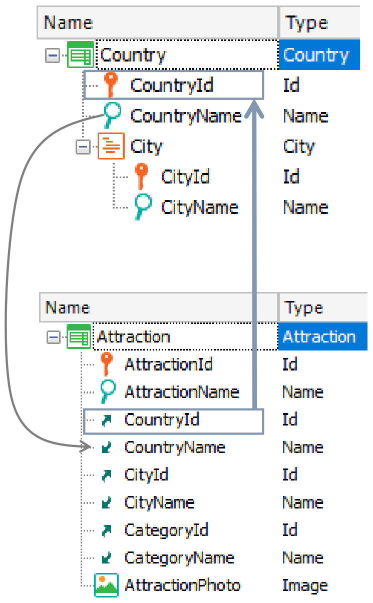
Different attribute names for the same concept

Subtype group

- GeneXus determines relations for attributes with the **same name**.

Country Transaction

Attraction Transaction



Name	Type
Country	Country
CountryId	Id
CountryName	Name
City	City
CityId	Id
CityName	Name

Name	Type
Attraction	Attraction
AttractionId	Id
AttractionName	Name
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name
CategoryId	Id
CategoryName	Name
AttractionPhoto	Image

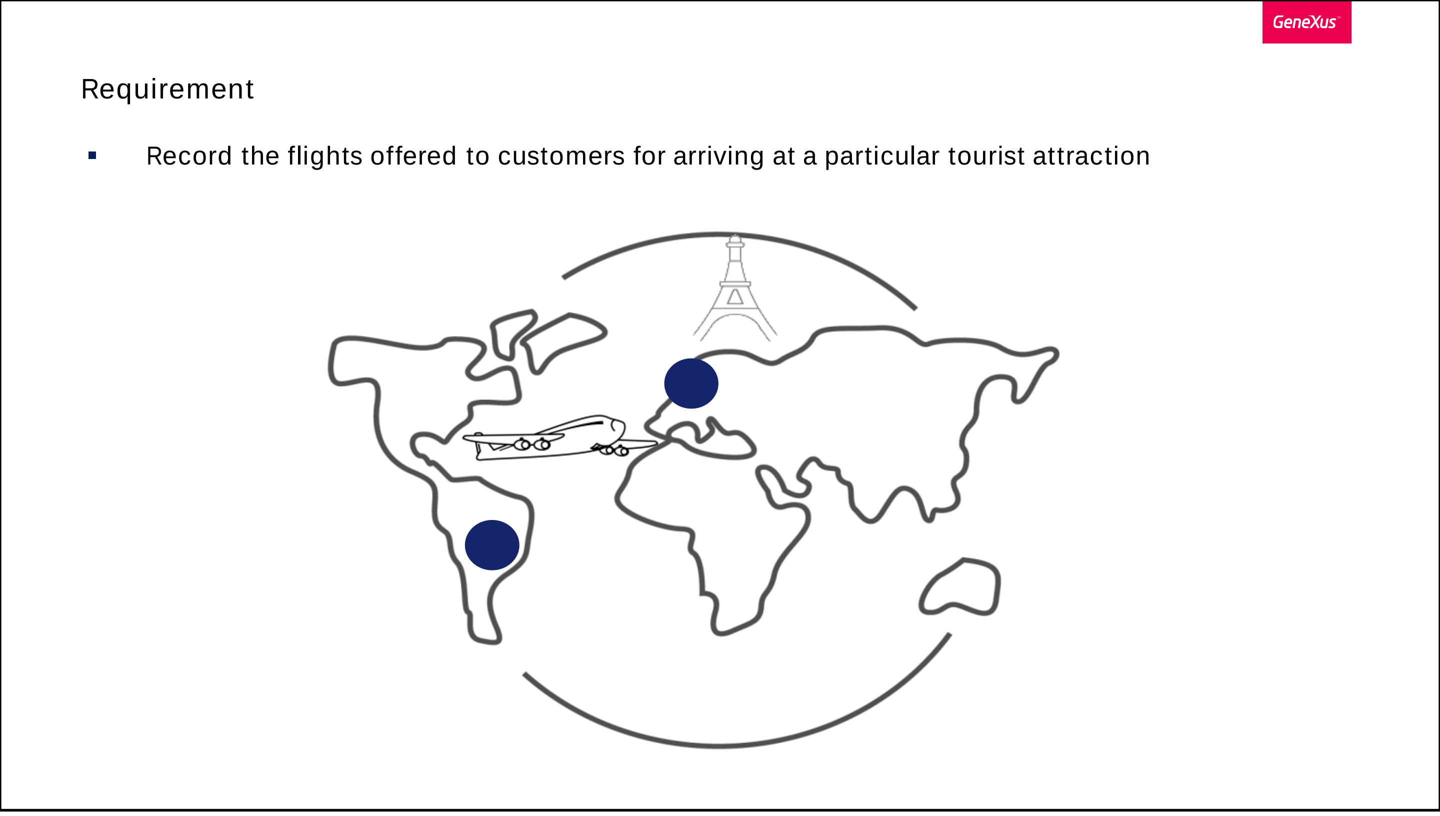
So far, we have seen that GeneXus defines relations between transactions –and between tables-, based on the names of attributes that it considers equal to one another.

For instance, the CountryId attribute is in the Attraction transaction, where it has **a foreign key role,** because it is present, under the same name, in the Country transaction, where it is primary key.

In turn, the CountryName attribute is also in both transactions under the same name: so, GeneXus understands that it is the same attribute. In this case, it is not a primary attribute, and therefore GeneXus will decide to store it in the COUNTRY table instead of in the ATTRACTION table.

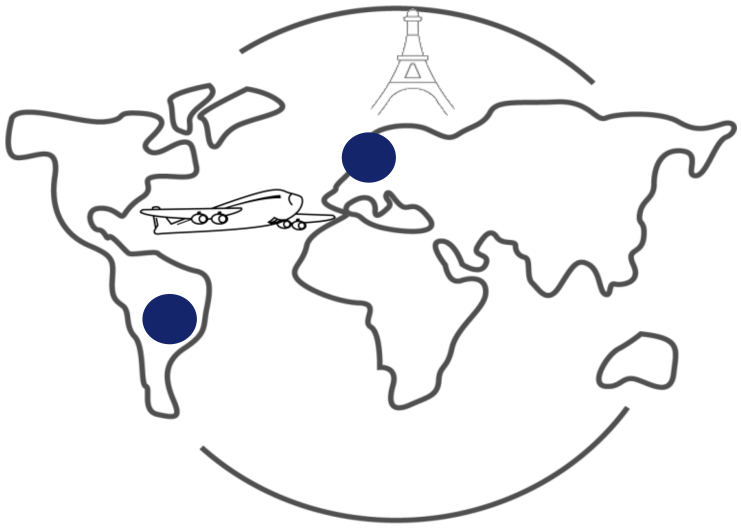
We can then say that **GeneXus always assumes that, if we use the same attribute name, then we are representing the same concept**.

However, there are cases where we might need to use different names for a single concept and indicate to GeneXus that the two names **have the same meaning**.



Requirement

- Record the flights offered to customers for arriving at a particular tourist attraction

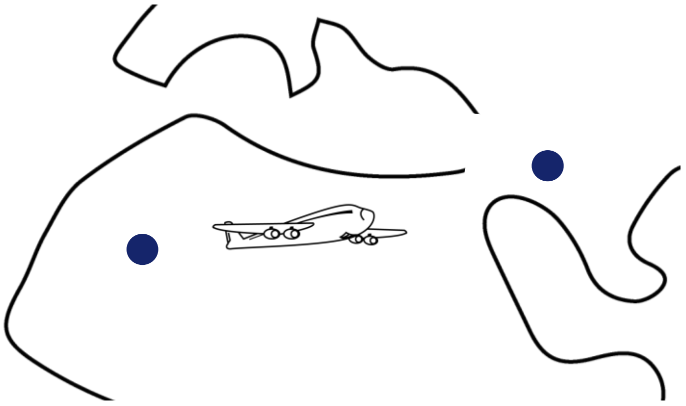


Let´s see how this works.

Let´s suppose that the travel agency needs to record the flights offered to customers for arriving at a particular tourist attraction.

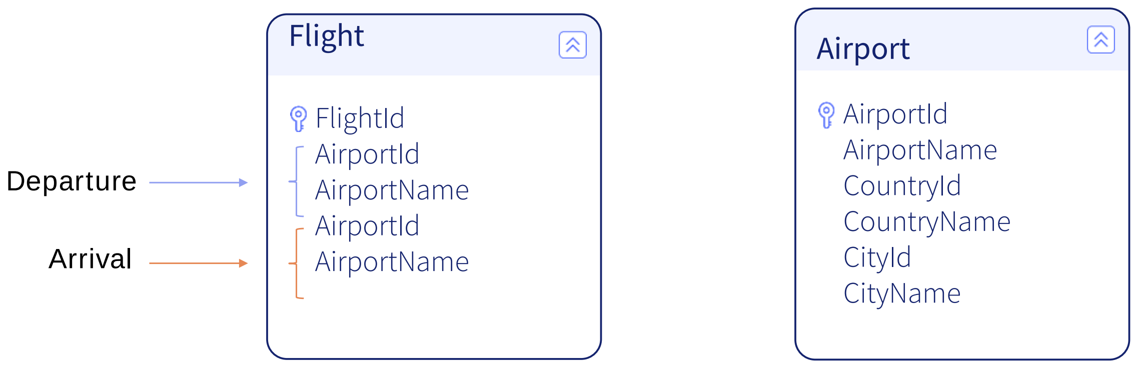
Requirement

- For each flight, the departure airport, as well as the arrival airport



We must record, for each flight, the departure airport, as well as the arrival airport.

Need for attributes with different names to represent the same concept



To represent this, first we will create a transaction under the name: Flight.

We define the FlightId attribute, which is automatically based on the ID domain.

Now let’s think **what other information we should record**.

As we said, each flight will have a departure airport and an arrival airport, but we will have to record each airport on its own, in order to later reference them from the flights.

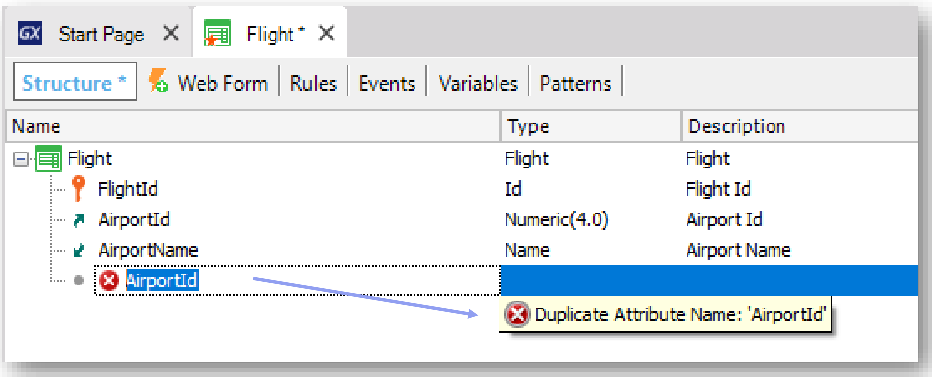
So, let’s leave the Flight transaction aside for a moment and create another transaction called Airport.

We then define that each airport has one identifier that is AirportId, one name that is AirportName, and is located in one country and in one city, so we will add the attributes: CountryId, CountryName, CityId and CityName. We save, and now we go back to see what our requirement was in the Flight transaction.

To each flight we must add its departure airport and its arrival airport.

GeneXus

Duplicated name



Name	Type	Description
Flight	Flight	Flight
FlightId	Id	Flight Id
AirportId	Numeric(4.0)	Airport Id
AirportName	Name	Airport Name
AirportId		

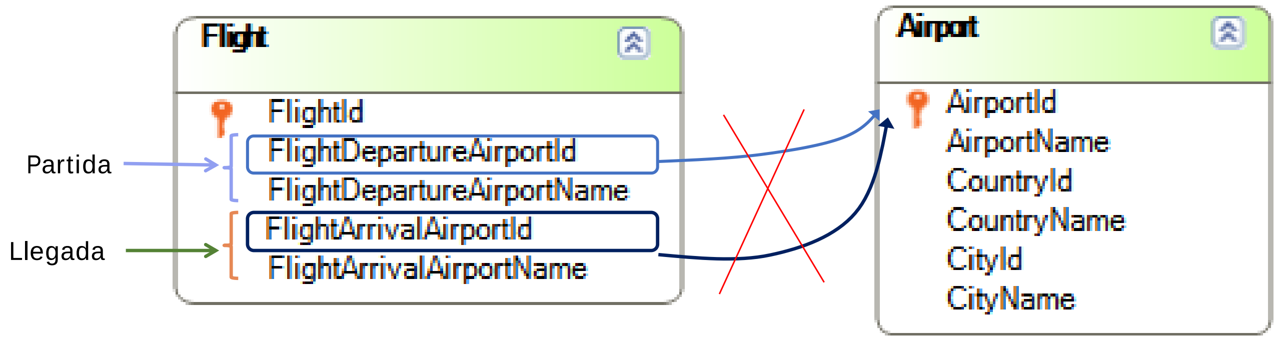
Duplicate Attribute Name: 'AirportId'

So, we go back to the Flight transaction and add the AirportId and AirportName attributes... but when we try to add AirportId again, GeneXus tells us there's an error! Which is: we're adding an attribute with a duplicated name!

And the same will happen with the AirportName attribute that we were going to add to represent the name of the arrival airport.

So, what can we do to enter two airports in one transaction? Obviously, we will have to use **names of different attributes** to store the flight's origin and destination information that we want to record.

Defining the solution…



Different attribute names
There is no relation
between transactions!

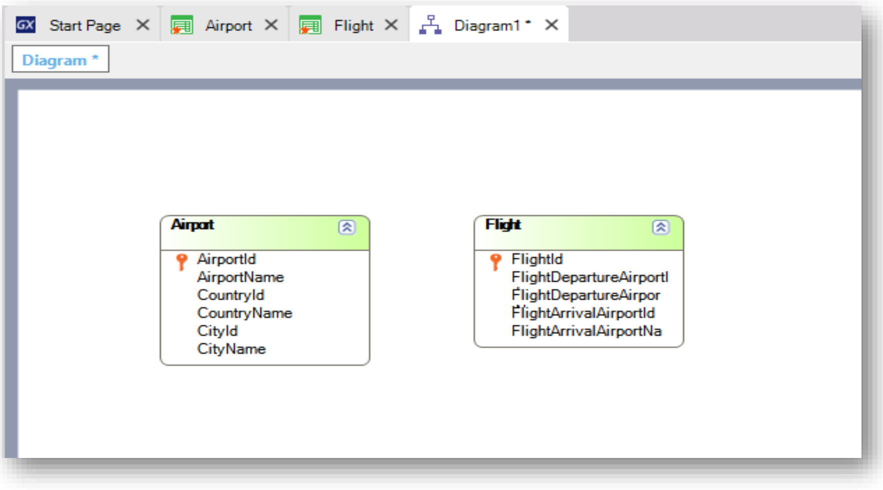
Let's then delete the attributes we originally entered and define attributes with new names.

We will call the flight's departure airport identifier with the name FlightDepartureAirportId, and the name of the departure airport will be FlightDepartureAirportName.

Well, now we have defined new attribute names, but to GeneXus these attribute names are not related to AirportId or AirportName.

As we said before, if we use different names in the Flight and Airport transactions to identify the airport concept, then GeneXus will not define any relation between the two transactions.

Diagram



Airport
- AirportId
- AirportName
- CountryId
- CountryName
- CityId
- CityName

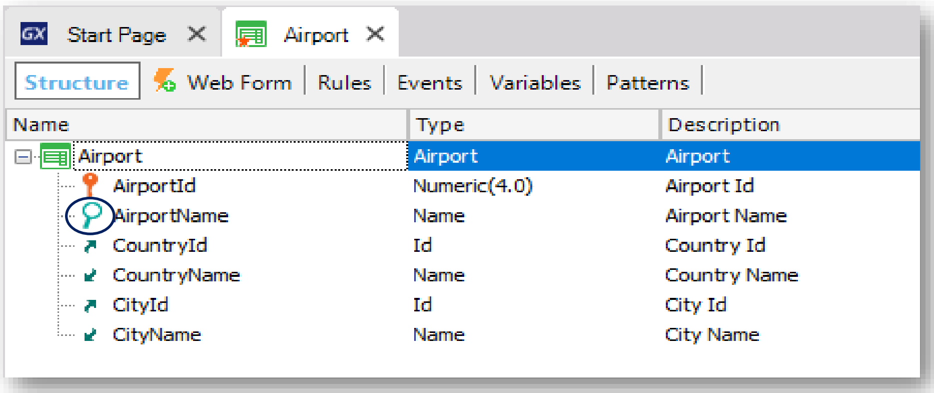
Flight
- FlightId
- FlightDepartureAirportI
- FlightDepartureAirpor
- FlightArrivalAirportId
- FlightArrivalAirportNa

To verify this, we will create a transaction diagram.

And we will drag the Airport and Flight transactions to see that, in fact, GeneXus does not find any relation between them because no foreign key was identified in Flight to allow the relation with Airport.

If a relation had been found, an arrow would appear between both transactions.

Description attribute



Name	Type	Description
Airport	Airport	Airport
AirportId	Numeric(4.0)	Airport Id
AirportName	Name	Airport Name
CountryId	Id	Country Id
CountryName	Name	Country Name
CityId	Id	City Id
CityName	Name	City Name

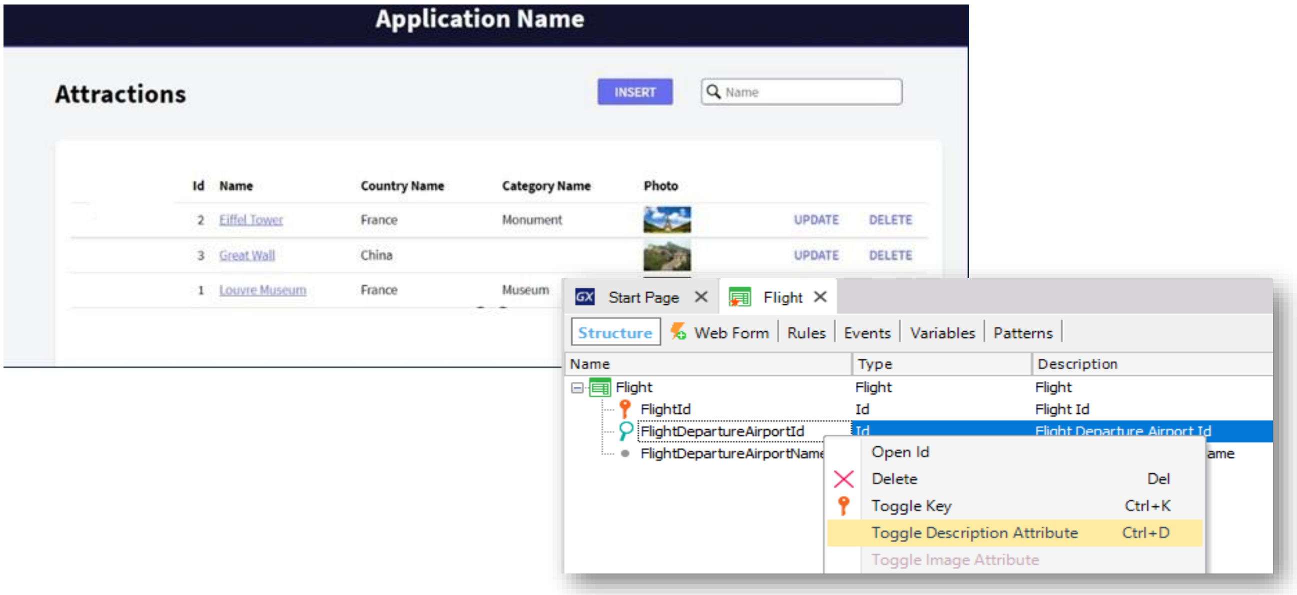
Another way to consider this is to focus on the way in which GeneXus shows the airport identifier attribute in the Flight transaction.

We can see that it's indicated with this symbol, which in the Airport transaction is also in AirportName.

This symbol is indicating that the attribute is the one that best describes the airport, in this case, or the flight, in the other case. When we create a transaction's structure, GeneXus chooses, based on the attributes' data types, which attribute best describes the transaction, but the user can change it, or decide that there should be no attribute of this class.

GeneXus

Description attribute



Application Name

Attractions

INSERT | Name

Id	Name	Country Name	Category Name	Photo		
2	Eiffel Tower	France	Monument		UPDATE	DELETE
3	Great Wall	China			UPDATE	DELETE
1	Louvre Museum	France	Museum			

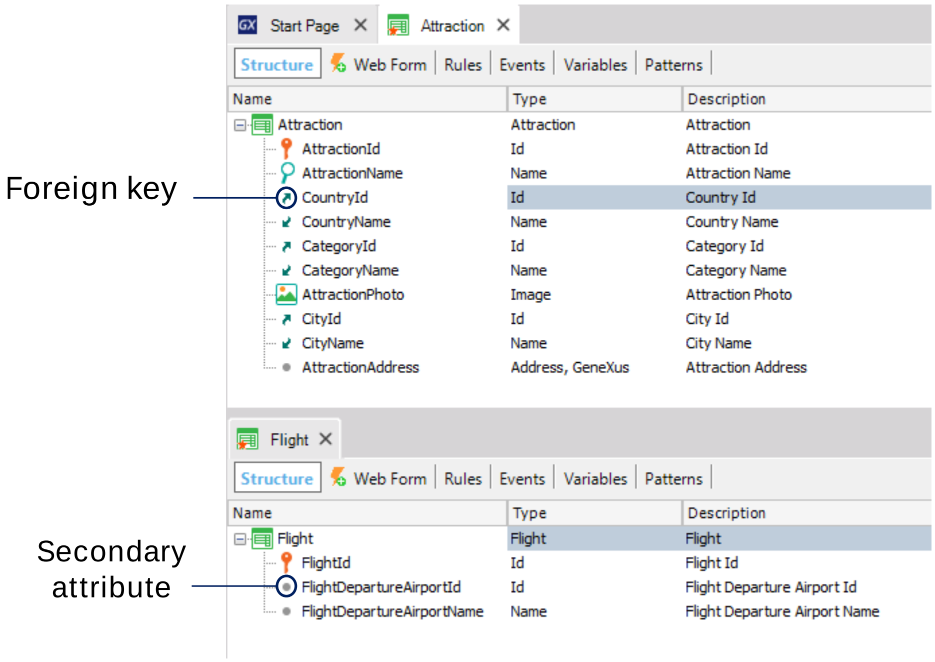
Name	Type	Description
Flight	Flight	Flight
FlightId	Id	Flight Id
FlightDepartureAirportId	Id	Flight Departure Airport Id
FlightDepartureAirportName		ame

This "describing attribute" is used, for example, in the Work With pattern, to allow filtering and ordering by it, amongst other options.

Let's remember the pattern we applied to Attraction.

In the Flight transaction, we do not want the departure airport to be the attribute that best describes the flight, so we remove it.

We can see that it is indicated with the square symbol, meaning that it is a secondary attribute and is not considered as foreign key.



Name	Type	Description
Attraction	Attraction	Attraction
AttractionId	Id	Attraction Id
AttractionName	Name	Attraction Name
CountryId	Id	Country Id
CountryName	Name	Country Name
CategoryId	Id	Category Id
CategoryName	Name	Category Name
AttractionPhoto	Image	Attraction Photo
CityId	Id	City Id
CityName	Name	City Name
AttractionAddress	Address, GeneXus	Attraction Address

Name	Type	Description
Flight	Flight	Flight
FlightId	Id	Flight Id
FlightDepartureAirportId	Id	Flight Departure Airport Id
FlightDepartureAirportName	Name	Flight Departure Airport Name

Let's compare this to the country identifier definition in the Attraction transaction.

In Attraction, the CountryId has an arrow pointing upwards, indicating that it is a foreign key attribute… but that is not the case of the FlightDepartureAirportId attribute in the Flight transaction.

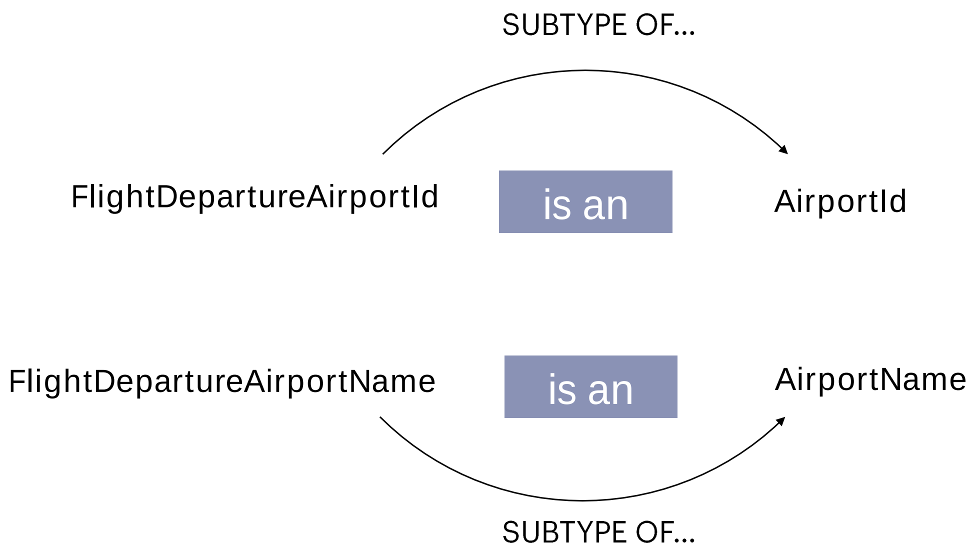
So, **how can we get GeneXus to relate different names to a single concept?**

We need FlightDepartureAirportId, even with a name different from AirportId, to be considered as such, that is: **as an airport identifier!**

And the same goes for the airport name!

Defining the solution…

- Through subtypes, it is possible to make two attributes with different names correspond to the same concept.



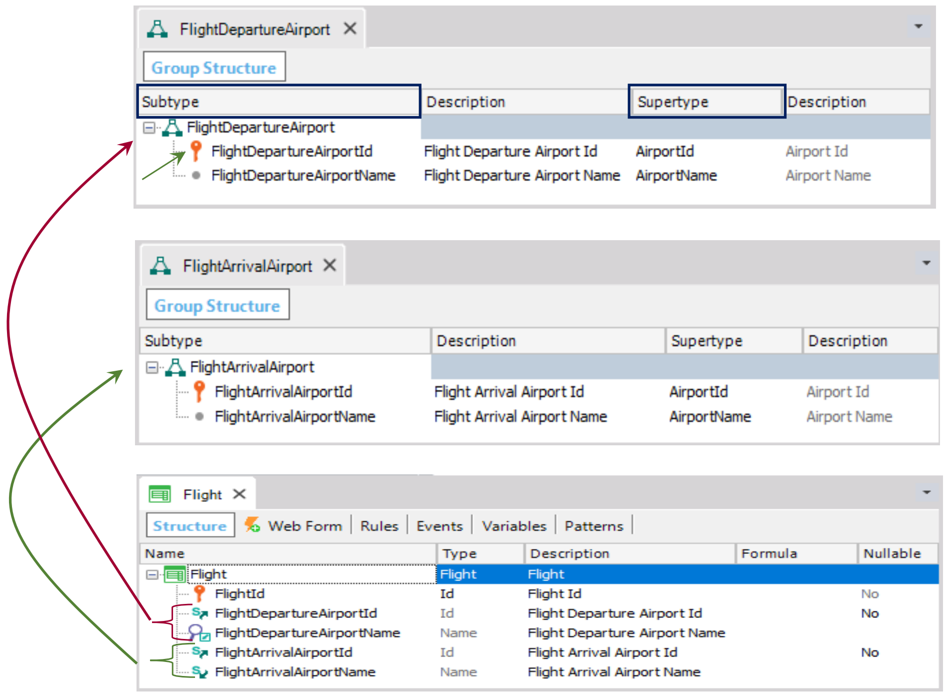
But, how can we make this happen?

The answer is: **by defining sub-types**.

When an attribute has a name different from another attribute already defined and they both represent the same concept, **we can "inform" GeneXus** that the new attribute is a sub-type of the other attribute.

From that moment on, to GeneXus, they will be exactly the same thing. So, GeneXus will consider the attribute FlightDepartureAirportId **just as if it was an AirportId, that it: it will identify is as foreign key in the Flight transaction.**

And we will do the same with FlightDepartureAirportName: we will indicate that it is a sub-type of AirportName.



FlightDepartureAirport

Group Structure

Subtype	Description	Supertype	Description
FlightDepartureAirport			
FlightDepartureAirportId	Flight Departure Airport Id	AirportId	Airport Id
FlightDepartureAirportName	Flight Departure Airport Name	AirportName	Airport Name

FlightArrivalAirport

Group Structure

Subtype	Description	Supertype	Description
FlightArrivalAirport			
FlightArrivalAirportId	Flight Arrival Airport Id	AirportId	Airport Id
FlightArrivalAirportName	Flight Arrival Airport Name	AirportName	Airport Name

Flight

Structure | Web Form | Rules | Events | Variables | Patterns

Name	Type	Description	Formula	Nullable
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightDepartureAirportId	Id	Flight Departure Airport Id		No
FlightDepartureAirportName	Name	Flight Departure Airport Name		
FlightArrivalAirportId	Id	Flight Arrival Airport Id		No
FlightArrivalAirportName	Name	Flight Arrival Airport Name		

SOLUTION

Now, let's go to the Flight transaction, where we will see that the FlightDepartureAirportId attribute has the symbol of the arrow pointing upwards, indicating that it will be considered as foreign key, and also the letter S symbol, meaning that it is an attribute defined as sub-type.

Let's now do the same to define the attributes that will enable us to record the flight's arrival airport.

To do that we will define attributes FlightArrivalAirportId and FlightArrivalAirportName.

Then we save.

We now create a new object of the type "Subtype group", with the name: FlightArrivalAirport.

We type the dot ('.')... and GeneXus suggests the attributes starting by "FlightArrivalAirport", so we select FlightArrivalAirportId.

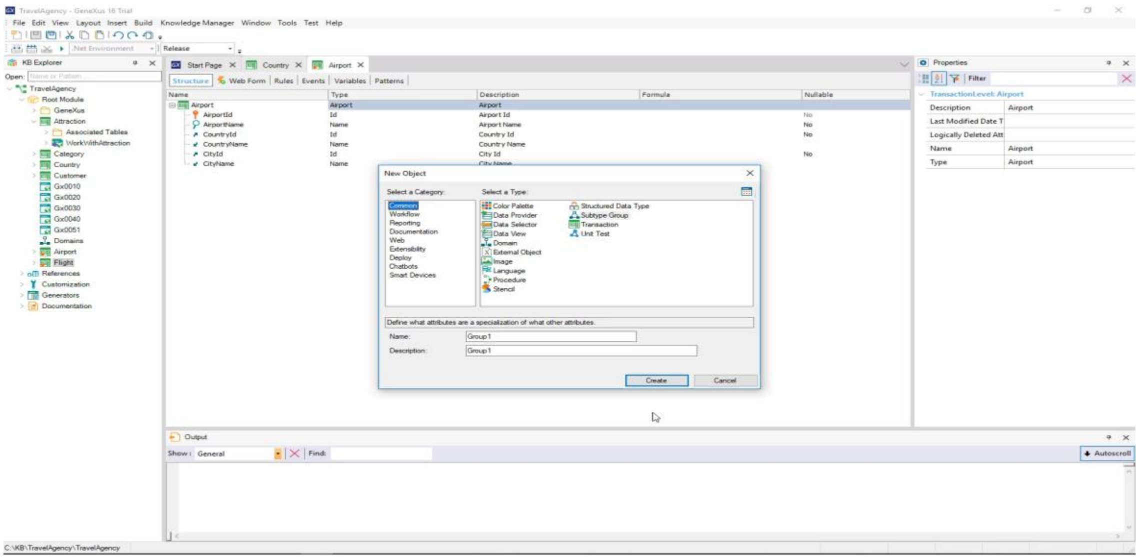
We press tab and declare that it will be a sub-type of AirportId.

Now we add FlightArrivalAirportName, and define that its super-type is AirportName.

Then we save.

Let's take a look at the structure of the Flight transaction again.

DEMO



Name	Type	Description	Formula	Nullable
Airport	Airport	Airport		
AirportId	Id	Airport Id		No
AirportName	Name	Airport Name		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		
CityId	Id	City Id		No
CityName	Name	City Name		

New Object

Select a Category: Common, Workflow, Reporting, Documentation, Web, Extensibility, Deploy, Chatbots, Smart Devices

Select a Type: Color Palette, Data Provider, Data Selector, Data View, Domain, External Object, Image, Language, Procedure, Stencil, Structured Data Type, Subtype Group, Transaction, Unit Test

Define what attributes are a specialization of what other attributes.

Name: Group1

Description: Group1

[DEMO: https://youtu.be/swgogPuGnOM]

Now, let’s see this in action.

The first thing we must do to define sub-types is create a group of sub-types.

So, we create a new object of the type Subtype group, and name it FlightDepartureAirport.

In the first line here we type the dot (‘.’), and GeneXus suggests the attributes that start by “FlightDepartureAirport” which we had already defined in the Flight transaction.

We then select FlightDepartureAirportId... press Tab, and for FlightDepartureAirportId to be a sub-type of AirportId we select the attribute AirportId as super-type.

We can say that the supertype is the original attribute, and the subtype is the attribute that conceptually matches that original attribute, but has another name.

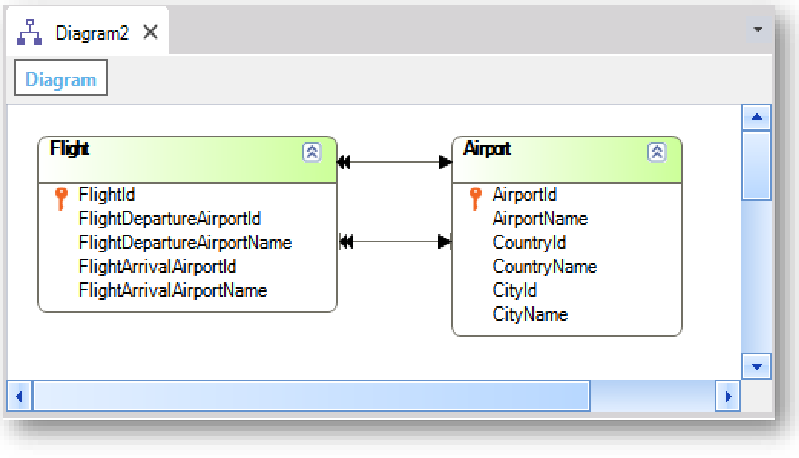
Now we add FlightDepartureAirportName, and define that its super-type is AirportName.

Then we save.

This attribute becomes the identifier of this subtype group, so we call it “**primary”**, and all the attributes we add in this group, such as FlightDepartureAirportName, will depend on it, like they do in the transaction.

GeneXus

Diagram: GeneXus has found a relation



The data will be checked for referential integrity at runtime!

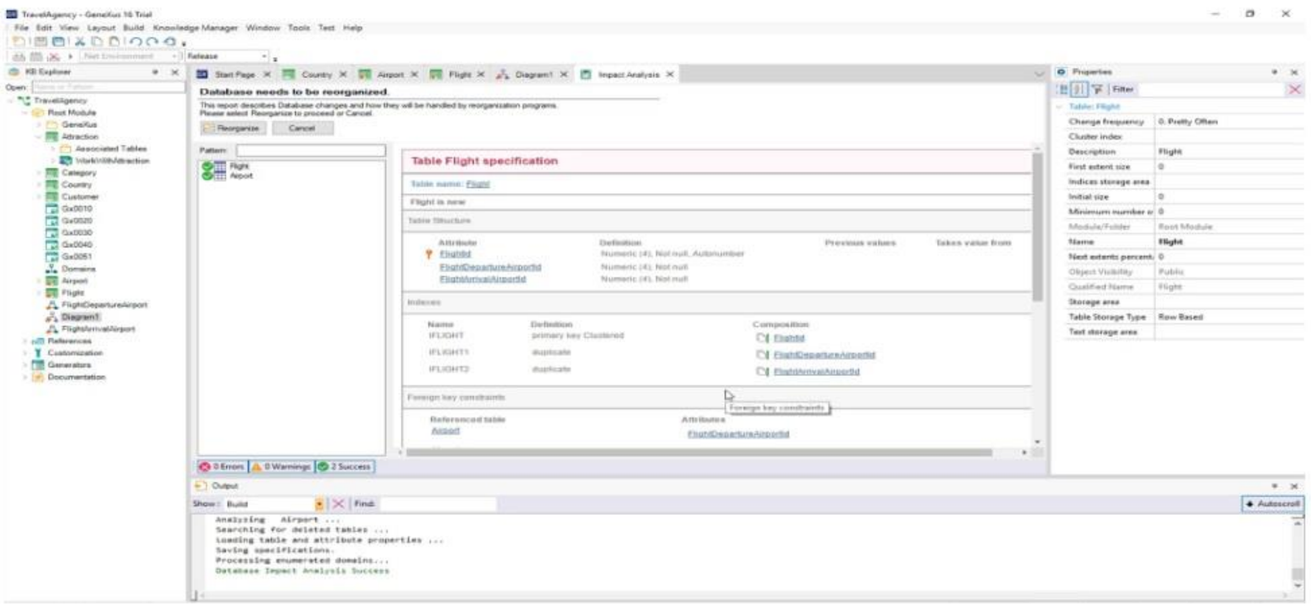
Let´s analyze again the transaction diagram we had previously created.

We can see that now, GeneXus does draw an arrow between the transactions: it considers the subtype identifier attributes of airport in Flight, exactly as if we had referenced AirportId.

We can see that GeneXus has found the relation between Flight and Airport.

Note that even though in GeneXus there´s only one arrow in the diagram, technically there should be two.

Reorganize…



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.

Table Flight specification

Table name: Flight

Flight is new

Table Structure

Attribute	Definition	Previous values	Takes value from
FlightId	Numeric (4), Not null, Autonumber		
FlightDepartureAirportId	Numeric (4), Not null		
FlightArrivalAirportId	Numeric (4), Not null		

Indexes

Name	Definition	Composition
IFLIGHT	primary key Clustered	FlightId
IFLIGHT1	duplicate	FlightDepartureAirportId
IFLIGHT2	duplicate	FlightArrivalAirportId

Foreign key constraints

Referenced table	Attributes
Airport	FlightDepartureAirportId

Properties — Table: Flight

Change frequency	0. Pretty Often
Cluster index	
Description	Flight
First extent size	0
Indices storage area	
Initial size	0
Minimum number o	0
Module/Folder	Root Module
Name	Flight
Next extents percent	0
Object Visibility	Public
Qualified Name	Flight
Storage area	
Table Storage Type	Row Based
Text storage area	

```
Analyzing   Airport ...
Searching for deleted tables ...
Loading table and attribute properties ...
Saving specifications.
Processing enumerated domains...
Database Impact Analysis Success
```

Let’s see how all this functions. To do so, we press F5…

The database must be reorganized, as the Flight and Airport tables need to be created. We agree, so we press Reorganize.

We will start by defining airports, so we will execute the Airport transaction:

We enter the “Guarulhos” airport and indicate the corresponding country: Brazil and city: Sao Paulo.

Now we will enter the “Charles de Gaulle” airport… so, we select: France, and the city is: Paris. Now we confirm.

Now we will go on to record a flight.

We now execute the Flight transaction… and as departure airport we will select “Guarulhos”, while our arrival airport will be “Charles de Gaulle”.

We can see that the labels in the attributes are not showing us that this one is the departure airport and this one the arrival one. If we go to the transaction’s form in GeneXus, and position ourselves on the field of the first airport.

We can see that the text shown in the label, that is to say, its Caption, is taken from the attribute’s ContextualTitle property. If we look for it, here is where it’ll appear.

We add “Departure”.

We do the same with the name, and add “Arrival” to the arrival airport.

We can look at the form, and let’s press F5 once again.

Now we will enter another flight.

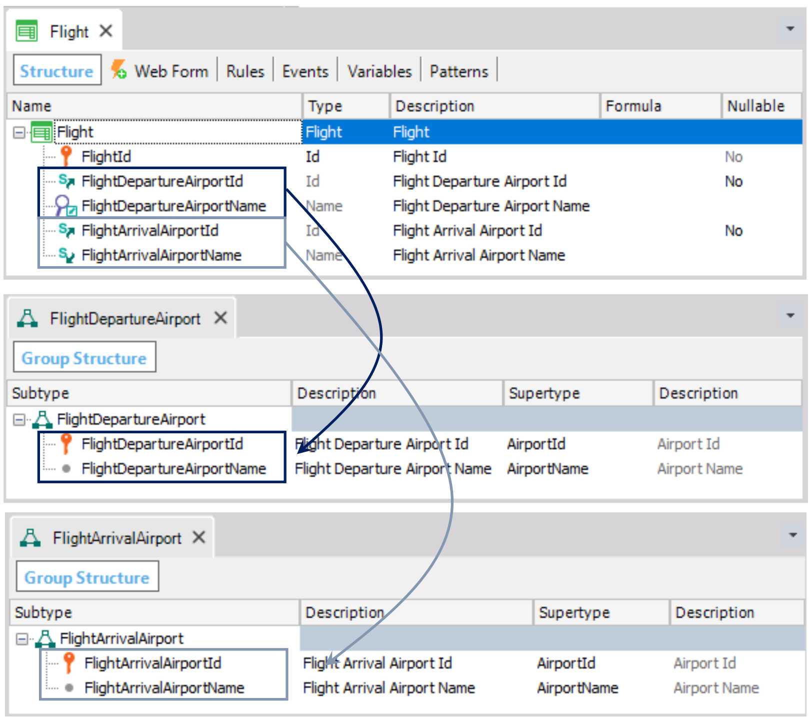
If we try to type in airport 15... we get a notice that the airport does not exist.

Data consistency is being controlled, and we are offered the selection list... we can see that we have the same controls and help as if the attributes were the foreign keys with their original names, like we saw in this video, **but in this case they are their sub-type attributes.**

And that was exactly the idea: **to define sub-types in order to determine that different attribute names correspond to the same concept!**

This is why this attribute... and this attribute... are interpreted as foreign keys, and these attributes will be inferred based on them.

But, how does GeneXus know that in this one it should infer the name of this airport, and not of this one?



Name	Type	Description	Formula	Nullable
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightDepartureAirportId	Id	Flight Departure Airport Id		No
FlightDepartureAirportName	Name	Flight Departure Airport Name		
FlightArrivalAirportId	Id	Flight Arrival Airport Id		No
FlightArrivalAirportName	Name	Flight Arrival Airport Name		

Subtype	Description	Supertype	Description
FlightDepartureAirport			
FlightDepartureAirportId	Flight Departure Airport Id	AirportId	Airport Id
FlightDepartureAirportName	Flight Departure Airport Name	AirportName	Airport Name

Subtype	Description	Supertype	Description
FlightArrivalAirport			
FlightArrivalAirportId	Flight Arrival Airport Id	AirportId	Airport Id
FlightArrivalAirportName	Flight Arrival Airport Name	AirportName	Airport Name

It's not because of the order in which the attributes are shown or because of the names we gave them. GeneXus knows because this attribute has been defined in the same group as this one. And this attribute was defined in a different group, with this one.

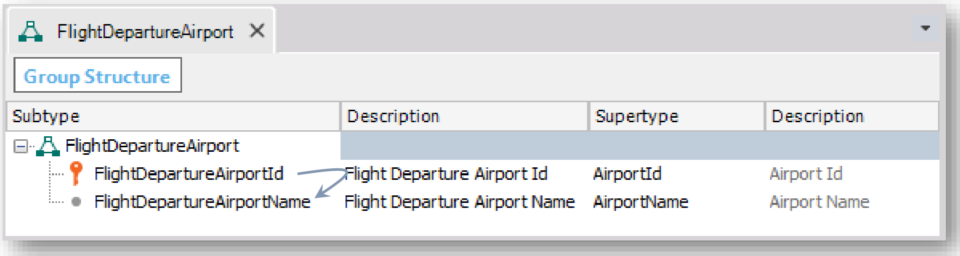
So, we have seen that using sub-types enabled us to represent a situation that occurs in reality, such as in this case, the flight with two airports with different roles: one for departing and the other for arriving.

Something we should note here is that, although we defined attributes with descriptive names that respectively refer to the departure and arrival roles, it was also highly important <u>to group, in the same group of sub-types, the attributes that correspond to one another</u>.

Note that we have not included all the sub-types in a single group, or the two primary sub-type attributes in one group and the two secondary attributes in another group. We have grouped together the attributes that define the departure airport, and in another group we included the attributes that define the arrival airport.

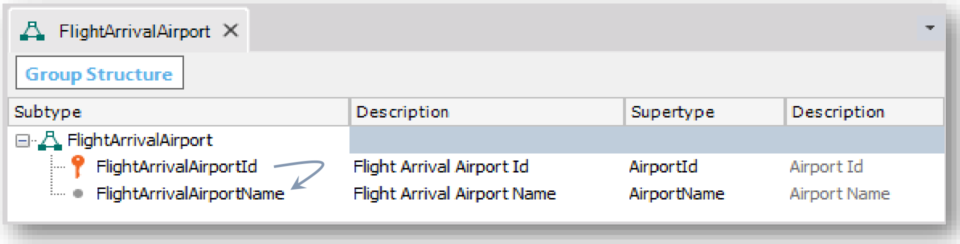
GeneXus

Subtype groups


FlightDepartureAirport

Group Structure

Subtype	Description	Supertype	Description
FlightDepartureAirport			
FlightDepartureAirportId	Flight Departure Airport Id	AirportId	Airport Id
FlightDepartureAirportName	Flight Departure Airport Name	AirportName	Airport Name


FlightArrivalAirport

Group Structure

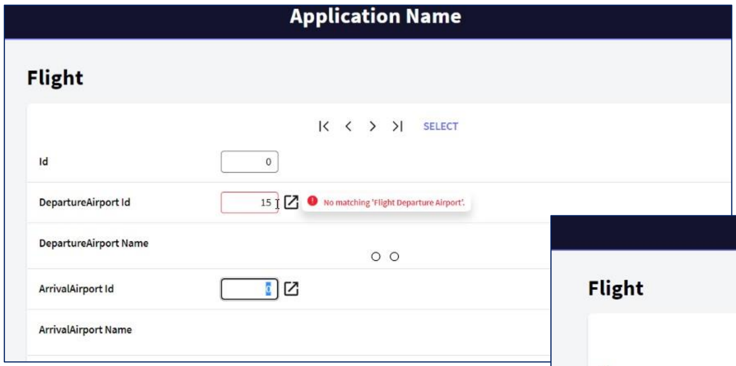
Subtype	Description	Supertype	Description
FlightArrivalAirport			
FlightArrivalAirportId	Flight Arrival Airport Id	AirportId	Airport Id
FlightArrivalAirportName	Flight Arrival Airport Name	AirportName	Airport Name

And this is so, because GeneXus understands with this group:

that when a value is entered for airport identifier FlightDepartureAirportId, then the name of the airport corresponding to **this identifier** must be loaded **in this FlightDepartureAirportName attribute,** and not in the other airport name we have in the transaction.

Likewise, GeneXus understands that when a value is typed for the airport identifier: FlightArrivalAirportId, the name of the corresponding airport must be loaded in the FlightArrivalAirportName attribute.

Subtype groups

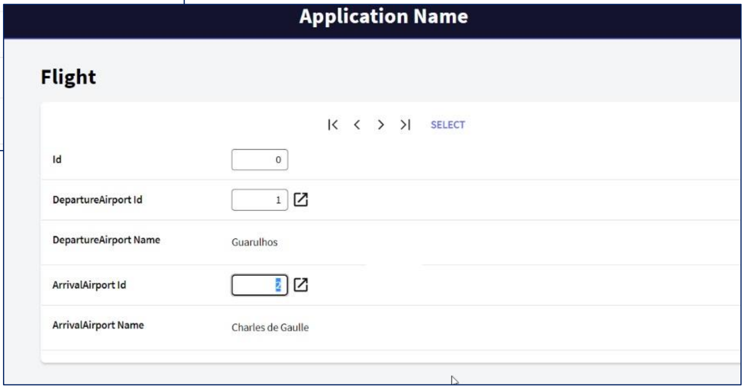


Application Name

Flight

Id	0
DepartureAirport Id	15
DepartureAirport Name	
ArrivalAirport Id	
ArrivalAirport Name	

No matching 'Flight Departure Airport'.



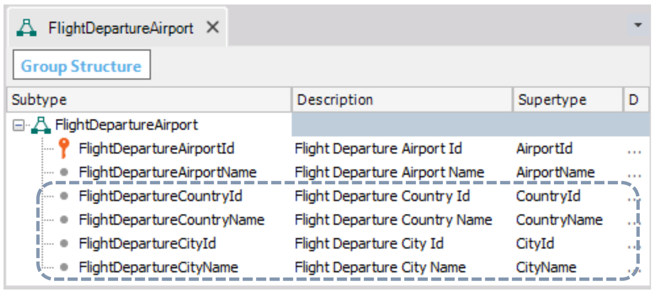
Application Name

Flight

Id	0
DepartureAirport Id	1
DepartureAirport Name	Guarulhos
ArrivalAirport Id	2
ArrivalAirport Name	Charles de Gaulle

- RI control
- Selection list
- Grouped inferences

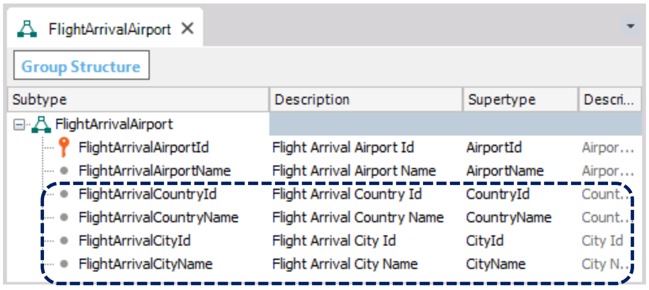
And what if for each airport we want to view its country and city?



FlightDepartureAirport

Group Structure

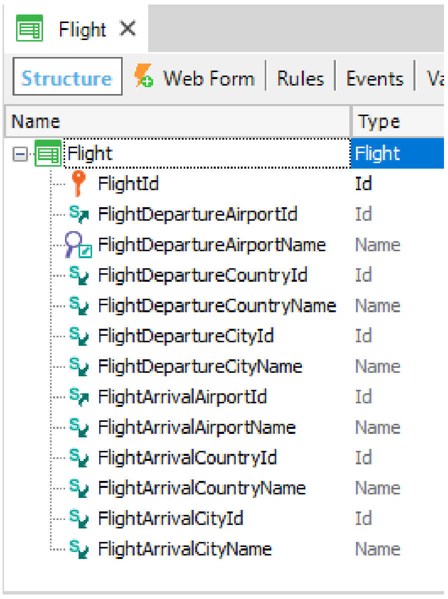
Subtype	Description	Supertype	D
FlightDepartureAirport			
FlightDepartureAirportId	Flight Departure Airport Id	AirportId	...
FlightDepartureAirportName	Flight Departure Airport Name	AirportName	...
FlightDepartureCountryId	Flight Departure Country Id	CountryId	.
FlightDepartureCountryName	Flight Departure Country Name	CountryName	.
FlightDepartureCityId	Flight Departure City Id	CityId	.
FlightDepartureCityName	Flight Departure City Name	CityName	.



FlightArrivalAirport

Group Structure

Subtype	Description	Supertype	Descri...
FlightArrivalAirport			
FlightArrivalAirportId	Flight Arrival Airport Id	AirportId	Airpor...
FlightArrivalAirportName	Flight Arrival Airport Name	AirportName	Airpor...
FlightArrivalCountryId	Flight Arrival Country Id	CountryId	Count...
FlightArrivalCountryName	Flight Arrival Country Name	CountryName	Count...
FlightArrivalCityId	Flight Arrival City Id	CityId	City Id
FlightArrivalCityName	Flight Arrival City Name	CityName	City N...



Flight

Structure | Web Form | Rules | Events | Va

Name	Type
Flight	Flight
FlightId	Id
FlightDepartureAirportId	Id
FlightDepartureAirportName	Name
FlightDepartureCountryId	Id
FlightDepartureCountryName	Name
FlightDepartureCityId	Id
FlightDepartureCityName	Name
FlightArrivalAirportId	Id
FlightArrivalAirportName	Name
FlightArrivalCountryId	Id
FlightArrivalCountryName	Name
FlightArrivalCityId	Id
FlightArrivalCityName	Name

Now let's suppose that in the Flight transaction, for each airport we want to see, besides its name, its country and city as well.

This may be solved by simply **defining more sub-type attributes, in each group necessary, with the adequate naming and with indication of their super-types. This will enable GeneXus to understand that, for the group's primary sub-type, it will have to infer all the remaining related information.**

Let's do it.

In this group of sub-types we will define the following attributes:

FlightDepartureCountryId as sub-type of CountryId, FlightDepartureCountryName as sub-type of CountryName, FlightDepartureCityId as sub-type of CityId, and FlightDepartureCityName as sub-type of CityName. We now save.

And we will also add these new attributes to the structure of the Flight transaction. We save again.

And go on to do the same for the other group of sub-types...

And we define:

FlightArrivalCountryId as sub-type of CountryId, FlightArrivalCountryName as sub-type of CountryName, FlightArrivalCityId as sub-type of CityId, and

FlightArrivalCityName as sub-type of CityName. We save.

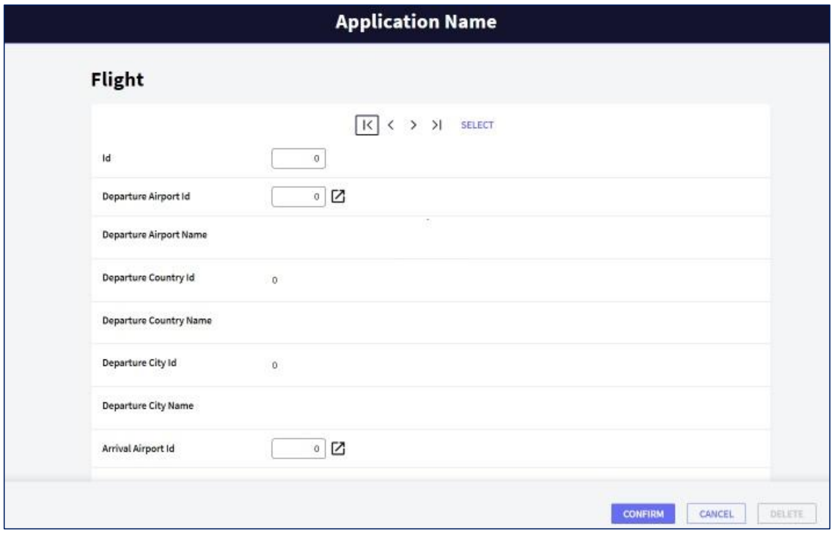
After saving, we also add these to the structure of the Flight transaction.

Like we did before, we change each new attribute's contextual title, so that the label on the screen indicates the attribute's role.

Then we press F5 again to execute the application.

GeneXus

Flight transaction


Application Name

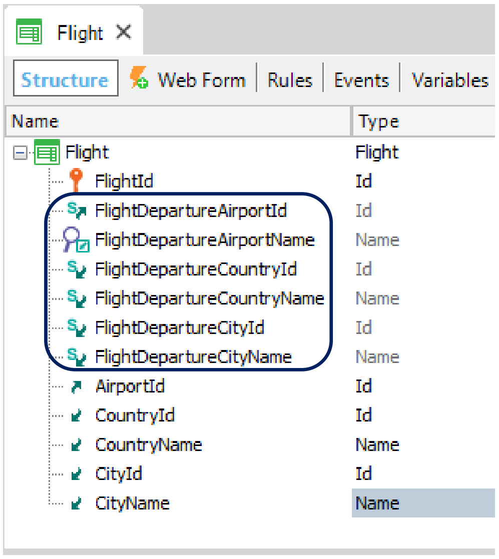
Flight

Id	0
Departure Airport Id	0
Departure Airport Name	
Departure Country Id	0
Departure Country Name	
Departure City Id	0
Departure City Name	
Arrival Airport Id	0

Now we open the Flight transaction, and when we check our first flight we can see, for each airport, its corresponding country and city.

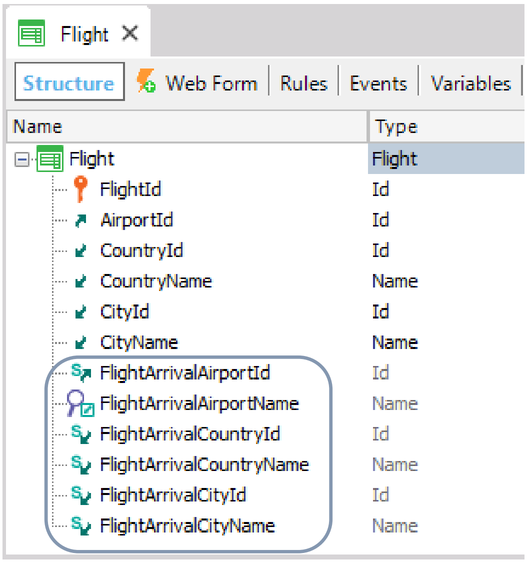
We have seen how to solve a double reference to the same concept with different roles, because the two airports must be obtained from the same table where each of them had a different role.

Other possible solutions



Name	Type
Flight	Flight
FlightId	Id
FlightDepartureAirportId	Id
FlightDepartureAirportName	Name
FlightDepartureCountryId	Id
FlightDepartureCountryName	Name
FlightDepartureCityId	Id
FlightDepartureCityName	Name
AirportId	Id
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name

Defining only one subtype group: "FlightDepartureAirport"



Name	Type
Flight	Flight
FlightId	Id
AirportId	Id
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name
FlightArrivalAirportId	Id
FlightArrivalAirportName	Name
FlightArrivalCountryId	Id
FlightArrivalCountryName	Name
FlightArrivalCityId	Id
FlightArrivalCityName	Name

Defining only one subtype group: "FlightArrivalAirport"

To end, it is important to know that also **these** could have been equally valid solutions.

In the first proposal, a single group of sub-types has been defined, the one corresponding to the **departure airport**, and the super-type attributes were left for the entry of the destination.

And this is totally valid.

In this other proposal... a single group of sub-types has also been defined, but in this case for the group corresponding to the **arrival airport**.

In sum, sub-types enable us to indicate to GeneXus how to associate different attribute names to a single concept. And, as we saw, validations, as well as all the behavior of sub-types will be identical as if we had used the super-type attributes.

There are a lot of other cases in which it's necessary to change an attribute's name in order to avoid a conflict or ambiguity. In this video we saw the case of multiple references from one table to the other, but these references don't have to be direct.

Summary and Remarks

Subtypes have allowed us to represent a reality in which a flight has two airports with different roles.

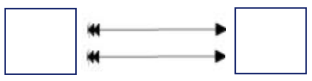
Subtype groups allow us to set both roles apart (We do not define a single group with all the subtypes).
The right way to do this is:

- To define a group of subtypes for each set of attributes matching one another.
- Each group of subtypes must necessarily include a primary attribute subtype (which in turn is a table's primary key), or a set of attributes that comprise a primary key.
- In each group of subtypes, define all the subtype attributes that need to be known and which belong to the base and/or extended table of the group's primary key.

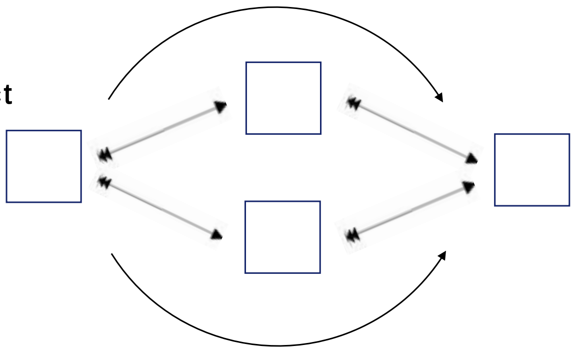
More use cases

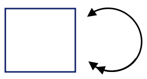
- Multiple references

Direct

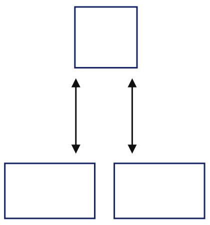


Indirect





- Recursive subtypes



- Specialization

From this table we have two different paths to get to this table. So we would need subtypes to differentiate them.

We also have the case when an entity must reference itself.

For example, an employee transaction, in whose information we have to register the employee's boss, who is also an employee.

Or in the case where we have an entity which registers general information, for example, of people (such as the name, telephone number, address, etc.) and then have entities which are specializations of that other entity, for example clients and passengers, who in particular are also people.

To finish, we will update the changes in GeneXus server. We add the comments...

And press Commit.