Related Transactions

GeneXus™

We create a Country transaction and include CountryId and CountryName in Attraction

We will create a transaction for countries: we will call it Country...

Here we create a country identifier attribute... CountryId, and we can see that GeneXus has assigned it the Id domain...

We create an attribute to store the name of the country, CountryName, which takes the Name domain.

We Save... note that just as it did with Customer, GeneXus has automatically created the form to add, edit and delete countries.

Let's go back to the Attraction transaction...

We want to assign a country to each attraction.

Note that typing the letter C displays the list of attributes existing in the Knowledge Base that begin with this letter.

We select CountryId and its entire definition is displayed.

In this transaction we will also include the CountryName attribute, because when we execute this transaction and select a country identifier

we will want to see the corresponding country name.

Let's focus on these 2 attributes included in more than one transaction.

We will find out what role they play here, in Attraction.

Let's remember that CountryId is the identifier or key in the Country transaction. To be more precise, from now on we will say that CountryId is the primary key of the Country transaction
… and when a primary key is included in another transaction, it has the role of foreign key.

Including an attribute that is a transaction primary key in another transaction allows us to relate both transactions.

Primary key and Foreign key

This means that, when executing the Attraction transaction, for this attribute we will have to enter a value that has been previously recorded through the Country transaction.

Let's see it at runtime. We press F5...

GeneXus analyzes the impact caused by the new definitions made on the Knowledge Base and informs us that a new table called Country will have to be created in the database with the CountryId and CountryName fields.

In addition, a new table called Attraction will also be created, with the fields AttractionId, AttractionName and CountryId.

Note that in the Attraction physical table that will be created by GeneXus, the CountryName attribute is not included even though we had included it in the Attraction transaction structure...

Data Base

Attraction Table

AttractionId	AttractionName	CountryId
1	Louvre Museum	2
2	Great Wall	3
3	Eiffel Tower	2
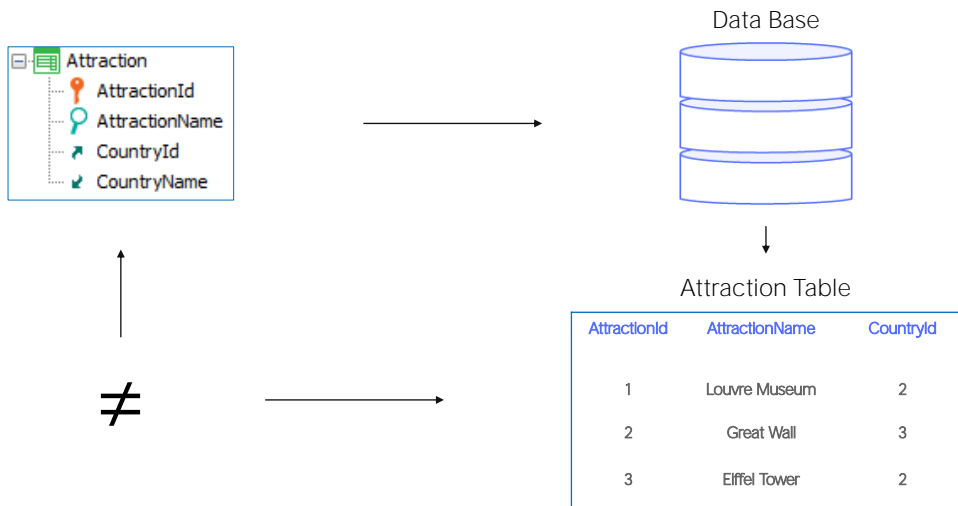
This happens because the TRANSACTION concept is not the same as the PHYSICAL TABLE concept:

Remember that TRANSACTION is the GX object that we created in the Knowledge Base to represent an object or actor of reality... and examining it GeneXus creates a PHYSICAL TABLE in the database, to store the data that will be entered when executing the transaction.

We must be aware that not all the attributes included in a transaction structure will later be stored in the physical table created based on this transaction..

Transaction and physical table

CountryId	CountryName
1	Brazil
2	France
3	China

AttractionId	AttractionName	CountryId
1	Louvre Museum	2
2	Great Wall	3
3	Eiffel Tower	2

Storing the country name in several physical tables would mean storing duplicate data. Instead, the country name can be retrieved from a single location where it is stored, that is to say, from the country table.

This structuring of the tables to avoid data inconsistencies is called database normalization.

We go back to the development environment and click on Reorganize...

The term "Reorganize" means to reorganize the database, that is to say, it refers to the task of making changes to it.

GeneXus creates the programs to change the database, uploads them to the server, and gives the order to execute them, making the necessary changes.

Next, it generates the necessary programs corresponding to the application itself. For example, for each new transaction that we have defined, programs are being generated in the selected programming language to enter, change and delete countries and tourist attractions.

GeneXus

Transaction and physical table

Primary Key →

Name	Type
Country	Country
CountryId	Numeric(4.0)
CountryName	Character(20)

→

Table name: Country

Country is new

Table Structure

Attribute
CountryId
CountryName

Foreign Key →

Name	Type
Attraction	Attraction
AttractionId	Numeric(4.0)
AttractionName	Character(20)
CountryId	Numeric(4.0)
CountryName	Character(20)

→

Table name: Attraction

Attraction is new

Table Structure

Attribute
AttractionId
AttractionName
CountryId
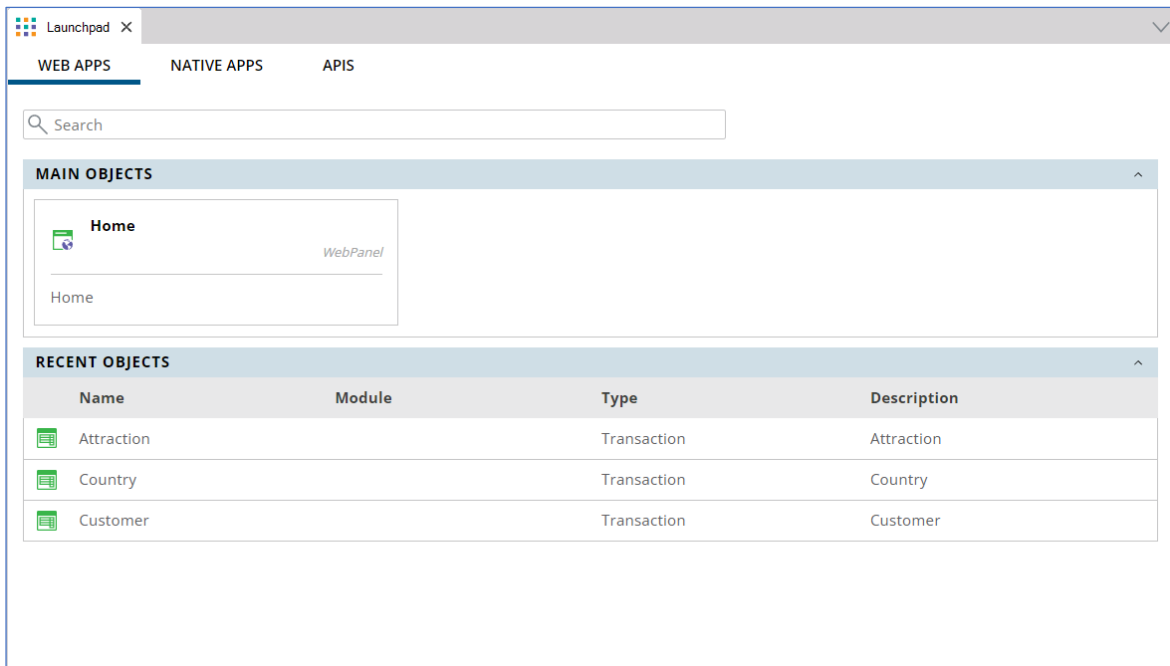
Now we have links to work not only with clients but also with attractions and countries.

Let's add data about some countries.

Since the CountryId attribute was previously defined as belonging to the Id domain and this domain has the Autonumber **property set to True... we don't have** to enter a value for the identifier because it will be numbered automatically.

We add Brazil, France and China...

We browse the data to confirm that they have been numbered...

And we execute the Attraction transaction.

We add the tourist attraction "Louvre Museum". Here again, we don't have to enter a value for the identifier so we type the name "Louvre Museum".

Now we have to indicate that the Louvre Museum is in France.

If we remember the identifier number of France we can enter it, but another option is to select it from a list by clicking on this arrow.

Note that this list displays all the countries that we have entered. We click on the link... or on the green check mark of the line on which France is located.

Note that the country identifier 2 has been loaded... we press the Tab key and the country name is displayed.

Note that we can't change the country name from here, because it is displayed only for reading purposes.

Remember that a selection arrow was automatically displayed next to CountryId and provided a list of available countries.

The arrow was displayed next to this attribute in particular because, as we've said before, here CountryId is a foreign key.

So, here the user will have to enter a value that has been previously registered as primary key value through the Country transaction... Therefore, GeneXus collaborates by generating and offering a list of countries available.

Here we can see, in GeneXus, the object implemented by that list.

Now we will see at runtime, how the Country and Attraction transactions check that the values entered for the CountryId attribute are consistent.

We will enter a new attraction such as the "Pyramids of Egypt". In the country field we enter the value 4 because we think it corresponds to Egypt, but an error message is displayed because country 4 doesn't exist!

We check the registered countries and see that we had only entered country 1, 2 and 3, but not 4.

Likewise, if we want to change an attraction that has already been entered and try to replace its country with another country that doesn't exist —4 again— we get the same error message.

That is to say, when we enter or change data through transactions, the associated data is automatically checked for consistency. Also, when we try to delete data through transactions, the necessary controls are made to maintain the consistency of the stored data.

Now, for instance, if we try to delete the country France, we see a message informing us that the deletion cannot be performed because related data exists in Attraction (remember that we entered the Louvre Museum that belongs to France).

Something very important to learn is that attributes must be named using exactly the same name when they are related to the same concept.

For example, if in the Attraction transaction instead of entering the CountryId attribute we had typed CountryIdentifier, for GeneXus, CountryId and CountryIdentifier **would be different attributes... and for this reason, in Attraction it wouldn't have checked if the value entered the country identifier exists in the country table... and it wouldn't have offered** the country selection list in the Attraction transaction.

Also, it wouldn't be possible to retrieve the name of the corresponding country... because CountryName could be referenced in the Attraction transaction due to the fact that CountryId plays the role of foreign key and retrieves its corresponding CountryName**... but** CountryIdentifier is not a **foreign key because it isn't the primary key of any transaction, so it isn't** possible to retrieve data associated with this attribute.

For each tourist attraction, we were asked to record the associated category:
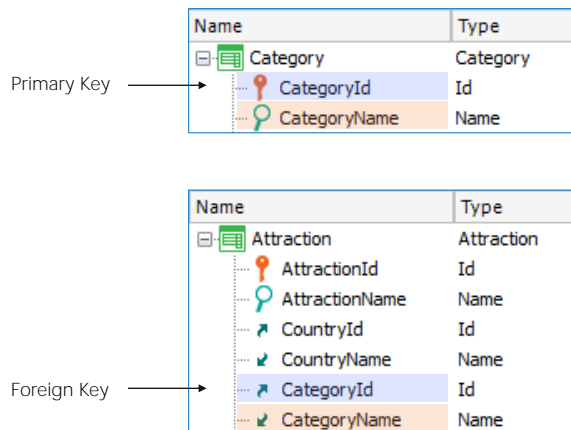
ATTRACTIONS

NAME:

COUNTRY:

IMAGE:

CATEGORY:
Monument
Museum
Park
....

Let's continue to represent more features of the travel agency's reality. We have been told that every attraction has an associated category to indicate if it is a monument, museum, park, and so on.

Name	Type
⊟ ▦ Category	Category
🔑 CategoryId	Id
◯ CategoryName	Name

Primary Key →

Name	Type
⊟ ▦ Attraction	Attraction
🔑 AttractionId	Id
◯ AttractionName	Name
↗ CountryId	Id
↙ CountryName	Name
↗ CategoryId	Id
↙ CategoryName	Name

Foreign Key →

And here we have the same situation that we saw with countries. We will **create a Categories transaction and assign the attraction's categories.**

Let's do it now.

We create the Category transaction.. with CategoryId**... and** CategoryName...

Now we add attributes to the Attraction transaction.

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		

Before trying this at runtime, we can allow the category not to be set, for example because we don't know its value at the time of entering the attraction.

This is done by changing the value of the Nullable property of the CategoryId attribute.

We set it to Yes.

This only makes sense for foreign keys that reference values from another table.

Defining more attributes

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

Let's implement another request of the travel agency: for each attraction, they want to enter its photo. So, in the Attraction transaction we enter an attribute called AttractionPhoto.

It will be of Image type, because this type makes it possible to store images.

Now we press F5 to apply the changes to the database and programs, and run the application.

Note that a new table will be created in the database to store the categories.
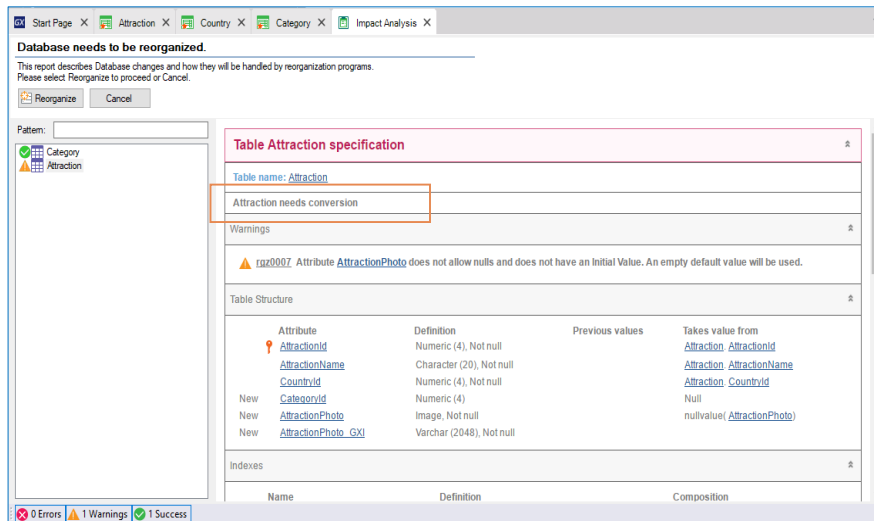
If we click on Attraction, we are told that the Attraction table has to be converted, which means that the CategoryId and AttractionPhoto attributes have to be added.

This element is added to store the file and to give the option to only reference a URL to it.

We press the REORGANIZE button...

The browser is opened and we start to enter some categories.

We enter these categories: Museum... and Monument.

We see that it allows us to enter a category and a photo.

We search for the Louvre Museum and we assign it the Museum category and a photo. We confirm.

Lastly, we send the changes we've made to GeneXus Server.

So far, we have seen:

- Modeling of entities of reality as transactions

- Difference between transaction and physical table

- Primary keys and foreign keys

- Referential integrity checks

- Possibility of leaving a foreign key blank (and not make a referential integrity check).

In short, so far we've seen that:

-The entities of reality are modeled by means of GeneXus transactions.

-There can be differences between the structure of a transaction and that of the associated table in the database, which in the case that we have seen has to do with the fact that some transaction attributes are "inferred" from another table.

The following concepts are involved here:

-Primary keys and foreign keys

-To avoid data inconsistencies, GeneXus builds the tables avoiding the use of redundant attributes (that's why it doesn't place the CountryName attribute in the Attraction table, because it gets it from the CountryId foreign key).

-By doing all this, for each transaction you can include checks that prevent referential inconsistencies in the generated programs, or in other words, you can ensure referential integrity. So, if the user tries to delete a country, for example France, it will include in the Country transaction a

control that will be triggered when the user clicks on Delete, and that will go to the database to check that none of the tables that reference countries contain a record with that country value that the user is trying to delete. Because if that happened, there would be, for example, an attraction that refers to a country that no longer exists.

-In the same way, in the Attraction transaction, for example, it will check for the existence of the country entered by the user.

-With the category, however, it is not the same, because it can take a zero value. That means that if it is left null then the check is not performed. However, it will be made if any value is entered.

Note: Databases usually make referential integrity checks independently of programs.

GeneXus™