Database Update

Using Business Components

GeneXus™

Business Component

&attraction

AttractionId	7
AttractionName	Forbidden city
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	2
CategoryName	Monument
AttractionPhoto	
AttractionAddress	

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	Christ the Redemmer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	Forbidden city	3	1	2	

+

LOGIC

&attraction.Insert()
&attraction.Update()
&attraction.Delete()

Business Component	**True**

In the previous video, we were given a detailed overview of familiar concepts to pave the way for understanding and using business components.

From the transaction structure with its logic (and by logic we mean controls for duplicates –not only primary key, but also candidate keys–, referential integrity, most of its rules and some of its events), a kind of data type similar to an SDT, but much more powerful, is obtained.

Then, it will be enough to define in almost any program a variable of that data type and manipulate it, which is what we will see next.

This variable, in its structure, will be handled similarly to an SDT. But it will also offer methods for doing something specific to a transaction: loading from the database, inserting, modifying, deleting; all this by executing the logic of the transaction, and then obtaining the generated messages and the results of success or failure.
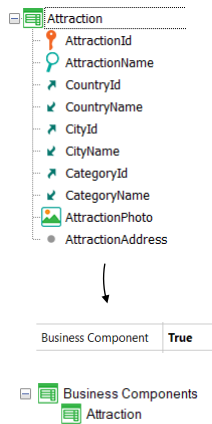
The way to obtain that kind of data in the knowledge base is simply by turning on the Business Component property of the transaction.
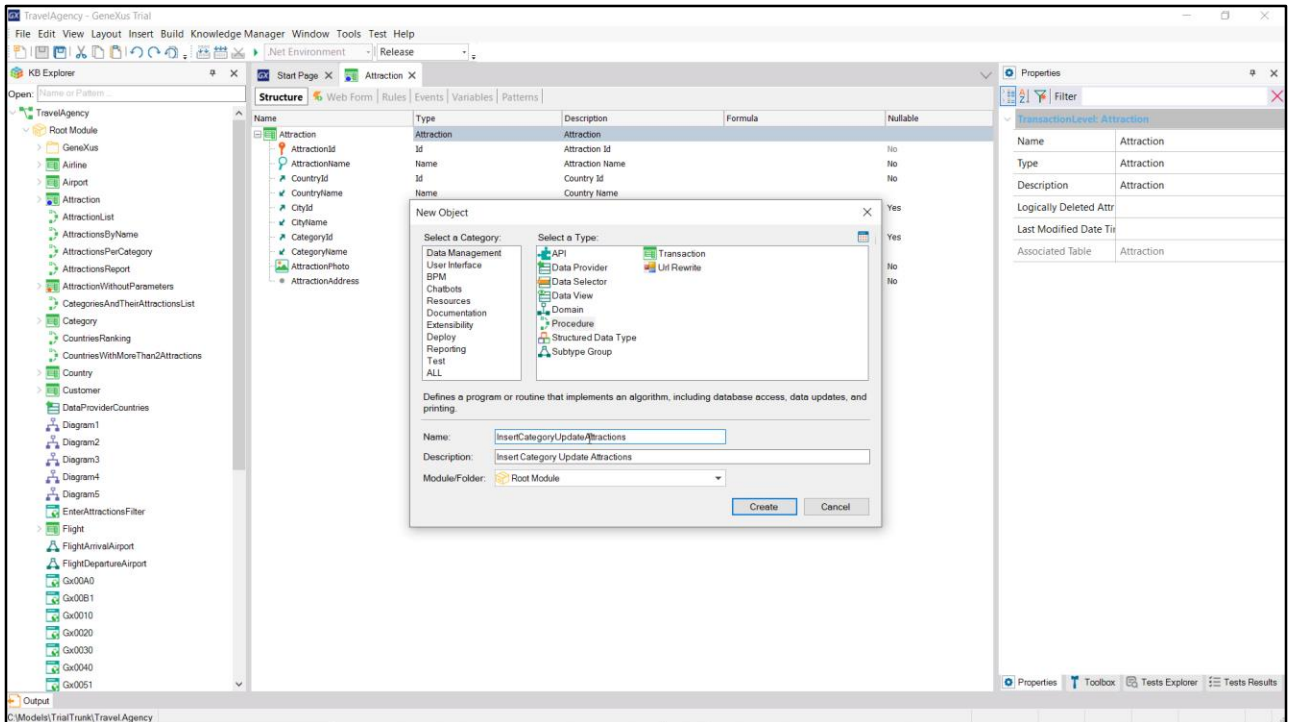
We will start by creating a simple procedure that we will modify to implement a more complete requirement later. Let's not pay attention to the name right now.

We'll start with something very simple: suppose we want to insert a new tourist attraction.

Insert through BC

The first thing we must do is create the Business Component of the Attraction transaction, so that it can be used in any object in the knowledge base. Therefore, we go to the transaction and among its properties find the one called Business Component. It is set to False by default. We change it to True. At first, we don't see any effect. However, if we save... we see this in the navigation listing.

Insert through BC
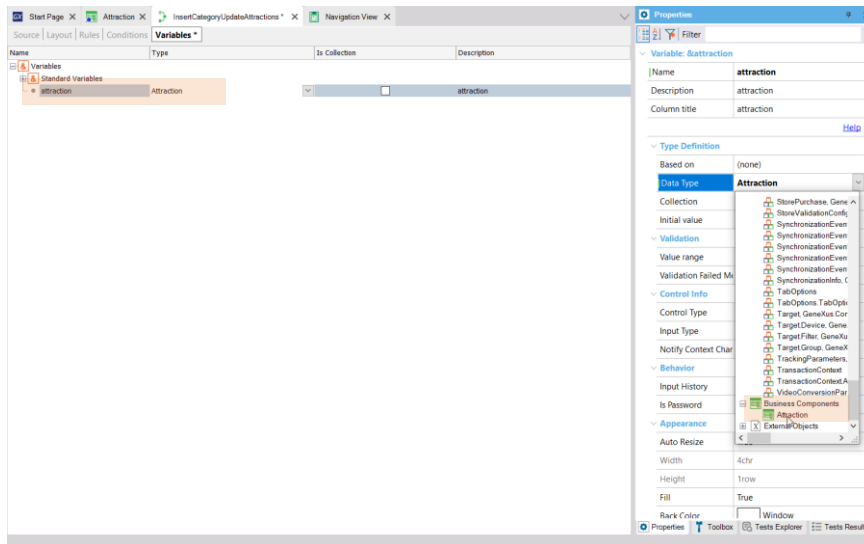
If now we go to the procedure and define in it a variable named Attraction, we see that because we selected a name matching that of the transaction, a data type of that name has been automatically used. What data type is that?

If we select the variable properties and open the combo... there is a Business Component group that for now only offers one value: not coincidentally, it is Attraction. Here's where we see the effect of having enabled the transaction property. The data type Business Component Attraction was created in the KB; it is available and can now be assigned to any variable.
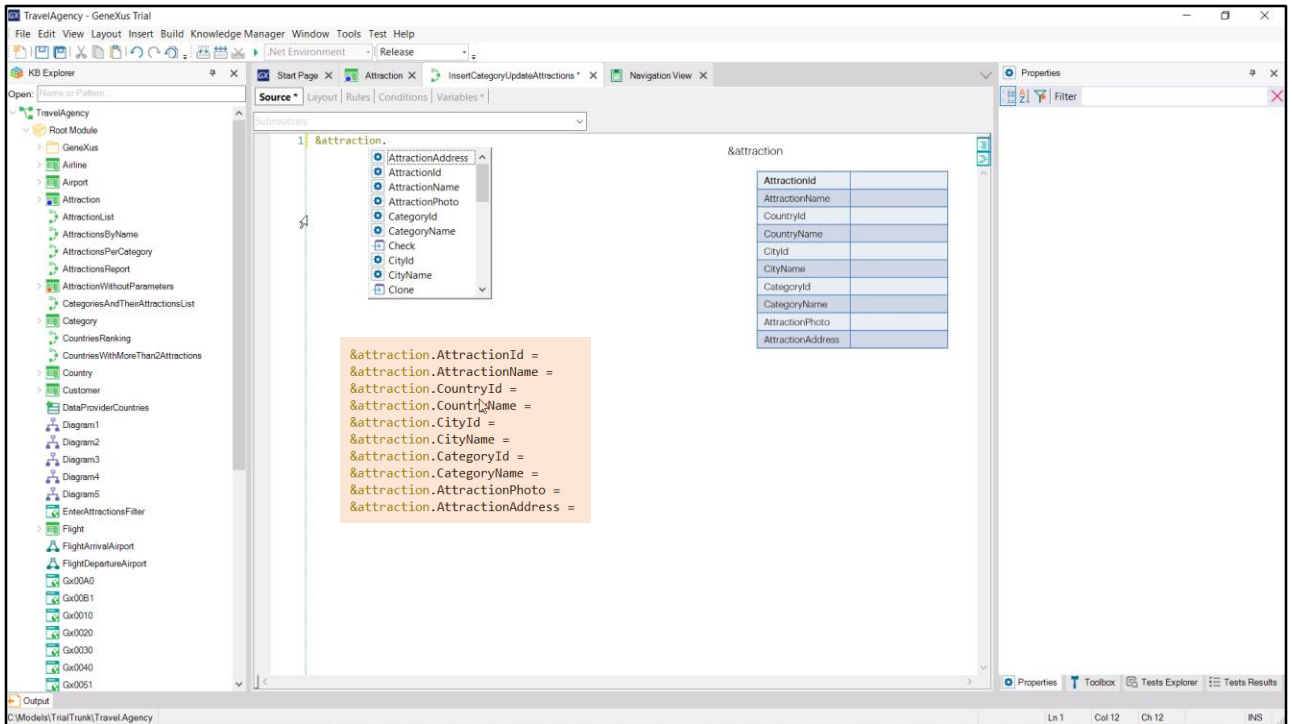
Having defined this variable in the procedure, memory space is automatically reserved to load all its elements as an SDT.

So, if we insert the variable and type a period, in this window we see the names of all the transaction's attributes, among other things, to use them in the structure, for example, by assigning them a value. In this way, we could...

To insert the Forbidden City tourist attraction, as we saw before when we used the transaction, we will have to complete the data.

We don't need to assign a value to the identifier, because it will be auto-numbered.

The name of the attraction will be... we know it's in China, it has the identifier 3, but what if we're wrong? We'd better look for that identifier with the Find formula, because one thing we are sure of is the name of the country, China.

That name is an inferred attribute in the transaction, so we don't have to assign a value to it here to insert the attraction.

We know the city is Beijing, so we find its identifier and assign it to the business component element and also remove the inferred element.

We assign a value for the category ID, looking for the one named "Monument."

And we will not assign any value to the photo and address.

So, the variable is loaded with the data of the attraction we want to insert in the table. All we have to do is insert it. To do so, we have the Insert method of the Business component variable, which, as we can see, will return a Boolean value: True if it could insert it; False otherwise. We can just invoke it and it will try to insert a new record exactly as the transaction does, that is, executing all the rules and validations. If it succeeds, then

the variable will be loaded with the ID given by the database and all of its own and inferred attributes.
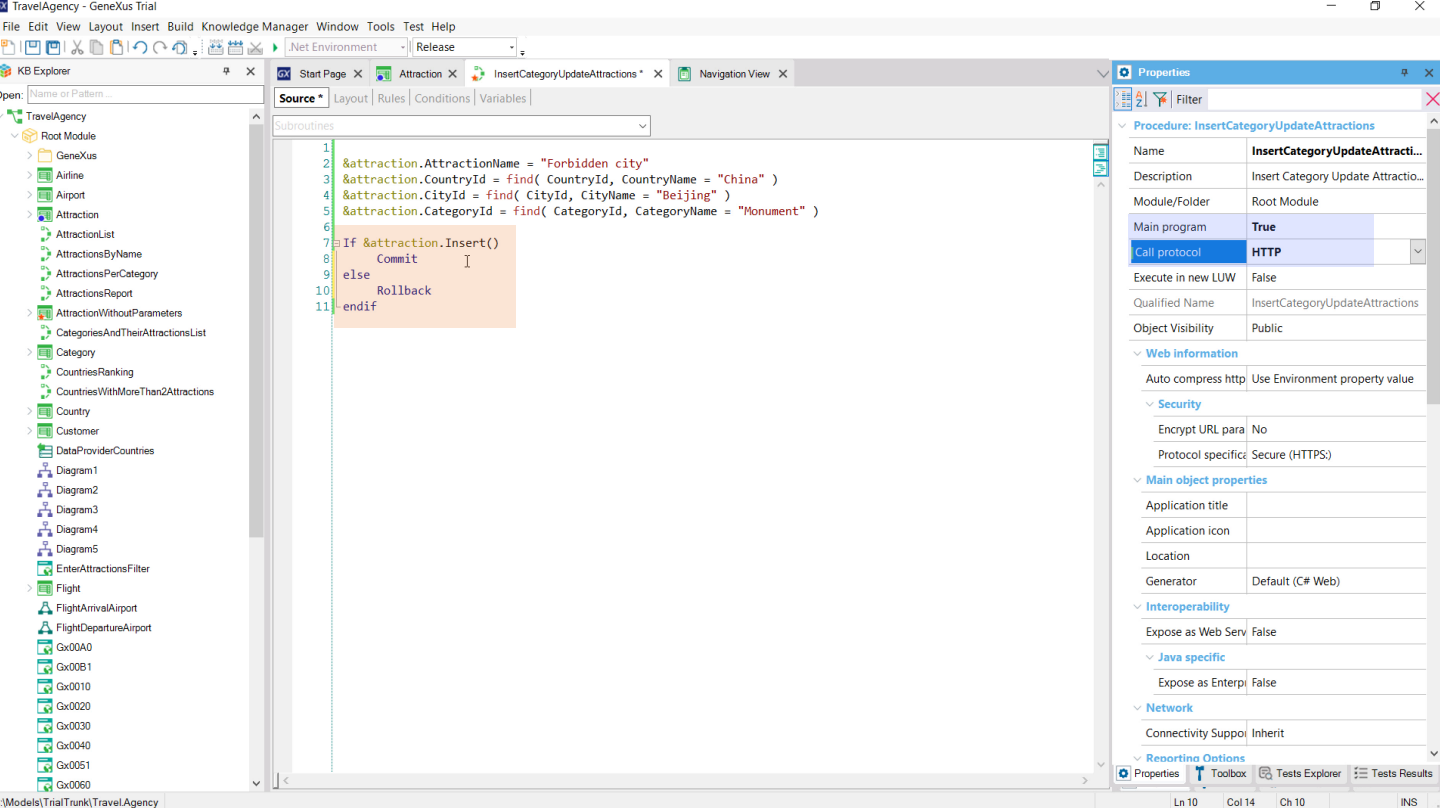
However, we should be aware that although the record is already in the table, it is in an unstable state. If there is a power outage or system crash for any other reason, that record will be gone when the database is reestablished.
This is because databases allow us to insert, edit and delete records as if at a logical level; whenever it seems convenient, we must indicate that all those operations we have been doing can be accepted. This is done with the Commit command. Therefore, if the insertion was successful, we commit. This is the action of making a commit on the database; that is, indicating that these operations should be fixed –made permanent– in the database. When a Commit is executed, all operations performed between the previous and the current commit will be accepted.

Just as we have the Commit operation to accept all those actions, the opposite one, called Rollback, allows us to undo everything that may have been done after the last Commit. Here we could make a Rollback if the operation was not successful... but it will not make much sense, because if the operation was not successful, we can assume that the record was not even inserted, so there will be nothing to undo.

To quickly run this procedure from the Developer Menu, we set it as main object and with HTTP invocation protocol.
We press F5.

Application Name

Attractions   INSERT   🔍 Name

Id	Name	Country Name	Category Name	Photo	City Name	Address		
1	Louvre Museum	France	Museum		Paris		UPDATE	DELETE
2	The Great Wall	China	Monument		Beijing		UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris		UPDATE	DELETE
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro		UPDATE	DELETE
5	Smithsonian Institute	United State	Museum		Washington		UPDATE	DELETE
6	Matisse Museum	France	Museum		Nice		UPDATE	DELETE
8	Forbidden city	China	Monument		Beijing		UPDATE	DELETE

If we open the Work With Attractions element, we will see that we already had attraction 7 for the Forbidden City, which we had inserted earlier through the transaction. Let's delete it (the identifier with value 7 will be lost; it will not be given again to the next attraction entered, which will be 8, as we will see now if we run the procedure).

Since it has no output, we don't see anything in the browser, but if we go back to Work With Attractions... it has actually inserted the Forbidden City.

Let's delete it again, to run the procedure again, but this time we'll add another attraction besides that one. For example, Notre Dame Cathedral.
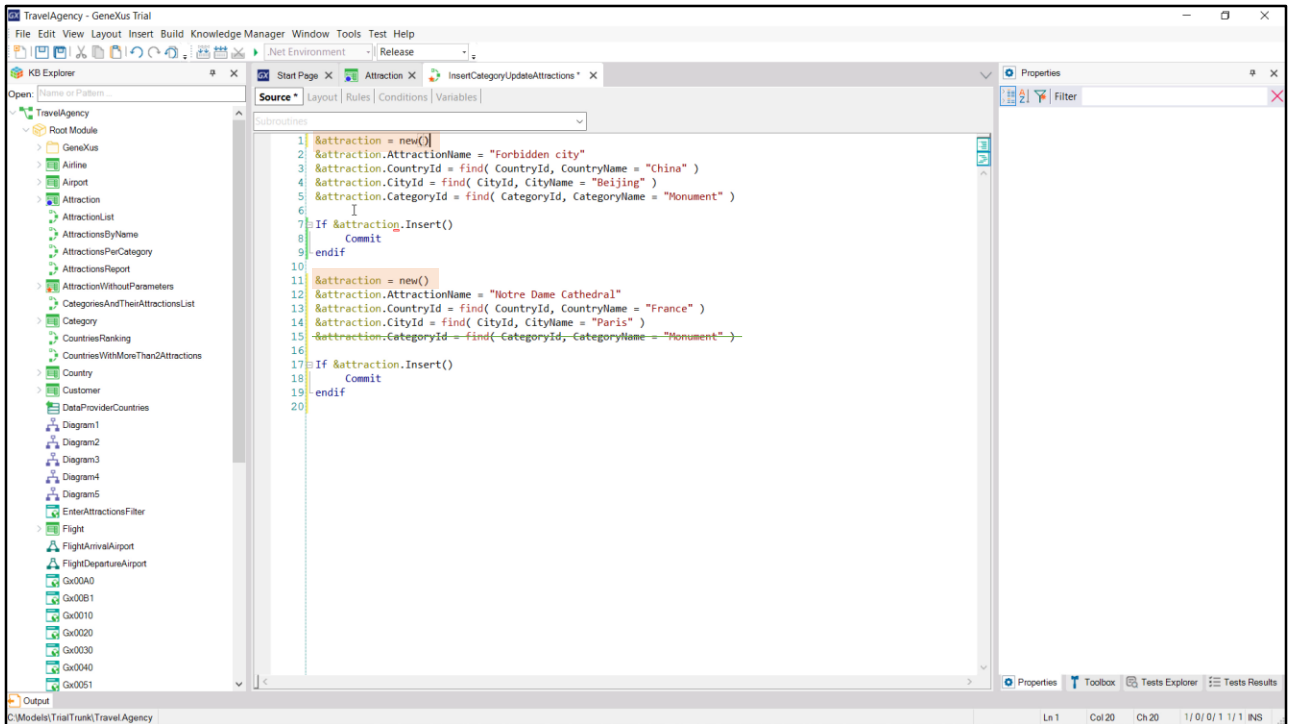
So, we should insert it here. We no longer need the information that the variable had. Actually, we need to at least clean it up, to make sure **nothing is left behind when we load the Cathedral's data. One way is to** ask for new memory for the variable, and it is done in this way...

OK. Now we have it insert the data.

Asking for new memory is important. If, for example, we had not assigned a value to the category identifier in order to leave it empty for the Notre Dame Cathedral, it would have been loaded with the value we assigned before. Therefore, it is always good practice to ask for memory before completing the data of a business component that we are going to insert. Even up here.

Finally, before running it, note that in this case we inserted the first attraction and we will commit if that insertion was successful. Let's do the same with the second one, and commit.

```
1    &attraction = new()
2    &attraction.AttractionName = "Forbidden city"
3    &attraction.CountryId = find( CountryId, CountryName = "China" )
4    &attraction.CityId = find( CityId, CityName = "Beijing" )
5    &attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )
6
7    If &attraction.Insert()
8        &attraction = new()
9        &attraction.AttractionName = "Notre Dame Cathedral"
10       &attraction.CountryId = find( CountryId, CountryName = "France" )
11       &attraction.CityId = find( CityId, CityName = "Paris" )
12       &attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )
13
14       If &attraction.Insert()
15           Commit
16       else
17           Rollback
18       endif
19   endif
20
```

But we might want to commit only once, after we've inserted both attractions. Or we could even try to insert the second attraction only if the first one was successful. For example, in that case, it would be like this.

Here we will only commit when the first and the second insertions were successful. It could happen that the first one was successful and the second one was not. Then the record of the first one will be saved in the database, but it will not be committed. So, maybe in that case, when the second one is not successful, we don't want the first record to remain in the database; and then we would use the Rollback command.

Let's run it. F5.

Application Name

Attractions

[INSERT]   🔍 Name

Id	Name	Country Name	Category Name	Photo	City Name	Address		
1	Louvre Museum	France	Museum		Paris		UPDATE	DELETE
2	✕ Great Wall	China	Monument		Beijing		UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris		UPDATE	DELETE
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro		UPDATE	DELETE
5	Smithsonian Institute	United State	Museum		Washington		UPDATE	DELETE
6	Matisse Museum	France	Museum		Nice		UPDATE	DELETE
9	Forbidden city	China	Monument		Beijing		UPDATE	DELETE
10	Notre Dame Cathedral	France	Monument		Paris		UPDATE	DELETE

Let's see the attractions, run the procedure, and go back to Work With Attractions. Now we have 9 and 10.

What if now we wanted to change the category of the Chinese Wall so that it is no longer a "Monument" and becomes a "Famous Landmark," as well as change its name by removing "The"?
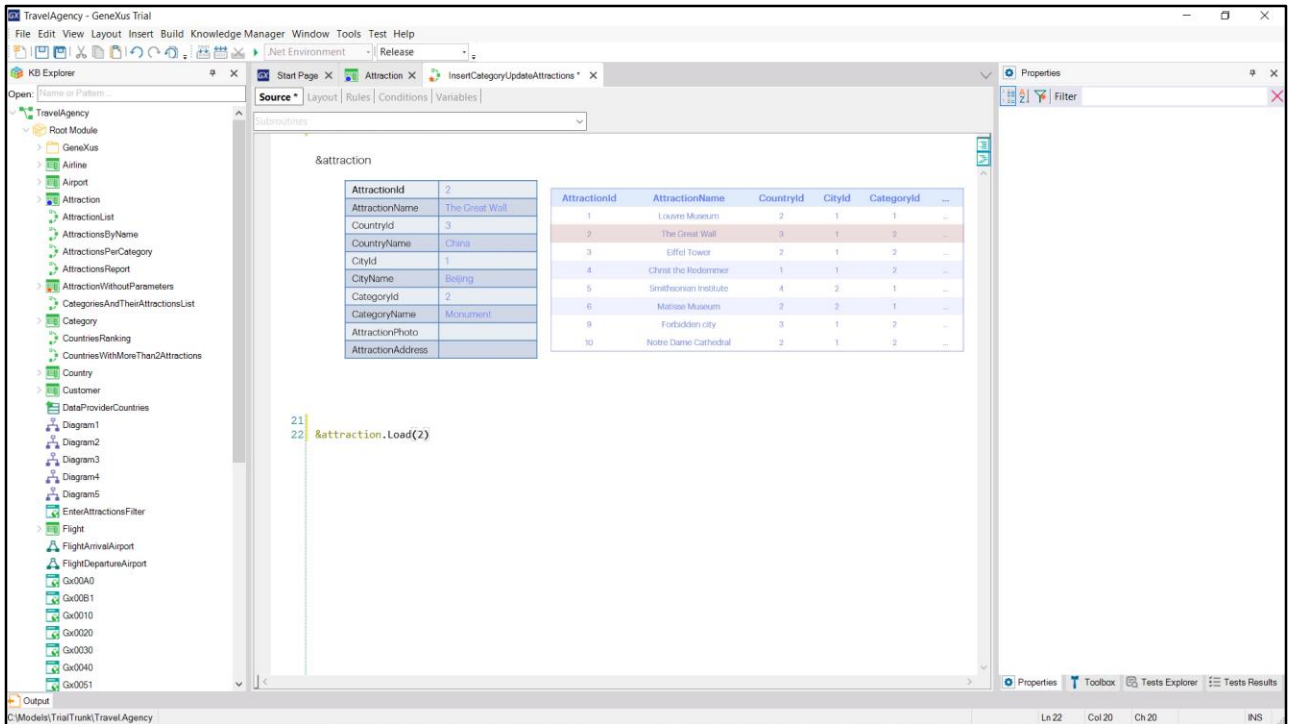
Let's leave the previous code with comments so that these two attractions
are not inserted again (if their identifiers weren't auto-numbered, the
insertions would fail because there is a duplicate key, but this is not the
case).

We need to make an Update, as we would do with the transaction. First of
all, we need to load the variable with the Attraction values we want to
change. We know its ID is 2. So we invoke the Load method of the variable
by passing it the key, 2, as a parameter.
When running this, it will go to the table to find the record 2, and if so, it
will load the variable BC with all values: its own, inferred ones and
formulas, as it happens when executing the transaction.
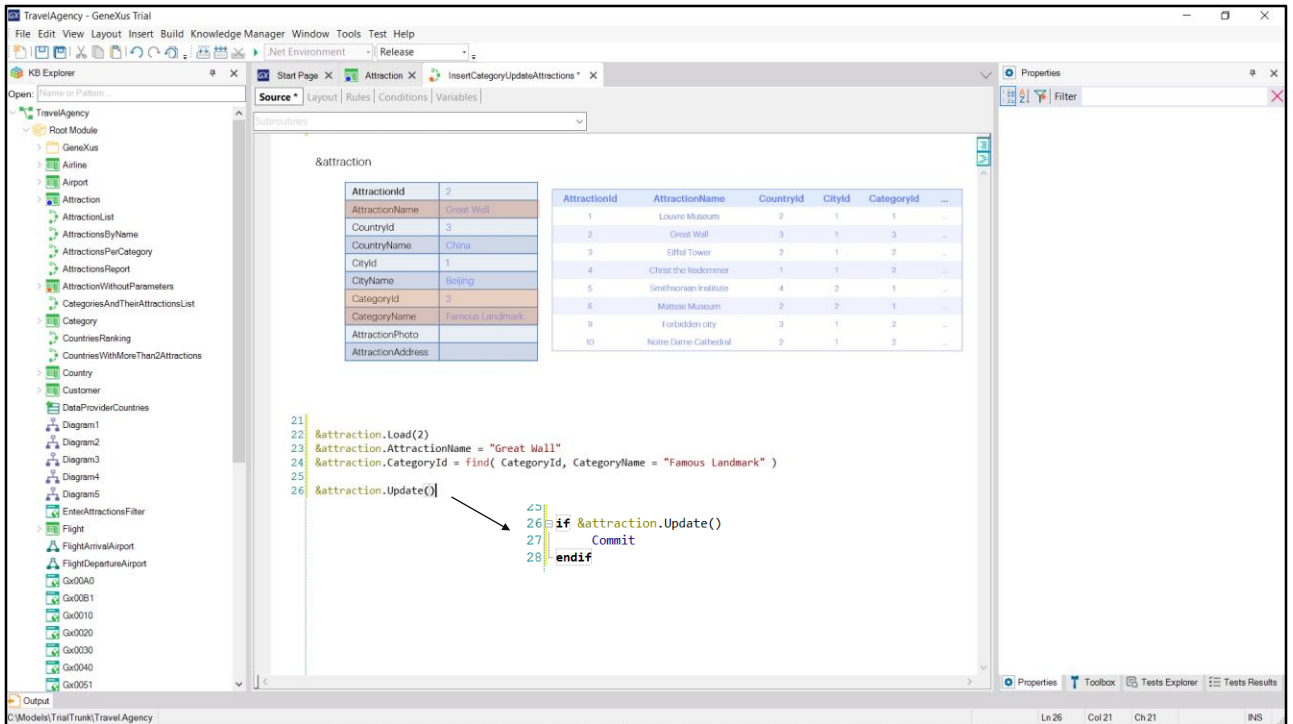
Of all these values, we need to change two: the name of the attraction...
And its category...

All we have to do is have the variable update this data in the table. To do so, we have the Update method.

When invoking it, exactly as in the transaction, all the rules, formulas, candidate key uniqueness and referential integrity controls will be run; if everything went well, the record will be updated in the table. And, of course, the variable will be loaded with the current data.

Again, we could ask for the result of the update to, for example, already perform a Commit.

Let's try it.

We run the procedure, and check the attractions. Note it's been modified.
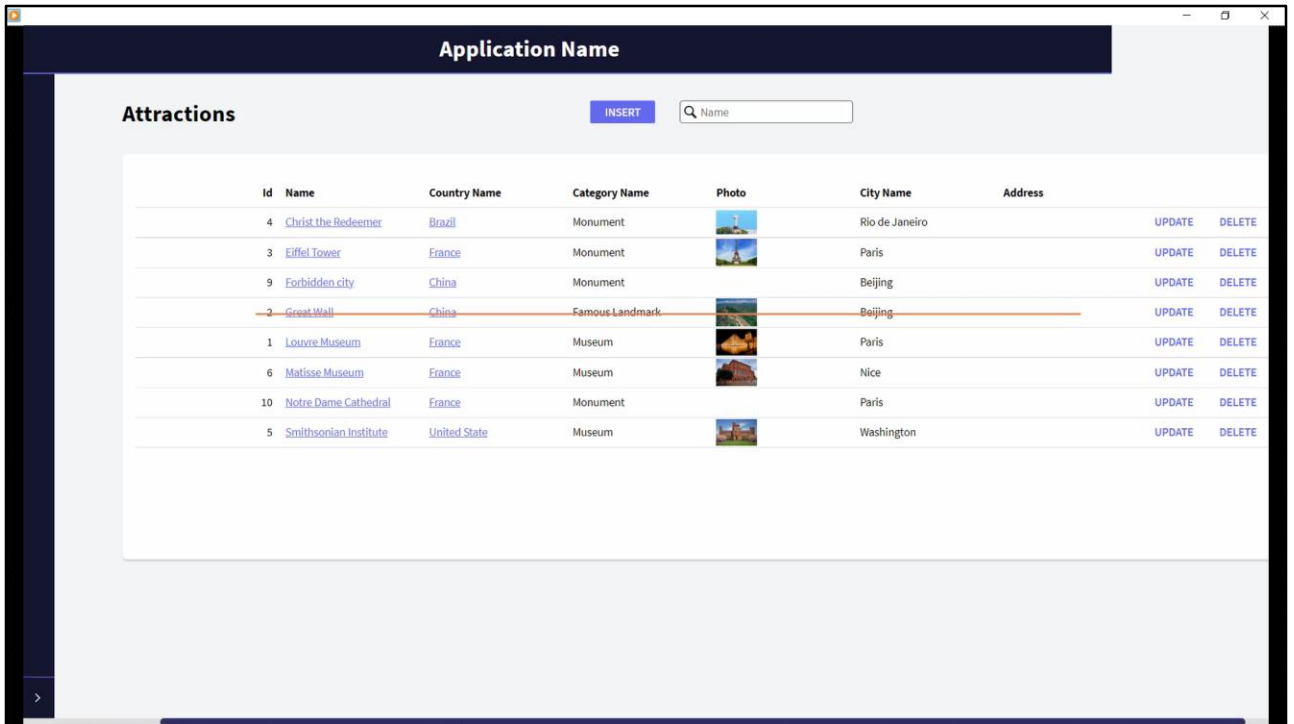
To complete the operations, what if now we wanted to delete an attraction by code?

For example, delete the Great Wall.
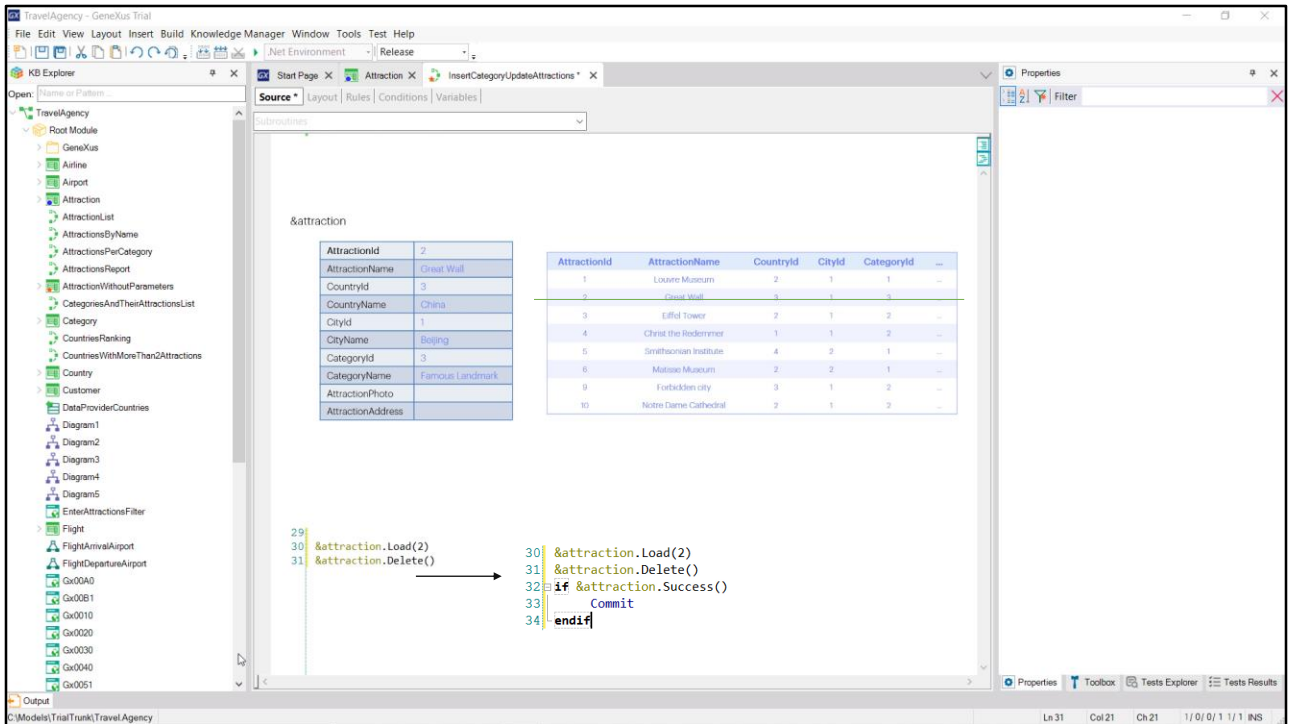
As we do in the transaction, we first have to load attraction 2 in the variable structure, and then simply give the order to delete.

This is done with the Delete command. We cannot repeat it enough: when this method is executed, all the transaction logic will be executed in Delete mode, including the referential integrity controls. Therefore, if there were related information, such as city tours featuring this attraction that we want to delete, the referential integrity control performed by the transaction –and by the Business Component– will prevent it. An error will occur and the record will not be deleted from the table.
However, if there are no errors the record will be deleted. The variable will be loaded with the data anyway, in case we want to do something with them.

The Delete method does not return the result of the operation, so to find out whether it was successful, we will have to query it explicitly using the Success method (the opposite is the Fail method). And, for example, Commit the deletion there, i.e., permanently delete the record. This method can be used after any operation, for example, after Load, to know if it found the requested record in the table.

Let's try what we've seen: We run the procedure and note that record 2

has indeed been deleted.
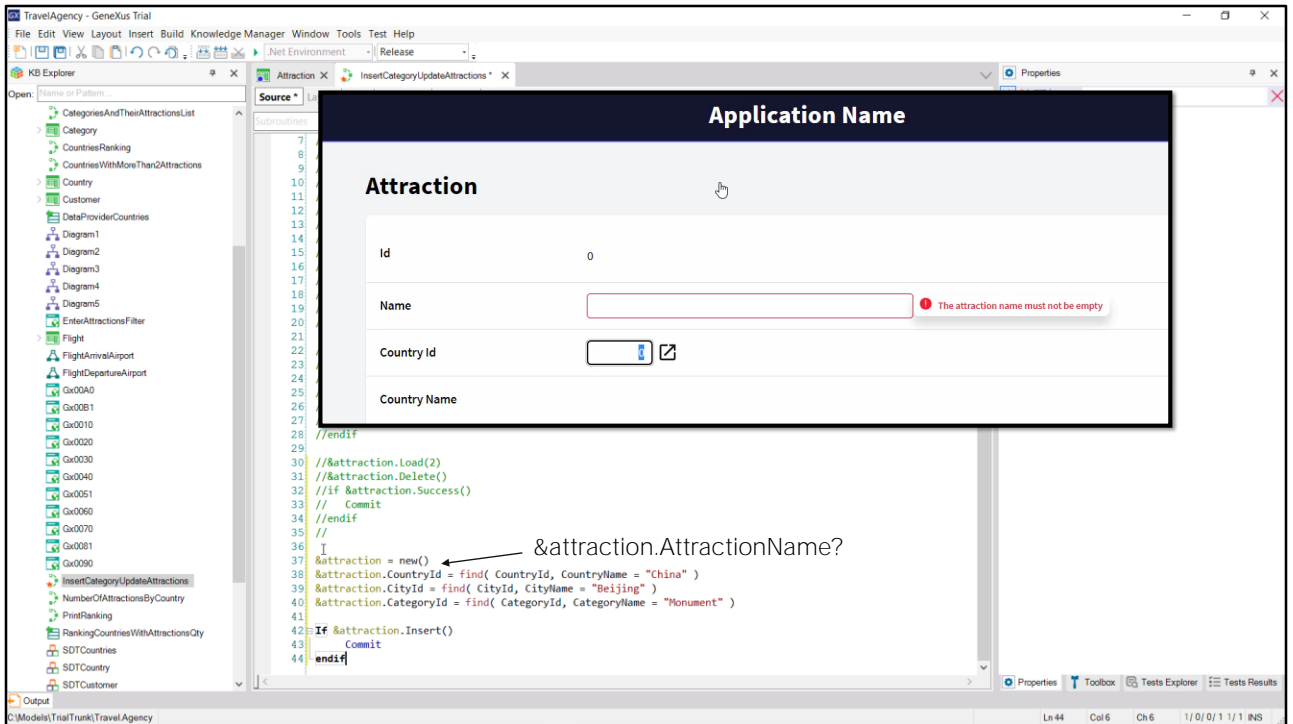
And what happens if we run the procedure again? Nothing happens. Why?

In this second run, attraction 2 no longer exists. Therefore, it has not been loaded or removed.

What if we now try to re-insert the Chinese Wall, but forget to enter its name? Let's try it.

We run the procedure. We open Work With Attraction, and don't see the new attraction inserted. Of course, we forgot to enter its name and the error rule we had declared is being triggered, preventing that record from being added to the table!

If we had done this through the transaction, the user would have received the message we specified in the error rule.

Where was that message when we tried to insert it through the business component?
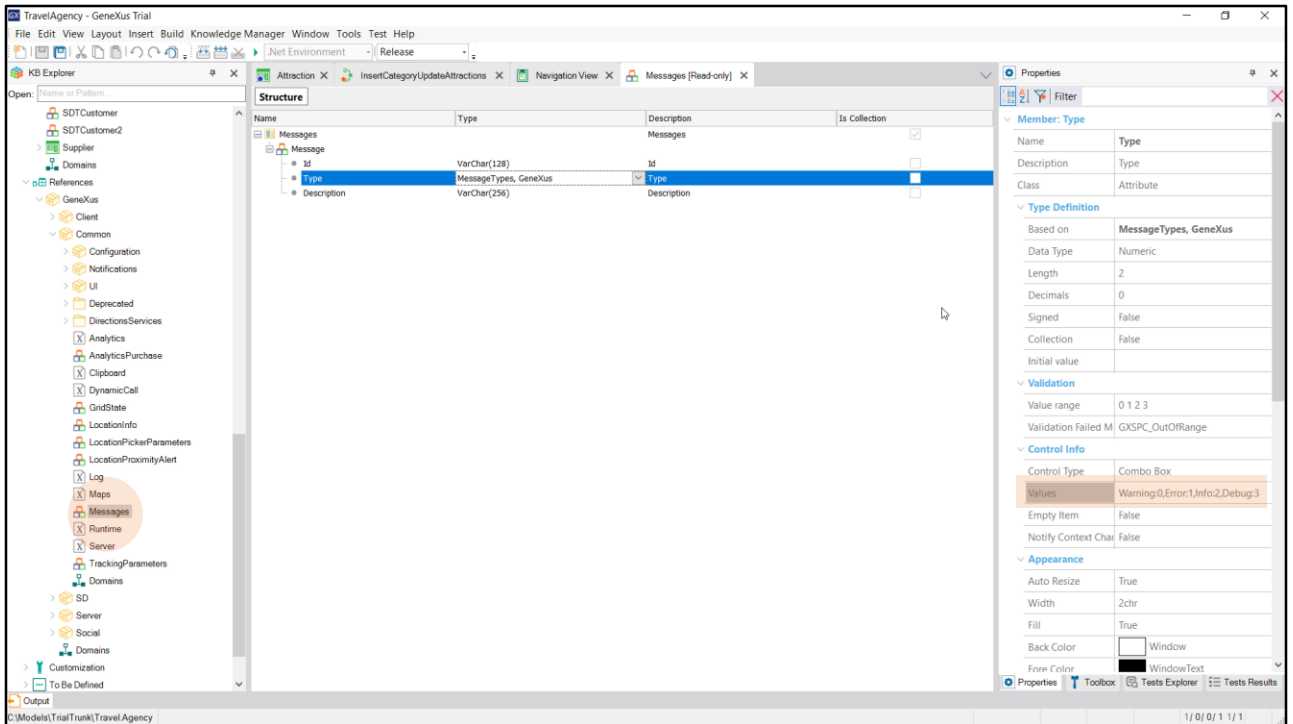
The transaction has a screen to show errors to the user, but the business component does not. Consequently, error handling is also left to the developer.

How do we obtain them?

The KB Explorer has a group of References, which we will talk about another time, but it will always contain a GeneXus module, with functionalities already implemented to be used in our KB. In particular, there we will find an SDT called Messages. It is a collection of items consisting of an ID, a type, and a description. The type is also predefined in the module. It is numbered and indicates the type of message in each case: warning, error, information, debug.

Error handling

```
&attraction = new()
&attraction.CountryId = find( CountryId, CountryName = "China" )
&attraction.CityId = find( CityId, CityName = "Beijing" )
&attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )

If &attraction.Insert()
    Commit
else
    &messages = &attraction.GetMessages()
    for &message in &messages
        print MessagePB
    endfor
endif
```

Values	Warning:0,Error:1,Info:2,Debug:3
Empty Item	False

Source * **Layout *** Rules | Conditions | Variables *

MessagePB

&message.Id ———————————————→ AttractionNameIsEmpty

& ———————————————→ 1

&message.Description ———————————————→ The attraction name must not be empty

```
Error( "The attraction name must not be empty", AttractionNameIsEmpty )
    if AttractionName.IsEmpty();
```

Every time an operation is executed on a Business Component variable, you can obtain the collection of messages generated by that operation with the GetMessages method. We will have to define a variable of the data type returned by that method in order to manipulate its result.

For example, just to be practical, let's show the result in a PDF file. We will run through the collection of messages generated with the for in command that allows running through collections, among other things. For this we define a variable of the data type of the message collection items. And we print every message of that collection in the output.

Let's enter the outputfile rule, and try it.

We run the procedure, and here is the output...

Only one item was obtained in the message collection, with an empty ID, of type 1 –which is an error. Its description is exactly what we entered in the error rule of the transaction.

If we look at the syntax of the error rule, we see that it allows a second parameter, which is optional. This parameter will allow defining the ID of the error for the business component's message. For example, we will

assign it this value. If we try now... there we see it.

Error handling

```
Msg( "The attraction name must not be empty", AttractionNameIsEmpty )
    if AttractionName.IsEmpty();


37  &attraction = new()
38  &attraction.CountryId = 22
39  //&attraction.CountryId = find( CountryId, CountryName = "China" )
40  &attraction.CityId = find( CityId, CityName = "Beijing" )
41  &attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )
42
43  If &attraction.Insert()
44      Commit
45  else
46      &messages = &attraction.GetMessages()
47      for &message in &messages
48          print MessagePB
49      endfor
50  endif
```

AttractionNameIsEmpty

0

The attraction name must not be empty

ForeignKeyNotFound

1

No matching 'Country'.

ForeignKeyNotFound

1

No matching 'CountryCity'.

Let's do one last test: let's modify the rule so that it is no longer an error, but a message.
Also, let's enter a non-existent country identifier as CountryId to see how **the insertion fails due to the referential integrity control. Let's try it.**

When trying to insert, three messages were displayed: the first one is a Warning message, and it would not have caused the insertion failure by itself. However, the second and third ones could have caused the failure because they are of the Error type. The internal ID is this, and the message displayed to the transaction user when the country integrity fails is, NOT COINCIDENTALLY, No matching 'Country'.

Also, a referential integrity error will be thrown when trying to insert the city, which depends on the country.

Business Compontent

```
☐ 🖳 Attraction
      📍 AttractionId
      🔑 AttractionName
      ↗ CountryId
      ↙ CountryName
      ↗ CityId
      ↙ CityName
      ↗ CategoryId
      ↙ CategoryName
      🖼 AttractionPhoto
      ● AttractionAddress

          +
        LOGIC
```

&attraction

AttractionId	7
AttractionName	Forbidden city
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	2
CategoryName	Monument
AttractionPhoto	
AttractionAddress	

+

&attraction.**Load**(Pk)
&attraction.**Insert**()
&attraction.**Update**()
&attraction.**Delete**()

&attraction.**Success**()
&attraction.**Fail**()
&attraction.**GetMessages**()

&attracion.**Mode**()
&attraction.**Save**()
&attraction.**InsertOrUpdate**()

Having said that, we have seen the most relevant aspects of updating the database with Business Components.

In this way, we saw that from the transaction it is possible to create a data type that is like an SDT, but that allows operations to be performed on the database from methods.
These operations preserve the logic of the transaction. Although we didn't go deeper into it, clearly not all the rules and events of the transaction will be incorporated into the business component. The ones that have to do with the interface obviously will not. The Parm rule is not taken into account, either.
In addition to allowing these operations on the database, we can check the result of the last operation performed, as well as obtain the messages generated, in a collection.

There are more methods to study. For example, the Mode method returns the mode of the business component: if it is Insert, Update, Delete.
The Save method will try Insert or Update according to the variable's mode.
Also, the InsertOrUpdate method will always try to insert, and if it fails because of a duplicate key, then it will try to update.

The Insert, Update, Delete and InsertOrUpdate methods can also be applied to Business Components collections.

Business Compontent

&attractions

AttractionId	
AttractionName	Notre Dame
CountryId	2
CountryName	
CityId	1
CityName	
CategoryId	2
CategoryName	
AttractionPhoto	
AttractionAddress	

AttractionId	2
AttractionName	Great Wall
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	3
CategoryName	Monument
AttractionPhoto	
AttractionAddress	

AttractionId	
AttractionName	Palace of Versailles
CountryId	2
CountryName	
CityId	1
CityName	
CategoryId	3
CategoryName	
AttractionPhoto	
AttractionAddress	

&attractions.**InsertOrUpdate**()   InsertOrUpdate()   InsertOrUpdate()   InsertOrUpdate()

So, if the &attractions variable were a collection of the Attraction Business Component –in this case, a collection of three Attraction items–, applying the InsertOrUpdate method would be the equivalent of running through the collection and applying the method individually to each item.

The result will be True if the individual results were all True.

Business Compontent
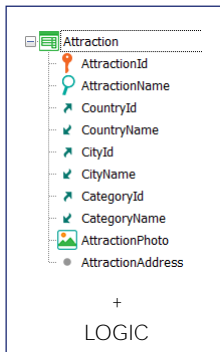
Only in Procedures?

&attraction

Attraction	
AttractionId	7
AttractionName	Forbidden city
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	2
CategoryName	Monument
AttractionPhoto	
AttractionAddress	

&attraction.**Load**(Pk)
&attraction.**Insert**()
&attraction.**Update**()
&attraction.**Delete**()

+

&attraction.**Success**()
&attraction.**Fall**()
&attraction.**GetMessages**()

&attracion.Mode()
&attraction.**Save**()
&attraction.**InsertOrUpdate**()

Attraction
- AttractionId
- AttractionName
- CountryId
- CountryName
- CityId
- CityName
- CategoryId
- CategoryName
- AttractionPhoto
- AttractionAddress

+

LOGIC

The last important thing to mention is that a variable of Business Component type can be used in any object that has some section of code, not only procedures. This means that we can update the database; for example, from a panel event that requests or displays data to the user, as we'll see.

It can also be done from events in transactions, although in that case there are some limitations that we will not see here.

More

- InsertCategoryUpdateAttractions proc

  Insert "Tourist site" category
  Update Attractions of Beijing

- CategoriesAndAttractions web panel

  Do:
  Insert "Tourist site" category
  Update Attractions of Beijing

  Undo:
  Delete "Tourist site" category
  Update Attractions of Beijing

- MassiveInsertRemove panel

  Remove Data:
  From Category
  From Attraction

In the following video, we will apply everything seen here in an example. You may skip it, except for the final part.

There:

We will implement again the procedure created here, so that it inserts a **new tourist attraction category: "Tourist Site." We'll change all the Beijing attractions that had the "monument" category and assign it the new one.**

We will see how to do this through an interactive web panel, which offers to do the above, but also to undo what was done, leaving everything as it was.

Lastly, we will create another web panel that will allow us to remove the information from both tables, Category and Attraction. This will be taken up again later, so we recommend watching at least this final part.

To be continued...

Of course there's a lot more to see, but we'll leave that for the next level. If you're interested, you can search for videos related to this topic in the GeneXus Analyst course.