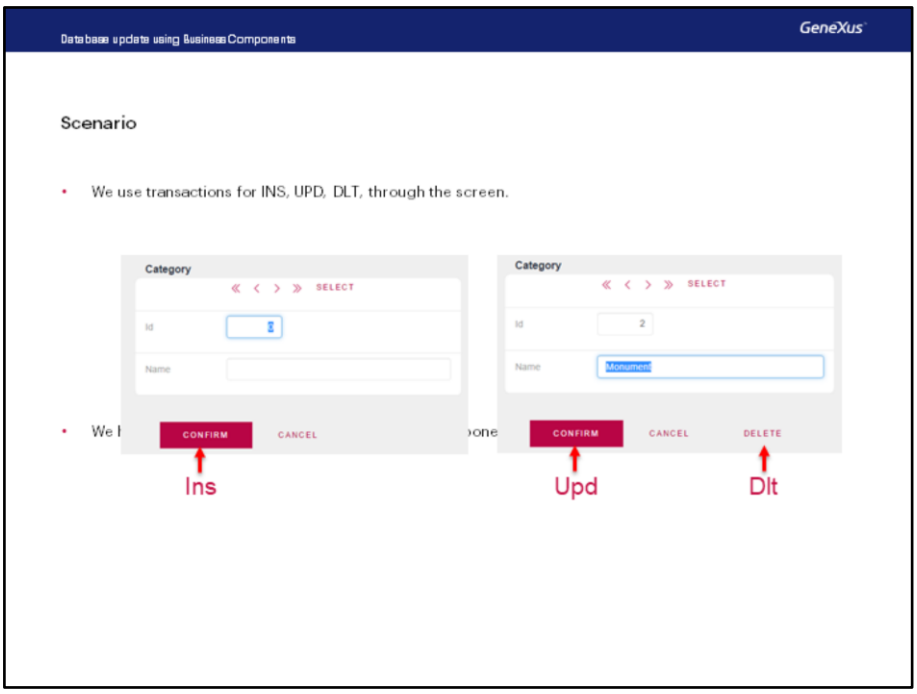
Database update using Business Components

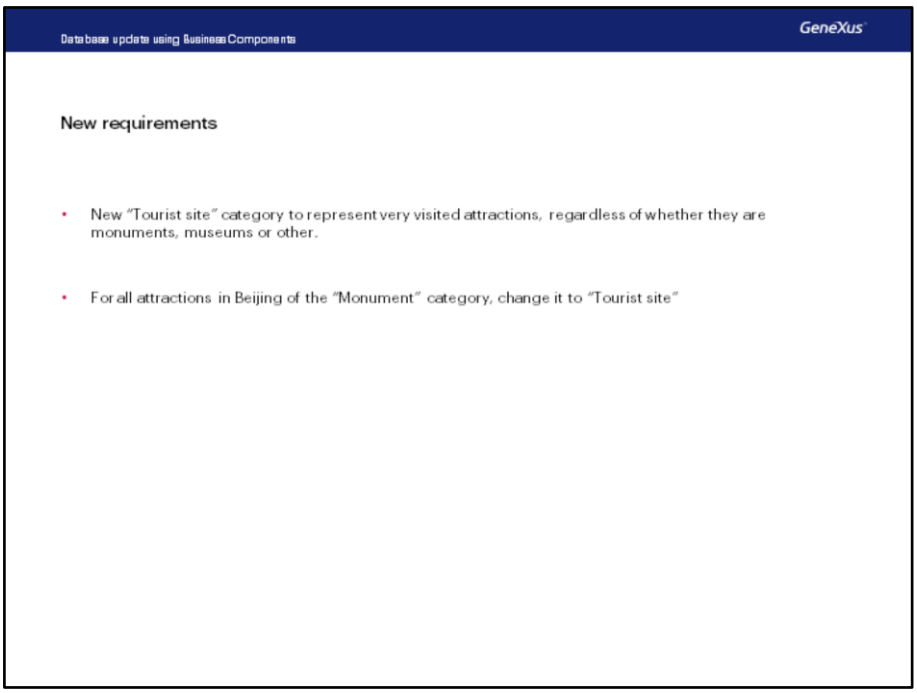
How to update data as in a transaction, but by code and without screen

GeneXus 16



Previously we used transactions to insert, modify or delete records from the database, and we always did this through its screen, with the buttons and controls we have available for that purpose.

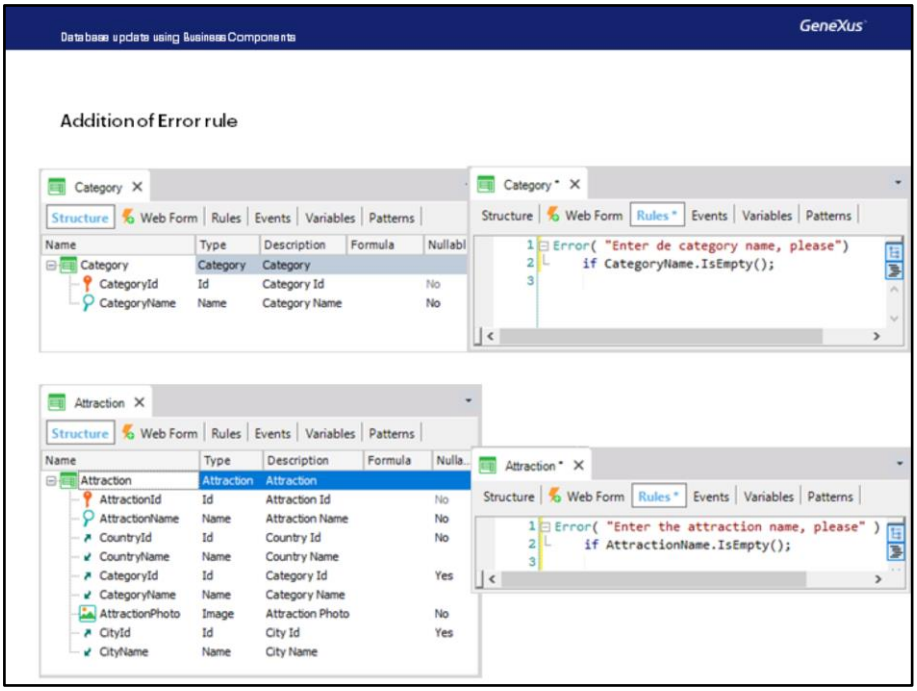
Here we will see how to do the same operations using commands written in a code, from any GeneXus object, and applying the Business Component concept.



We can explain it with an example. Suppose that, in our travel agency, we have some tourist attractions visited so often by tourists that they should be classified differently in order to organize trips to those particular attractions.

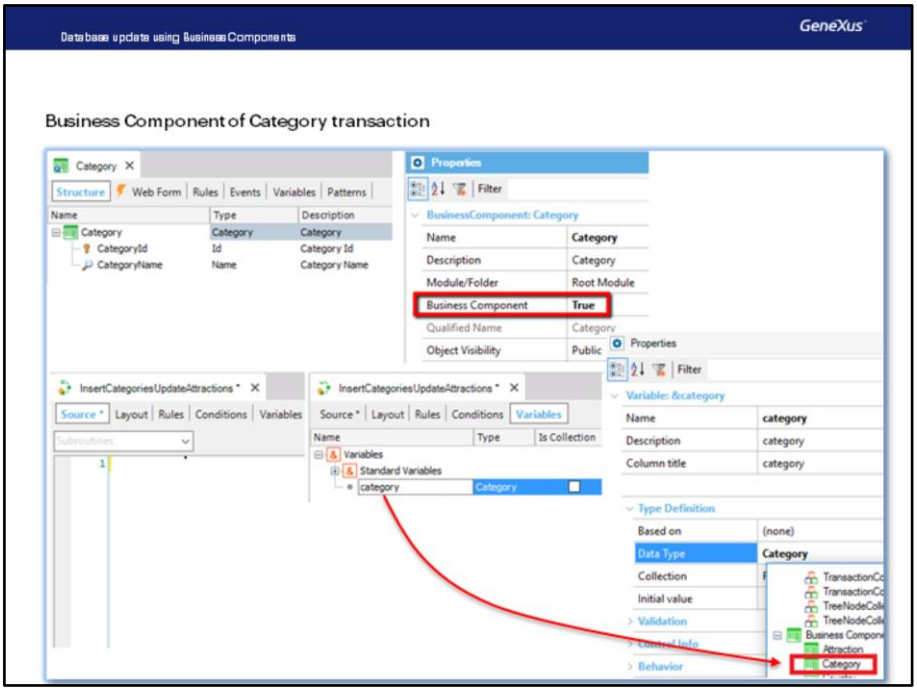
For example, in Beijing, since monuments are so popular, we should not classify them as such but rather as “tourist sites”, a category we did not have and that will be useful for us to identify the attractions that are more visited, regardless of the attraction type with which they were classified before.

We will need to create that new category and change all the Beijing attractions in the “Monument” category to the “Tourist site” category.



To do this we must insert a new category in the CATEGORY table and then go over the attractions of the ATTRACTION table filtering by city equal to “Beijing” and by category equal to “Monument”, in order to change the category code to “Tourist site”.

Before we move on, we see that we have added an Error rule to both the Category transaction and to Attraction, in order to control that the name is not empty. We will explain what we need this for in just a moment.



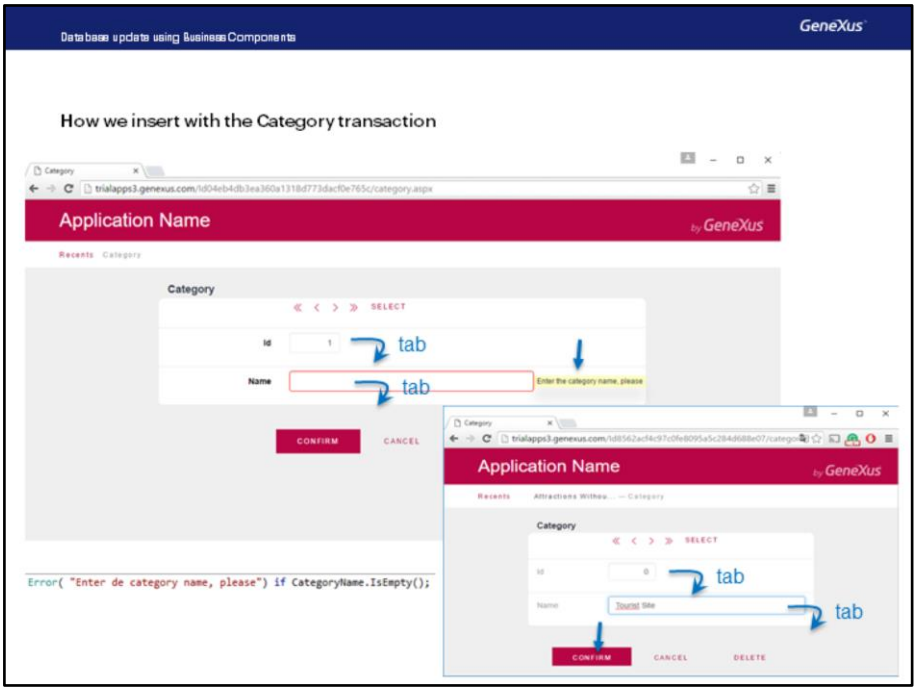
To implement the creation of the new category and update of the attractions we will create a procedure object that we will call InsertCategoryUpdateAttractions.

The first thing we have to do now is create the “Tourist site” category, for which we use a Business Component of Category. To create it we go to the Category transaction, find the **Business Component** property and assign it the True value.

When we do this, GeneXus creates a structure from the Category transaction that will have similar functionalities to those of the transaction that will be associated with it.

To be able to use this definition we must create a variable of the type of this business component of Category. So, we go to the tab of variables in the procedure and we create a variable under the name &category. We will see that GeneXus automatically assigns to it the Category data type. This data type corresponds to the Category Business Component, created from the Category transaction.

If we go to the properties of the variable and click on Data Type, we will find a group called Business Component. When we open it we will find the Category Business Component, with an icon like the one for a transaction. We then confirm that GeneXus created a new data type based on the Category transaction and we will see that it shares many aspects with it.

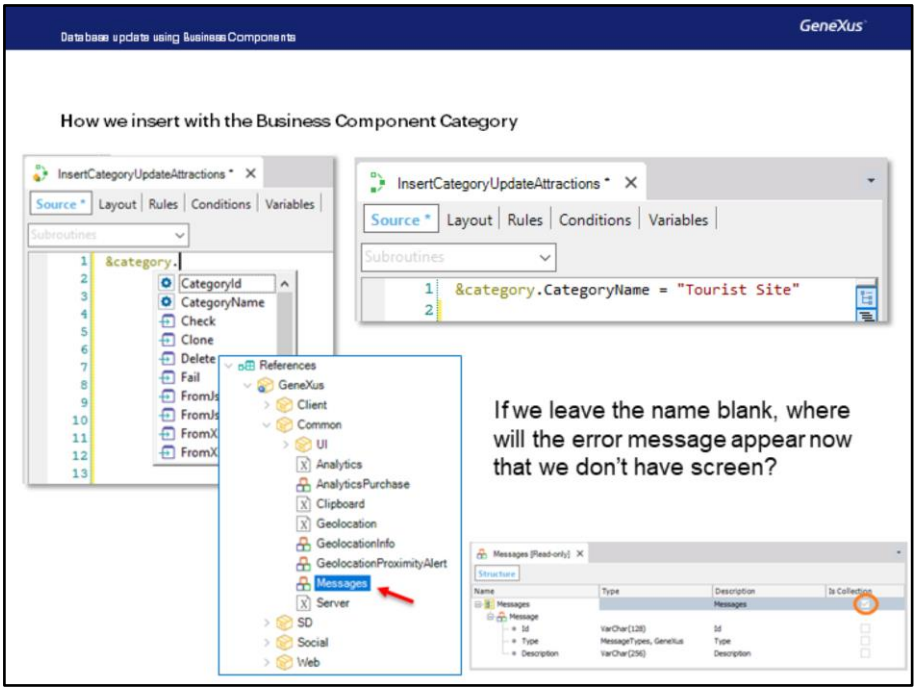


We should bear in mind that if we execute the Category transaction, by deafult, is expected that the user will insert a new record.

The first thing we would do would be to stand on the category identifier. But since the CategoryId attribute is autonumbered, it will not be necessary to assign a value to it. So we press tab to exit the field and we leave it empty.

Then we should write the name of the category. But what would happen if we leave it empty? The Error rule that controls if we leave the CategoryName attribute without a value will be triggered!

In the name of the category we would write “Tourist site”, and last, after assigning the values we wanted to the attributes of the category, we press the Confirm button so that the data is saved to the database and we consider the insertion as concluded.



When we use a Business Component, the sequence of operations is exactly the same. To insert a new record we would assign values to the fields of the category and confirm the changes.

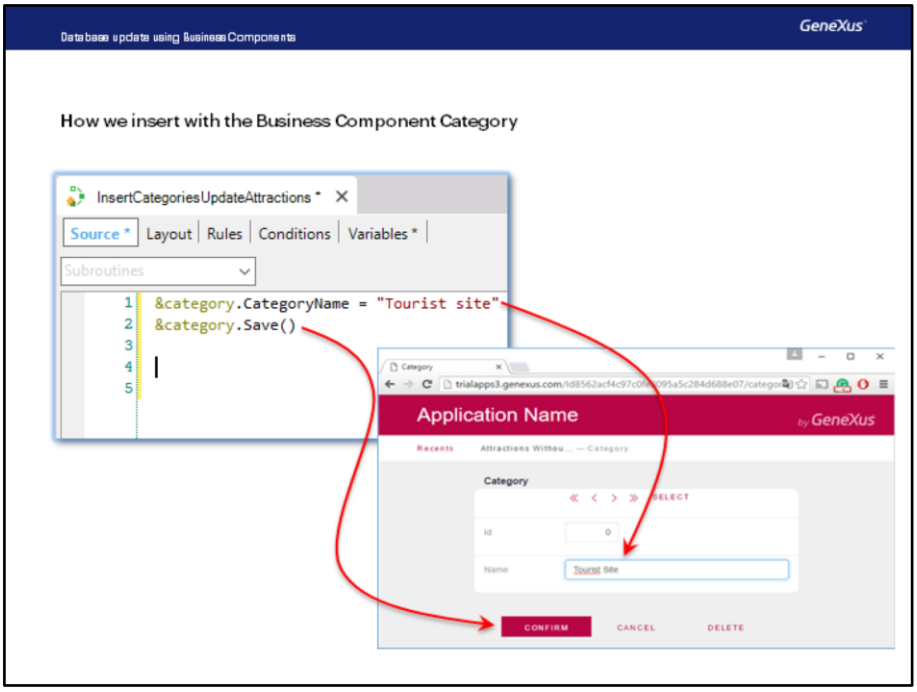
To do this we insert the &category variable in the source. If we press the point we will see that several methods available appear (like Save, Delete, Check) and CategoryId and CategoryName also appear, to which we will be able to assign values.

Since we know that the category identifier is autonumbered, we do not assign value to it, and by default it will also assume that we will be inserting. As we did with the transaction screen, we maintain the name of the category.

What is interesting about working with a Business Component is that the rules defined in the transaction that was used to create it will also be triggered. Therefore, if in our example we omitted to assign a value to the name of the category, the Error rule that controls this would have been triggered, as is happens when we use the screen of the Category transaction. It's just that this will not happen when this assignment is found in the code, but rather when the Business Component is instructed to save.

We must make clear that not all rules will be triggered because we are executing the Business Component through code, so no rules calling objects will screen will be triggered (like for example those calling another transaction or a web panel); and the Parm rule will not be triggered either if it was defined in the transaction.

It is then clear that if we leave the category without a name, then the record will not be inserted, but where will the error message appear now that we don't have the screen? In every knowledge base, GeneXus creates a predefined SDT called Messages that is a collection of items.

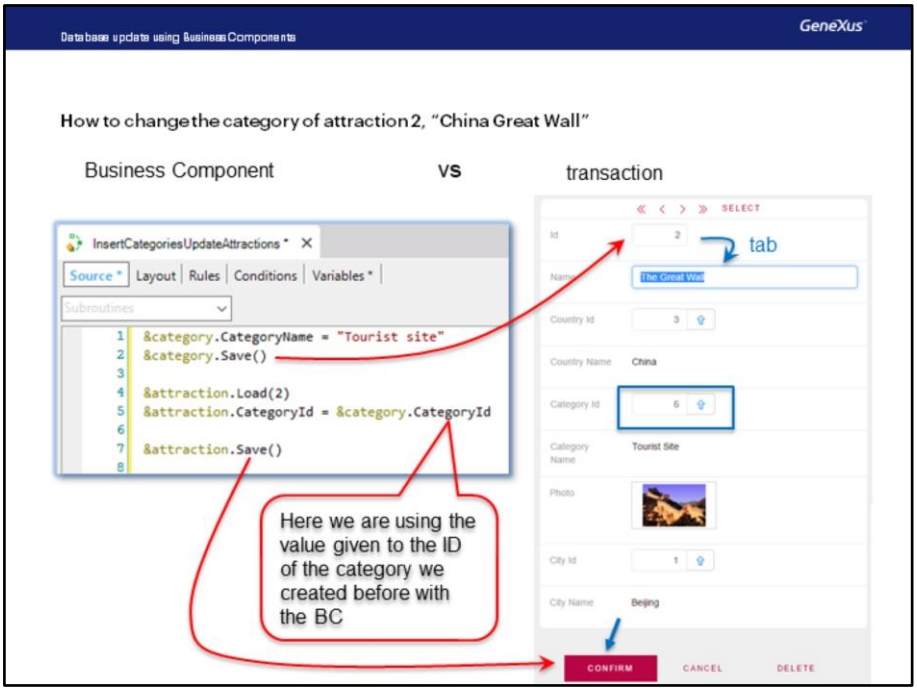


Considering our example, if we were working with the screen of the transaction, once we assign the name of the category we should press the Confirm button. With Business Components we use the Save() method:

&category.CategoryName = “Tourist site”
&category.Save()

For each business component variable, like &category, when we do a Save operation, a collection of messages is loaded in memory with all the warning or error messages produced as a result of that operation.

Though we have a very simple manner of obtaining and going through it, we will not be considering that now.



So far, we have created the "Tourist site" category. Now we will change the category of China's monuments.

Suppose that we only have to change the category of the China Great Wall, whose id= 2. So, what do we do in the transaction? We open the Attraction transaction, insert a 2 in the ID, exit the field, and GeneXus will bring us all the data on the China Great Wall.
Then we change the values we want, which in this case is the category, and we save (by pressing Confirm).

When we confirm, the change takes place, triggering all the corresponding rules. For instance, had we deleted the name of the transaction, then the update would have failed. If, on the other hand, we had changed the ID of category with a non-existing one, then an error would have occurred since GeneXus will control consistency between values of related tables (referential integrity), and the modification would not be possible.

The behavior will be the same in the case of the business component.

To stand on a transaction we use the Load command and insert the ID of the attraction between parentheses, in this case number 2.
Then we assign the new value of the category (which is the value of the ID of the "Tourist site" category that we created before) and we end by doing a Save().

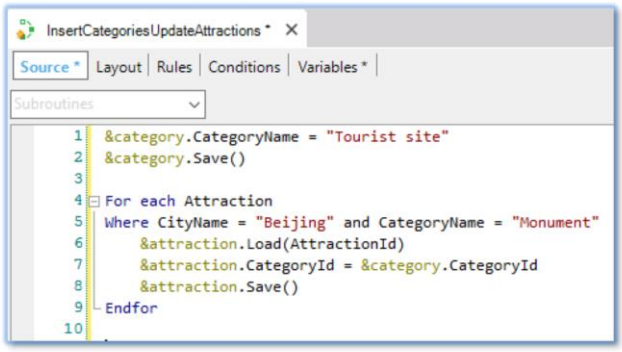
&Attraction.**Load**(2)

//Here we are using the value of the ID of the category we created before with the BC.
&Attraction.CategoryId = &category.CategoryId

&Attraction.**Save**()

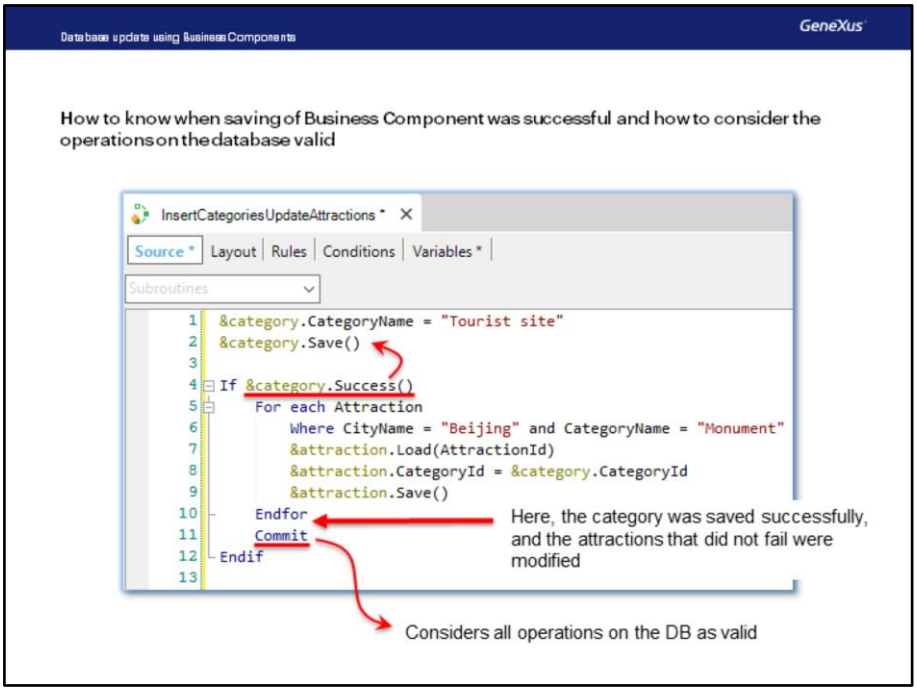
Since we wrote the Load of an existing record, the Save will know that we want to update (and not insert).

How to change the category of all attractions in Beijing that were monuments... with Attraction business component



But our requirement is not to change only the category of the China Great Wall, but also de category of all attractions in Beijing that are monuments. To do that we go over all those attractions.

Since we are now going over records with a For each, we will do the Load of the ID of the attraction we find in each iteration.



Because every Save() may turn out to be successful, or not, and depending on the business rules and on referential integrity, the records could have been inserted or modified in the database, or maybe not.

To know whether a Save() command has been successful or not, we have the Success() method that returns True when it has been successful, or it returns False otherwise. The reason for the Save() command to fail could be the triggering of an Error rule whose saving it did not allow, or perhaps a system fall at the time of saving causing the record to not be saved. Or also a possible fail in referential integrity.

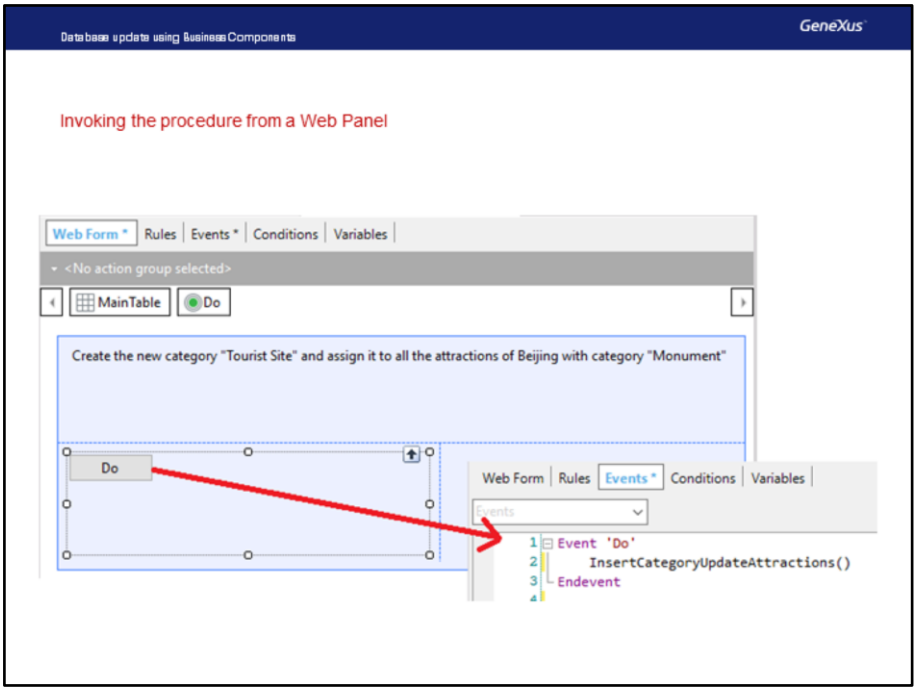
When the insertion of the new category is not successful, then the Save of the attractions will obviously fail because that category will be inexistent. So we condition the modification of the attractions to the result of the Success() method.

All that's left is to inform the database that the records we inserted or updated are correct, and that we want them entered so that they will not be lost in the event of a system fall. We do this by adding the Commit command.

At this point, we know that the category was successfully recorded and that the attractions were modified (for all those not producing an error due to a business rule preventing the modification). Now we need the database to consider all these updates valid. We inserted a record and modified other records. In the event of a failure in the system (for instance, due to a power blackout), then the databases will be recovered undoing all the operations for which a Commit was not performed.

The **Commit** is a special command that enables us to indicate to the database the end of the block of operations that we want done altogether (and if this is not possible, the undoing of them all). When this command is executed, the database will save the data in such a way that it will not

be lost in the event of a system fall or power failure.

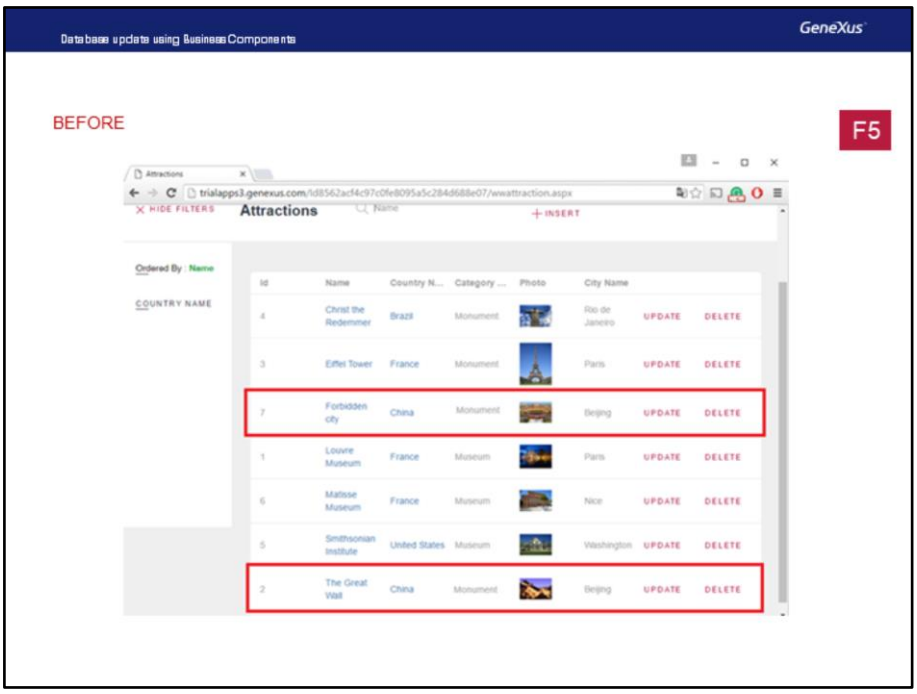


To execute our procedure we need another GeneXus object to invoke it.

We can do this from the Web panel object. We have seen already that a Web Panel is a tool we may design in accordance with out needs in order to fulfill a variety of functions, such as viewing data from the database, making a start screen for our system, or for entering data.

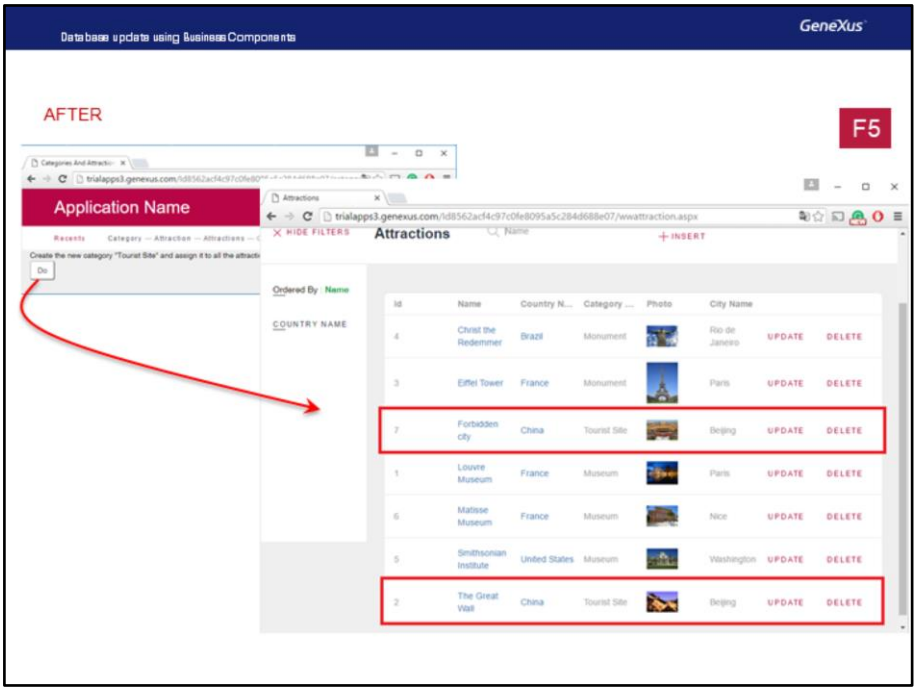
In our case we will create a web panel with a button that, when pressed, will call the procedure we created before.

In the event associated with the button we invoke the InsertCategoriesUpdateAttractions procedure, which does not require any parameters.



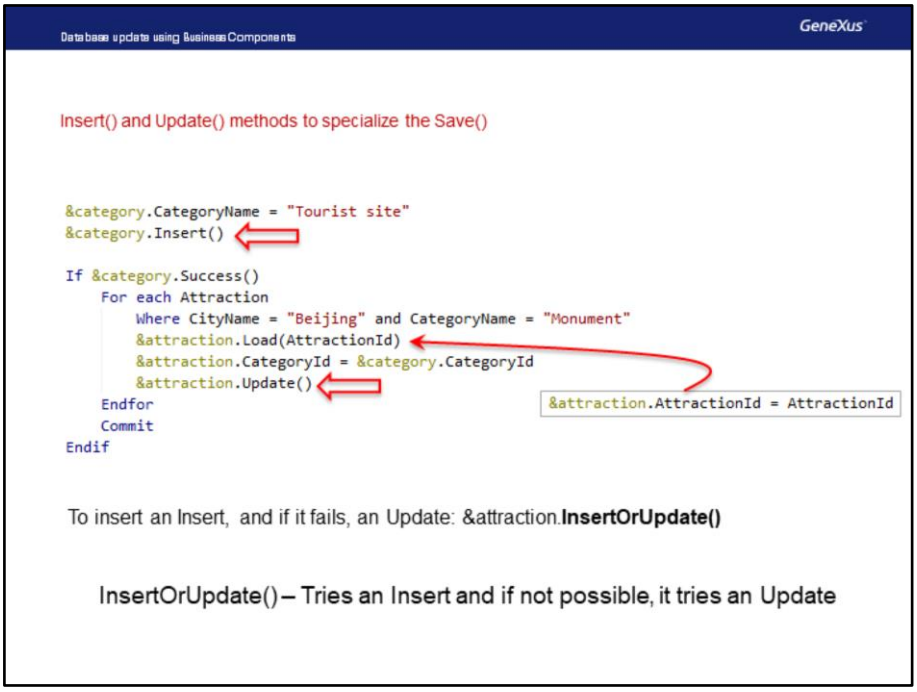
We execute the application and note that China's attractions have the Monument category.

If we check the detailed information we will confirm that they are both located in Beijing.



When we open the CategoriesAndAttractions webpanel and press the Do button to return to the attractions we will see that the Tourist Site attraction has been created and assigned to the two attractions in Beijing with the Monument category.

We see that, in the case of the category, GeneXus inferred that it had to do an Insert, and in the second case, when we changed the value of the attraction's category, it was an Update, as it happens with a transaction. This is why, in the procedure, we use the Save() method, which has the intelligence to save data, regardless of the operation carried out.



We saw that the Save() method has the intelligence to save the operation, regardless of whether it was a data insertion or modification.

But we also have the Insert() and Update() methods enabling us to specifically indicate the operation that we want to carry out.

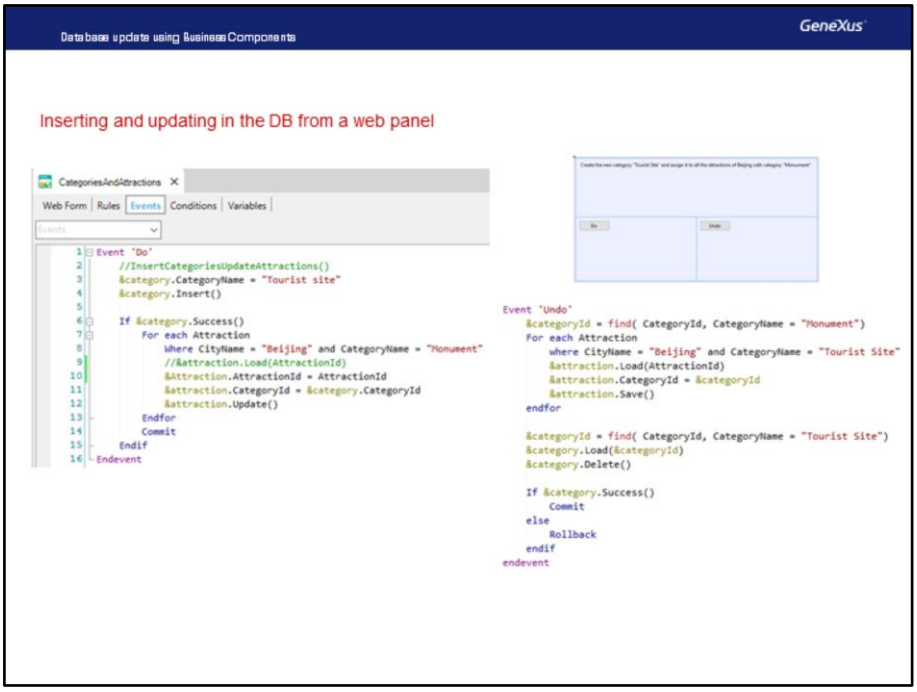
We modify the code to use these methods; Insert for the case of categories, and Update for attractions.

Note that in the case of attractions we would not have required the Load, since we had the possibility or directly writing:

&Attraction.AttractionId = AttractionId

Since we made clear that we want to do an attraction Update, GeneXus knows that we want to update a record.

In addition to the Insert() and Update() operations to indicate that we specifically want an insertion or a modification, we also have the InsertOrUpdate() operation, through which an attempt will be made to do an Insert when we assign new values to the attributes. If this is not possible (because, for instance, there is already a record with that key), then an Update will be tried.



Something else important to point out is that, as seen, it is possible to insert and update records from two different tables within the same object (in this case, a procedure), though we could have also put this code directly in the event of the web panel, since updates to the database through business components may be done from any object (with some limitations for the case of transactions).
If we want to undo what was done, we add the “Undo” button to the web panel we delete from it the recently created “Tourist site” category, and then change the category of the attractions that we had changed to “Tourist site” back to “Monuments”.

How could we delete the category through the transaction? We would first find it by selecting its ID, and then we would press the **Delete** button. With business components the procedure will be the same, with a procedure with that name.

When we do not have the ID, we search for it using the Find formula. This is an inline formula for finding the first record fulfilling the condition indicated. For that record it returns the value of the attribute we specify.

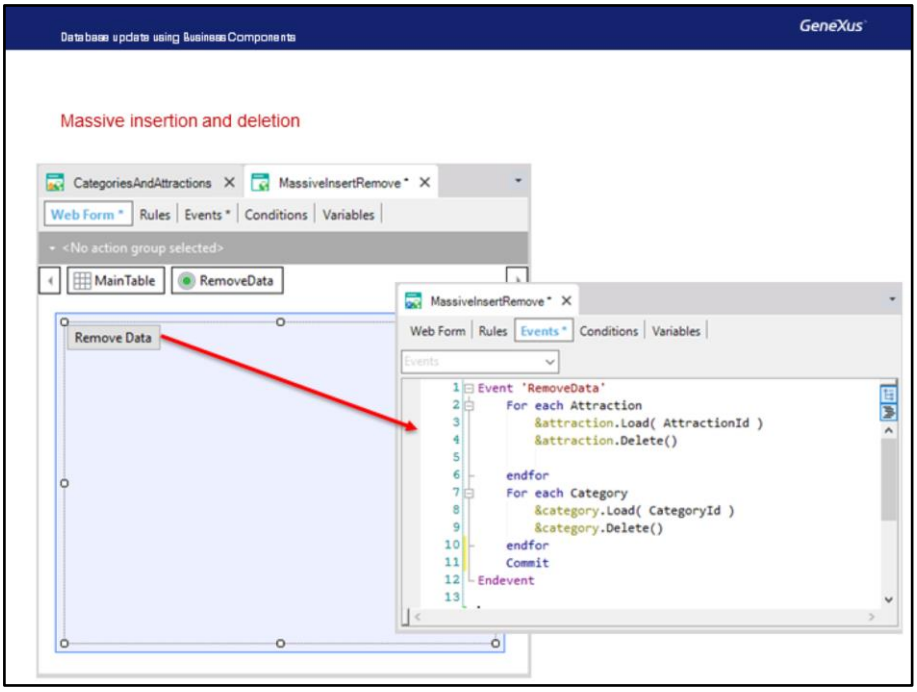
If we deleted the “Tourist Site” category through the transaction, the delete would fail because there are attractions with that category assigned. The Business Component will do exactly the same, controlling referential integrity at the time when the code of the Delete() is executed. We cannot delete the “Tourist Site” category because there are attractions with that category that would end up lost.

So, before we do this, we must change the category of these attractions by restoring the previous category of “Monument”.

Since, in general, we do not remember the values of identifiers but we do remember their names, it will be safer if we search for them in the database using names instead of trying to remember. So, in our case, in order to recover the ID value of the Monument category we will use the Find formula.
For each attraction in Beijing with the “Tourist Site” category we load, in the &Attraction business component variable, all its data and change only the data of CategoryId by assigning to it the one corresponding to “Monument” and then we update.

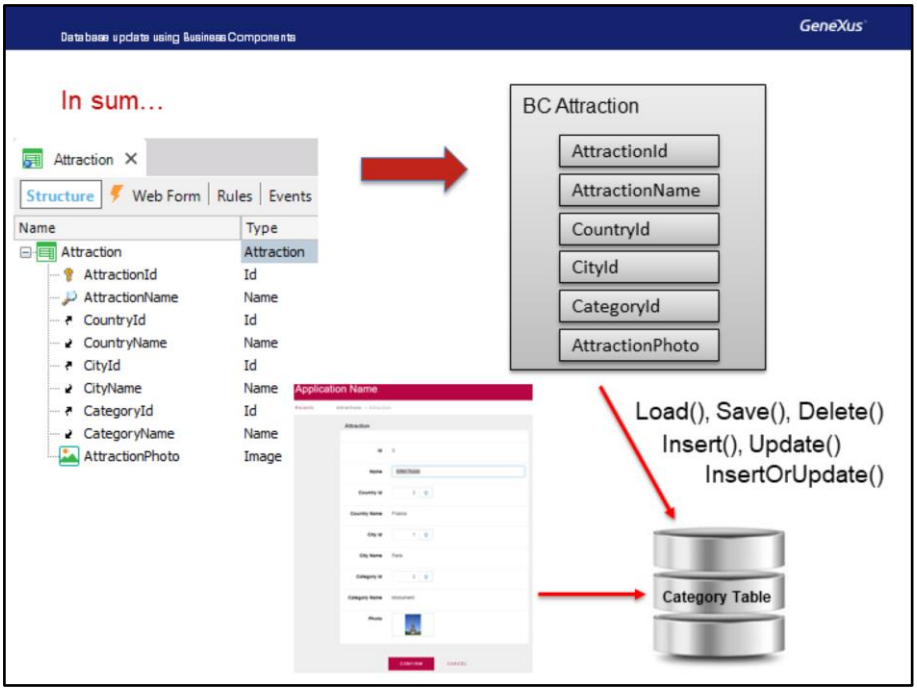
Now we can delete the “Tourist Site” category. To end we must remember the Commit to confirm that the deletions are definitely done on the database and to avoid the recovery of previously deleted records in the event of a system fall as it happens with the rollback performed by databases upon recovering.

If any of the attractions failed in its update, then the Delete of &category will also fail. So we could directly ask if the deletion of the category has been successful for doing the Commit. If not, then we will not want any of the modifications on the database to be effected, so we can proceed to cause the rollback, that is: undo everything that was done.



Note that, if we wanted to delete all the previous attractions, instead of changing their category we should delete all those records from the ATTRACTION table. And we could do this through a For each with Delete. To view this we will create a web panel “MassiveInsertRemove” with a “Remove data” button.

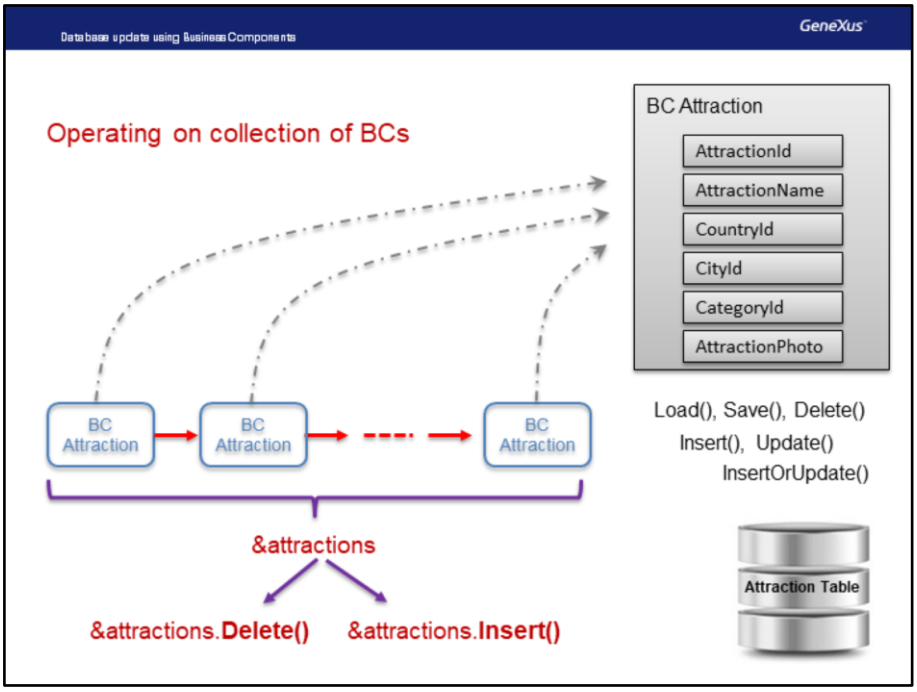
Note that the order in which deletion are made is in fact important, because we known that business components control referential integrity, so we cannot delete the categories first.



To summarize what we have seen: we learned that a Business Component is a special kind of data type built on the basis of a transaction, as a mirror for it. It enables us to do everything that the transaction does, except what relates to its screen.

That's why, when we have the Business Component (mirror) of the trn, we can then define a variable of that data type that will be like a structured SDT, with the same structure as the transaction. And it is through the Load, Save and Delete (or Insert, Update and Delete) methods that we can do the same operations that we use in an interactive manner in the transactions, ensuring that the business rules are triggered "as if" the transaction were executed. What business rules? The ones that have no relation to working in an interactive manner through a screen, or that invoke another screen. This is why we say that executing a Business Component is like executing the transaction in a "silent" manner.

For a single level transaction like the Attraction transaction, the structure of the Business Component will contain not only one element per attribute physically present in the associated table, but also the attributes like CategoryName, or CountryName or CityName that appear in the structure of the transaction as inferred through the foreign keys. They appear with the same function as in transactions: to possibility of inferring values after doing the Load.



The Insert, Update and Delete operations may be done also in a massive manner on a collection of BCs, which we still do not know how to load. This means that, if we had a collection variable of Business Components, such as having all the attractions loaded in a &attractions variable where each item is of the Business Component Attraction type, then we could delete them all by doing &attractions.Delete(), without the need for going over the whole collection, item by item, to delete them individually.

Further ahead we will see an example that adds records in a massive manner with Insert.

GeneXus™

The power of doing.

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