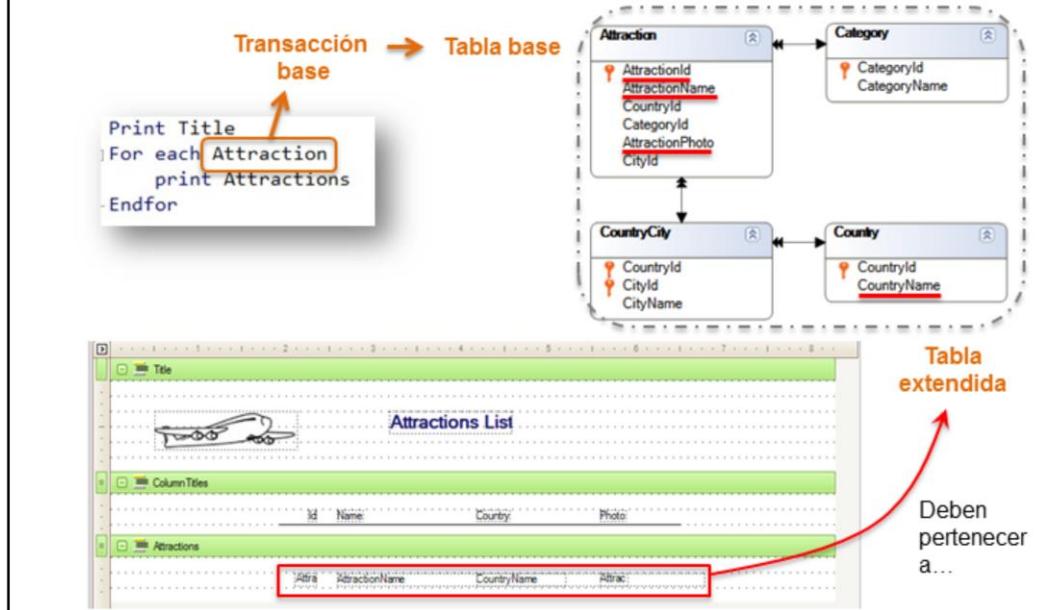


Más sobre el comando For Each

GeneXus™ 15

Tabla base

Repaso: Tabla base y Tabla Extendida de un For Each



Recordemos que GeneXus determina la tabla base del for each teniendo en cuenta el nombre de la transacción que declaramos al lado del for each (que debe ser la transacción cuya tabla física asociada queremos recorrer).

Además, los atributos declarados dentro del for each (printblocks, where, order, etc.), deben pertenecer a la tabla extendida de la tabla base del for each.

En el ejemplo presentado en la diapositiva, la tabla base del for each será ATTRACTION, o sea la tabla que se recorrerá; se accederá a su tabla extendida para acceder a los datos requeridos.

Repaso: Navegación resultante

Procedure AttractionsList Navigation Report

Name	AttractionsList	Environment	C# Default (C#)
Description	Attractions List	Spec. Version	15_0_1-106211
Output Devices	File	Form Class	Graphic
Main	Yes	Program Name	AttractionsList2
		Call Protocol	HTTP
		Parameters	

Levels

For Each Attraction (Line: 10)

Order: AttractionId **PK**
Index: IATTRACTION

Navigation Start from: FirstRecord
filters: Loop while: NotEndOfTable
Join location: Server

Toda la tabla

Attraction (AttractionId)
Country (CountryId)

The diagram shows four tables: Attraction, Category, CountryCity, and Country. Attraction has primary key AttractionId and attributes AttractionName, CountryId, CategoryId, AttractionPhoto, and CityId. Category has primary key CategoryId and attribute CategoryName. CountryCity has primary key CountryId and attributes CityId and CityName. Country has primary key CountryId and attribute CountryName. Relationships are shown with double-headed arrows: Attraction to Category, Attraction to CountryCity, and CountryCity to Country. A red dashed line with arrows points from the 'Toda la tabla' text to the Attraction table, and another red dashed line points from the 'Country' table to the CountryId attribute in the Attraction table.

El listado de navegación nos informa claramente que la tabla base es ATTRACTION, que la recorrida será ordenada por la clave primaria de dicha tabla: AttractionId, y que se recorrerá toda la tabla, accediendo a la tabla COUNTRY (para recuperar CountryName, el país de la atracción).

Determinación de la tabla base

- ¿Es obligatorio especificar transacción base para un for each?

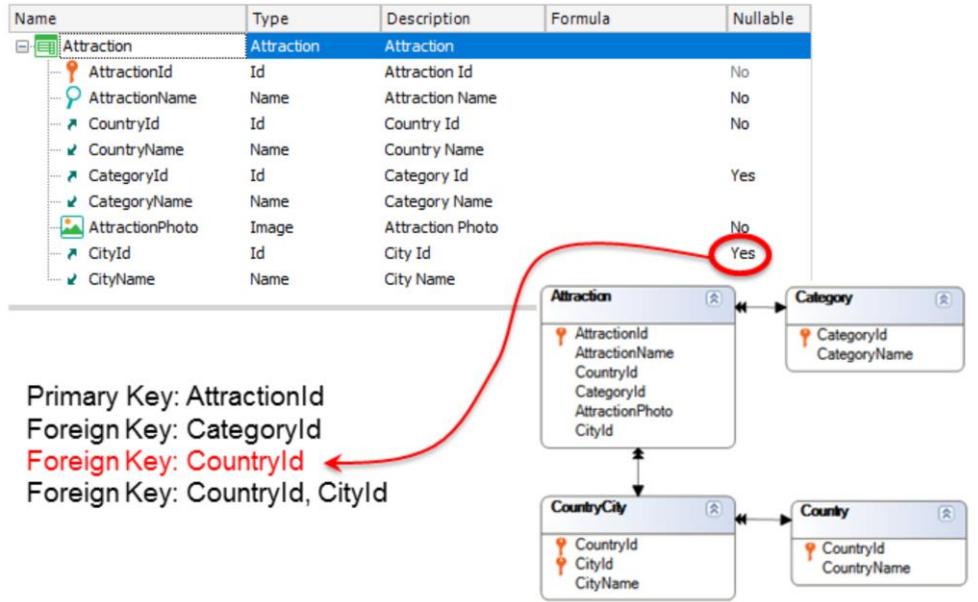
```
print Title
print ColumnTitles
For each
    print Attractions
endfor
```

- Respuesta: No. GeneXus podrá calcular la tabla base del for each a partir de los atributos que participen del comando. La forma en que encuentra la tabla base no será abordada en este curso.

Índices y su relación con las consultas a la BD

Cláusula order del for each

Claves e índices creados automáticamente por GeneXus



Si observamos la transacción Attraction, podemos ver que GeneXus define a nivel de la tabla física asociada cuatro claves: la primaria, y tres foráneas.

La tercera, la clave por CountryId que no se evidencia en el diagrama de tablas es creada únicamente para los casos en que el usuario deje vacío el valor de CityId.

Puesto que {CountryId, CityId} forman una clave foránea compuesta, si no fuera posible que el usuario dejara nulo el valor de CityId, es decir, si no pudiera no indicar valor de ciudad, entonces sería innecesaria una clave foránea por CountryId, dado que si existe un registro en CountryCity para ese país, es porque al ingresarlo ya se había controlado que existiera ese país en la tabla Country.

La clave foránea por CountryId aparece, entonces, sólo porque se habilitó la propiedad Nullable para CityId.

Claves e índices creados automáticamente por GeneXus

Attribute	Order	Description
Attraction Indexes		Attraction
IAAttraction	Primary Key	Automatic Index
• AttractionId	Ascending	Attraction Id
IAAttraction2	Foreign Key	Automatic Index
• CategoryId	Ascending	Category Id
IAAttraction1	Foreign Key	Automatic Index
• CountryId	Ascending	Country Id
• CityId	Ascending	City Id

Los **índices** son vías de acceso eficiente a los datos

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

Por cada PK y FK GeneXus crea un índice en la tabla asociada.

Excepción: un índice por {A, B} ya es, en particular, un índice por {A}

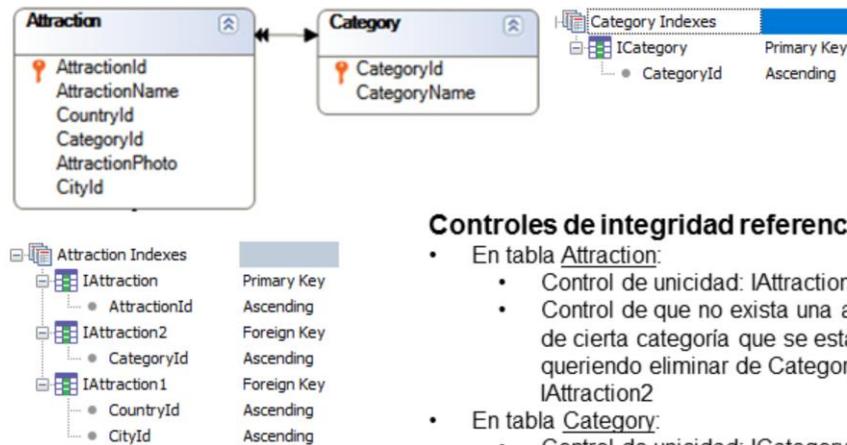
Los **índices** son vías de acceso eficiente a los datos. Podemos pensar por ejemplo, en un libro de cocina con muchas páginas que contienen recetas, el cual tiene varios índices (índice alfabético, índice por tipos de comidas, etc.). De igual forma, las tablas que almacenan registros tienen índices también.

GeneXus al crear tablas físicas crea para ellas un índice por el atributo primario de la tabla (es decir, por su clave primaria sea simple o compuesta) y un índice por cada clave foránea. Esto lo hace para que sean más eficientes los controles de consistencia de los datos entre tablas, como veremos en la siguiente página.

Si editamos la tabla Attraction en GeneXus, se muestra automáticamente la estructura que nos presenta su conformación. Pero si vamos a la solapa Indexes, podemos ver los índices que se crearán sobre esa tabla, en la base de datos.

Podemos ver que se crearán tres índices, con los nombres que vemos. Uno por la Primary Key, y dos por las Foreigns Keys. ¿Por qué no se crea un índice por CountryId solo? Porque es innecesario. Si tenemos un índice compuesto por CountryId, CityId, ese índice ya es, en particular, un índice por CountryId.

Claves e índices creados automáticamente por GeneXus

**Controles de integridad referencial:**

- En tabla Attraction:
 - Control de unicidad: IAttraction
 - Control de que no exista una atracción de cierta categoría que se está queriendo eliminar de Category: IAttraction2
- En tabla Category:
 - Control de unicidad: ICategory
 - Control de que cuando se va a insertar una atracción de cierta categoría en Attraction, ésta exista: ICategory

Estos índices, como decíamos, se crean para hacer eficientes los controles de integridad referencial que GeneXus realiza automáticamente en las transacciones.

Los **índices por clave primaria** se crean en las tablas para hacer eficiente el control de duplicados, y también para hacer eficiente la búsqueda cuando desde otra transacción se está queriendo insertar o modificar la clave foránea que refiere a esa clave primaria. En el ejemplo, cuando desde Attraction se está ingresando una nueva atracción, y hay que chequear que exista una categoría en la tabla Category con ese valor de CategoryId. Allí se utiliza el índice por PK de Category (ICategory).

Los **índices por clave foránea** se crean en las tablas para que cuando desde una transacción que tiene la clave primaria a la que esa clave foránea refiere, en nuestro caso Category, se quiera eliminar un registro, se pueda saber rápida y eficientemente si existe algún registro relacionado, para, en ese caso, impedir la eliminación. En nuestro caso, si vamos a eliminar una categoría desde la transacción Category, GeneXus debe saber, para permitirlo, que no existe ninguna atracción con esa categoría. Entonces usa el índice IAttraction2 de Attraction.

Claves candidatas e índices

- Para una entidad podemos tener más de un atributo o conjunto de atributos que la identifican, es decir, que sus valores no pueden repetirse.

Name	Type
Customer	Customer
PK → CustomerId	Numeric(4,0)
CustomerName	Character(20)
CustomerLastName	Character(20)
CK → CustomerDNI	DNI
CK → CustomerPassportNumber	PassportNumber
CustomerAddress	Address, GeneXus
CustomerPhone	Phone, GeneXus
CustomerEMail	Email, GeneXus
CustomerAddedDate	Date

- Una **clave candidata** se define creando un **índice único**.

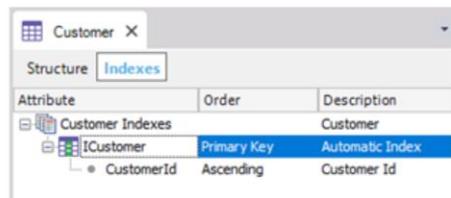
Como habíamos visto en la clase sobre relaciones 1 a 1, para cada nivel de cada transacción es obligatorio definir el atributo o conjunto de atributos que conforman el identificador del nivel. Ese identificador se traducirá a nivel de la tabla física en la clave o llave primaria de la tabla. Con esto estamos diciendo que los valores de este atributo o conjunto de atributos no podrán repetirse.

Pero en muchos casos hay más de un atributo o conjunto de atributos que deben cumplir esa condición. Por ejemplo, para el cliente elegimos identificarlo con un número interno de nuestro sistema, pero también podríamos tener como atributo secundario su DNI, documento nacional de identidad, expedido por su país, o incluso su número de pasaporte, que también deben ser únicos. Como tenemos que elegir a uno de los tres (CustomerId, CustomerDNI, CustomerPassportNumber) para identificar a la entidad (en nuestro caso elegimos CustomerId), si no hacemos nada más los otros quedarán como atributos secundarios, pudiendo repetirse.

¿Cómo le decimos a GeneXus que tanto CustomerDNI como CustomerPassportNumber son **claves candidatas**, para que él nos asegure que no se repitan para clientes diferentes? Ya habíamos visto que era definiendo un índice por cada clave candidata.

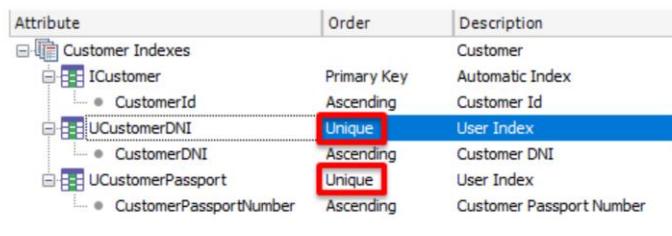
Claves candidatas e índices

- La tabla Customer sólo tiene definido un índice por PK.



Attribute	Order	Description
Customer Indexes		Customer
ICustomer	Primary Key	Automatic Index
CustomerId	Ascending	Customer Id

- El desarrollador debe crear índices **unique** para definir claves candidatas.



Attribute	Order	Description
Customer Indexes		Customer
ICustomer	Primary Key	Automatic Index
CustomerId	Ascending	Customer Id
UCustomerDNI	Unique	User Index
CustomerDNI	Ascending	Customer DNI
UCustomerPassport	Unique	User Index
CustomerPassportNumber	Ascending	Customer Passport Number

Si observamos los índices que automáticamente se han definido en la tabla Customer, vemos que tenemos únicamente el índice por clave primaria.

Debemos crear un índice por el atributo CustomerDNI, e indicarle que será de tipo **Unique**. Es decir, indicarle que no podrán repetirse sus valores.

Y lo mismo para el atributo CustomerPassportNumber.

De esta manera, GeneXus interpretará que debe utilizar cada índice unique que tenga definida la tabla para controlar la unicidad de esos valores. Es decir, si se está ingresando un nuevo cliente y el usuario digita un DNI que ya existe para otro cliente, la transacción disparará un error informando sobre esta situación y no permitirá grabar el registro nuevo.

Otros índices que debe crear el desarrollador para optimizar consultas

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

AttractionId	AttractionName	CountryName	AttractionPhoto

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: 10)

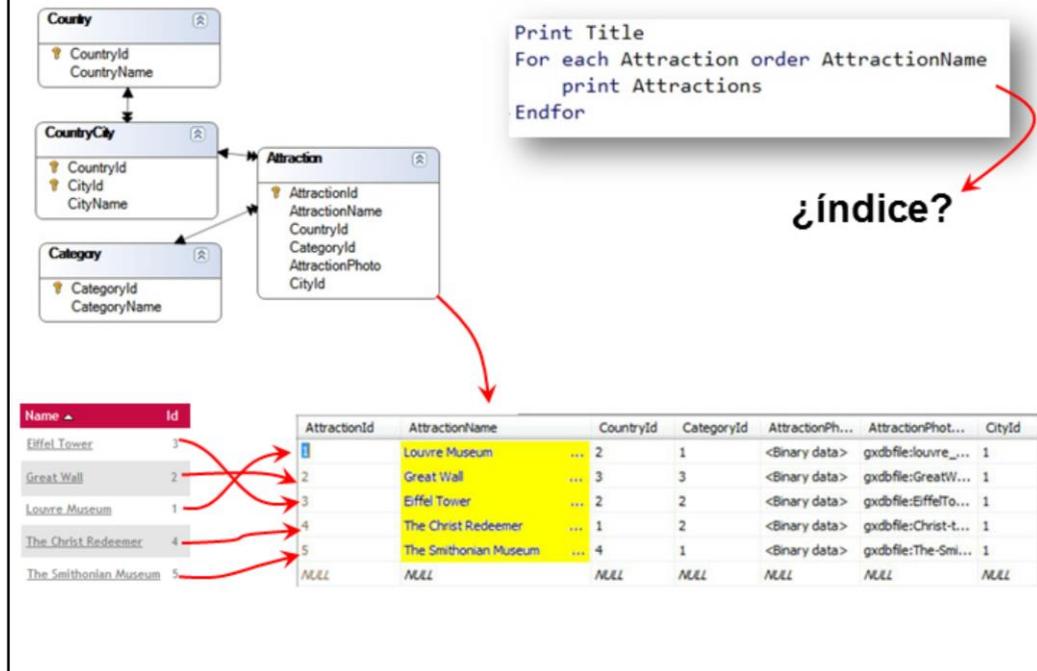
Order: [AttractionName](#)
! No index

Navigation Start from: FirstRecord
filters: Loop NotEndOfTable
while:
Join location: Server

[Attraction](#) ([AttractionId](#))
[Country](#) ([CountryId](#))

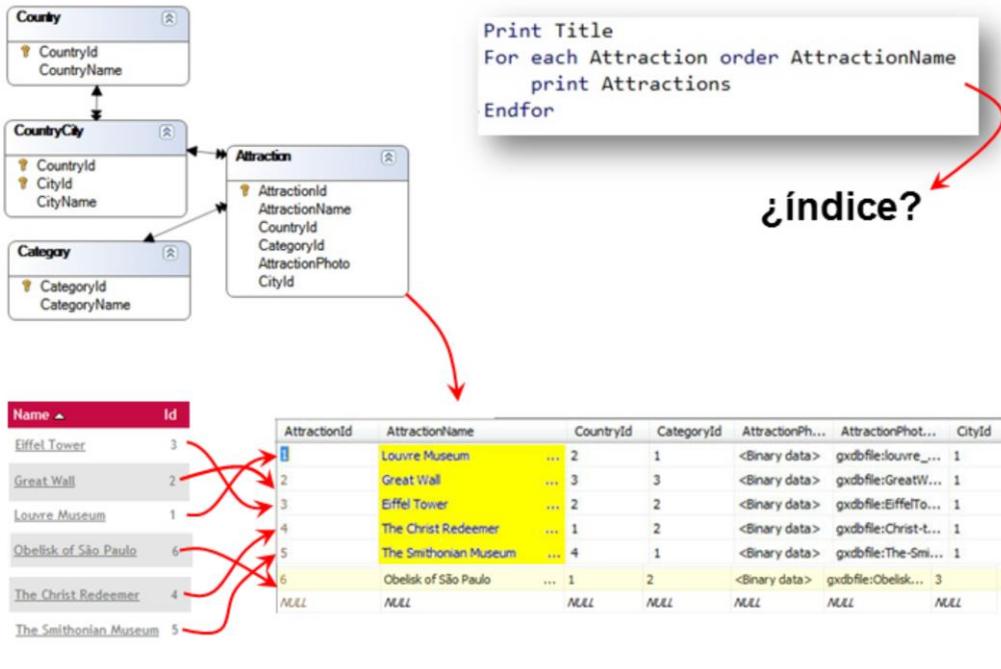
Ya habíamos visto que si agregamos una cláusula order para ordenar por nombre de atracción, el listado de navegación nos da un aviso, informándonos de que en la base de datos no existe un **índice** por el atributo por el que necesitamos ordenar la información, por lo que podríamos tener baja performance para esta consulta.

Es que al indicarle un atributo por el que ordenar, GeneXus intenta que la ordenación sea eficiente y por lo tanto busca si existe un índice por ese atributo. Como no lo encuentra, nos lo hace saber.



Supongamos que la tabla ATTRACTION tiene los datos que se muestran. Si necesitamos obtener sus registros ordenados por el atributo AttractionName, entonces tendrán que reordenarse los registros ya que por defecto están ordenados por el atributo que es clave primaria.

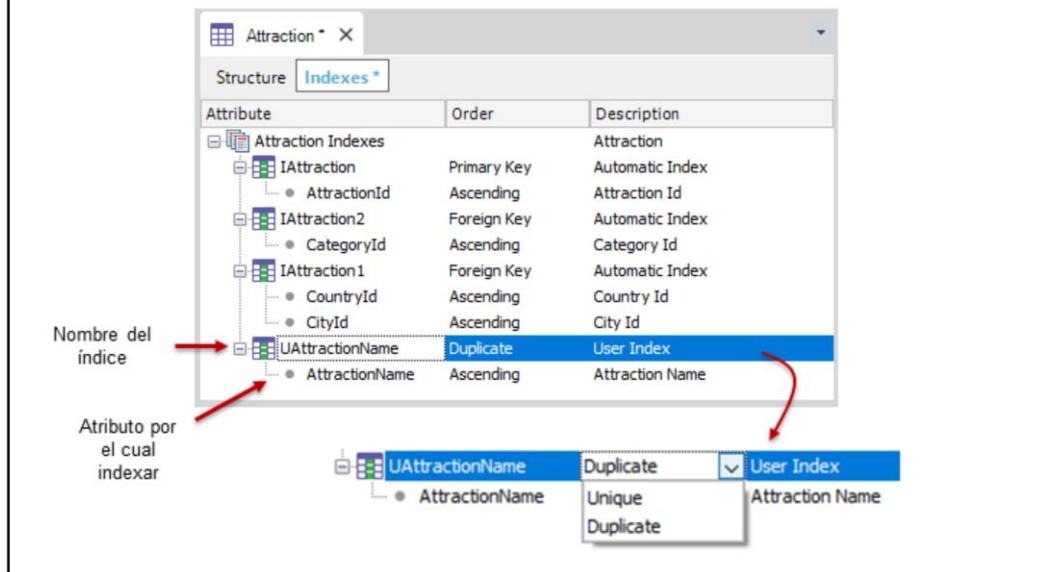
Cuando se define una consulta, si hay un índice físico creado en la tabla por el atributo a ordenar, GeneXus lo usará. Pero en este caso la consulta se necesita ordenada por un atributo secundario: AttractionName. Y GeneXus nos advierte en el listado de navegación asociado al objeto, que no hay un índice definido.



La existencia del índice optimizaría la consulta. Pero la desventaja de crear un índice es que, a partir de allí, debe ser mantenido. Es decir, si los usuarios van agregando, modificando o eliminando atracciones en la tabla ATTRACTION, debe reacomodarse el índice (o sea, los punteros del índice deben reacomodarse de forma tal de tener incluidas las nuevas atracciones, donde correspondan, para mantener el orden).

Crear un índice desde GeneXus para una tabla de la base de datos es sencillo y puede hacerse en cualquier momento. Y así como lo creamos, podemos eliminarlo en cualquier momento.

Definición de índices de usuario (unique o duplicate)



Definir un índice para una tabla de la base de datos es sencillo y puede hacerse en cualquier momento.

¿Cómo? Buscamos la tabla, la abrimos y vamos a la sección relacionada a los índices definidos.

Los tres primeros que vemos en el ejemplo, que aparecen anteceditos por el prefijo "I", son los creados automáticamente por GeneXus a partir de las claves primaria y foráneas.

Necesitamos crear uno nuestro, es decir de usuario. Para ello presionamos enter, tras lo que aparecerá el nombre por defecto UAttraction. Lo modificamos a nuestro gusto (agregándole Name al final, por ejemplo). El prefijo "U" es por User.

Deseamos que este índice esté compuesto por el atributo AttractionName, ordenado en sentido ascendente.

Si fuera un requisito que los nombres de atracciones no pudieran repetirse, podemos controlarlo indicando que el índice sea **Unique**, y no **Duplicate**, como ya vimos. Si definimos para un índice que sea Unique, se controlará **automáticamente** cuando se ingrese una atracción (o modifique su nombre), que no exista otra con el mismo nombre –utilizando este índice–. En nuestro ejemplo los nombres pueden repetirse (por ejemplo pensemos que cada país suele tener un Obelisco), así que para este índice por AttractionName, dejamos el valor: Duplicate.

Otros índices que debe crear el desarrollador para optimizar consultas

Procedure AttractionsList2 Navigation Report

Name	AttractionsList2	Environment	C#	Default (C#)
Description	Attractions List2	Spec. Version	15_0_1-106211	
Output Devices	File	Form Class	Graphic	
Main	Yes	Program Name	AttractionsList2	
		Call Protocol	HTTP	
		Parameters		

Levels

For Each Attraction (Line: 10)

Order: **AttractionName**
Index: UATTRACTIONNAME

Navigation Start FirstRecord
filters: from:
Loop NotEndOfTable
while:
Join Server
location:
=Attraction (AttractionId)
=Country (CountryId)

... Y a partir de allí lo utiliza cada vez que lo necesite.

Nos informa que utilizará el índice que se acaba de crear.

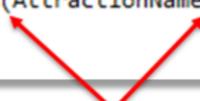
Así como lo creamos, en cualquier momento podemos eliminarlo, y al hacer F5 y reorganizar, volveremos a la situación de la que habíamos partido antes de crearlo.

La decisión de si crear o no el índice dependerá del DBMS con el que se cuente, de la frecuencia con la que se ejecutarán consultas que deban ordenar por AttractionName, y de la frecuencia con la que se actualicen los datos de la tabla.

Orden descendente

- ¿Cómo hacemos para ordenar el listado de atracciones por nombre de atracción, pero en orden alfabético inverso?
- Rodear de paréntesis los atributos que se quieran ordenar en forma descendente:

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



¿Cómo hacemos para solicitar un orden descendente? Simplemente rodeando de paréntesis curvos al atributo o atributos.

Órdenes compatibles con los filtros

Ordenar en forma compatible con los filtros

```

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

```



Name	Id	AttractionId	AttractionName	CountryId	CategoryId	AttractionPh...	AttractionPhot...	CityId
Eiffel Tower	3	1	Louvre Museum	2	1	<Binary data>	gxdfile:louvre_...	1
Great Wall	2	2	Great Wall	3	3	<Binary data>	gxdfile:GreatW...	1
Louvre Museum	1	3	Eiffel Tower	2	2	<Binary data>	gxdfile:EffeTo...	1
Obelisk of São Paulo	6	4	The Christ Redeemer	1	2	<Binary data>	gxdfile:Christ-t...	1
The Christ Redeemer	4	5	The Smithsonian Museum	4	1	<Binary data>	gxdfile:The-Smi...	1
The Smithsonian Museum	5	6	Obelisk of São Paulo	1	2	<Binary data>	gxdfile:Obelsk...	3
		NULL	NULL	NULL	NULL	NULL	NULL	NULL

Supongamos que lo que nos interesa es obtener un listado de las atracciones cuyos nombres estén alfabéticamente entre un par de valores recibidos por parámetro. Por ejemplo, entre la "F" y la "N".

Para eso especificamos las cláusulas where que se ven arriba.

Tener varias cláusulas where es equivalente a tener una sola, donde las condiciones se conjugan con el operador lógico "and". Es decir, se considerarán sólo los registros que cumplan con todas las condiciones **a la vez**.

Si vamos a filtrar por AttractionName, y tenemos un índice creado por ese atributo, nos convendrá siempre **ordenar por AttractionName** para optimizar la consulta. De hacerlo,

Ordenar en forma compatible con los filtros

```

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

```

Levels

For Each Attraction (Line: 10)

Order: AttractionName
 Index: UATTRACTIONNAME

Navigation filters:
 Start: AttractionName >= &NameFrom
 from:
 Loop: AttractionName <= &NameTo
 while:

Join location:
 Server
 =Attraction (AttractionId)
 =Country (CountryId)

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: 10)

Order: AttractionName
 | No index

Navigation filters:
 Start: AttractionName >= &NameFrom
 from:
 Loop: AttractionName <= &NameTo
 while:

Join location:
 Server
 =Attraction (AttractionId)
 =Country (CountryId)

¡Consulta optimizada!

Observemos que ordenando por el atributo por el que estamos filtrando por menor o igual y por mayor o igual hace que no se recorra toda la tabla. En caso de existir índice creado por el desarrollador, GeneXus utiliza ese índice y la consulta estará optimizada.

En caso de no existir índice, y dependiendo del DBMS, se creará en forma temporal y luego de utilizarse se eliminará. Pero los manejadores suelen tener estrategias de optimización que podrían no requerir crear estos índices temporales. No profundizaremos en esto.

Ordenar en forma compatible con los filtros

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

¡Consulta no optimizada!

Levels
For Each Attraction (Line: 10)

Order: AttractionId ← PK
Index: IATTRACTION

Navigation
Start from: FirstRecord
Loop while: NotEndOfTable ← Recorrerá toda la tabla...

Constraints: AttractionName >= &NameFrom
AttractionName <= &NameTo } ...aplicando para cada registro estas restricciones

Join location: Server

=Attraction (AttractionId)
=Country (CountryId)

Observemos que si no especificamos cláusula order, GeneXus ordenará por clave primaria, y deberá recorrerse toda la tabla para saber si una atracción está dentro del rango del where o no.

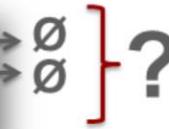
Poder condicionar la aplicación de
los órdenes y los filtros.

Cláusulas **When**

```

For each Attraction order AttractionName
  Where AttractionName >= &NameFrom
  Where AttractionName <= &NameTo
  print Attractions
Endfor

```



```

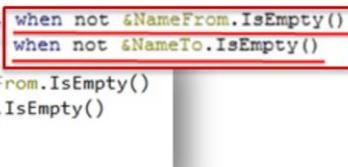
For each Attraction
  Where AttractionName >= &NameFrom when not &NameFrom.IsEmpty()
  Where AttractionName <= &NameTo when not &NameTo.IsEmpty()
  print Attractions
Endfor

```

```

For each Attraction order AttractionName
  Where AttractionName >= &NameFrom when not &NameFrom.IsEmpty()
  Where AttractionName <= &NameTo when not &NameTo.IsEmpty()
  print Attractions
Endfor

```



¿Qué resultado se obtendrá para el for each de arriba si las variables &NameFrom y &NameTo están vacías? Si existiera una atracción con nombre vacío, será la única devuelta, pues será la única que cumplirá ambas condiciones. En caso contrario, ninguna atracción será listada.

¿Es posible condicionar los ordenamientos y los filtros, para que sólo se apliquen ante determinadas circunstancias? Por ejemplo, que sólo se aplique el primer where **cuando** la variable &NameFrom no esté vacía. Y que sólo se aplique el segundo where **cuando** la variable &NameTo no esté vacía. La respuesta es sí. Lo conseguimos condicionando las cláusulas where con **when**, como vemos en el segundo for each. Sólo se aplicará cada where cuando la condición del when se satisfaga. Así, en ejecución, cuando dejemos ambas variables vacías, no se aplicará ninguno de los where, por lo que saldrán listadas todas las atracciones de la tabla. Si la variable &NameFrom está vacía pero &NameTo no, no se aplicará el primer where pero sí el segundo, por lo que se listarán todas las atracciones cuyo nombre será menor o igual a &NameTo.

De la misma manera puede condicionarse la aplicación o no de un order, como mostramos en el tercer for each. De hecho puede especificarse una sucesión de órdenes condicionados, de manera que el primero cuya condición se satisfaga sea el elegido.

Vea más de órdenes y filtros en el wiki de GeneXus (ie: <http://wiki.genexus.com/commwiki/servlet/wiki?6075,Order+clause>).

Cláusula **When none**

Cuando no hay registros a recuperar en el for each

```

For each Attraction
  Where AttractionName >= &NameFrom
  Where AttractionName <= &NameTo
  print Attractions
Endfor

```

&NameFrom 'A' &NameTo 'B'

AttractionId	AttractionName	CountryId	CategoryId	Attr...	Attr...	CityId
1	Louvre Museum ...	2		<Bna...	gxdbfi...	1
2	Great Wall ...			<Bna...	gxdbfi...	1
3	Eiffel Tower ...			<Bna...	gxdbfi...	1
4	The Christ Rede...		2	<Bna...	gxdbfi...	1
5	The Smithsonian ...		1	<Bna...	gxdbfi...	1
6	Obelisk of São P...			<Bna...	gxdbfi...	3
NAL	NAL	NAL	NAL	NAL	NAL	NAL

```

For each Attraction
  Where AttractionName >= &NameFrom
  Where AttractionName <= &NameTo
  Print Attractions
when none
  print warningMessage
endfor

```

warningMessage

There is no attraction in the range

↘

Si se incluye un for each, se considera independiente.

¿Qué pasa cuando ninguno de los registros de la tabla base cumple con las condiciones?

Supongamos que queremos en ese caso imprimir en la salida un mensaje que lo advierta... para eso programamos la cláusula **when none**.

Todos los comandos que se escriban entre el when none y el endfor se ejecutarán secuencialmente y **en el único caso en que no se hayan encontrado registros de la tabla base del for each que cumplieran las condiciones**.

En nuestro caso hemos decidido imprimir un mensaje, pero se podrían escribir una serie de comandos, como otro for each, por ejemplo.

Como la ejecución de lo que siga al **when none** implicará que no se encontró lo que se buscaba, si allí se escribe un for each, no se anidará al del when none. Será como un for each independiente.

Resumen

Sintaxis del for each

```
For each BaseTransaction  
  order att1, att2, ... , attn [when condition]  
  order att1, att2, ... , attn [when condition]  
  where condition [when condition]  
  where condition [when condition]  
  
  main code  
  
  When none  
  .....
```

endfor

Como ya hemos visto, la **tabla base** de un For each se determina a partir de la transacción base especificada; el resto de los atributos mencionados, tanto en el cuerpo del For each (main code) como en las cláusulas Order y Where, deberán pertenecer a la tabla extendida de esa tabla base (por eso aparecen los subrayados en la sintaxis que presentamos arriba).

Los atributos mencionados en el bloque When none no son considerados.

Dejamos en gris todo lo que ya habíamos visto antes. Aquí se agregan las cláusulas **when** y **when none**.

Más adelante veremos que se agregan más cláusulas a este fundamental comando de acceso a la base de datos

GeneXus™

Videos

training.genexus.com

Documentación

wiki.genexus.com

Certificaciones

training.genexus.com/certificaciones