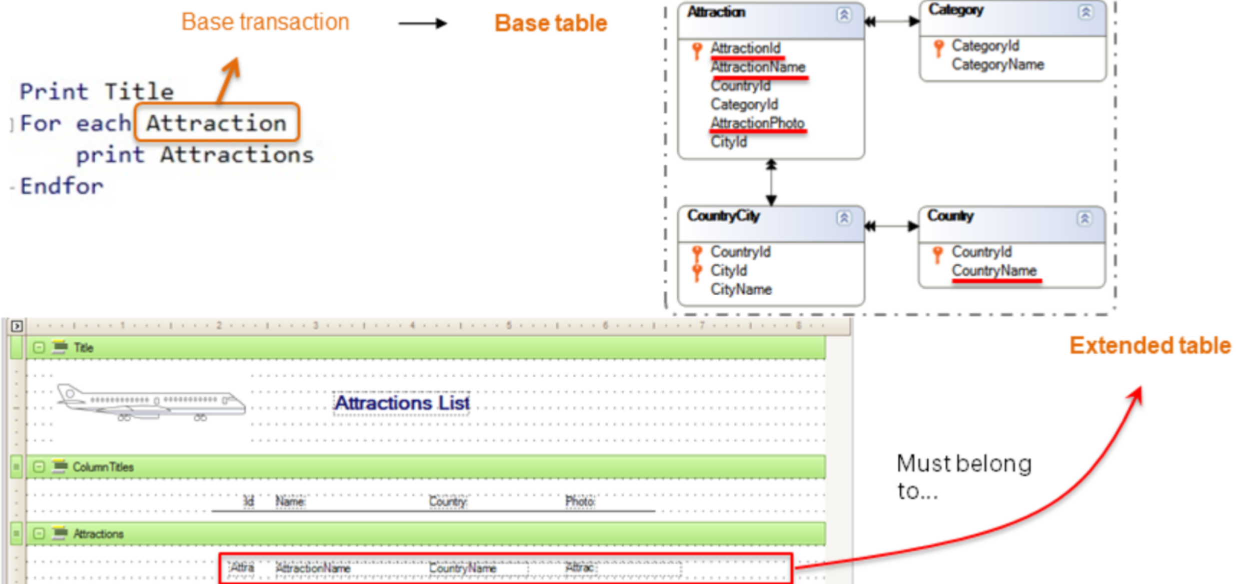
More about the For Each Command

GeneXus™ 16

GeneXus

Base table

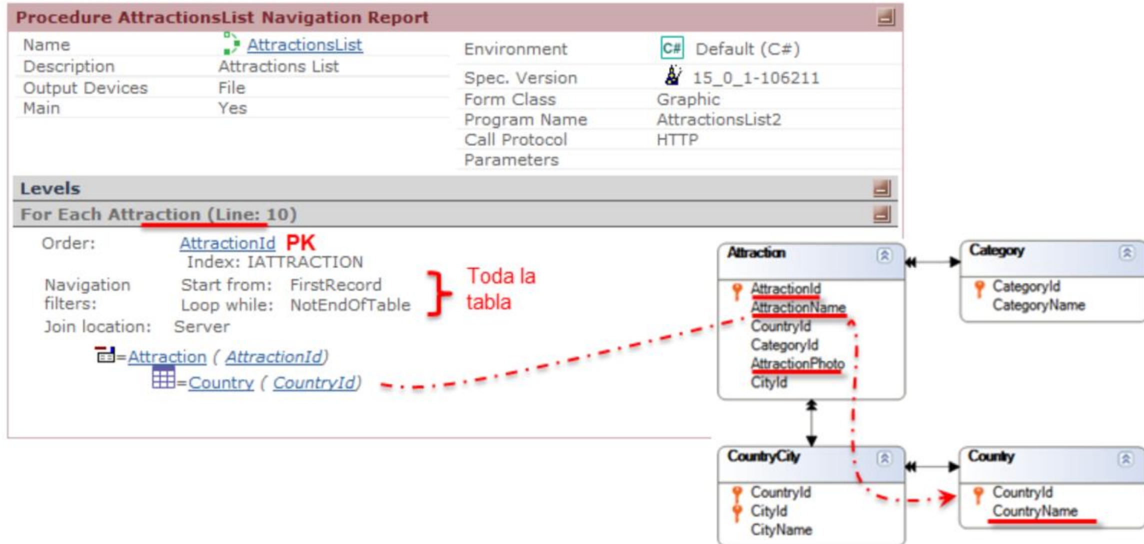


Remember that GeneXus determines the base table of the For each by the transaction name that is mentioned next to it. In the example: Attraction
Why? Because it is the name of the transaction **whose associated physical table** we want to navigate.

In addition, all the other attributes inside the for each (printblocks, where, order, etc) must belong to the extended table of the base table of the For each.

In the example shown in the slide, the base table of the For Each command –the base table that will be run through– will be ATTRACTION; its extended table will be accessed in order to reach the required data.

Review: Navigation list



The navigation list informs us that the base table is ATTRACTION, that the navigation order will be determined by the primary key, AttractionId, and that the entire table will be run through, accessing COUNTRY –in order to retrieve CountryName, the attraction country–.

Determining the base table

- Must we indicate a base transaction for a For Each command?

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

- **Answer**:
  No. GeneXus will be able to calculate the base table of the ForEach comand from the attributes included in the command. The way in which it finds the base table will not be covered in this course.

GeneXus

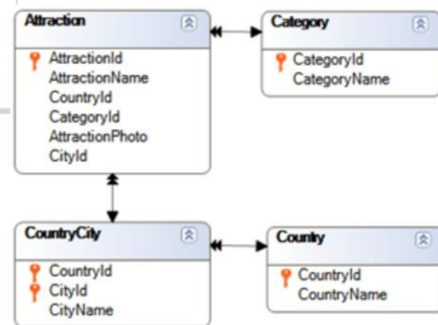
Indexes and their relationship with database queries

Order clause of the For Each command

Keys and indexes automatically created by GeneXus

Name	Type	Description	Formula	Nullable
Attraction	Attraction	Attraction		
AttractionId	Id	Attraction Id		No
AttractionName	Name	Attraction Name		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		
CategoryId	Id	Category Id		Yes
CategoryName	Name	Category Name		
AttractionPhoto	Image	Attraction Photo		No
CityId	Id	City Id		Yes
CityName	Name	City Name		

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



If we consider the Attraction table, we can see that GeneXus defines four keys at the associated physical table level: the primary key and three foreign keys.
The third key by CountryId, which is not displayed in the table diagram, is created oOnly for those cases when the user leaves the CityId value empty.
Since {CountryId, CityId} make up a compound foreign key, if it wasn't possible for the user to leave the CityId value null –that is to say, if he always had to enter a city value– a foreign key by CountryID would be unnecessary, because for a record to exist in CountryCity for that country, a control must have already been made to check for the country's existence in the Country table.
The foreign key for CountryId appears, then, only because the Nullable property has been enabled for CityId.

GeneXus

Keys and indexes automatically created by GeneXus

Attribute	Order	Description
Attraction Indexes		Attraction
IAttraction	Primary Key	Automatic Index
AttractionId	Ascending	Attraction Id
IAttraction2	Foreign Key	Automatic Index
CategoryId	Ascending	Category Id
IAttraction1	Foreign Key	Automatic Index
CountryId	Ascending	Country Id
CityId	Ascending	City Id

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

For each PK and FK, GeneXus creates an index in the associated table

Exception: an index by {A, B} already is, in particular, an index by {A}

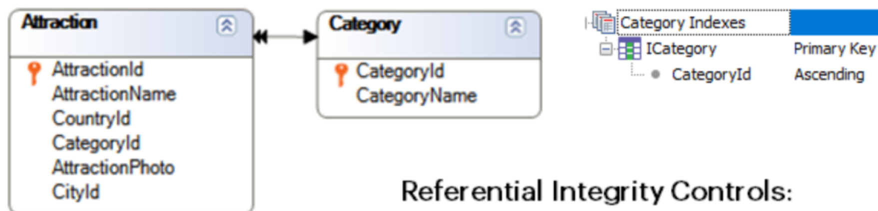
Indexes are an efficient way to access data. For example, we can consider a cook book with many pages containing recipes, and with several indexes (alphabetic index, food type index, and so on). Likewise, the tables that store records also have indexes.

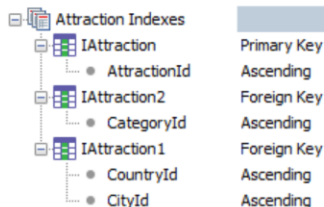
When creating physical tables, GeneXus creates for them one index by the primary attribute of the table (that is to say, by its primary key, whether it is simple or compound), and one index by each foreign key. This is done in order to control the consistency of data between tables more efficiently, as we will see in the following page.

If we edit the Attraction table in GeneXus, the structure that indicates how it is made up is automatically displayed. But if we open the Indexes tab, we can see the indexes that will be created over that table, in the database.

We can see that three indexes will be created, with the names displayed. One by the Primary Key, and two by the Foreign Keys. Why isn't an index by CountryId created? Because it's not necessary. If we have a compound index by CountryId, CityId, this is already an index by CountryId.

Keys and indexes automatically created by GeneXus





Referential Integrity Controls:

- In the Attraction table:
    - Uniqueness Control: IAttraction
    - It checks that there are no attractions of a certain category that we're trying to delete from Category: IAttraction2
- In the Category table:
    - Uniqueness Control: ICategory
    - When trying to insert an attraction in a certain category of Attraction, it checks that it exists: ICategory

As we said, these indexes are created to improve the efficiency of the referential integrity controls automatically made by GeneXus in transactions.

Primary key indexes are created in tables to make duplicate controls more efficient, and also to enhance searches when trying to insert or change the foreign key that makes reference to that primary key from another transaction. For example, when we're inserting a new attraction from Attraction, and we need to check whether the Category table contains a category with that CategoryId value. In this case we use the PK index of Category (ICategory).

Foreign key indexes are created in tables for cases when we want to delete a record from a transaction that has the primary key to which that foreign key makes reference –in our case, Category. They allow us to know, quickly and efficiently, if there is any related record in order to prevent the record from being deleted. In this case, if we are going to delete a category from the Category transaction, GeneXus must know that there aren't any attractions with that category in order to allow it. So, it uses the IAttraction2 index of Attraction.

Candidate keys and indexes

- For an entity we may have more than one attribute or set of attributes that identify it; that is to say, their values cannot be repeated.

	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

- A **candidate key** is defined as a **unique index**.

As we had said in the class about 1 to 1 relationships, for every level in every transaction we must define the attribute or set of attributes that make up the level identifier. At the physical table level, this identifier will be translated into the primary key of the table. This means that the values of this attribute or set of attributes must not be repeated.

But in many cases, there's more than one attribute or set of attributes that must meet this condition. For example, we choose to identify customers with an internal number in our system, but we may also use as secondary attribute his DNI or ID card –issued in his country– or even this passport number, which must also be unique. Since we have to select one of these three attributes (CustomerId, CustomerDNI, CustomerPassportNumber) to identify the entity (in this case we choose CustomerId), if we don't do anything else the others will be used as secondary attributes, and they may be repeated.

How do we tell GeneXus that both CustomerDNI and CustomerPassportNumber are **candidate keys**, so that it makes sure that they are not repeated for different customers? We had already seen that we had to define an index for every candidate key.

Candidate keys and indexes

- The Customer table only has one index defined by PK

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

- The developer must create **unique indexes** to create candidate keys.

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

If we look at the indexes that have been automatically defined in the Customer table, we can see that we only have the primary key index.

We need to create an index for the CustomerDNI attribute and set it as **Unique**. That is to say, tell it that its values cannot be repeated.

The same applies to the CustomerPassportNumber attribute.

In this way, GeneXus will infer that it must use every unique index that the table has defined to control the uniqueness of these values. That is to say, if, when entering a new customer the user types an ID card that already exists for another customer, the transaction will trigger an error to inform him about this situation and will not allow saving the new record.

Other indexes that must be created by the developer to optimize queries

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

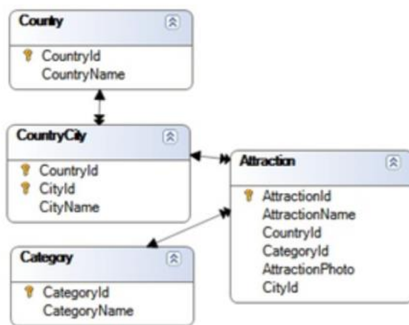
Attractions
AttractionPhoto
AttractionId AttractionName CountryName

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

We had already seen that if we add an Order clause by attraction name, the navigation list shows a warning to inform us that the database doesn't have an **index** for the attribute by which we need to order the data. As a result, the query may have poor performance.

When we give GeneXus an attribute by which to order data, it tries to order it in an efficient way; therefore, it looks for an index by that attribute. Next, it informs us that it can't find one.




```
Print Title
For each Attraction order AttractionName
    print Attractions
Endfor
```

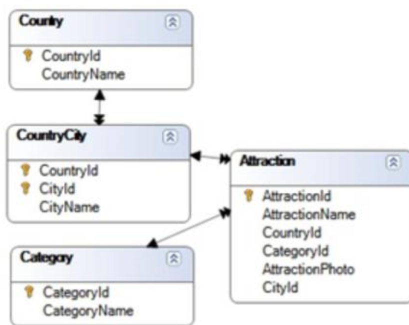
Index?

Name	Id
Eiffel Tower	3
Great Wall	2
Louvre Museum	1
The Christ Redeemer	4
The Smithonian Museum	5

AttractionId	AttractionName	CountryId	CategoryId	AttractionPh...	AttractionPhot...	CityId
1	Louvre Museum ...	2	1	<Binary data>	gxdbfile:louvre_...	1
2	Great Wall ...	3	3	<Binary data>	gxdbfile:GreatW...	1
3	Eiffel Tower ...	2	2	<Binary data>	gxdbfile:EiffelTo...	1
4	The Christ Redeemer ...	1	2	<Binary data>	gxdbfile:Christ-t...	1
5	The Smithonian Museum ...	4	1	<Binary data>	gxdbfile:The-Smi...	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Suppose that the ATTRACTION table contains the data that is displayed. If we need to obtain its records ordered by the AttractionName attribute, the records will have to be reordered, because by default they are ordered by the attribute that is a primary key.

When a query is created, if there is a physical index created in the table for the attribute to order by, GeneXus will use it. In this case, the query has to be ordered by a secondary attribute: AttractionName. In the navigation list associated with the object, GeneXus warns us that an index hasn't been created.




```
Print Title
For each Attraction order AttractionName
    print Attractions
Endfor
```

Index?

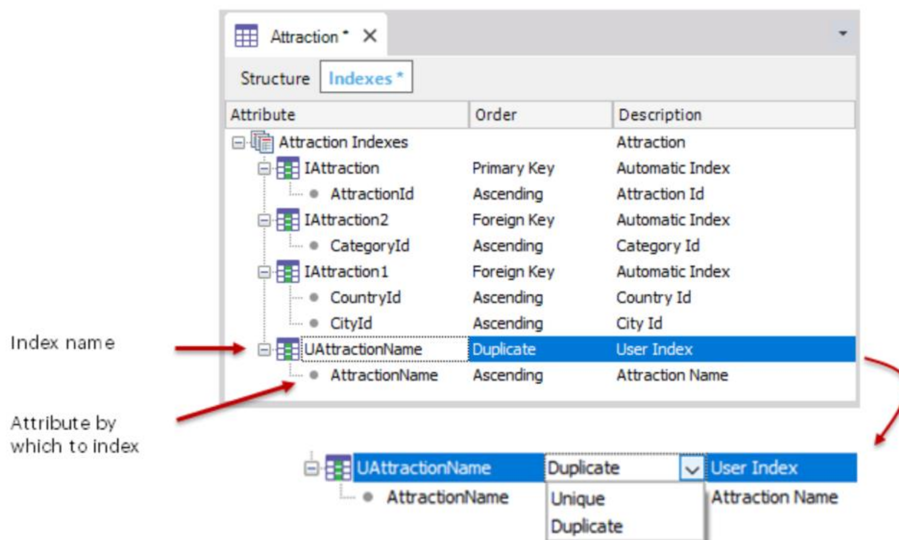
Name ▲	Id
Eiffel Tower	3
Great Wall	2
Louvre Museum	1
Obelisk of São Paulo	6
The Christ Redeemer	4
The Smithonian Museum	5

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

The existence of an index would optimize the query. On the other hand, the disadvantage of creating an index is that it must be maintained. That is to say, if users add, change or delete attractions in the ATTRACTION table, the index must be rearranged (the index pointers must be rearranged so that the new attractions are included in the corresponding locations to keep the order).

Creating an index in GeneXus for a database table is a simple task, and it can be done at any time. In addition to creating it, we can delete it at any time.

Creation of user indexes (unique or duplicate)



Creating an index for a database table is a simple task, and it can be done at any time.

How? We look for the table, open it and go to the section related to the indexes defined.

The first three that we see in the example, which are preceded by the prefix "I", are those automatically created by GeneXus based on the primary and foreign keys.

We need to create our own index, that is to say, a user index. To do so, we press Enter, which will display the default name UAttraction. We change it as needed (by adding Name at the end, for example). The prefix "U" corresponds to User.

We want this index to be made up by the AttractionName attribute, arranged in ascending order.

If a requirement stated that attraction names must not be repeated, we can control it by indicating that the index be **Unique**, and not **Duplicate**, as we've seen before. If we set an index as Unique, when entering an attraction (or changing its name) a control will be **automatically** made to check that there isn't another one with the same name –using this index. In our example, names can be repeated (for example, consider that countries usually have an obelisk), so for this index by AttractionName, we leave the value: Duplicate.

Other indexes that must be created by the developer to optimize queries

Database needs to be reorganized.

This report describes Database changes and how they will be handled by reorganization programs.
Please select Reorganize to proceed or Cancel.

Reorganize | Cancel

Pattern:
Attraction

Una vez definido el índice, GeneXus debe reorganizarla tabla, para crearlo.

Table Attraction specification

Table name: Attraction

Attraction needs conversion

Table Structure

Attribute	Definition	Previous values	Takes value from
AttractionId	Numeric (4)Not null Autonumber		
AttractionName	Character (50)Not null		
CountryId	Numeric (4)Not null		
CategoryId	Numeric (4)		
AttractionPhoto	Image Not null		
AttractionPhoto_GXI	Varchar (2048) Not null		
CityId	Numeric (4)		

Indexes

	Name	Definition	Composition
	IATTRACTION	primary key Clustered	AttractionId
	IATTRACTION2	duplicate	CategoryId
	IATTRACTION1	duplicate	CountryId CityId
New	UATTRACTIONNAME	duplicate	AttractionName

Foreign key constraints

Referenced table	Attributes

Once we've done this, the database will be reorganized by pressing F5 to create this new index. Remember that the navigation list of the report informed us that we didn't have an index to solve the query. Let's see what it says after reorganizing...

Other indexes that must be created by the developer to optimize queries

...From then on, it uses it as necessary.

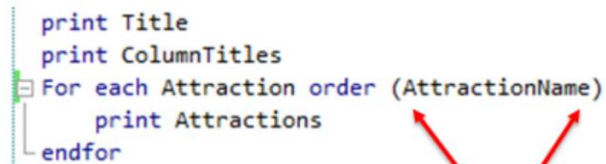
We're told that it will use the index that has just been created.

Once we create it, we can delete it at any time. By pressing F5 and reorganizing, we go back to the status it had before creating it.

The decision of whether to create the index or not will depend on the DBMS being used, of the frequency with which queries will be made to order by AttractionName, and the frequency with which the table data is updated.

Descending order

- How do we order the list of attractions by attraction name in descending alphabetical order?
- Enclose in brackets the attributes to be arranged in descending order:



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

How do we configure it to use descending order? Simply by placing the attribute or attributes between brackets.

Orders compatible with filters

Ordering that is compatible with filters

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

Name ▲	Id
Eiffel Tower	3
Great Wall	2
Louvre Museum	1
Obelisk of São Paulo	6
The Christ Redeemer	4
The Smithonian Museum	5

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

Now let's suppose that we're interested in obtaining a list of attractions whose names are shown in alphabetical order between two values received in a parameter. For instance, between "F" and "N".

To do so, we enter the Where clauses indicated above.

Several Where clauses are equivalent to only one, where conditions are combined with the "and" logical operator. That is to say, to be considered, records must meet all conditions **at once**.

If we're going to filter by AttractionName, and we have an index created by that attribute, we should always **order by AttractionName** to optimize the query.

GeneXus

Ordering that is compatible with filters

```
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 from: AttractionName >= &NameFrom
Loop while: AttractionName <= &NameTo
Join location: Server
=Attraction (AttractionId)
=Country (CountryId)

Optimized query!

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 from: AttractionName >= &NameFrom
Loop while: AttractionName <= &NameTo
Join location: Server
=Attraction (AttractionId)
=Country (CountryId)

Note that ordering by the attribute being used to filter values lower than or equal to the filter value, and higher than or equal to the filter value, causes the table not to be run through. If there is an index created by the developer, GeneXus uses that index and the query will be optimized.

If there isn't an index and, depending on the DBMS, it will be temporary created and after being used it will be deleted. However, management systems usually have optimization strategies that may not require the creation of temporary indexes. We will not address this topic any further.

Ordering that is compatible with filters

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

Non-optimized query!

Note that if we don't enter the Order clause, GeneXus will order by primary key, and the entire table will have to be run through to know if an attraction is included in the Where range or not.

Applying conditions to orders and filters

When clauses

For each command / When clause to condition orders and filters

GeneXus

```
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
```

```
when not &NameFrom.IsEmpty()
when not &NameTo.IsEmpty()
```

What result will be obtained for the above For Each command if the &NameFrom and &NameTo variables are empty? If there is an attraction with an empty name, it will be the only one returned because it will be the only one that meets both conditions. Otherwise, there won't be any attractions listed.

Is it possible to add conditions to orders and filters so that they are applied only under certain circumstances? For example, to only apply the first Where **when** the &NameFrom variable is not empty. And to apply second Where **when** the &NameTo variable is not empty. The answer is yes. We can do it by adding conditions to Where clauses with **when**, as we can see in the second For Each. Where clauses will only be applied when the condition is met. Thus, at runtime, when both variables are left empty, none of the Where clauses will be applied and all the attractions in the table will be listed. If the variable &NameFrom is empty but &NameTo is not, the first where clause will not be applied. However, the second one will be applied, so all the attractions whose name is lower than or equal to &NameTo will be listed.

In the same way, we can add a condition to an order to have it applied or not, as we showed in the third For Each command. In fact, a series of conditioned orders can be indicated, in order to choose the first one whose condition is met.

Read more about orders and filters in the GeneXus wiki
(http://wiki.genexus.com/commwiki/servlet/wiki?6075,Order+clause)

When none clause

When no records are recovered in the for each

&NameFrom 'A' &NameTo 'B'

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

AttractionId	AttractionName	CountryId	CategoryId	Attra...	Attra...	CityId
1	Louvre Museum ...	2		na...	gxdbfil...	1
2	Great Wall ...	3	3	Bina...	gxdbfil...	1
3	Eiffel Tower ...	2	2	na...	gxdbfil...	1
4	The Christ Rede...	1	2	na...	gxdbfil...	1
5	The Smithonian ...	4		Bina...	gxdbfil...	1
6	Obelisk of São P...	1		<Bina...	gxdbfil...	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ø

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

warningMessage

There is no attraction in the range

If a For Each command is included, it is considered an independent command.

What happens when none of the records in the base table meet the conditions?

Suppose that in this case we want to print a message in the output to warn about this... to do so we program the **when none** clause that closes the For Each command.

All the commands written between when none and endfor will be executed in sequence, **only in cases when no records of the For each base table that meet the conditions can be found.**

In this case, we have decided to print a message, but we may also type a series of commands, such as another For Each command, for example.

Since what will be executed after the **when none** clause will imply that the search was unsuccessful, if we type a For Each command there, the when none clause will not be nested. It will be like a standalone For Each command.

GeneXus

Summary

As we already know, GeneXus determines the **base table** of the For Each command by the transaction name that is mentioned next to it; the rest of the attributes mentioned, both in the For Each command body (main code) and in the Order and Where clauses, will have to belong to the extended table of that base table (that's why part of the syntax is underlined above).

The attributes mentioned in the when none block are not taken into account.

We gray out everything that we've seen before. Here, the **when** and **when none** clauses are added.

Later on we will see that more clauses are added to this essential command to access the database.

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications