

## Loading compound data types

GeneXus object: Data Provider

**GeneXus**<sup>™</sup>

In previous videos we have seen the concept of structured data type, the possibility of defining them as simple or collection data types, and some examples of manual loading, although we have mentioned that there is another, higher level way to perform the loading.

New requirement: Ranking of countries

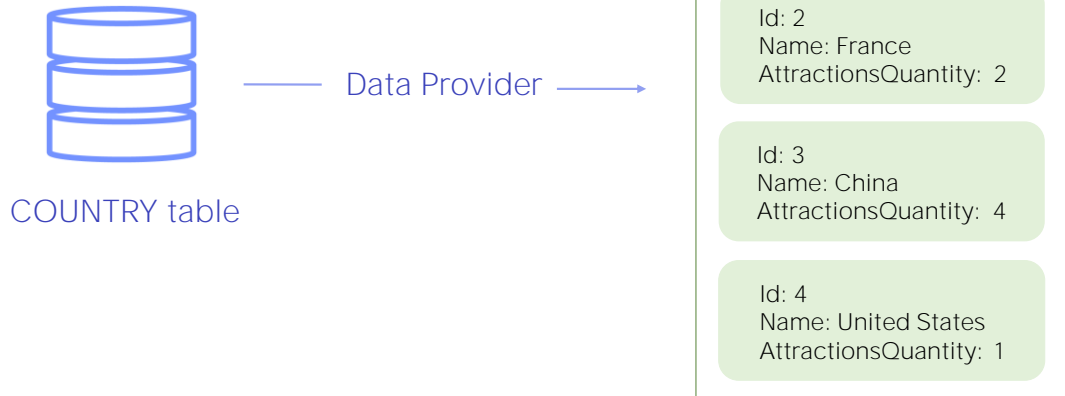
## Ranking

Country name	Attractions quantity
France	3
United States	2
Egypt	1
Brazil	1
China	1
Uruguay	0

Remember that we still have to solve a request made by the Travel Agency to implement a ranking of countries according to the number of tourist attractions they offer.

In other words, we must show all the countries, ordered from highest to lowest, by that amount.

## Collection structure



To load the data from the collection we will use a GeneXus object of Data Provider type.

This object allows us to load a data structure, for example from information obtained from the database, and returns this structure already loaded.

## New requirement: Ranking of countries

The screenshot illustrates the configuration of a Data Provider in GeneXus. The KB Explorer on the left shows a list of data types, with 'SDTCountries' selected. An arrow labeled '1' points to this selection. An arrow labeled '2' points to the 'Source' field of the 'DPRankingCountriesWithAttractionsQty' Data Provider, where 'SDTCountries' has been dragged. An arrow labeled '3' points to the 'Properties' window for this Data Provider. In the Properties window, the 'Output' property is set to 'SDTCountries' and the 'Collection' property is set to 'False', both of which are highlighted with red boxes.

Drag the SDT to the source of the Data Provider

We create a Data Provider object and name it RankingCountriesWithAttractionsQty.

GeneXus places us in the Source section of the Data Provider.

This is where we will declare how we want the data to be loaded into the collection to be returned. Note how easy it is to declare the loading: What we are going to do, from the KB Explorer window, is simply drag the SDTCountries structured data type over the source of the Data Provider.

When doing so, GeneXus automatically writes several lines of text.

If we open the Data Provider properties, we can see that GeneXus assigned the name of the SDTCountries collection to the Output property. This means that the Data Provider will return a collection of SDTCountries structured data type, loaded with data.

Since SDTCountries is already a collection, it isn't necessary to configure the Collection property of the Data Provider to True.

It should be mentioned that if this Collection property is set to True, the Data Provider will return a collection of the SDT indicated in the Output property, regardless of how complex its structure may be.

## New requirement: Ranking of countries

The first screenshot shows the source code for the `SDTCountries` structured data type. The code is as follows:

```
1 SDTCountries
2 {
3   SDTCountriesItem
4   {
5     Id = /*Id value*/
6     Name = /*Name value*/
7     CountryAttractionsQuantity = /*Country Attractions Quantity value*/
8   }
9 }
```

An arrow points from the text "Structured data type name" to the `SDTCountries` label on line 1.

The second screenshot shows a zoomed-in view of the `SDTCountriesItem` substructure. The code is as follows:

```
1 SDTCountries
2 {
3   SDTCountriesItem
4   {
5     Id = /*Id value*/
6     Name = /*Name value*/
7     CountryAttractionsQuantity = /*Country Attractions Quantity value*/
8   }
9 }
```

An arrow points from the text "Substructure of the collection's item." to the `SDTCountriesItem` label on line 3.

Let's see what GeneXus wrote in the source.

We recognize the name of the `SDTCountries` structured data type, which is a collection. And then between brackets is the substructure of the collection item.

## New requirement: Ranking of countries

The image shows two windows from the GeneXus IDE. The top window, titled 'SDTCountries', displays the 'Structure' view. It shows a table with columns 'Name', 'Type', and 'Is Collection'. The 'SDTCountries' entity is highlighted in blue and has a checked 'Is Collection' box. Below it, the 'SDTCountriesItem' sub-entity is shown with three properties: 'Id' (Type: Id), 'Name' (Type: Name), and 'CountryAttractionsQuantity' (Type: Numeric(4.0)).

Name	Type	Is Collection
SDTCountries		<input checked="" type="checkbox"/>
SDTCountriesItem		
Id	Id	<input type="checkbox"/>
Name	Name	<input type="checkbox"/>
CountryAttractionsQuantity	Numeric(4.0)	<input type="checkbox"/>

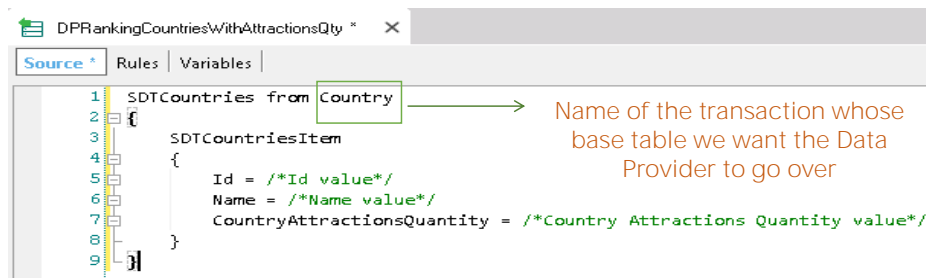
The bottom window, titled 'DPRankingCountriesWithAttractionsQty', shows the 'Source' view. It contains the following code:

```
1 SDTCountries
2 {
3   SDTCountriesItem
4   {
5     Id = /*Id value*/
6     Name = /*Name value*/
7     CountryAttractionsQuantity = /*Country Attractions Quantity value*/
8   }
9 }
```

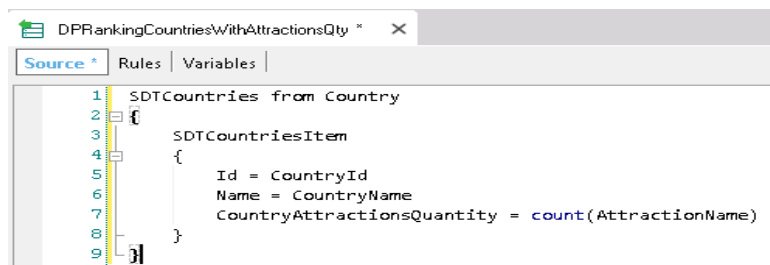
Let's compare this with the structure of the SDT:

We see that GeneXus represented the structure of SDTCountries in text form. And left the members ID, Name, and AttractionsQuantity of the substructure of SDTCountries ready to load their value.

## New requirement: Ranking of countries



Indicate the attributes or calculations with which the elements of the collection are loaded:



We want to load the collection from the content of the COUNTRY table.

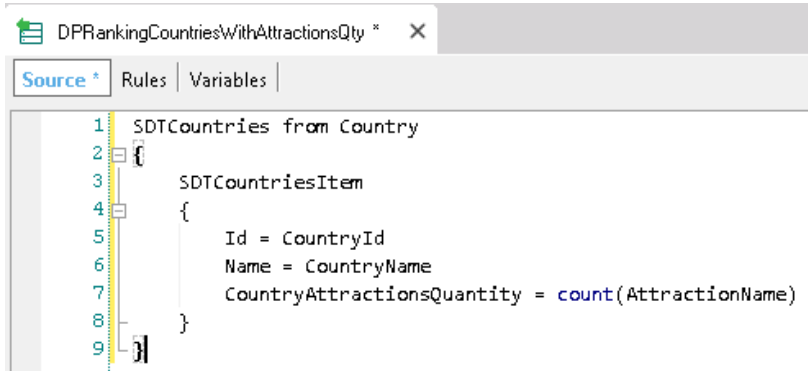
Then, we must instruct the Data Provider to run through the table. To this end we use the From clause, and next to it we will indicate the name of the transaction whose base table we want to run, as we have done to indicate the base transaction of the For Each command. So in this case we write: From Country

If the transaction had more than one level, then in order to specify a certain level, associated with a certain base table that we want to navigate, we would have to write the name of the transaction, period, name of the level.

Then, we will indicate that we want to load the ID element with the value of the CountryId attribute, load the Name item with the value of the CountryName attribute, and load the AttractionsQuantity item with the number of tourist attractions in each country, so to this member we assign the result of the inline formula Count(AttractionName).

Let's review a concept we have already studied: this inline formula will navigate the ATTRACTION table by the attribute we have indicated in brackets. In addition, the table navigated by the Data Provider – COUNTRY–, and the table navigated by the formula –ATTRACTION– have a common attribute that is CountryId; this formula will count the attractions of the country navigated by the Data Provider each time.

## New requirement: Ranking of countries



```
1 SDTCountries from Country
2 {
3   SDTCountriesItem
4   {
5     Id = CountryId
6     Name = CountryName
7     CountryAttractionsQuantity = count(AttractionName)
8   }
9 }
```

The base table of the Data Provider Is COUNTRY

So what we have done is simply: to declare a table to be navigated by the Data Provider, and for each record accessed, we have indicated the values we want to assign to a new item in the country collection.

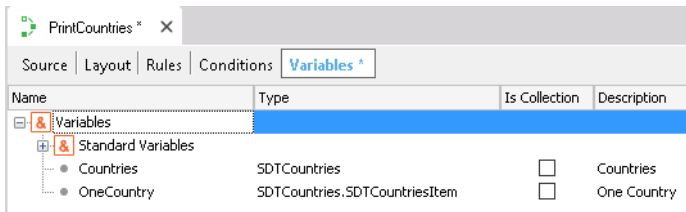
Since the Data Provider runs through the COUNTRY table, we usually say that the base table of the Data Provider is COUNTRY.

The final result will be that the data of all the countries in the database, each with its number of attractions, will have been stored in the collection in memory.

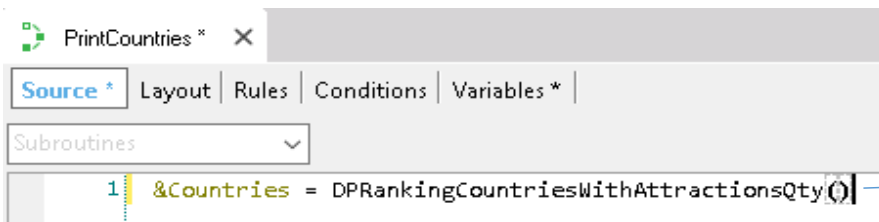


## New requirement: Ranking of countries

- 1) Create a procedure object
- 2) Define the variables:



- 3) In the source of the procedure, invoke the Data Provider:



The variable &Countries receives what the Data Provider returns.

Remember that we already have in our knowledge base a procedure named CountriesRanking, so we are going to modify it.

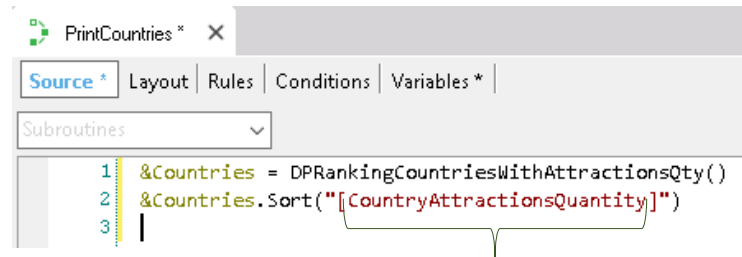
First, in the variables section we will define a variable &Countries based on the SDTCountries data type.

We then go to the procedure Source, and from here, to this Countries variable, we assign the result returned by the Data Provider that we have just created.

In this way we are invoking the Data Provider and it will return a collection of countries, which will be loaded in the variable &Countries.

## New requirement: Ranking of countries

To implement a ranking, we must order the collection from highest to lowest quantity of attractions...We use the **Sort** method:



```
1 &Countries = DPRankingCountriesWithAttractionsQty()
2 &Countries.Sort("[CountryAttractionsQuantity]")
3 |
```

SDT field to sort the collection

The brackets within parentheses indicates the reverse order, that is, from high to low.

Remember that the requirement is to view a ranking of all countries ordered from highest to lowest according to the number of attractions they have registered.

Therefore, we still need to order the collection we got loaded. That is to say, to order the items of the collection of countries, before it is shown, from highest to lowest according to the number of attractions they have registered.

To solve this we have the Sort method. The syntax is as follows:

```
&Countries.Sort("CountryAttractionsQuantity")
```

But in this way the collection of countries will be ordered from lowest to highest by the number of attractions and we need it to be ordered from highest to lowest, because we want to implement a ranking.

To indicate the reverse order, inside the quotes we will add straight parentheses.

New requirement: Ranking of countries

To go over a collection stored in memory and print each element in the printblock, we have the *For ... in* structure



```
PrintCountries * X
Source * | Layout | Rules | Conditions | Variables *
Subroutines
1 &Countries = DPRankingCountriesWithAttractionsQty()
2 &Countries.Sort("[CountryAttractionsQuantity]")
3
4 Print Title
5
6 For &OneCountry in &Countries
7     print Country
8 Endfor
9
```

Once the collection is sorted, we must run through it to show it in the printblock, so we will need to define a variable based on the element of the collection.

So we define the `&oneCountry` variable based on the data type corresponding to the element in the collection.

Then we declare the structure: `For &oneCountry in &Countries`

## New requirement: Ranking of countries

Insert the variables in the Country printblock...

The screenshot shows the GeneXus IDE interface. On the left, a dialog box titled "Select Members" is open, showing a tree view of the "SDTCountries" object. The "Country..." member is selected, and a green arrow points from it to the "Country" printblock in the layout editor on the right. The layout editor shows a "Country" printblock with the following structure:

Ranking	
Country name	Attractions quantity
&OneCountry.Name	&OneCountry.CountryAttractionsQ

Now let's go to the layout and insert the &oneCountry variable.

New requirement: Ranking of countries

## Ranking

Country name	Attractions quantity
France	3
United States	2
Egypt	1
Brazil	1
China	1
Uruguay	0

To see the ranking running we select the Run option.

And we see the PDF list with all the countries that were registered in the database, each one with its corresponding number of attractions and in the requested order.

Where clause

```
SDTCountries from Country
Where CountryName <> "France"
{
    SDTCountriesItem
    {
        Id = CountryId
        Name = CountryName
        AttractionsQuantity = count(AttractionName)
    }
}
```

Optionally, Data Providers accept the Where clause to filter, as well as the For Each command... for example, if we don't want the list to include France, how would we go about it?

We can add the Where CountryName clause ... different ... from France.

Another option to implement the requirement...

Name	Type	Description	Is Collection
SDTCountry		SDTCountry	<input type="checkbox"/>
• Id	Numeric(4.0)	Id	<input type="checkbox"/>
• Name	Character(20)	Name	<input type="checkbox"/>
• AttractionsQuantity	Numeric(4.0)	Attractions Quantity	<input type="checkbox"/>

SDTCountry from Country

```
{
  Id = CountryId
  Name = CountryName
  AttractionsQuantity = Count(AttractionName)
}
```

Output	
Infer Structure	No
Output	<b>SDTCountry</b>
Collection	<b>True</b>
Collection Name	<b>RankingCountries</b>

Name	Type	Is Collection	Description
& Variables			
& Standard Variables			
& CountriesCollection	SDTCountry	<input checked="" type="checkbox"/>	Countries Collection

Another way to implement this same requirement is from the SDT set as simple and not as a collection.

In this case, the collection must be built by the Data Provider, and for that its Collection property must be set to True.

In this way, we're telling the Data Provider to return a collection of elements of SDTCountry type. Also, note that the Collection Name property is displayed and a name has been automatically assigned to the collection.

When invoking the Data Provider from the procedure source, we must define the &CountriesCollection variable as a collection of the structured data type SDTCountry.

Note that the syntax used to invoke the Data Provider does not change, only that the variable that receives the result is based on a simple SDT. Therefore, in order to receive the collection returned by the Data Provider we must declare it as a collection by selecting the IsCollection check box.

## Summarizing...

Data Provider

Name	Type	Description	Is Collection
SDTCountries		SDTCountries	<input checked="" type="checkbox"/>
SDTCountriesItem			
• Id	Numeric(4,0)	Id	<input type="checkbox"/>
• Name	Character(20)	Name	<input type="checkbox"/>
• AttractionsQuantity	Numeric(4,0)	Attractions Quantity	<input type="checkbox"/>

```
SDTCountries from Country
{
  SDTCountriesItem
  {
    Id = CountryId
    Name = CountryName
    AttractionsQuantity = count(AttractionName)
  }
}
```

Output	
Infer Structure	No
Output	<b>SDTCountries</b>
Collection	False

Name	Type	Description	Is Collection
SDTCountry		SDTCountry	<input type="checkbox"/>
• Id	Numeric(4,0)	Id	<input type="checkbox"/>
• Name	Character(20)	Name	<input type="checkbox"/>
• AttractionsQuantity	Numeric(4,0)	Attractions Quantity	<input type="checkbox"/>

```
SDTCountry from Country
{
  Id = CountryId
  Name = CountryName
  AttractionsQuantity = Count(AttractionName)
}
```

Output	
Infer Structure	No
Output	<b>SDTCountry</b>
Collection	<b>True</b>
Collection Name	<b>RankingCountries</b>

In sum, we have two ways for a Data Provider to return a collection of elements:

One of them is to define a structured data type of collection type; after dragging it to the Data Provider source, it is automatically configured to return a collection of that type.

The other option is to define a structured data type that is not a collection, and then setting the Data Provider properties to have it build the collection.

In this way we have seen the power of Data Providers to load information into a data structure in memory. We saw how easy it was to declare the data we wanted to load, with GeneXus solving everything necessary to carry it out.



*GeneXus*<sup>TM</sup>

[training.genexus.com](http://training.genexus.com)  
[wiki.genexus.com](http://wiki.genexus.com)