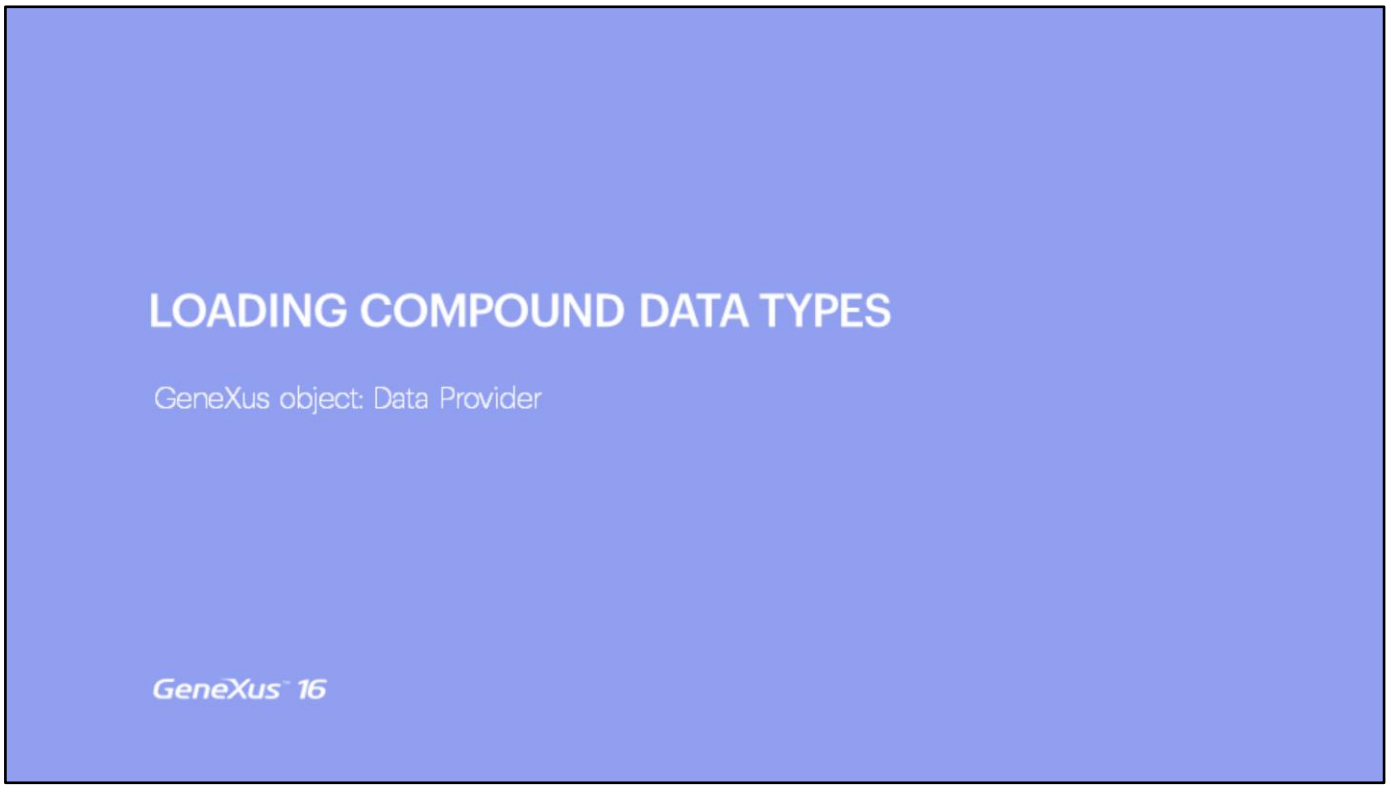
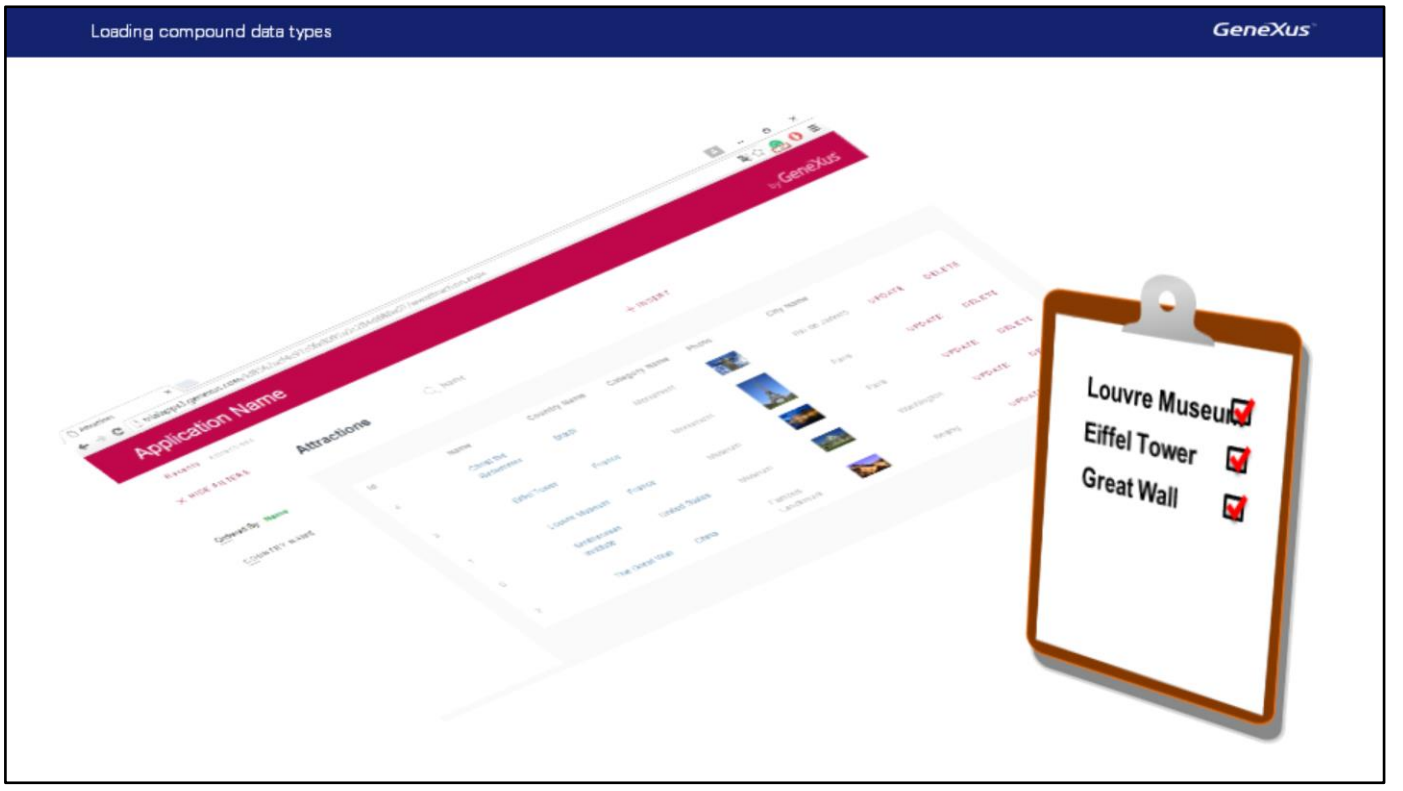
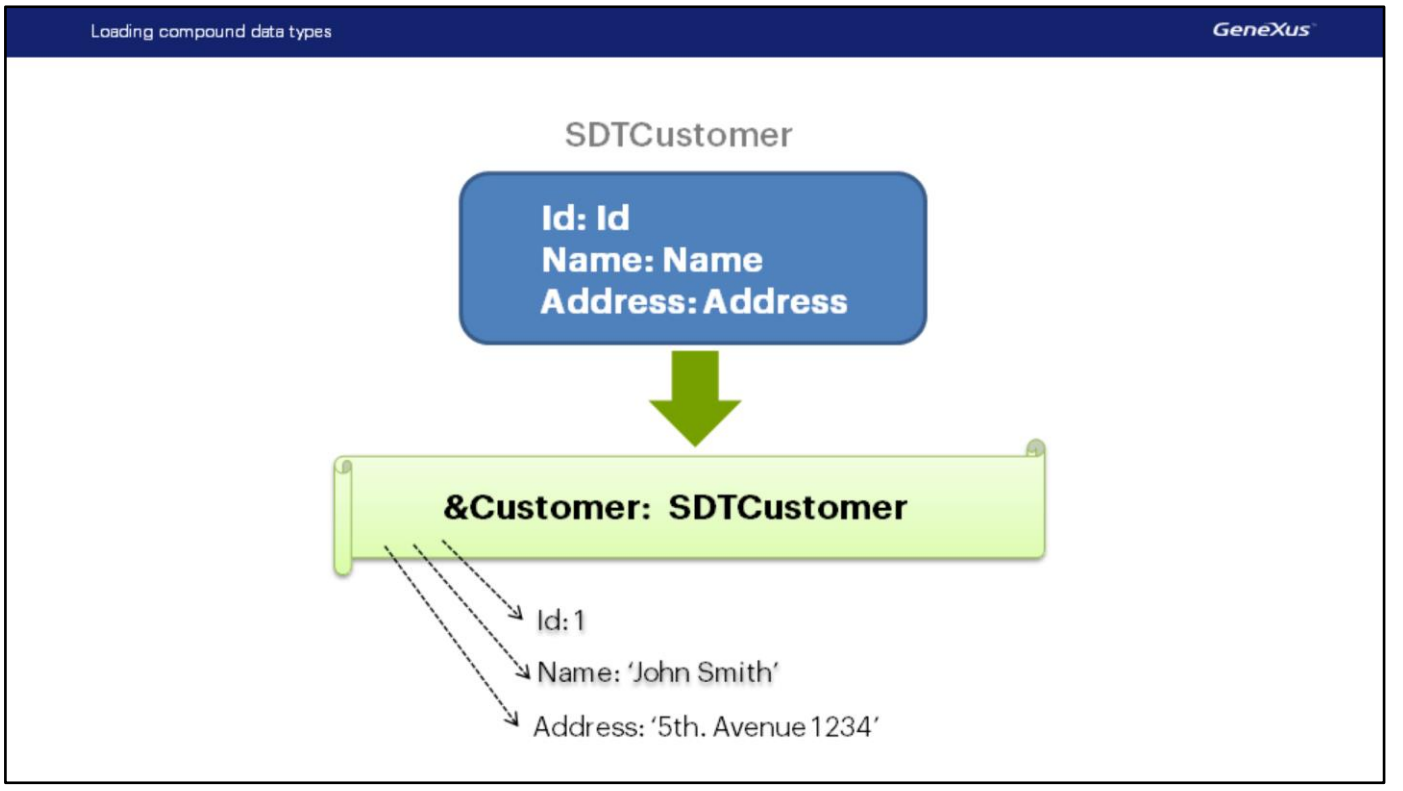


On more than one occasion, we need to store in memory **a list of elements** which hold the same type of information with different values stored.



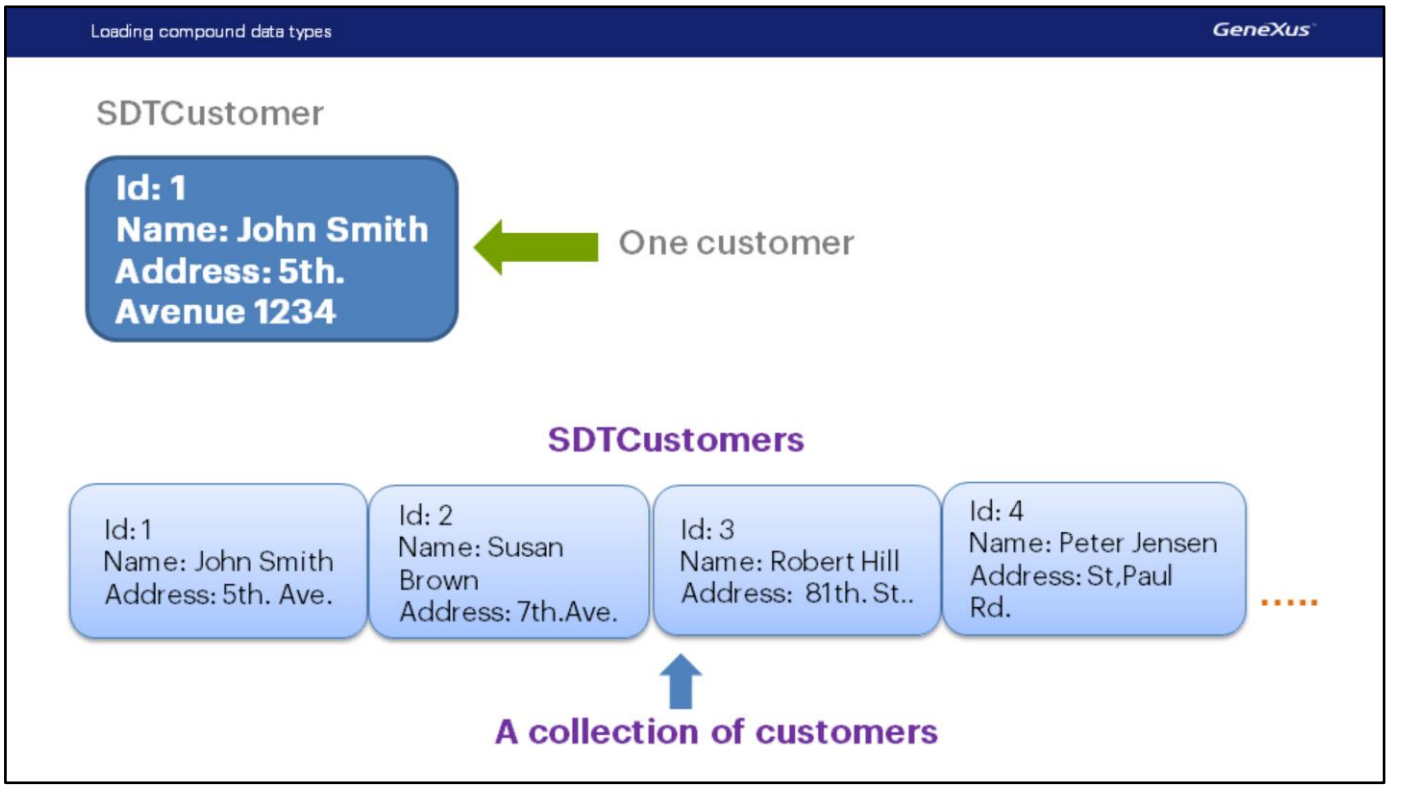
For instance, the travel agency might need to carry out operations with a group of customers who share a common feature, or we might be asked to process information about specific data on a group of tourist attractions and this could mean that we must load the lists in memory temporarily.



To solve this kind of requirement, we must create a structure in memory **capable of storing a collection of elements.**

We have seen that **structured data types,** as SDTCustomer in this example, enable us to define structures that store several data corresponding to **an element.**

In this graphical representation we are then storing in memory the identifier, the name and the address of **one** customer.



To store several elements with customer data, we saw that we need to define a structured data type and indicate that it is a collection.

New requirement: Ranking of countries

Ranking of countries based on the number of tourist attractions they offer:

Let's suppose that we have to implement a new requirement for the travel agency.

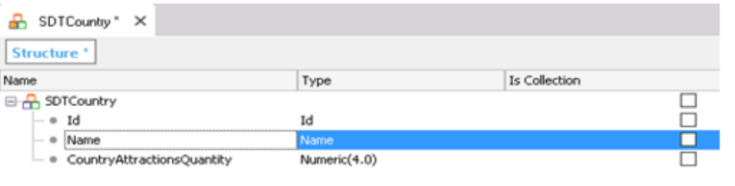
The travel agency wishes to provide its customers with a ranking of countries based on the number of tourist attractions offered.

This means that we must show all countries, sorted from larger to smaller in what refers to number of attractions offered.

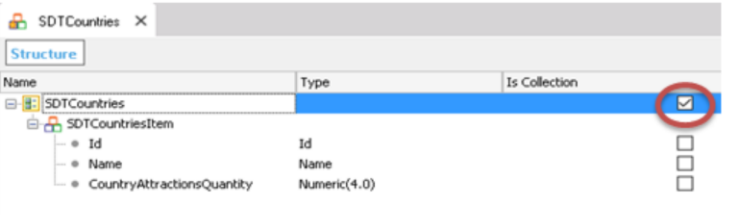
To solve this requirement we must first load all the countries stored in the database in a structure, each with its corresponding number of tourist attractions. And then we will have to sort them according to that number from the greater to the smaller amount, to then show them either on a web panel or a listing.

New requirement: Ranking of countries

To save one country in memory:



To save a collection of countries in memory:



Let's start by creating a new object of the structured data type , which we will call SDTCountries.

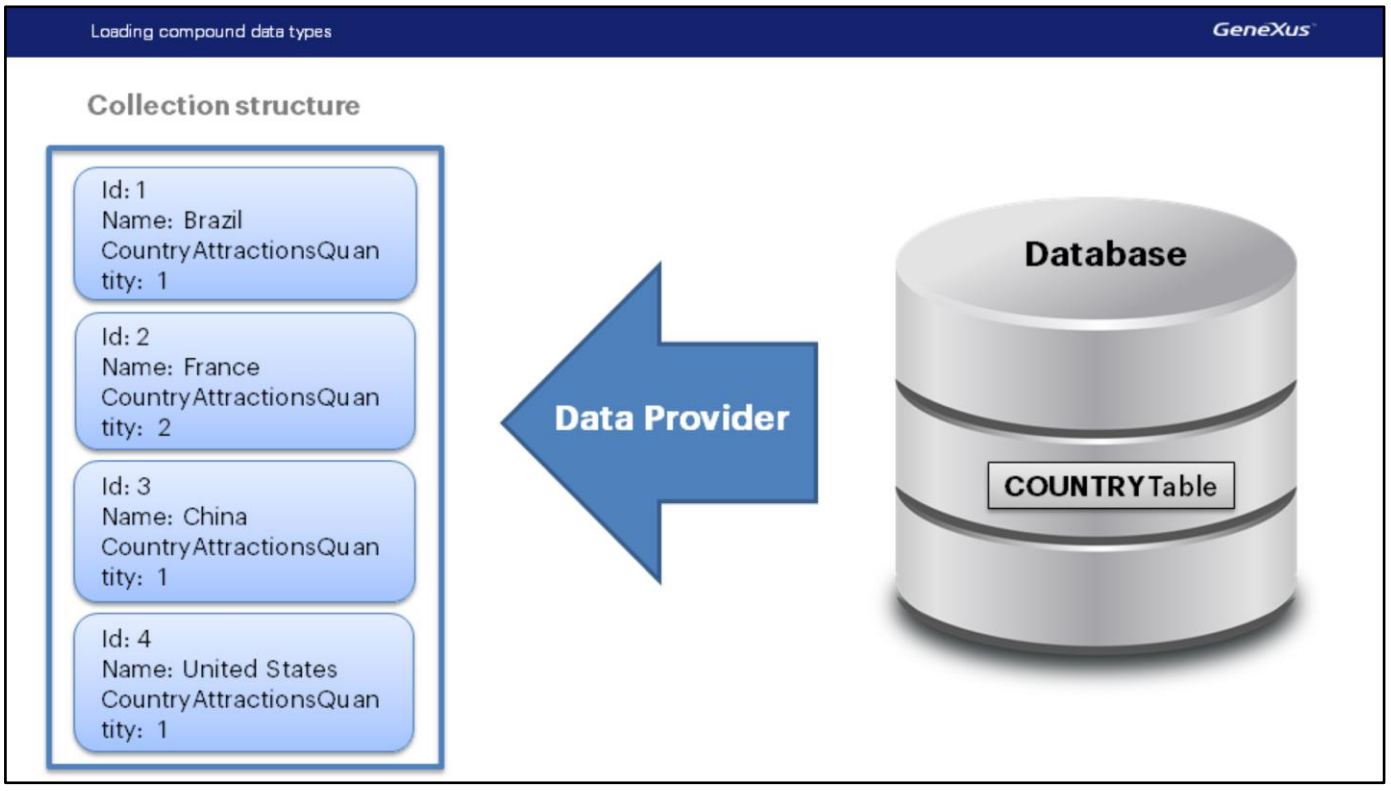
For each country, we need its identifier, name and number of registered tourist attractions.

With the first definition in the slide, we indicate that we will be saving in memory the information relative to one single country.

But we need to design a ranking of countries, so we must store several countries. Therefore we check the IS COLLECTION box to have a structure that will allow us to store data of several countries (as shown in the second definition in the slide).

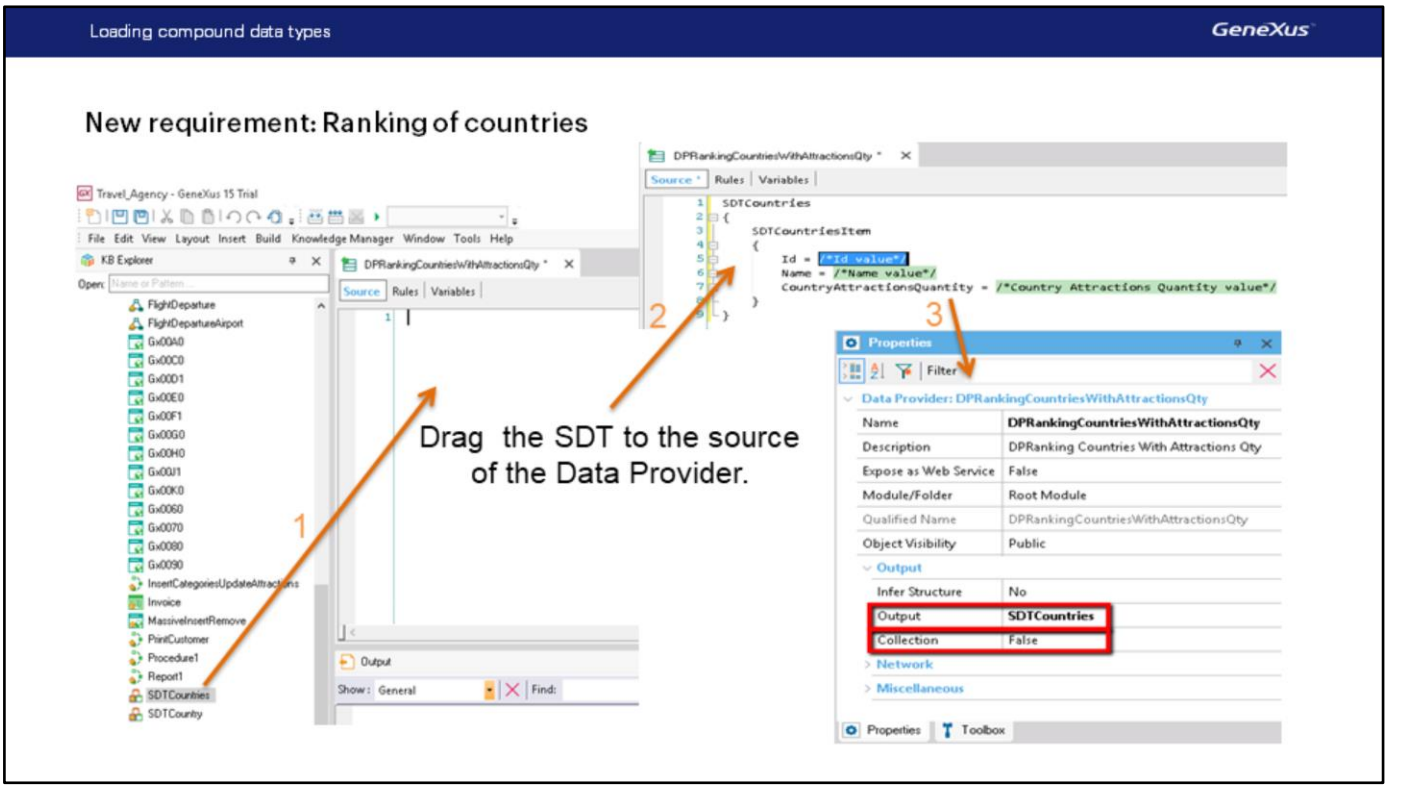
This structure includes the same members we defined at the start, though grouped into a substructure called SDTCountriesItem . This substructure was automatically created when we indicated the collection checkmark.

Each item will store one country's data, and the collection will store the group of countries' data.



To load the data of the collection we will use a GeneXus object called **Data Provider.**

The Data Provider object enables us to load a data structure, for example, based on data from the database, and it returns the loaded structure.



We create a Data Provider object in GeneXus and name it: DPRankingCountriesWithAttractionsQty.

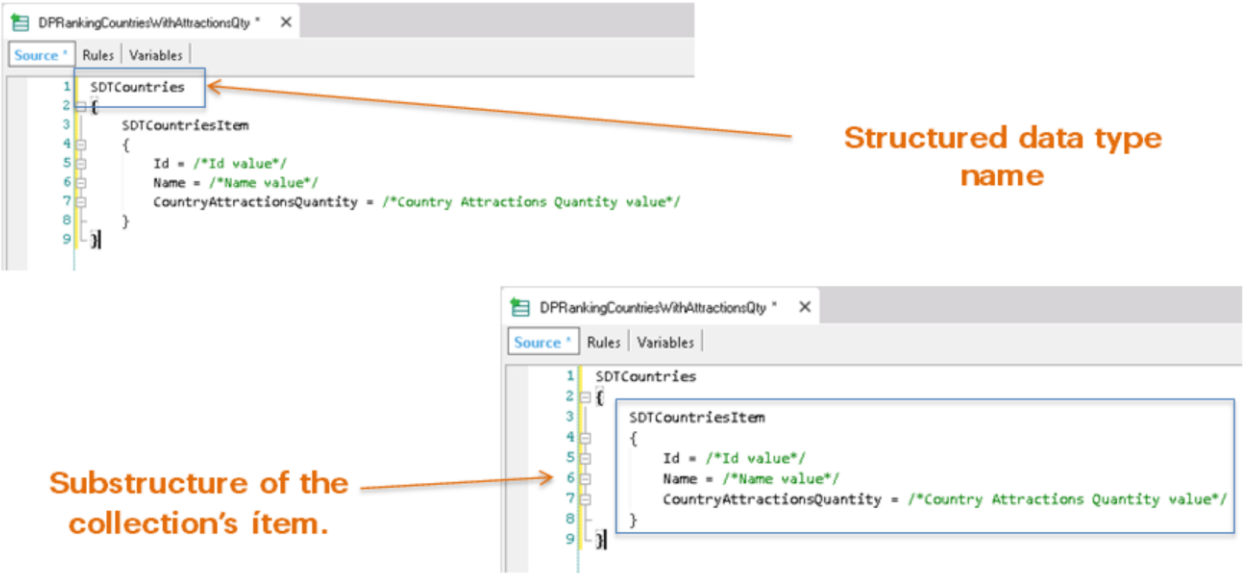
We can see that GeneXus positions us in the Source section of the Data Provider. Here is where we will declare how we want the data to be loaded in the collection we want returned. See how easy it is to declare the load: Go to the window of the KB Explorer and find the structured data type SDTCountries, then drag it to the source of the Data Provider.

We can see that GeneXus automatically wrote several lines of text.

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

Since the SDTCountries is already a collection, we will not need to configure the Collection property of the Data Provider with True value. We would have to do this if we wanted the Data Provider to return a collection based on a simple structured data type.

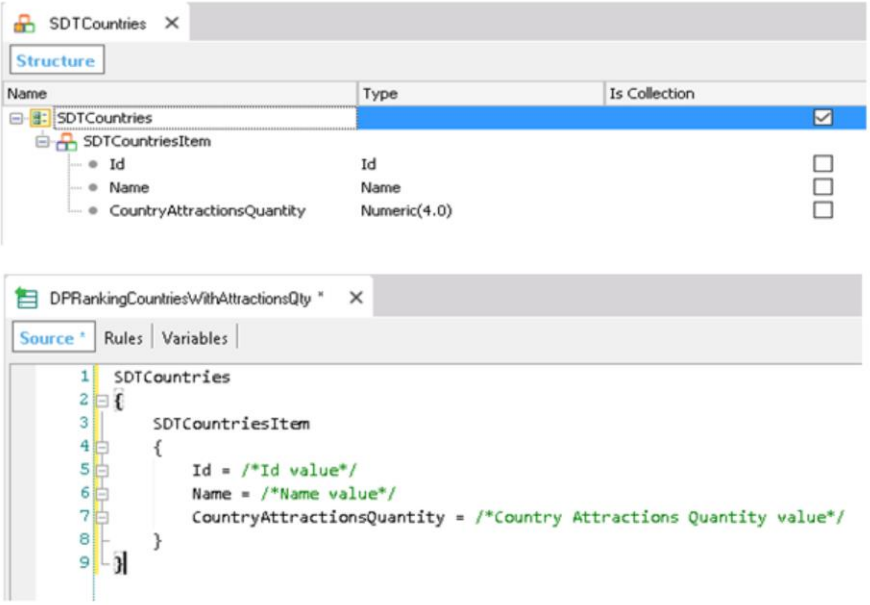
New requirement: Ranking of countries



Now let's analyze what GeneXus wrote in the source.

We recognize the name of the structured data type SDTCountries that is a collection. And inside, the substructure of the collection's item is between braces.

New requirement: Ranking of countries



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

We can see that GeneXus represented, as text, the structure of the SDTCountries, and provided members Id, Name and CountryAttractionsQuantity from the SDTCountriesItem substructure for loading their values.

New requirement: Ranking of countries

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

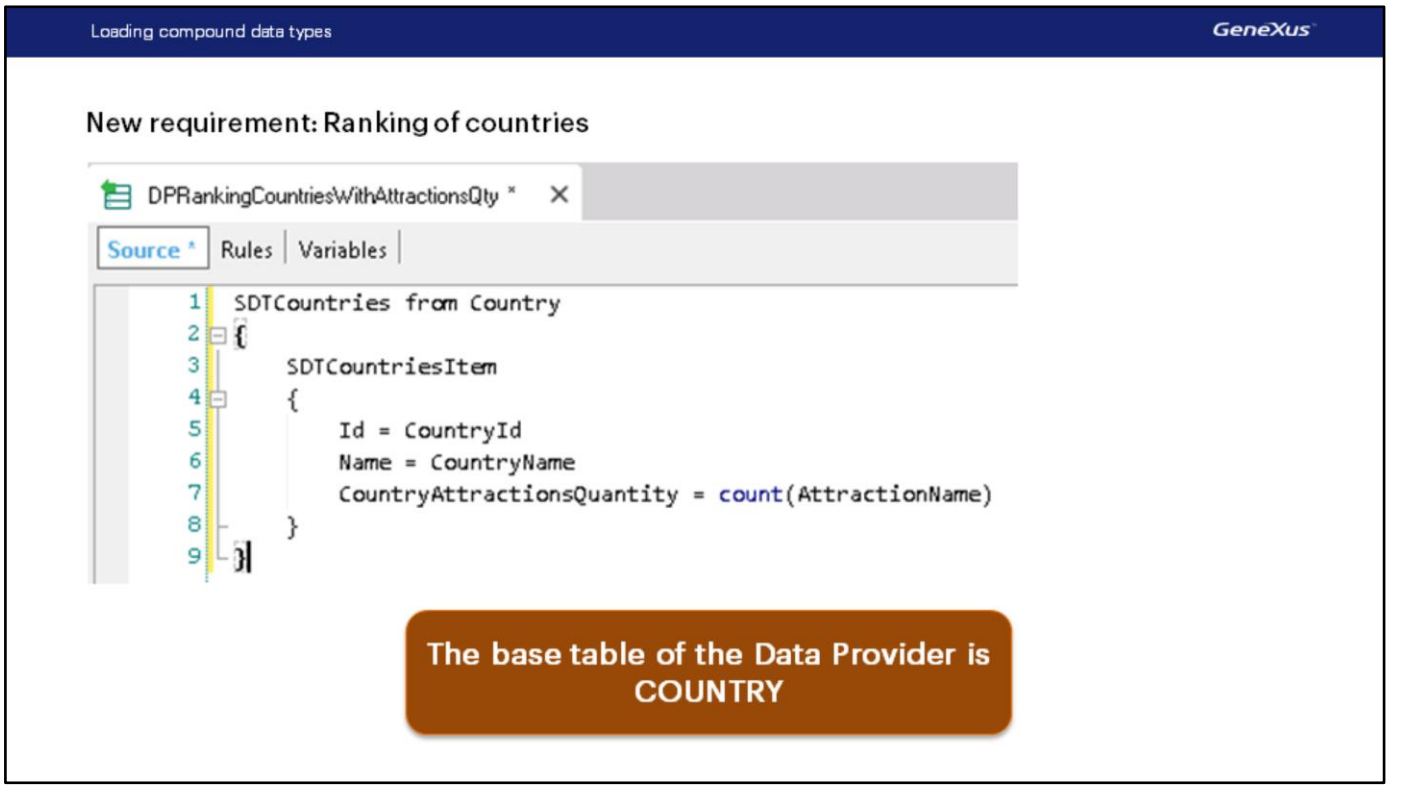
Since we will be loading this collection based on the contents of the COUNTRY table, we must indicate to the Data Provider that it must go over that table. To do this we use the From clause and next to it we include the name of the transaction whose base table we want to go over.

In our case, from Country.

When the transaction has more than one level, in order to specify a level associated to a particular base table we want to navigate, we must write the **name of the transaction, dot, and the name of the level.**

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

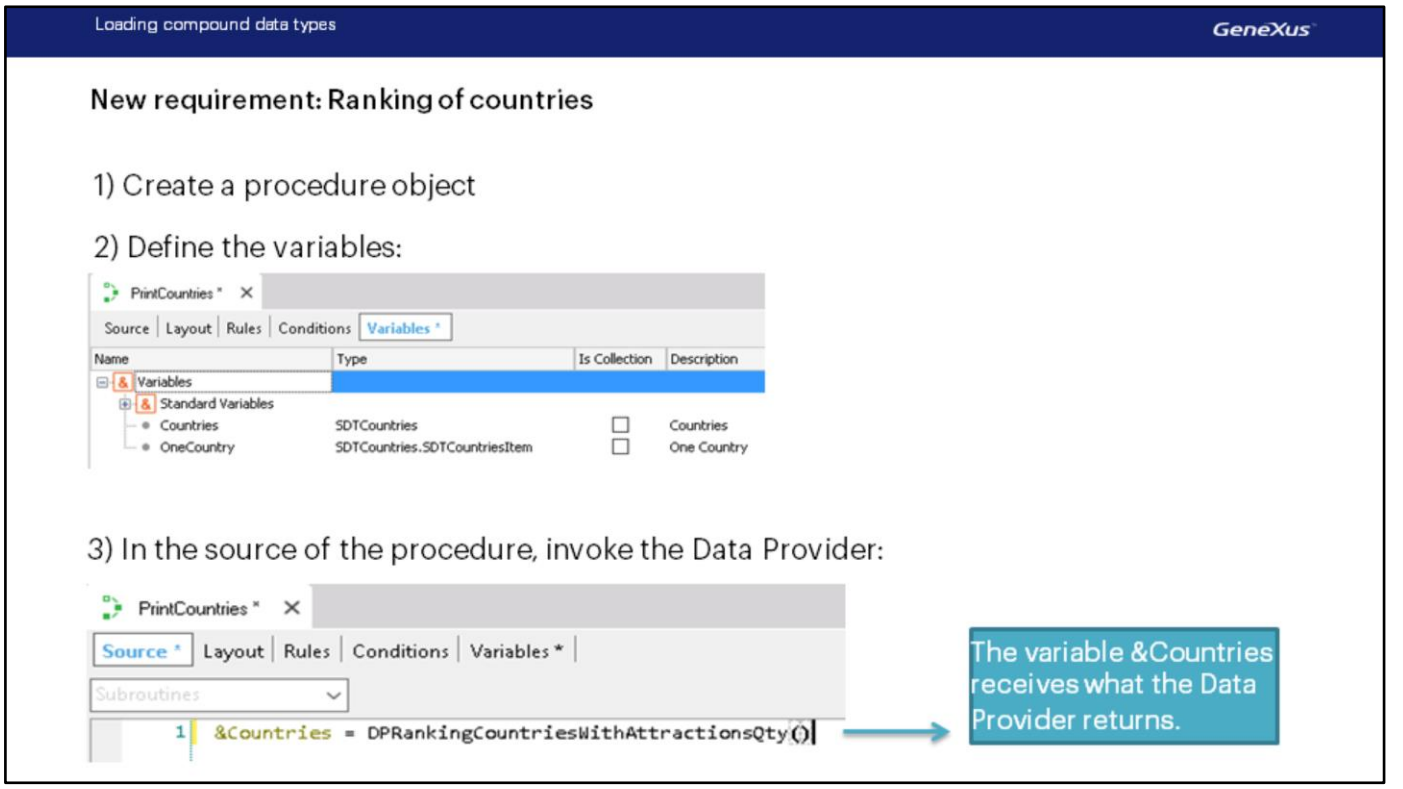
Let's review a concept we have already seen: this inline formula defined will navigate the ATTRACTION table by the attribute indicated inside the parentheses. Also, since there is an attribute in common between the tables navigated by the Data Provider and the formula , that is CountryId , then the formula will count the attractions **of the country** navigated by the Data Provider every time.



What we did was simply to **declare one table to be navigated by the Data Provider, and for each record accessed, we indicated the values we want to assign to a new item in the collection of countries.**

Since the Data Provider goes over the COUNTRY table, we usually say that **the base table of the Data Provider is COUNTRY:**

The final outcome will be that the collection in memory will store the data of all the countries in the database, each with its own number of attractions.



Now we create a Procedure object to view the contents of the collection of countries. We call this procedure "PrintRanking"

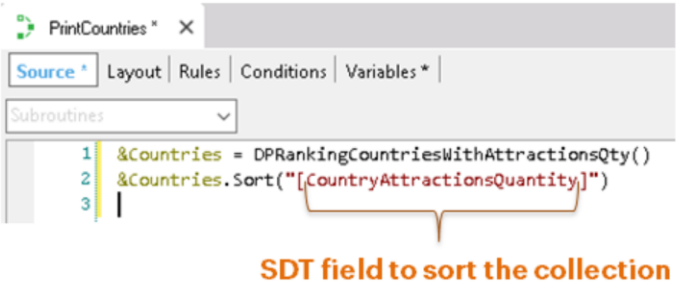
We go to the Variables section of this procedure and define a &Countries variable of the SDTCountries data type.

Then we go to the source, and we load this variable of the collection type, assigning to it the result returned by the Data Provider we just created.

With the instruction we wrote we are **invoking** the Data Provider, which will return a collection of countries that will be loaded in the &Countries variable.

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:



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

Now let's recall the exact requirement of the travel agency: to view a ranking of all countries **sorted from highest to lowest number of recorded attractions.**

Therefore what is left to do is sort the collection we loaded. That is: sort the items of the country collection prior to showing it, in order from highest to lowest number of attractions recorded.

To solve this we have the Sort method, and the syntax is as follows:

&Countries.Sort("CountryAttractionsQuantity")

But this will sort the country collection from lesser to greater number of attractions and we need the opposite sorting because we need to implement a ranking.

So, to indicate the reverse order, inside the quotation marks we add brackets.

New requirement: Ranking of countries

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

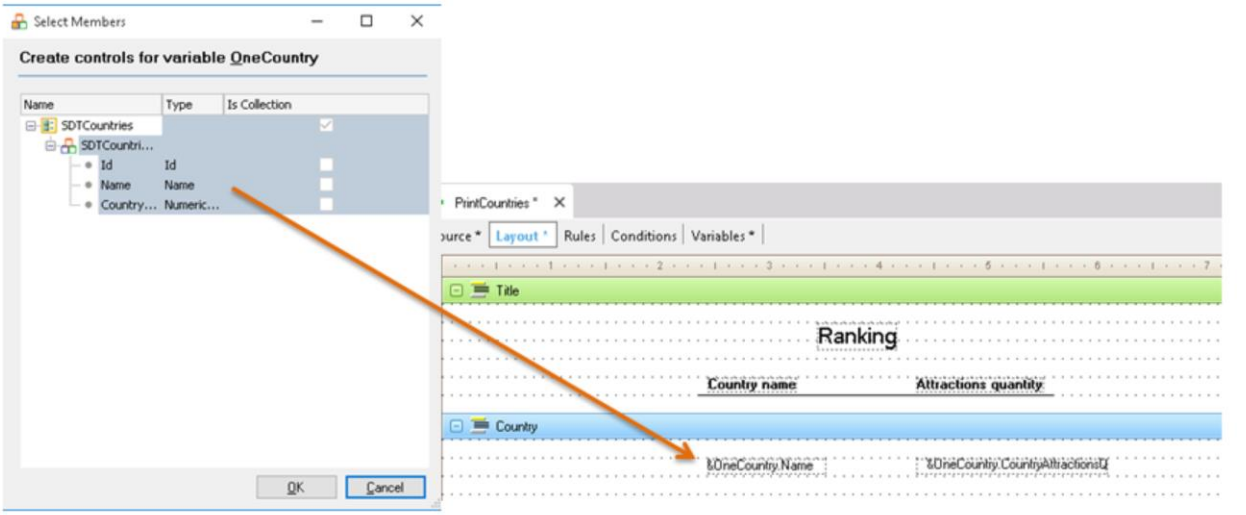
The only thing remaining now is to go over the collection again with the Data Provider and print each item in it, with the stored data.

To go over a collection stored in memory we have the **For element in Collection** command.

We define a &oneCountry variable to load in it each element we iterate from the collection.

New requirement: Ranking of countries

Insert the variables in the Country printblock...



In the Layout" section, we call this printblock "Country", and we select: Insert / Variable and choose &oneCountry

New requirement: Ranking of countries

Ranking

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

So now all we have to do is define the necessary properties to print the list in PDF format.

We go to the procedure's properties and in "Main program" choose the value True.

Then, in "Main object properties", we select "Call protocol" and choose "HTTP".

Finally we have to insert the OutputFile rule in the rule section: we select Insert / Rule and choose the rule OutputFile.

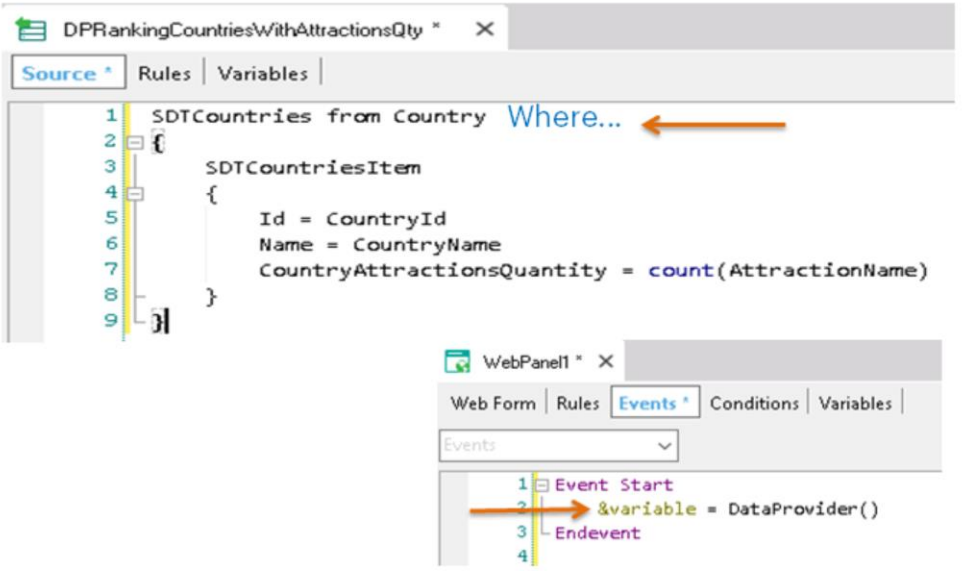
We finish by naming the file as "Ranking.PDF" and the format we will use as "PDF".

The development of the requirement is now complete. We select run over the procedure to view the ranking in runtime!

And we view the pdf listing with all the countries stored in the database, each with its own number of attractions, and the order of countries as requested!

So, we have seen the power of Data Providers to load data in a data structure in memory, specifically in the case of collection type. And we saw how simple it is to declare what we wanted to load, and then GeneXus solved everything needed to carry it out.

New requirement: Ranking of countries



Data Providers optionally admit the Where clause to filter, like the For each command... Further ahead we will see other examples where Data Providers are used. They may also be invoked, in other object sections, such as in web panel events.

GeneXus™

The power of doing.

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