &categoryId **WHEN** not &categoryId. isempty()  
CountryId = &countryId **WHEN** not &countryId. isempty()  
AttractionName >= &AttractionName **WHEN** not &AttractionName. isempty()

**Join location:** Server

Attraction ( AttractionId ) INTO AttractionName CategoryId CountryId  
 Country ( CountryId ) INTO CountryName  
 Category ( CategoryId ) INTO CategoryName

| Category     | Country       | Attraction            |
|--------------|---------------|-----------------------|
| Monument     | Brazil        | Christ the Redemmer   |
| Tourist site | Italy         | Cinque Terre          |
| Monument     | France        | Eiffel Tower          |
| Tourist site | China         | Forbidden city        |
| Tourist site | Scotland      | Glenfinnan Viaduct    |
| Tourist site | United States | London Bridges        |
| Monument     | England       | London Towers         |
| Museum       | France        | Louvre Museum         |
| Museum       | France        | Matisse Museum        |
| Tourist site | China         | Meet the Emperor      |
| Tourist site | Italy         | Rifugio Nuvolau       |
| Museum       | United States | Smithsonian Institute |
| Tourist site | China         | The Great Wall        |

Ejecutemos el web Panel, que definimos como main para facilitar la ejecución.

Si observamos el listado de navegación, vemos que los filtros aparecen en la sección de Constraints. Esto no es únicamente por la inexistencia de un índice que permita optimizar, sino porque contienen las condiciones When. Todo Where condicional se mostrará en la sección de Constraints, pero eso no significa que la consulta no vaya a optimizarse. Volveremos sobre esto.

Aquí vemos el listado de todas las atracciones, ya que ninguno de los 3 filtros se habrá aplicado. Observemos que se listan ordenadas por nombre de atracción, tal como pedimos.

Si ahora pedimos listar las atracciones de la categoría 3 que es Tourist Site, también se muestra el resultado ordenado por AttractionName y desordenado por país.

ListAttractions \* X

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

Subroutines

```

1 print AttractionTitles
2
3 for each Attraction
4   order CategoryName when not &categoryId.IsEmpty()
5   order CountryName when not &countryId.IsEmpty()
6   order AttractionName
7   where CategoryId = &categoryId when not &categoryId.IsEmpty()
8   where CountryId = &countryId when not &countryId.IsEmpty()
9   where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
10  print attractionInfo
11 -endfor
12

```

stAttractions Navigation Report

ListAttractions

List Attractions

File

Environment:

Spec. Version:

Form Class:

Program Name:

Parameters:

Default (.NET)

17\_0\_10-159906

Graphic

ListAttractions

in: &categoryId, in: &countryId, in: &AttractionName

For Each Attraction (Line: 4)

Order:

CategoryName WHEN &categoryId.IsEmpty()

No index!

CountryName WHEN &countryId.IsEmpty()

No index!

AttractionName OTHERWISE

No index!

Navigation filters:

Start from: FirstRecord

Loop while: NotEndOfTable

Constraints:

CategoryId = &categoryId WHEN not &categoryId.IsEmpty()

CountryId = &countryId WHEN not &countryId.IsEmpty()

AttractionName >= &AttractionName WHEN not &AttractionName.IsEmpty()

Join location:

Server

Attraction ( AttractionId ) INTO AttractionName CountryId CategoryId

Country ( CountryId ) INTO CountryName

Category ( CategoryId ) INTO CategoryName

Pero supongamos que en caso de no filtrar por categoría, es decir, que esta variable esté vacía, los queremos ordenados justamente por nombre de categoría, y en cambio si sí se seleccionó un valor para &categoryId (por ejemplo el de Tourist Site) allí queremos que salga ordenado por nombre de país (si es que no se seleccionó país, es decir, se dejó vacía esta variable) y solo en caso contrario (es decir, en caso en que se filtre por categoría y país), queremos ordenar por nombre de atracción.

Vemos que en el listado de navegación nos aparecen las cláusulas order condicionales, donde la última es incondicional. A diferencia de las cláusulas where que hacen un AND entre ellas, solamente una de las cláusulas order se aplicará. Para ello la primera condición que sea True hará que su cláusula order sea la elegida. Solo se ordenará por la incondicional si ninguna de las condiciones de las cláusulas order anteriores se satisfizo. Por supuesto, podríamos no colocar cláusula order incondicional, y allí el orden quedará indefinido si no se satisface ninguna de las condiciones.

| Category     | Country       | Attraction            |
|--------------|---------------|-----------------------|
| Monument     | France        | Eiffel Tower          |
| Monument     | Brazil        | Christ the Redemmer   |
| Monument     | England       | London Towers         |
| Museum       | France        | Louvre Museum         |
| Museum       | United States | Smithsonian Institute |
| Museum       | France        | Matisse Museum        |
| Tourist site | China         | Forbidden city        |
| Tourist site | Scotland      | Glenfinnan Viaduct    |
| Tourist site | China         | Meet the Emperor      |
| Tourist site | Italy         | Rifugio Nuvolau       |
| Tourist site | China         | The Great Wall        |
| Tourist site | Italy         | Cinque Terre          |
| Tourist site | United States | London Bridges        |

| Category     | Country       | Attraction         |
|--------------|---------------|--------------------|
| Tourist site | China         | Meet the Emperor   |
| Tourist site | China         | The Great Wall     |
| Tourist site | China         | Forbidden city     |
| Tourist site | Italy         | Rifugio Nuvolau    |
| Tourist site | Italy         | Cinque Terre       |
| Tourist site | Scotland      | Glenfinnan Viaduct |
| Tourist site | United States | London Bridges     |

| Category     | Country | Attraction       |
|--------------|---------|------------------|
| Tourist site | China   | Forbidden city   |
| Tourist site | China   | Meet the Emperor |
| Tourist site | China   | The Great Wall   |

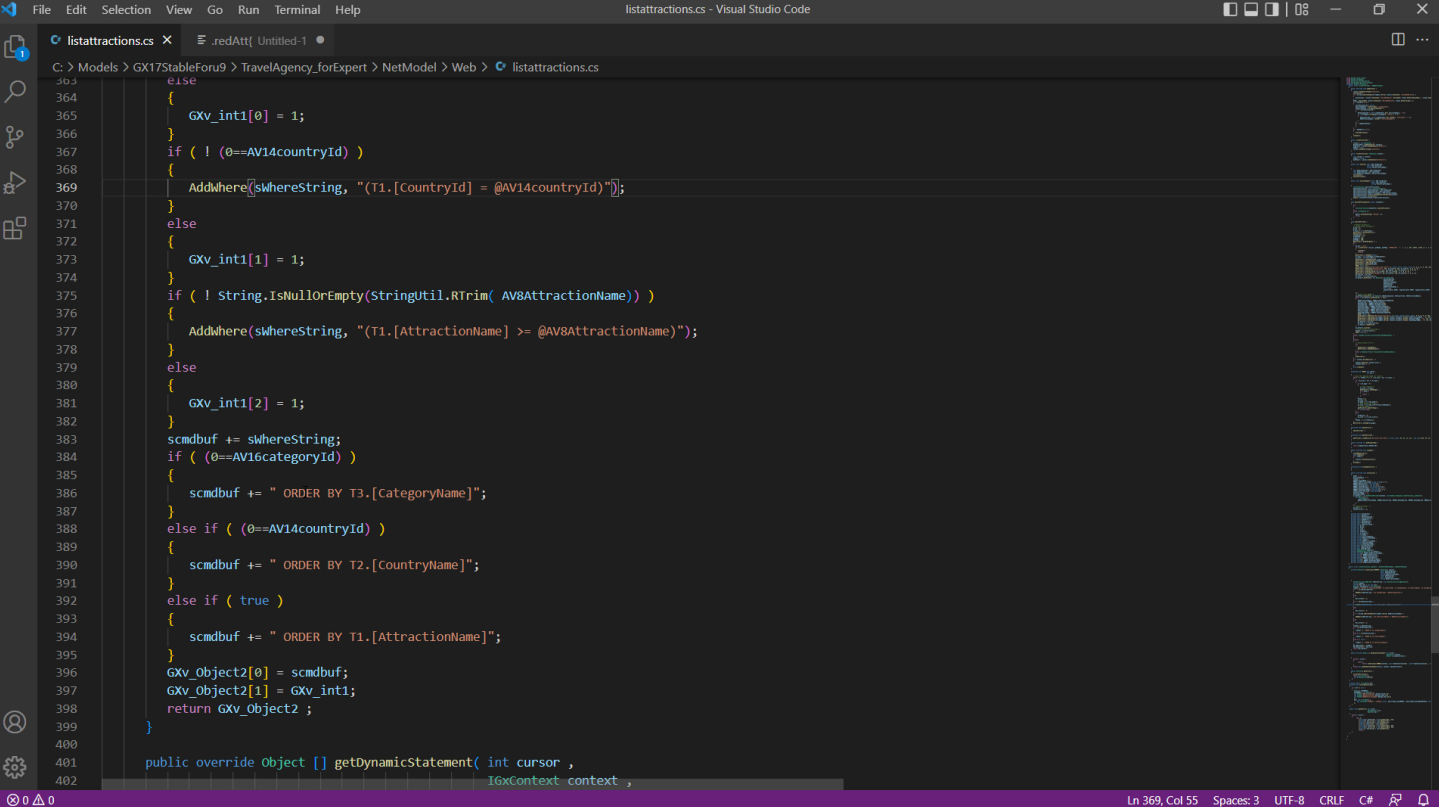
Vemos rápidamente que si no seleccionamos categoría se muestra ordenado por ella.

En cambio si seleccionamos categoría y dejamos el país sin seleccionar, entonces saldrá ordenado por país y desordenado por lo demás.

Y si ahora seleccionamos categoría y país, sale ordenado, ahora sí, por nombre de atracción.

```
File Edit Selection View Go Run Terminal Help listattractions.cs - Visual Studio Code
listattractions.cs x .redAtt[ Untitled-1
C: > Models > GX17StableForu9 > TravelAgency_forExpert > NetModel > Web > listattractions.cs
350 short A5CategoryId ,
351 short A3CountryId ,
352 string A20AttractionName )
353 {
354     System.Text.StringBuilder swhereString = new System.Text.StringBuilder();
355     string scmdbuf;
356     short[] GXv_int1 = new short[3];
357     Object[] GXv_Object2 = new Object[2];
358     scmdbuf = "SELECT T1.[AttractionName], T1.[CountryId], T1.[CategoryId], T2.[CountryName], T3.[CategoryName], T1.[AttractionId] FROM ((([Attraction] T1 INNER JOIN
359     if ( ! (0==AV16categoryId) )
360     {
361         AddWhere(swhereString, "(T1.[CategoryId] = @AV16categoryId)");
362     }
363     else
364     {
365         GXv_int1[0] = 1;
366     }
367     if ( ! (0==AV14countryId) )
368     {
369         AddWhere(swhereString, "(T1.[CountryId] = @AV14countryId)");
370     }
371     else
372     {
373         GXv_int1[1] = 1;
374     }
375     if ( ! String.IsNullOrEmpty(StringUtil.RTrim( AV8AttractionName)) )
376     {
377         AddWhere(swhereString, "(T1.[AttractionName] >= @AV8AttractionName)");
378     }
379     else
380     {
381         GXv_int1[2] = 1;
382     }
383     scmdbuf += swhereString;
384     if ( (0==AV16categoryId) )
385     {
386         scmdbuf += " ORDER BY T3.[CategoryName]";
387     }
388     else if ( (0==AV14countryId) )
389     {
```

Si vamos a investigar el fuente, para ver cómo construye la sentencia SQL que envía al manejador... vemos que primero arma la primera parte fija del select (la de los atributos a ser seleccionados y de qué tablas con los joins para acceder a la extendida...pero luego complementa dinámicamente a partir de la evaluación de las variables la parte del Where (agregando Wheres cuando las variables no estén vacías).



```
364     {
365         GXv_int1[0] = 1;
366     }
367     if ( ! (0==AV14countryId) )
368     {
369         AddWhere(swWhereString, "(T1.[CountryId] = @AV14countryId)");
370     }
371     else
372     {
373         GXv_int1[1] = 1;
374     }
375     if ( ! String.IsNullOrEmpty(StringUtil.RTrim( AV8AttractionName)) )
376     {
377         AddWhere(swWhereString, "(T1.[AttractionName] >= @AV8AttractionName)");
378     }
379     else
380     {
381         GXv_int1[2] = 1;
382     }
383     scmdbuf += swWhereString;
384     if ( (0==AV16categoryId) )
385     {
386         scmdbuf += " ORDER BY T3.[categoryName]";
387     }
388     else if ( (0==AV14countryId) )
389     {
390         scmdbuf += " ORDER BY T2.[countryName]";
391     }
392     else if ( true )
393     {
394         scmdbuf += " ORDER BY T1.[AttractionName]";
395     }
396     GXv_Object2[0] = scmdbuf;
397     GXv_Object2[1] = GXv_int1;
398     return GXv_Object2 ;
399 }
400
401 public override Object [] getDynamicStatement( int cursor ,
402     IGxContext context ,
```

Y para el ORDER BY de la sentencia SQL hace algo similar, solo que con if anidados, para reflejar justamente lo que decíamos antes, que solo una cláusula order será agregada al Select.

Estas evaluaciones para obtener la sentencia SQL final que se envía al DBMS para que resuelva la consulta se realizan dinámicamente, en tiempo de ejecución. Cada vez que se ejecuta este listado deberá ejecutar esta sección de código para componer la consulta final.



## For each

For each **Attraction**

```

order CategoryName when &categoryId.IsEmpty()
order CountryName when &countryName.IsEmpty()
order AttractionName

where CategoryId = &categoryId when not &categoryId.IsEmpty()
where CountryId = &countryId when not &countryId.IsEmpty()
where AttractionName >= &attractionName when not &attractionName.IsEmpty()

```

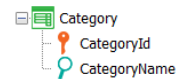
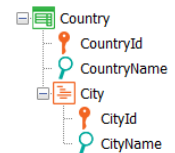
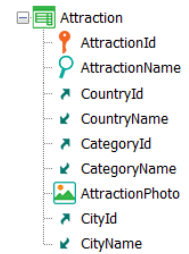
```

print info // CategoryName, CountryName, AttractionName

```

endfor

## endfor



Por lo que si este for each se incluyera dentro de otra estructura repetitiva que se ejecutara para millones de registros el costo del armado dinámico de la consulta podría ser importante.

For each **Attraction**

```
order CategoryName, CountryName, AttractionName
```

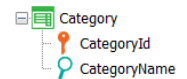
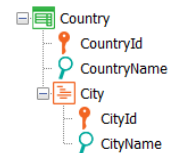
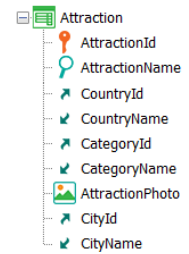
```
where CategoryId = &categoryId when not &categoryId.IsEmpty()
```

```
where CountryId = &countryId when not &countryId.IsEmpty()
```

```
where AttractionName >= &attractionName when not &attractionName.IsEmpty()
```

```
print info // CategoryName, CountryName, AttractionName
```

```
endfor
```



En el ejemplo que vimos utilizamos órdenes condicionales porque nos interesaba desplegar la información ordenada en forma distinta en base a condiciones. Es decir, las cláusulas order cumplieron un requerimiento lógico de la consulta. Eran parte de la letra del problema, digamos. Aunque no eran necesarias en este caso. Pensemos que bastaba con elegir este orden incondicional para satisfacer el requerimiento.

For each **Attraction**

```

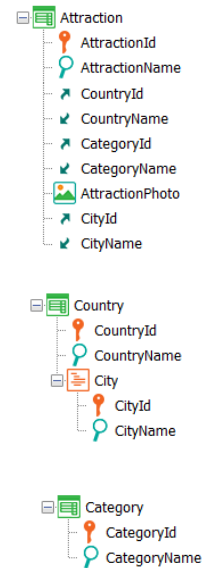
order CategoryId when not &categoryId.IsEmpty()
order CountryId when not &countryId.IsEmpty()
order AttractionName when not &attractionName.IsEmpty()

where CategoryId = &categoryId when not &categoryId.IsEmpty()
where CountryId = &countryId when not &countryId.IsEmpty()
where AttractionName >= &attractionName when not &attractionName.IsEmpty()

print info // CategoryName, CountryName, AttractionName

```

endfor



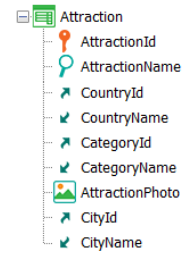
Pero muchas veces, así como vimos para el caso de una única cláusula order incondicional, ésta se especifica con objetivo de optimización y no es un requerimiento. En esos casos elegir órdenes compatibles con los filtros suele ser una buena práctica, sobre todo en el caso de DBMSs poco inteligentes.

Así, por ejemplo, si no nos importara en qué orden saldrá listada la información, podríamos colocar estas otras cláusulas order. Esto se traducirá dinámicamente del siguiente modo: si &categoryId no está vacía, entonces sabemos que la consulta se parecerá a...

```

For each Attraction
  order CategoryId
  where CategoryId = &categoryId
  where CountryId = &countryId when not &countryId.IsEmpty()
  where AttractionName >= &attractionName when not &attractionName.IsEmpty()
  print info // CategoryName, CountryName, AttractionName
endfor

```



```

For each Attraction
  order CategoryId
  where CategoryId = &categoryId
  where CountryId = &countryId
  where AttractionName >= &attractionName
  print info // CategoryName, CountryName, AttractionName
endfor

```

```

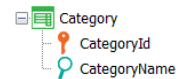
For each Attraction
  order CategoryId
  where CategoryId = &categoryId
  where AttractionName >= &attractionName
  print info // CategoryName, CountryName, AttractionName
endfor

```

```

For each Attraction
  order CategoryId
  where CategoryId = &categoryId
  where CountryId = &countryId
  print info // CategoryName, CountryName, AttractionName
endfor

```



...esta, donde dependiendo de si &countryId está o no vacía, y si &attractionName está o no vacía, tendremos la consulta final así, así o así.

Observemos que en cualquier caso como existe índice por CategoryId, al menos la primera cláusula Where estará optimizada.

For each **Attraction**

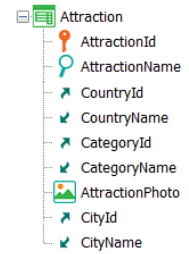
```

order CategoryId when not &categoryId.IsEmpty()
order CountryId when not &countryId.IsEmpty()
order AttractionName when not &attractionName.IsEmpty()
where CategoryId = &categoryId when not &categoryId.IsEmpty()
where CountryId = &countryId when not &countryId.IsEmpty()
where AttractionName >= &attractionName when not &attractionName.IsEmpty()

print info // CategoryName, CountryName, AttractionName

```

endfor



For each **Attraction**

```

order CountryId
where CountryId = &countryId
where AttractionName >= &attractionName
print info // CategoryName, CountryName, AttractionName
endfor

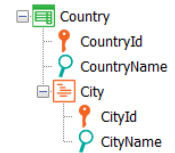
```

For each **Attraction**

```

order CountryId
where CountryId = &countryId
print info // CategoryName, CountryName, AttractionName
endfor

```



Si en cambio &categoryId está vacía, entonces si &countryId no, la consulta quedará así o así, dependiendo de si &attractionName no está vacía o sí lo está.

En cualquiera de los dos casos, estará optimizada en relación al filtro por CountryId, dado que tiene un índice, por ser clave foránea.

For each **Attraction**

```

order CategoryId when not &categoryId.IsEmpty()
order CountryId when not &countryId.IsEmpty()
order AttractionName when not &attractionName.IsEmpty()
where CategoryId = &categoryId when not &categoryId.IsEmpty()
where CountryId = &countryId when not &countryId.IsEmpty()
where AttractionName >= &attractionName when not &attractionName.IsEmpty()

print info // CategoryName, CountryName, AttractionName

```

endfor

For each **Attraction**

```

order AttractionName
where AttractionName >= &attractionName
print info // CategoryName, CountryName, AttractionName
endifor

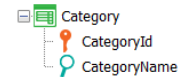
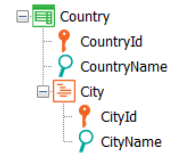
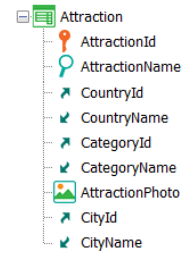
```

For each **Attraction**

```

print info // CategoryName, CountryName, AttractionName
Endfor

```



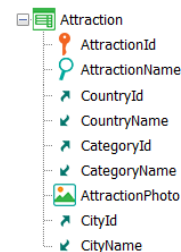
Si en cambio también &countryId está vacía, entonces si &attractionName no lo está la consulta quedará así. Y si sí lo está, quedará de este otro modo pero el order será indefinido. Eso significa que puede variar de DBMS en DBMS e incluso entre ejecuciones sucesivas.

En el primer caso como no sabemos de la existencia de un índice por AttractionName, no sabemos qué tan optimizada estará la consulta.

```

for each Attraction
  order CategoryId when not &categoryId.IsEmpty()
  order CountryId when not &CountryId.IsEmpty()
  order AttractionName when not &AttractionName.IsEmpty()
  where CategoryId = &categoryId when not &categoryId.IsEmpty()
  where CountryId = &countryId when not &countryId.IsEmpty()
  where AttractionName >= &AttractionName when not &AttractionName.IsEmpty()
  print attractionInfo
endfor

```



For Each Attraction (Line: 4)

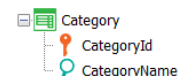
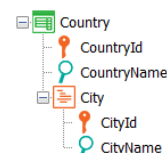
**Order:**      CategoryId WHEN not &categoryId. isempty()  
                  Index: IATTRACTION2  
                  CountryId WHEN not &countryId. isempty()  
                  Index: IATTRACTION1  
                  AttractionName WHEN not &AttractionName. isempty()  
                  No index!

**Navigation filters:** Start from:    FirstRecord  
                                  Loop while:    NotEndOfTable

**Constraints:**    CategoryId = &categoryId WHEN not &categoryId. isempty()  
                          CountryId = &countryId WHEN not &countryId. isempty()  
                          AttractionName >= &AttractionName WHEN not &AttractionName. isempty()

**Join location:**    Server

=Attraction ( AttractionId ) INTO AttractionName CountryId CategoryId  
                  =Country ( CountryId ) INTO CountryName  
                  =Category ( CategoryId ) INTO CategoryName



Si observamos el listado de navegación veremos que los filtros siguen mostrándose en la sección de Constraints, aunque sepamos que dependiendo de los valores de las variables algunos deberían mostrarse en los Navigation filters. Es que el listado de navegación no realiza la descomposición que hicimos antes. Debemos comprender, entonces, que el escenario será mejor que este que puede parecer a primera vista si no se tiene en cuenta todo lo anterior.

**For each** *BaseTrn<sub>1</sub>, ... , BaseTrn<sub>n</sub>*

```

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**

¿Qué más decir de los órdenes condicionales?

No se soportan en cortes de control.

No aplican a generadores legacy Cobol y RPG.

Si las condiciones tienen atributos, son considerados como instanciados, es decir son evaluados antes de comenzar la navegación y no cambian durante ella.

Con esto terminamos de explorar el tema de los orders de las consultas.





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
```

Por supuesto esto que vimos para el for each vale para grupos de Data Providers y grids con tabla base, así como para consultas con In en Data Selectors.