

Database Update

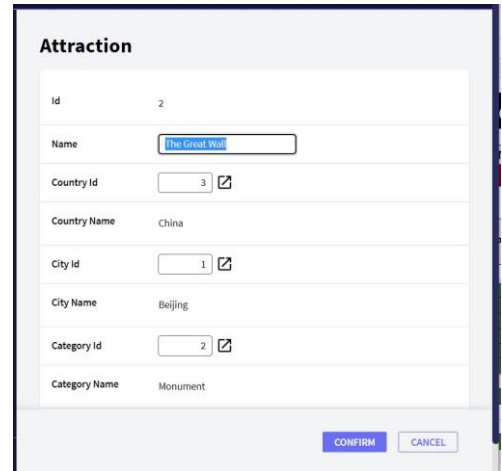
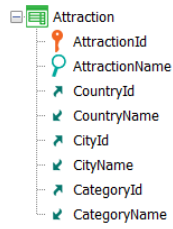
What happens when a foreign key accepts nulls?

GeneXus[™]




1. Update Through a Form



Insert, Update, Delete



Attraction

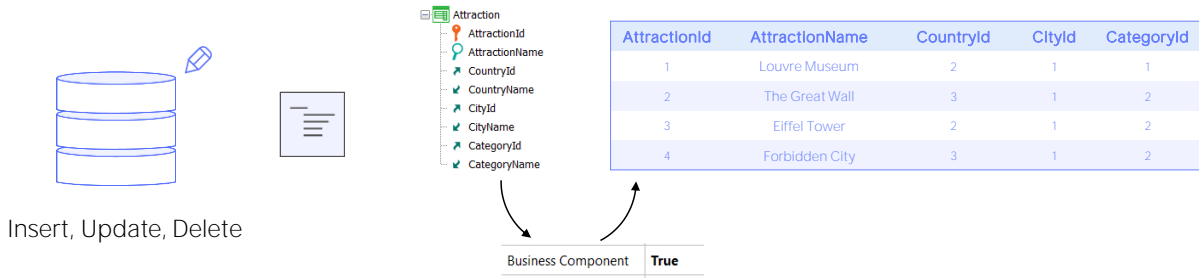
Id	2
Name	<input type="text" value="The Great Wall"/>
Country Id	<input type="text" value="3"/> 
Country Name	China
City Id	<input type="text" value="1"/> 
City Name	Beijing
Category Id	<input type="text" value="2"/> 
Category Name	Monument

2. Update Through Code



There are two ways to update the database: interactively through the transaction object, using its screen, or through code.

1. Business Component: Insert(), Update(), Delete()



2. Procedure: New, For Each, Delete

To update through code there were two options:

Using the business component of the transaction, through its methods, or exclusively within a procedure, through the New, For Each commands with direct assignment of the attributes to be modified, and the Delete command.



	Uniqueness Controls	Referential Integrity Controls	Rule/Event Execution
Business Component	✓	✓	✓
Proc (new, for each, delete)	✓		

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	2

As we know, both options control the uniqueness of records, that is, no records with a repeated primary or candidate key are ever allowed to be left in the database. In the example, it will never be possible to have two attractions with the same identifier. And if, for example, AttractionName were a candidate key, no two attractions with the same name would be allowed either.

There's a big difference between updating via a Business Component and doing it via direct update commands in a procedure: only Business Components will execute the logic coming from the rules and events of the transaction, and only in this way will referential integrity be controlled through a program. Why do we say **"through a program"**?

The image shows two screenshots of the GeneXus IDE's Structure view. The top screenshot shows the 'Attraction' entity with its attributes: AttractionId (Id, No), AttractionName (Name, No), AttractionAddress (Address, GeneXus, No), CountryId (Id, No), CountryName (Name, No), CityId (Id, No), CityName (Name, No), CategoryId (Id, Yes), and CategoryName (Name, No). The 'CategoryId' attribute is highlighted in blue. The bottom screenshot shows the 'Category' entity with its attributes: Category (Category, No), CategoryId (Id, No), and CategoryName (Name, No). The 'Category' attribute is highlighted in blue.

Name	Type	Description	Formula	Nullable
Attraction	Attraction	Attraction		
AttractionId	Id	Attraction Id		No
AttractionName	Name	Attraction Name		No
AttractionAddress	Address, GeneXus	Attraction Address		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		No
CityId	Id	City Id		No
CityName	Name	City Name		No
CategoryId	Id	Category Id		Yes
CategoryName	Name	Category Name		No




Name	Type	Description	Formula	Nullable
Category	Category	Category		
CategoryId	Id	Category Id		No
CategoryName	Name	Category Name		No

If we have the Attraction transaction that records the tourist attractions that always belong to a country and city and to some category (like monument, museum, etc.), then clearly in the associated table we will have foreign keys. In particular, the CategoryId attribute will be a foreign key to the Category table.

The GeneXus developer will not have to program in the transaction the referential integrity control for each foreign key because it will already be automatically included in the generated program. It's part of the **transaction's** logic.

Travel Agency

Attraction

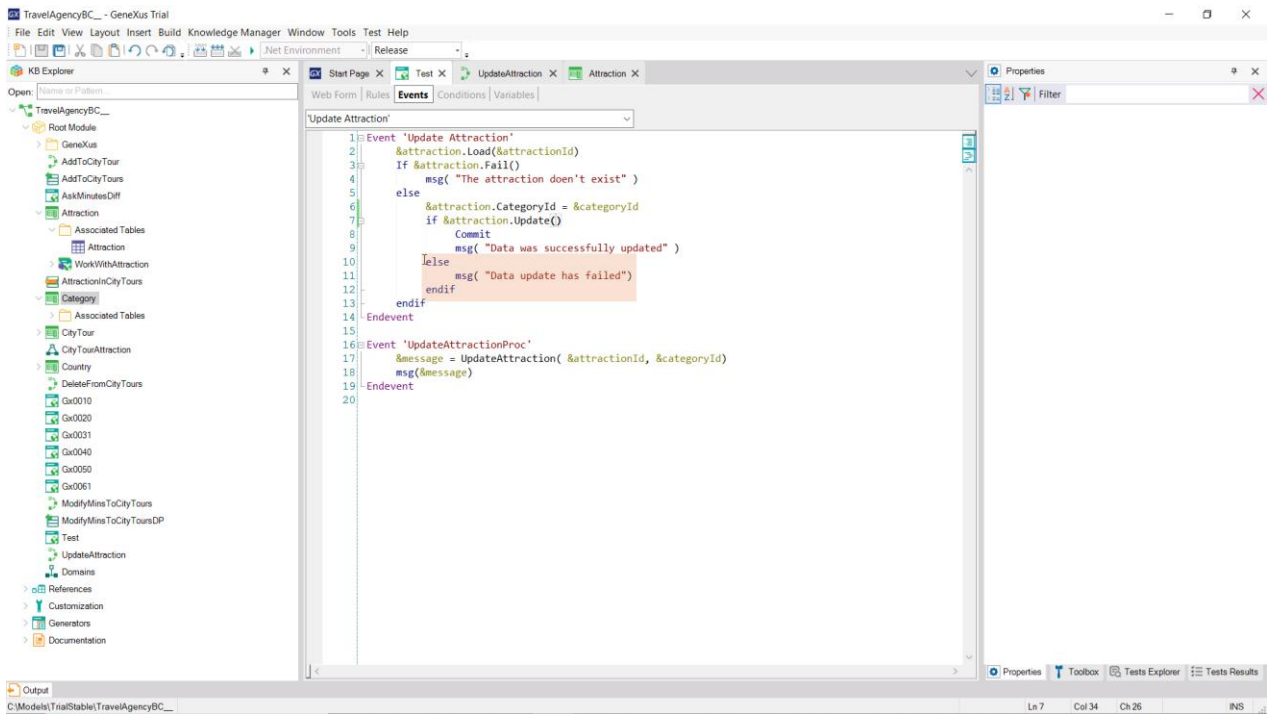
Id	2
Name	<input type="text" value="The Great Wall"/>
Country Id	<input type="text" value="3"/> 
Country Name	China
City Id	<input type="text" value="1"/> 
City Name	Beijing
Category Id	<input type="text" value="12"/>  ● No matching 'Category'.
Category Name	

So, when the user wants to assign an existing attraction to a non-existent category, the transaction informs us that this is not possible. It's making a referential integrity control, before the database itself does it. 0

The image shows a web interface titled "Travel Agency". It features two input fields: "Attraction id" and "Category id". The "Attraction id" field is a text box with a small icon on the right. The "Category id" field is a dropdown menu with a blue background and a small icon on the right. Below these fields are two buttons: "Update Attraction BC" on the left and "Update Attraction PROC" on the right. A mouse cursor is visible in the center of the page.

Let's suppose that we implemented a web panel that allows the user to enter an attraction ID and a category ID, to change the category of the attraction to this other one.

There are two options to do it through code: one is through the Attraction Business Component, and the other will be through a procedure.

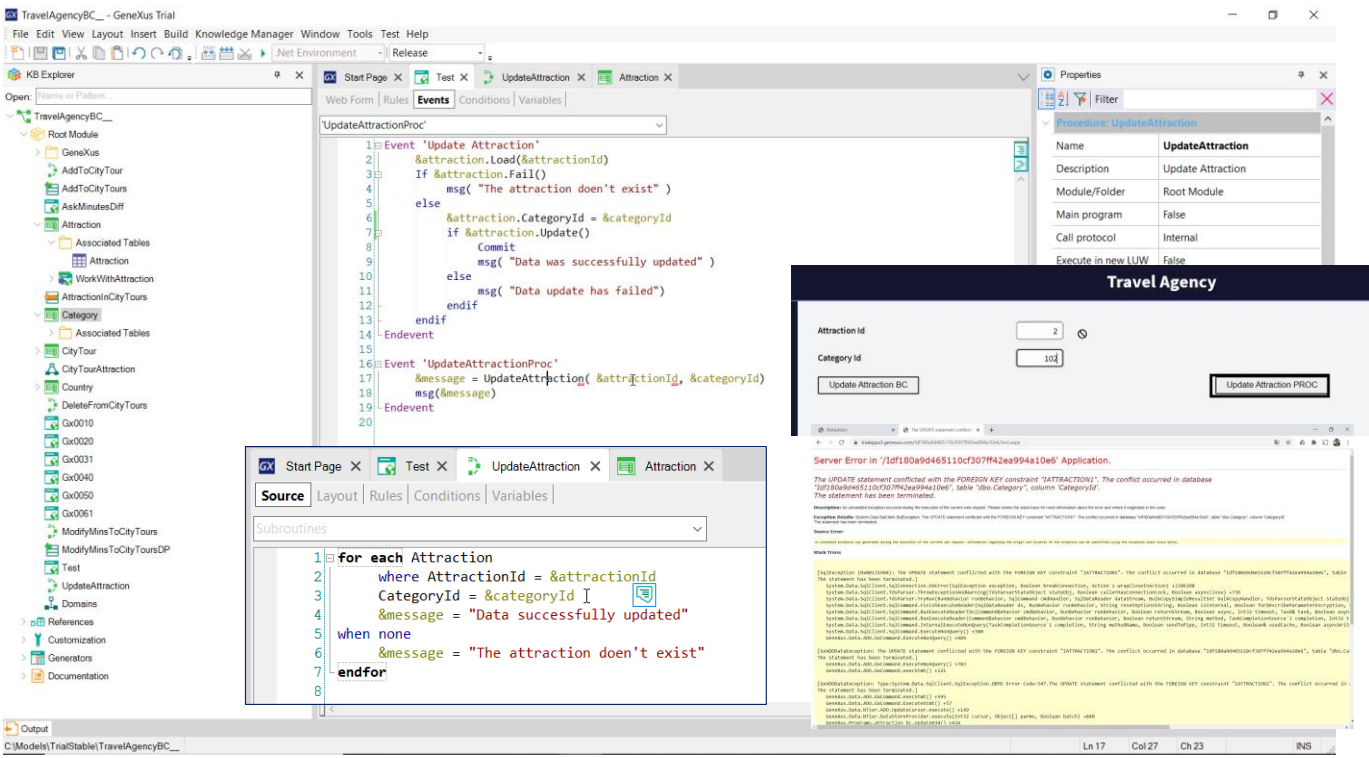


Let's see how we would implement the first one.

With the Business Component property activated, we define a variable of this type, and in the event associated with the button we load it from the &AttractionId variable that the user enters through the screen.

If it exists, that is, the Load does not fail, then we change the category for the one entered by the user, and try to update it. This will work just like the transaction, since the code associated with that Update method will check for the existence of that category and only then send the update order to the database. And since that code will find that there is no such category, it will not try to update the database; it will return False and the "else" will be executed.

Let's try it at runtime.



On the other hand, if we want to do the same update through a procedure, as we see here, we send the attraction and category identifier to this procedure, and in it we search for the attraction and the category is directly updated.

When we run it with a non-existent category... ouch! A database exception will occur and the user will have to deal with this unfriendly screen. Here, the program is not performing the necessary referential integrity controls. But the database is making them. This the reason for this error.

We could capture the error to make something more friendly so that the program doesn't cancel abruptly. To this end, the Error_handler rule is used. As a parameter we must enter the name of a subroutine that we must define in the object, and there we will program its behavior in case of a database error, asking first for that error. You can see details about this in our wiki.



The screenshot displays the GeneXus IDE interface. On the left, the 'Preferences' window shows a tree view of the project 'TravelAgency'. Under 'Data Stores', the 'Default (SQL Server)' data store is selected. On the right, the 'Properties' window shows the configuration for 'DataStore: SQL Server'. The 'Declare referential integrity' property is set to 'Yes'.

DataStore: SQL Server	
Type	DataStore
Description	SQL Server
Access technology settings	
Creation/Reorganization information	
Database schema	
Primary key definition	Primary key
Declare referential integrity	Yes
Default tables storage area	
Default indices storage area	
Default temporary storage area	
Generate COMMENT ON statements	No
Database information	

We can also remove the referential integrity statements from the database so that it doesn't make the controls. Although it is not recommended, we have this property at the Data Store level in case we need it.



If select CategoryId from Category where CategoryId =102 ...



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	102
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	2

Name	Type	Description	Formula	Nullable
Attraction	Attraction	Attraction		
AttractionId	Id	Attraction Id		No
AttractionName	Name	Attraction Name		No
AttractionAddress	Address, GeneXus	Attraction Address		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		No
CityId	Id	City Id		No
CityName	Name	City Name		No
CategoryId	Id	Category Id		Yes
CategoryName	Name	Category Name		No

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	null
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	2




By default, then, if we don't change that property, the referential integrity may or may not be controlled programmatically, but it will always be controlled by the database.

However, we had seen a case where a non-existent foreign key value was allowed. It's just that strictly speaking, it wasn't a value. That's when we allowed the foreign key to be null. How did we tell the database that we would allow this for a foreign key?

In the transaction structure we had the Nullable column, which was set to No by default, but could be changed to Yes.

Travel Agency

Attraction

Id	2
Name	<input type="text" value="The Great Wall"/>
Country Id	<input type="text" value="3"/> 
Country Name	China
City Id	<input type="text" value="1"/> 
City Name	Beijing
Category Id	<input type="text" value="0"/> 
Category Name	

This meant that when the transaction was executed, and the category identifier was left empty, the program didn't make any referential integrity controls, and neither did the database.

empty ≠ null

Numeric
CategoryId 0

Numeric
CategoryId null

The empty value and the null value are two very different things.

The empty value is a value. Depending on the data type, it will be one of the possible values of that type. Thus, if the type is numeric, the empty value is zero. Databases have empty values specified according to each data type. But, strictly speaking, the null value is not a value. In fact, **it's** a NON-VALUE. Saying that CategoryId is Null is the same as saying that its value is not specified and is not known. **It's** not the same as saying it is empty.

empty ≠ null



For AttractionId = 2 → CategoryId = 0



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	2



For AttractionId = 2 → CategoryId = null



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	null
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	2



Therefore, empty and null values are two very different things in databases. If we are implementing a program and send the order to change the category of attraction 2 to 0 in the database, what we will get is none other than the screen with the database exception for referential integrity failure. On the other hand, if we change it to null, there will be no problem, just like when we did it through the transaction.

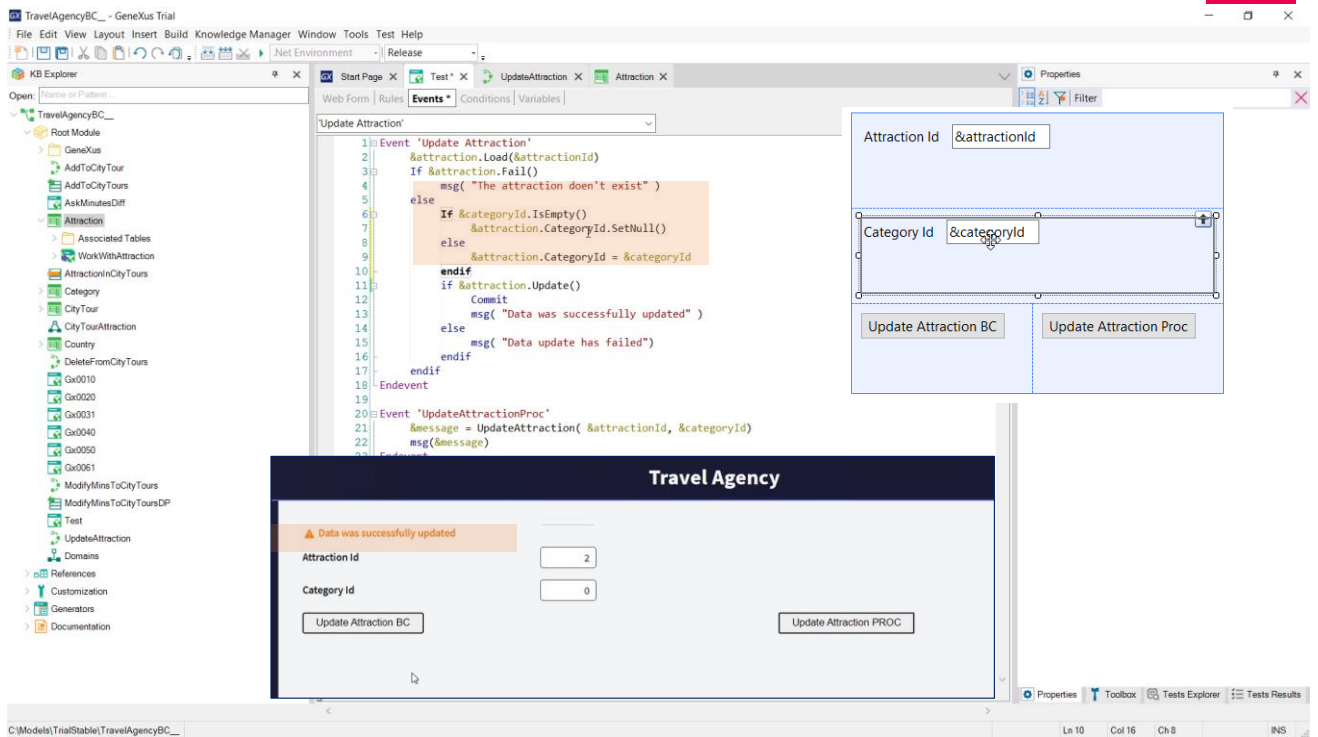
So, the question is: Why is the transaction asking the database to change the Null and instead the business component is sending the empty value, zero?

The screenshot shows the GeneXus IDE interface. On the left, a form is being designed with fields for Attraction, Country, City, and Category. The 'Category' field is highlighted. In the center, a table lists the attributes of the 'Attraction' entity, with 'CategoryId' highlighted. On the right, the Properties window for 'CategoryId' is open, showing various configuration options. The 'Nulls in Forms' property is set to 'Empty as Null'.

name	Type	Description	Formula	Nullable
Attraction	Attraction	Attraction		No
AttractionId	Id	Attraction Id		No
AttractionName	Name	Attraction Name		No
AttractionAddress	Address, GeneXus	Attraction Address		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		No
CityId	Id	City Id		No
CityName	Name	City Name		No
CategoryId	Id	Category Id		Yes
CategoryName	Name	Category Name		No

Attribute: CategoryId	
Name	CategoryId
Description	Category Id
Title	Category Id
Column title	Category Id
Contextual Title	Id
Formula	
Nulls in Forms	Empty as Null
Class	Attribute
Qualified Name	CategoryId
Type Definition	
Supertype	
Based on	Id
Data Type	Numeric
Length	4
Decimals	0
Signed	False
Autonumber	True
Autonumber start	1
Autonumber step	1
Autonumber for rep	True
Initial value	
Validation	
Value range	
Validation failed M	

In the properties of the foreign key attribute, CategoryId, we see one called “Nulls in Forms” that is set to “Empty as Null.” This means that if the attribute accepts nulls, as in this case, every time the value is left empty in the form, it will be considered to be Null when sent to the database and not empty.



But this doesn't apply when the update is made through the Business Component, which doesn't have a form.

Therefore, if the `&CategoryId` variable is empty, this element will be left with an empty and not null value, and this is why the referential integrity fails when trying to update. We have to specify the null value if the variable is empty.

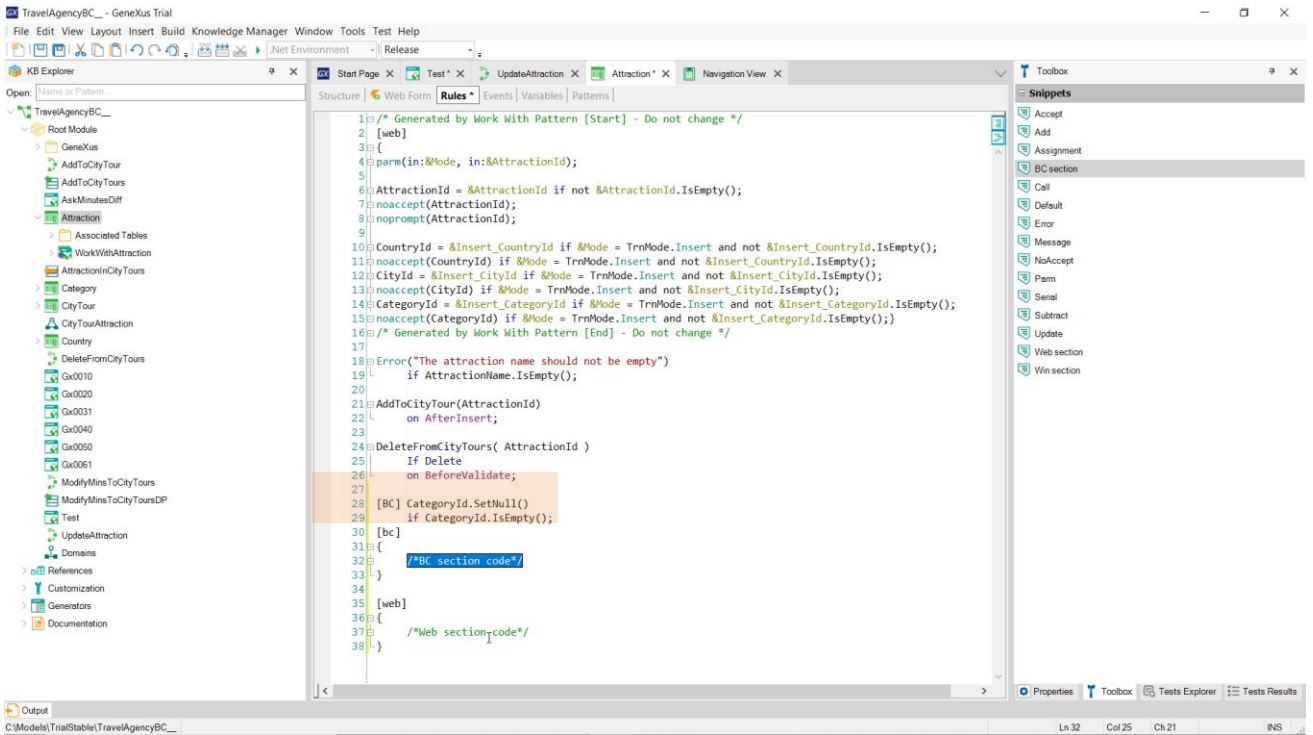
There are two ways to do so: Either we do it locally, only here, or we do the same mapping that gets the attribute property, but this time also valid in the Business Component, wherever it is used.

For the first thing, you only have to specify that if the `CategoryId` variable is empty, the Business Component element will be set to null. The `SetNull` method applies to attributes that accept null values and their corresponding ones in the Business Component variables, as in this case.

If we try it now, we see that we can do it.

The drawback of this alternative is that if we want to leave this foreign key null through a Business Component variable anywhere else, we will have to do the same.

As we were saying, the other, more general option is to incorporate this `SetNull` as a rule in the transaction. Here we remove it then...

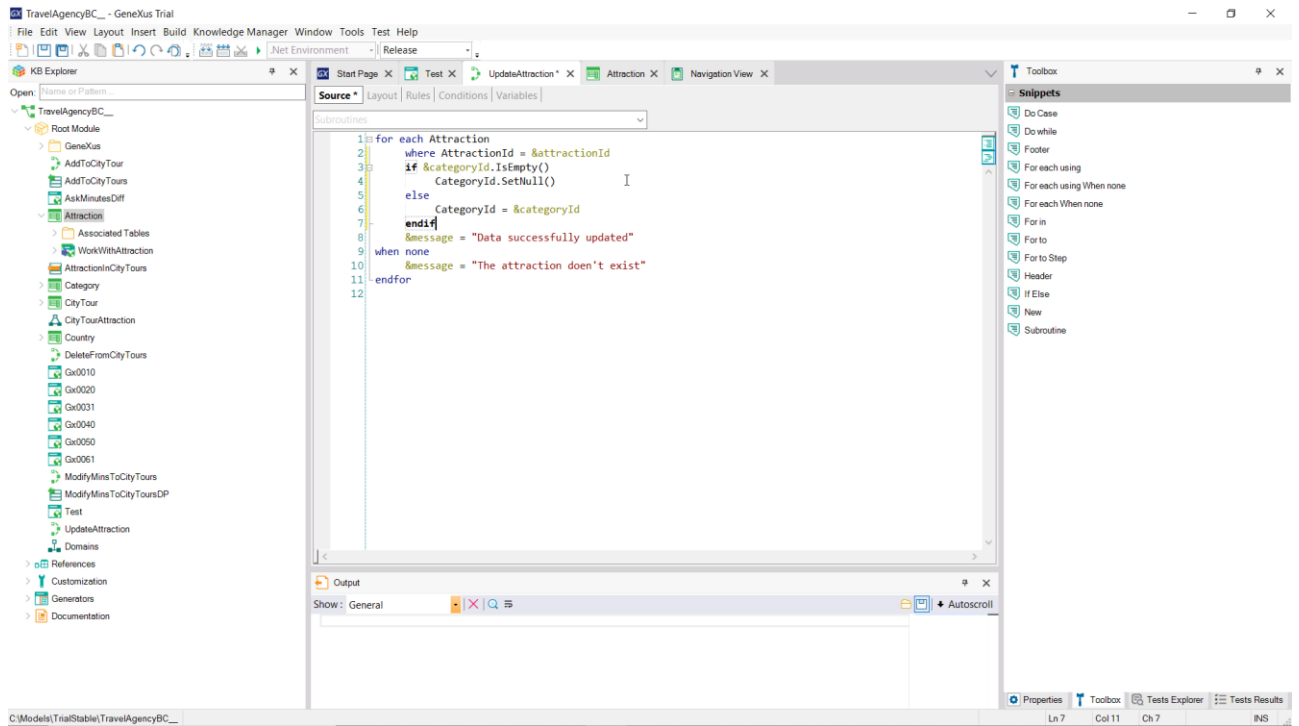


And go to the transaction to specify that the attribute is set to null when its value is left empty. This is already done by default when using the transaction screen, thanks to this property; so, strictly speaking, we can ask that this rule is only executed for the Business Component.

We can always indicate that a rule is only executed when it is in the Web transaction, or when it is in the Business Component, by placing this mark.

Actually, if we want to write a set of rules that only apply to the Web transaction, or that only apply to the Business Component, we do it in this way.

Let's try it. We see that it is indeed another solution.



The solution here can only be the local one. Let's try it...

And we see that it works.

*GeneXus*TM

training.genexus.com
wiki.genexus.com