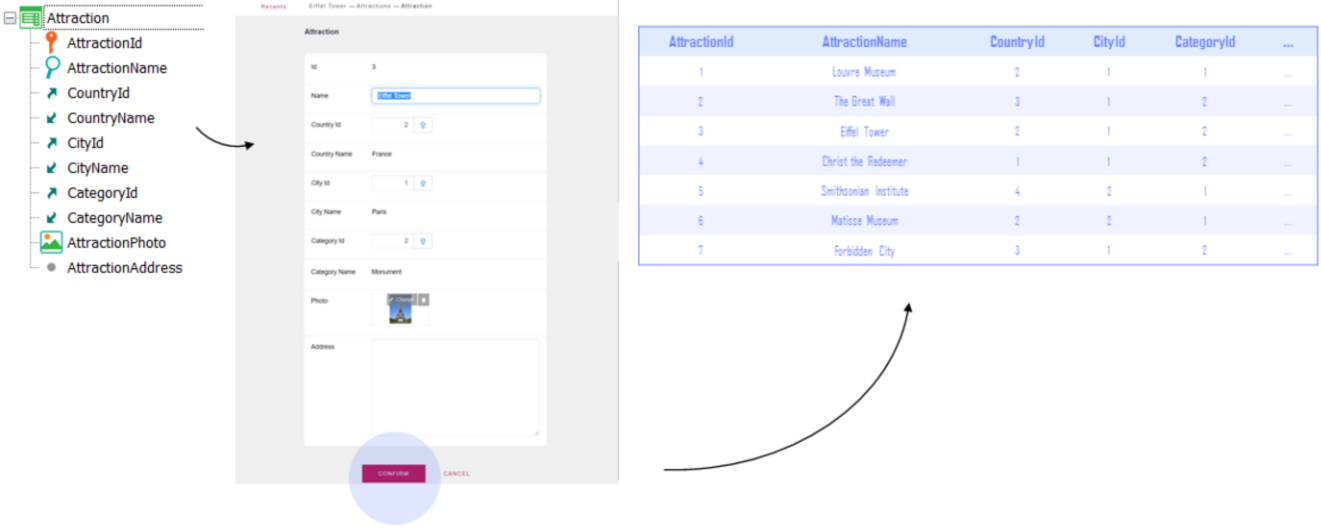
Database Update

Using Business Components. Justification.

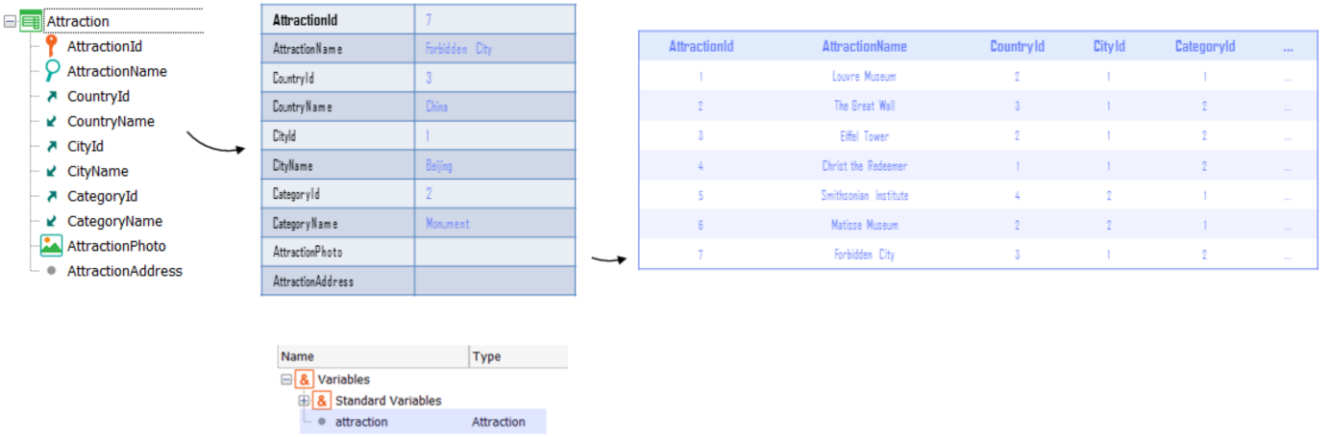
GeneXus™

1. Transaction: Insert, Update, Delete



So far, we have only updated database information through transactions, i.e. interactively through a graphical interface.

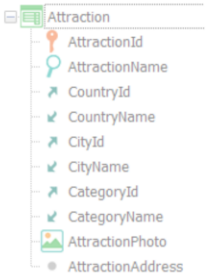
1. **Business Component**: Insert(), Update(), Delete()



Next, we will look at how to update the database information using code.

We will give priority to one of the two ways: updating through Business Components, and we will see why. We will work with the transaction structure as if it were an SDT variable, taking into account the rules of the transaction. Through that variable we will insert, change or delete data from the database. It will be like using the transaction, but without its screen.

2. **Procedure**: New, For each, Delete

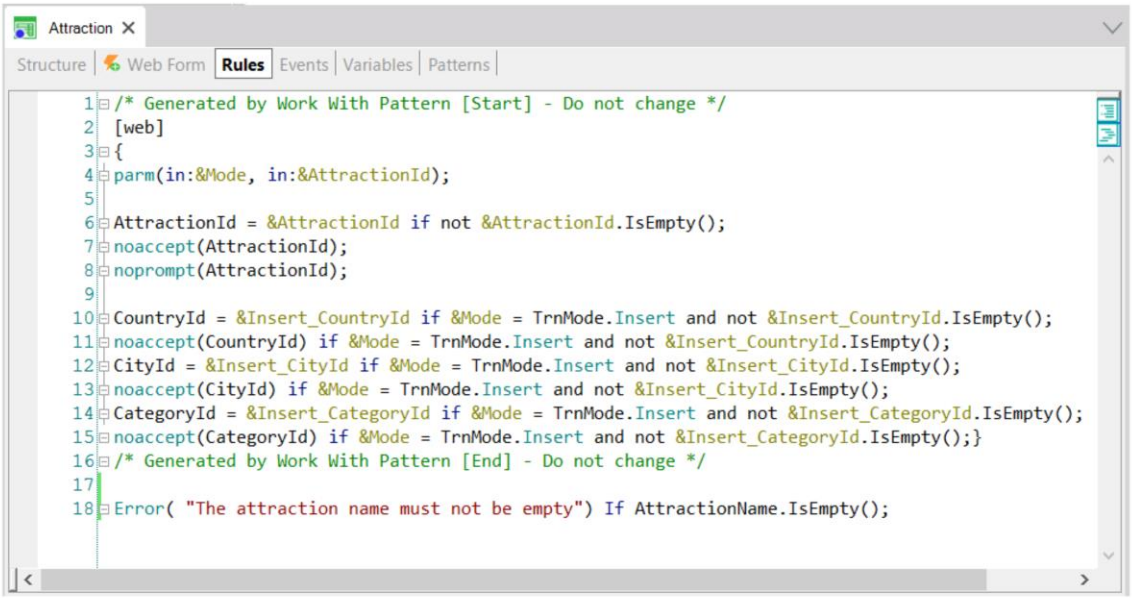


AttractionId	AttractionName	CountryId	CityId	CategoryId	...
1	Louvre Museum	2	1	1	...
2	The Great Wall	3	1	2	...
3	Eiffel Tower	2	1	2	...
4	Christ the Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	Forbidden City	3	1	2	...

The other way to do it is through special commands that can only be used within procedure-type objects. We'll see them later, in another video. But in this case there is independence from the transaction. We work directly on the tables, which has its drawbacks.

So let's study this first option, the one with the highest level, and therefore the one that basically will be more relevant to us.

Insert through the transaction



We will first take a closer look at what happens when we want to insert a new tourist attraction using the transaction.

We have changed the order of the attributes, so that the city is now part of the country, which affects the form.

We will add an error rule to avoid leaving the transaction name empty.
If now you press the Control key and the space bar, this little window opens and offers you all the elements that you could put in this part of the code. If you type the letter "N" you'll find the attribute you're looking for. This is a way to avoid making typing errors. The other way is to select Insert > Attribute, whose shortcut is Control+Shift+A, and there select Attraction Name.

To this transaction we had applied the Work With pattern, and that's why all these other rules are appearing, automatically added by the pattern to, for example, allow invoking the transaction from the Work With, passing it the mode (insert, update, delete) and the attraction identifier.

To more easily understand what we are trying to do, we would rather be able to invoke the transaction directly, without parameters, as it was before the pattern was applied. We can remove it by eliminating the instance from here... or, since we will need it for later, save this transaction with another name; for example, this one.

Parallel transactions

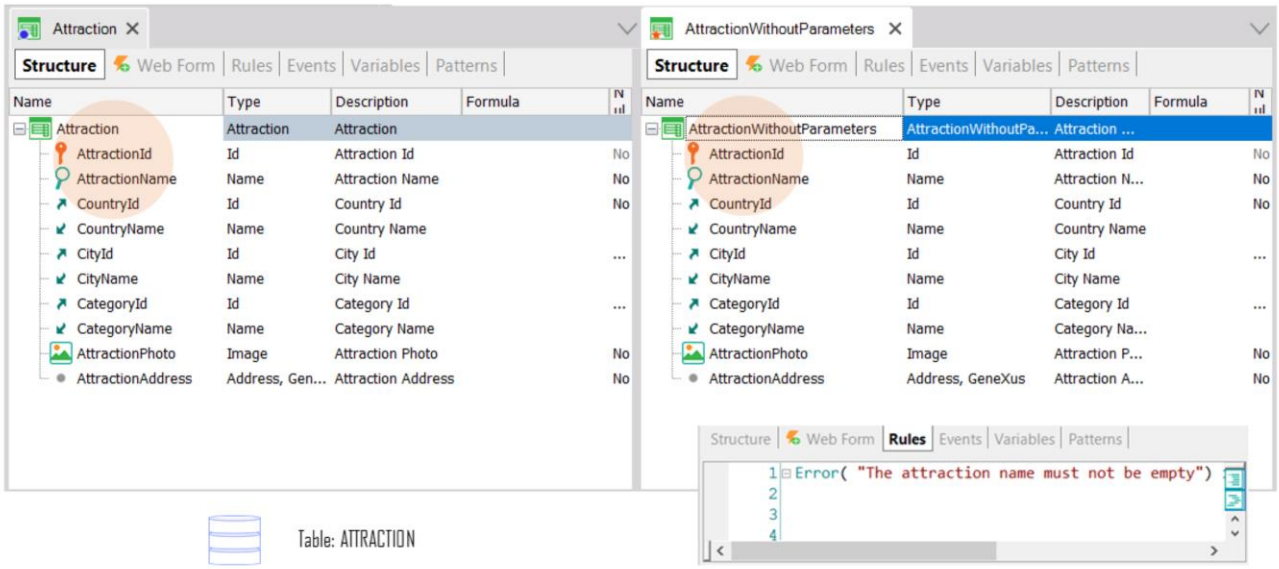
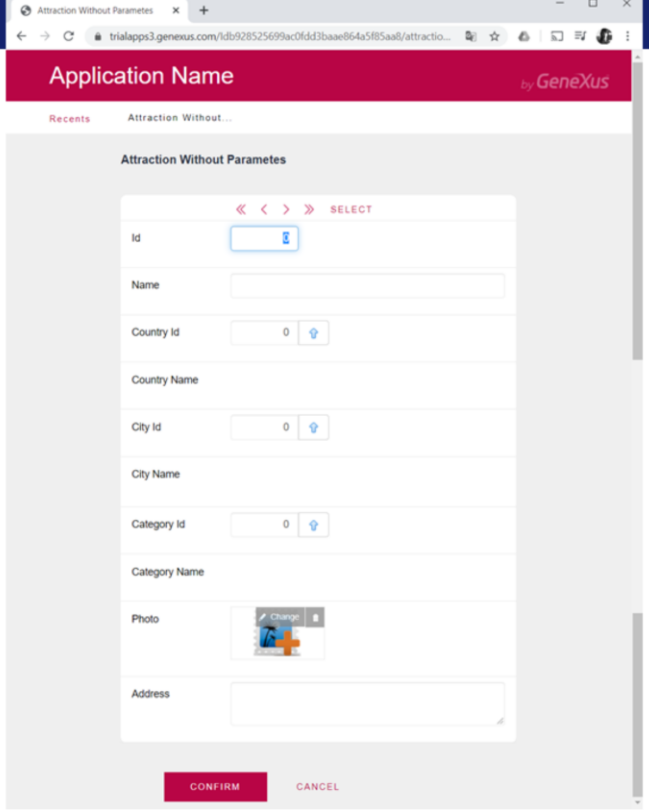


Table: ATTRACTION

This transaction will be identical to the other one, except for its name and the fact that it doesn't have the pattern applied. The rules that came from the pattern and the events have remained, so we just delete them. We leave the error rule and delete all the events that have been added by the pattern.
Note that the table on which this transaction will insert, change, and delete records is exactly the same as the table in the Attraction transaction. Why? Because the identifier is the same. These transactions are called parallel transactions. They share the table, but the programs are independent.
To this one, for example, we could remove even the Error rule, and therefore it would allow inserting records that the other would not.

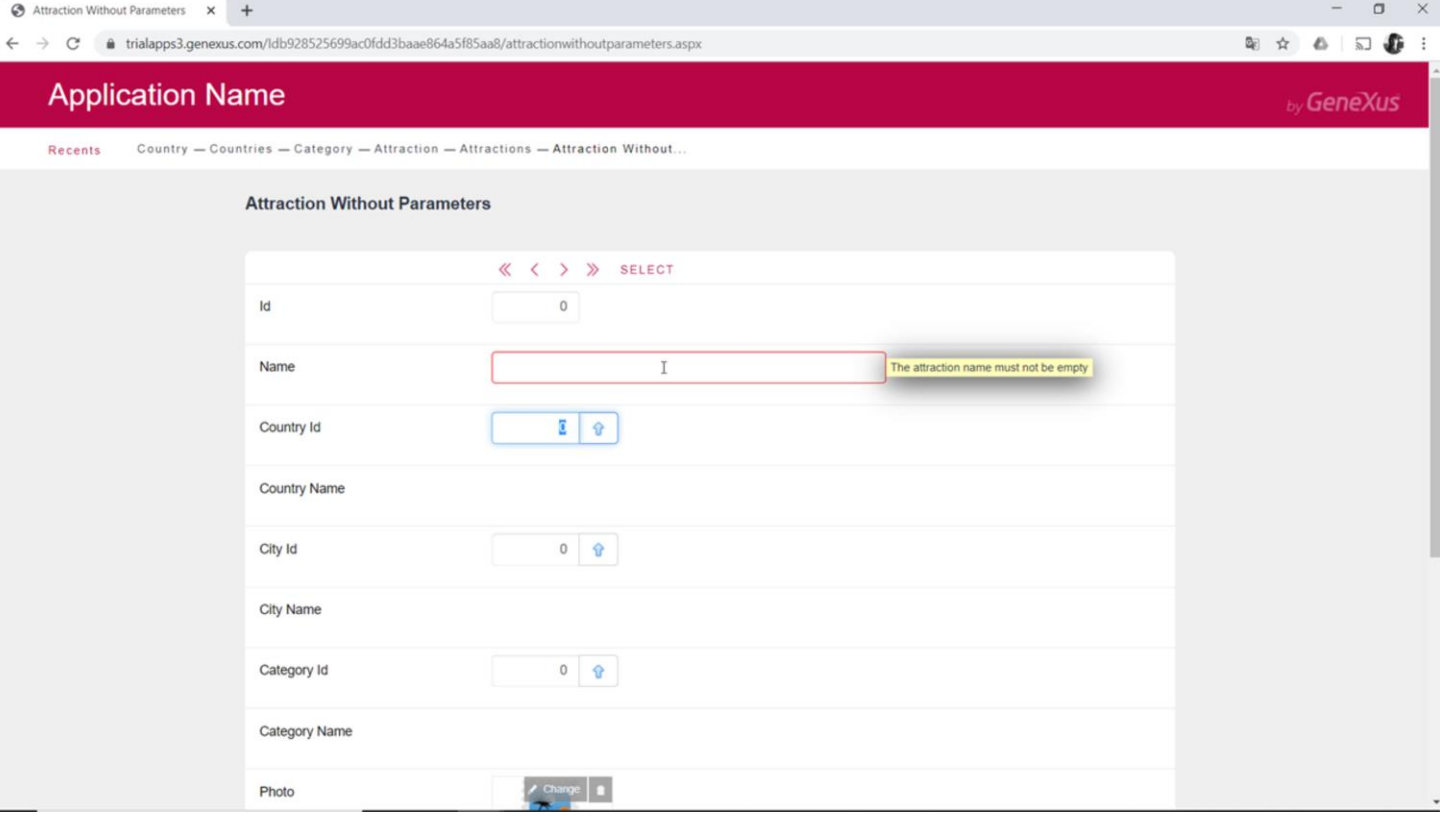


AttractionId	AttractionName	CountryId	CityId	CategoryId	...
1	Louvre Museum	2	1	1	...
2	The Great Wall	3	1	2	...
3	Eiffel Tower	2	1	2	...
4	Christ the Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...

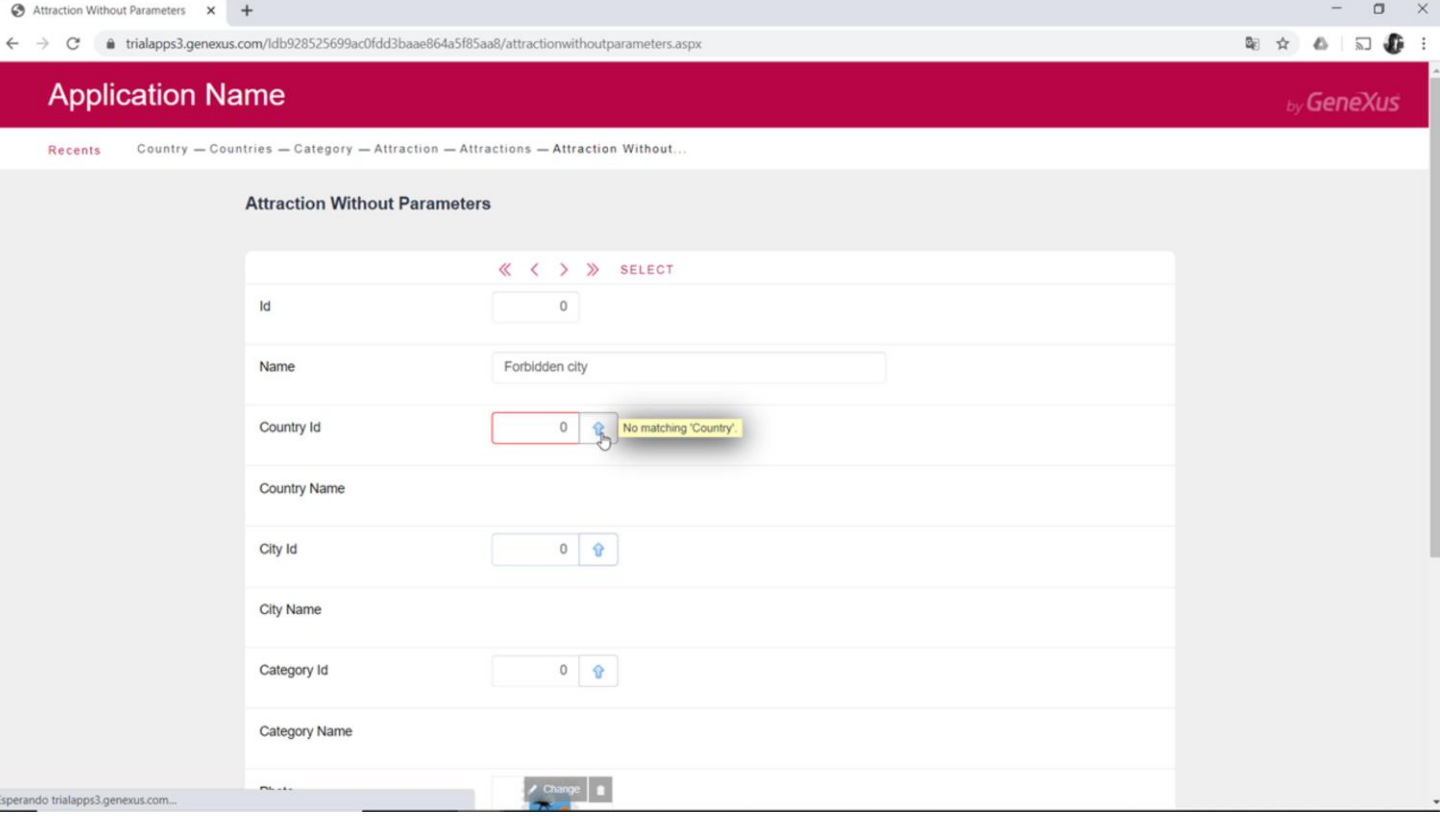
We press F5 to run it.

It opens with empty fields, and the identifier field is active, waiting for the user to enter a value and infer which operation will be processed. It can be an insertion, or otherwise an update or deletion.

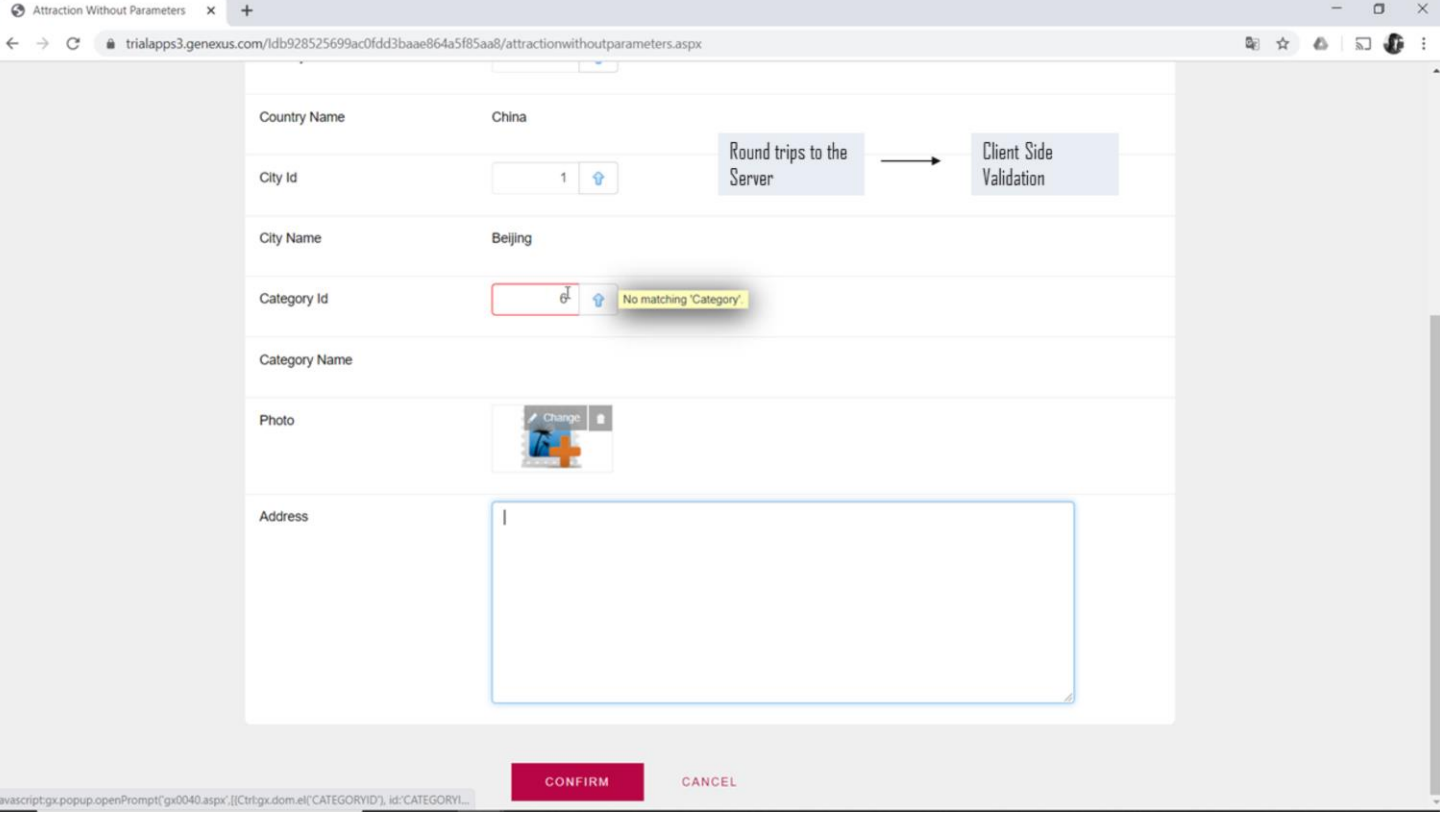
Thus, when leaving the field, GeneXus will search for a record with the entered value in the table. If it exists, the transaction will remain in Update mode, and the fields with the corresponding values. On the other hand, if it does not exist, the transaction will be in Insert mode and all the fields will be empty (unless there is some Default rule that depends only on the mode, or some assignment that is only conditioned with If Insert. For example, AttractionName equals something If Insert. In this case, there is none).



Since AttractionId is auto-numbered, the user will probably leave the value 0 when trying to insert a new attraction. The transaction will then be in Insert mode. If he now leaves the next field, AttractionName, without entering a value, an error message will be displayed because of the rule we had programmed. But even so, it allows us to continue entering the others because after confirming we will not be allowed to save anyway, and it will continue showing the message. So, let's enter the Forbidden City.



In the case of foreign keys, when you exit the field it is checked whether a record with that value exists in the associated table. Otherwise, the "No matching" referential integrity error is thrown. We can also continue, but, as with the other error, it won't let us save either. The Forbidden City is in China, and as we choose the key its name is already inferred. The same happens for the city.

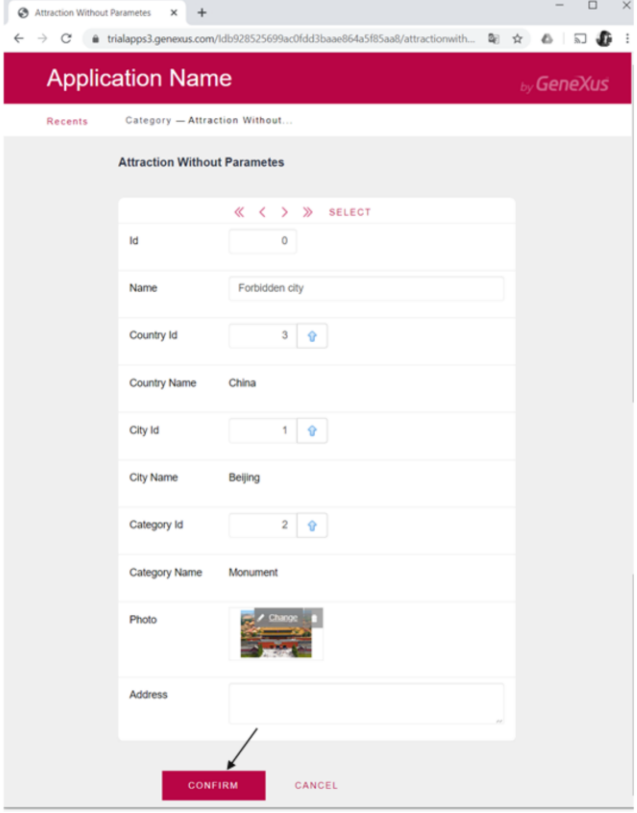


Here is another foreign key, the category, which cannot be left with a non-existent value, but which can be left empty. Remember we indicated that the attribute would accept nulls. But we know its category will be Monument.

Note that to make these referential integrity checks and bring the inferred names it was necessary to go to the server, which is the one that actually controls the database. All the rules and controls that are triggered field by field in the user's browser are there to make the user experience more agile, without any downtime. This is called Client Side Validation. But all this will have to be repeated on the server when the user confirms.

Of course, the fields that are not default foreign keys –that do not accept nulls–, and those that don't have explicit rules to prevent it can be left empty. Let's add a photo, but no address.

And confirm. We're informed that the addition has been successfully made. What happened in the background?



AttractionId	AttractionName	CountryId	CityId	CategoryId	...
1	Louvre Museum	2	1	1	...
2	The Great Wall	3	1	2	...
3	Effel Tower	2	1	2	...
4	Christ the Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	forbidden City	3	1	2	...

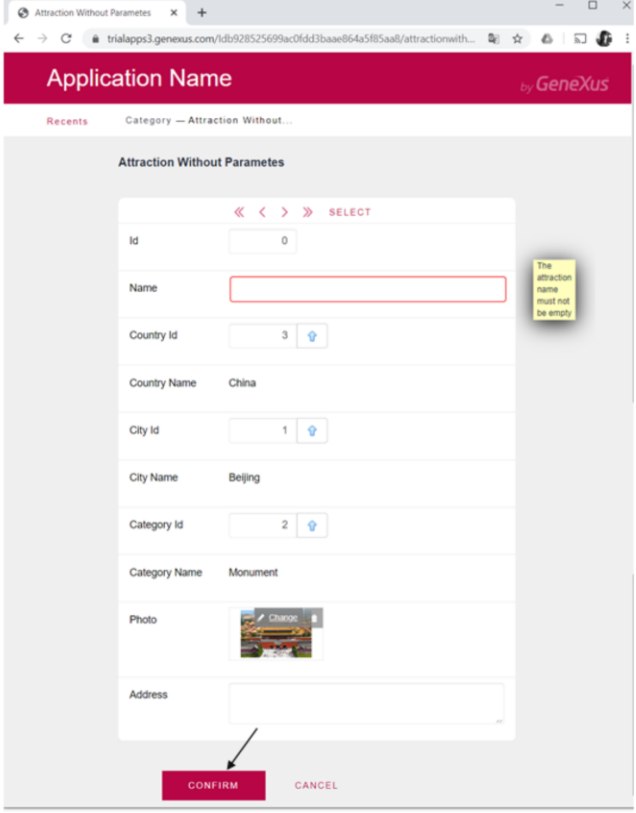
AttractionId	0
AttractionName	forbidden City
CountryId	3
CountryName	
CityId	1
CityName	
CategoryId	2
CategoryName	
AttractionPhoto	
AttractionAddress	

After pressing Confirm, all the information entered by the user in the fields must travel to the server, which will start again from scratch to ensure that there are no security violations. The browser is always a hostile environment. The server is in charge of making the program do what it is coded to do, in a time frame that is transparent to the user. Also, it is the only one allowed to operate on the database.

You can imagine, just for practical purposes, that it is like taking all the attributes of the transaction structure and building with them a SDT that is loaded with the values that the user interactively gave them in the form (the ones that matter are the non-virtual ones; that is, the ones that will be physically in the table. Here they are these ones).

With this structure loaded on the server everything is executed from scratch: the validations, rules and formulas, passing through each element just as the user did interactively.

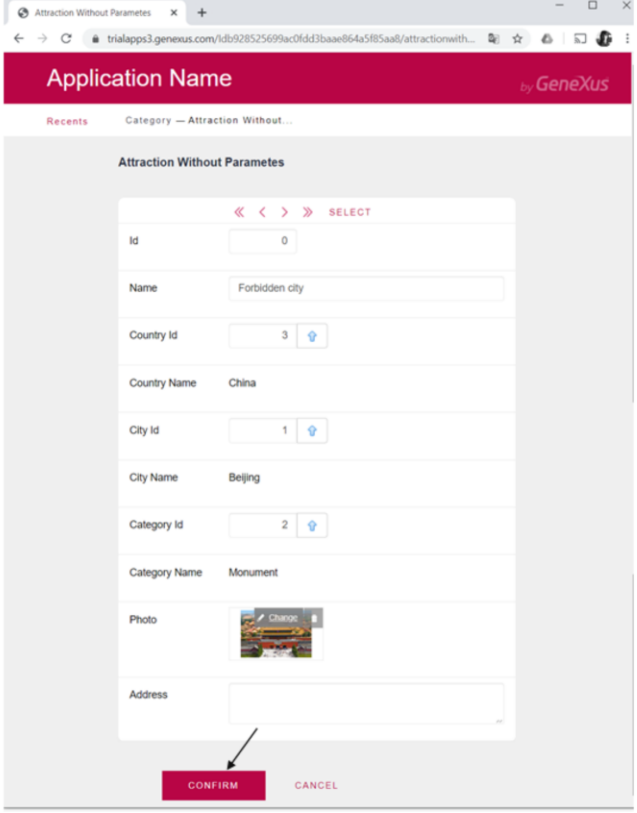
If it finishes without any errors, then it inserts the record in the database.



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 Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...

AttractionId	0
AttractionName	
CountryId	3
CountryName	
CityId	1
CityName	
CategoryId	2
CategoryName	
AttractionPhoto	
AttractionAddress	

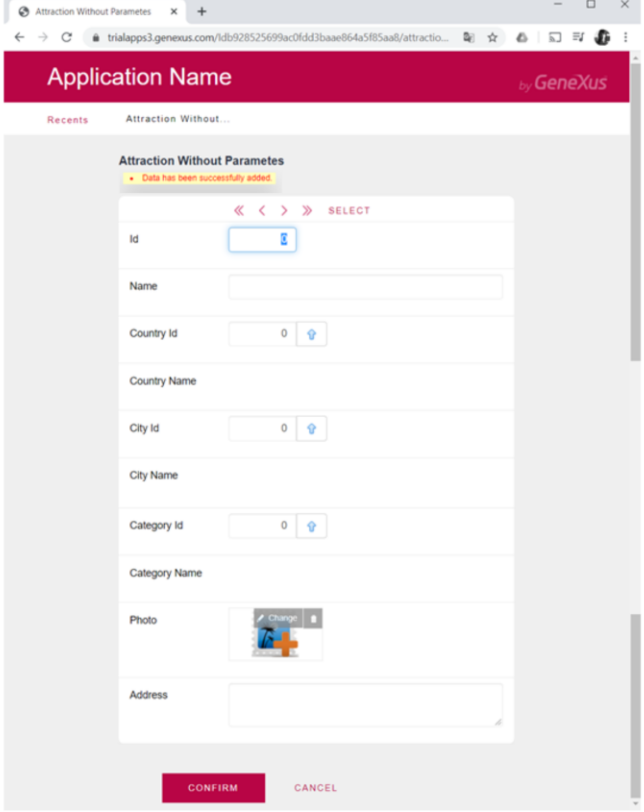
If after pressing Confirm we leave the AttractionName empty, the error rule will be triggered and the insertion in the database will not be allowed, showing the error to the user in the browser. The same will happen if we leave a non-existent foreign key value.



AttractionId	AttractionName	CountryId	CityId	CategoryId	...
1	Louvre Museum	2	1	1	...
2	The Great Wall	3	1	2	...
3	Effel Tower	2	1	2	...
4	Christ the Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	forbidden City	3	1	2	...

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

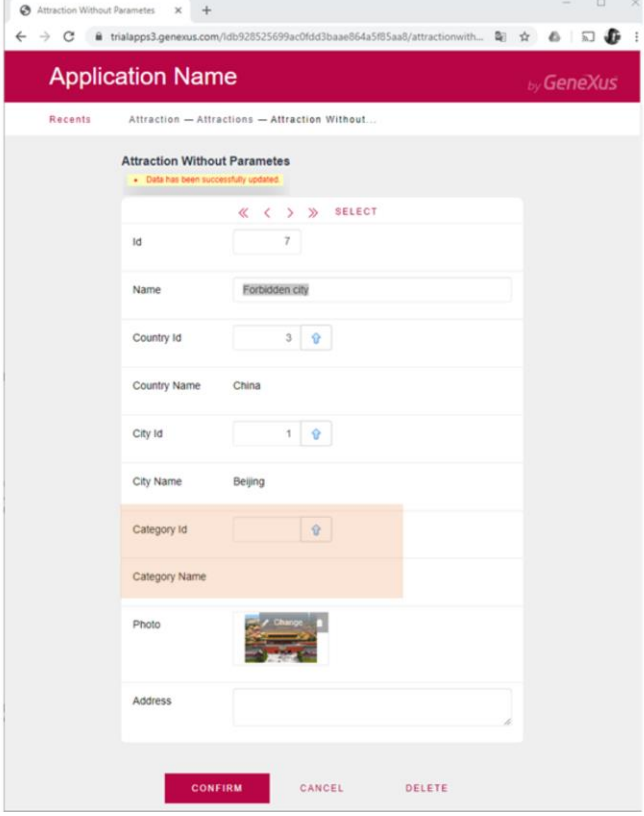
But if everything goes well, in this case as AttractionId is auto-numbered, when it is inserted in the table it will be given the corresponding number and the SDT will also be updated, with that data and with the corresponding inferred data and formulas (there are no formulas here) in case something else has to be done with it (if it were a two-level transaction, we would still need to work with the lines).

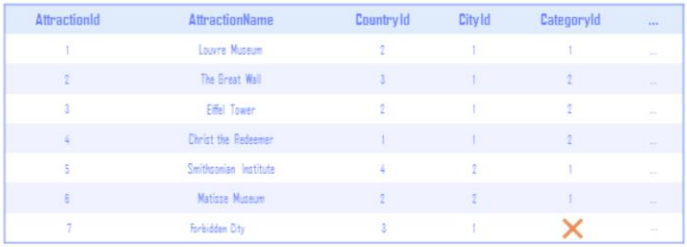


AttractionId	AttractionName	CountryId	CityId	CategoryId	...
1	Louvre Museum	2	1	1	...
2	The Great Wall	3	1	2	...
3	Effel Tower	2	1	2	...
4	Christ the Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	forbidden City	3	1	2	...

AttractionId	
AttractionName	
CountryId	
CountryName	
CityId	
CityName	
CategoryId	
CategoryName	
AttractionPhoto	
AttractionAddress	

What we see at runtime is that after the insertion the screen is empty, with the message that the data was successfully entered. The transaction returns to Insert mode; that is, it is ready again for the user to enter a new attraction. We can also think that this structure is deleted from the server, to be created again when the process is restarted.





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 Redeemer	1	1	2	...
5	Smithsonian Institute	4	2	1	...
6	Matisse Museum	2	2	1	...
7	Forbidden City	3	1	✕	...

AttractionId	7
AttractionName	Forbidden City
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	✕
CategoryName	Monu...
AttractionPhoto	
AttractionAddress	

If now in the key we enter this value, 7, and exit the field... we will see this in the browser. The transaction will have gone to the server, which in turn will go to the database to query for the existence of a record with that value. It will find it. Again, we can think that it loads all its values (the physical ones, inferred ones, and formulas) in a structure like the previous one and sends it to the client, with the information that now the transaction will be in Update mode.

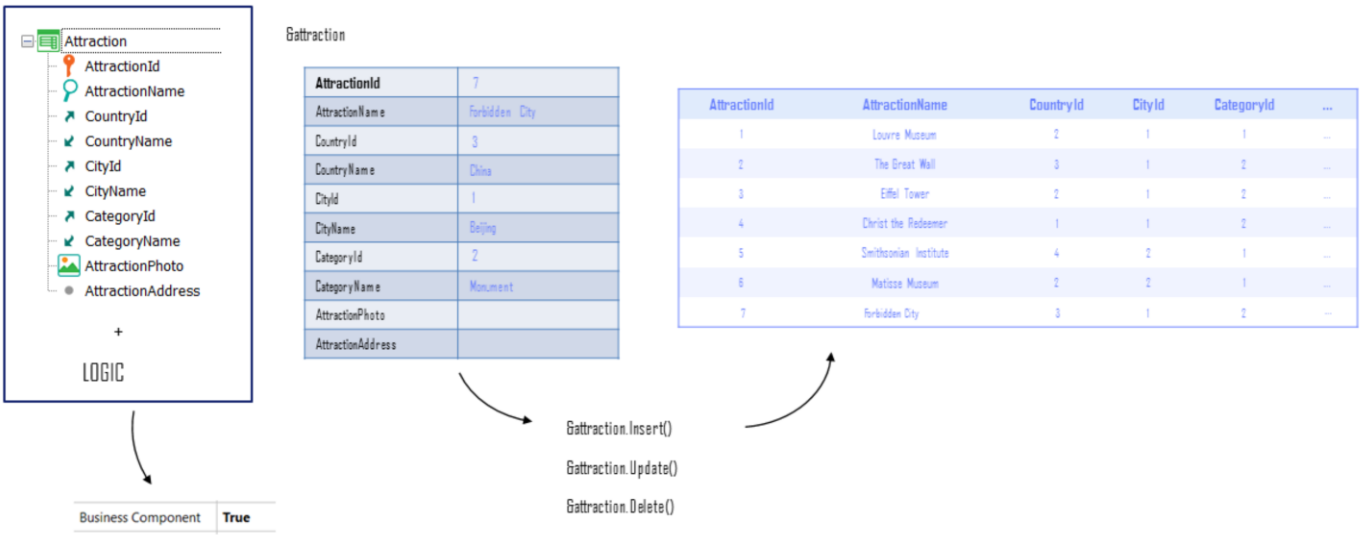
And again, the user will interactively make the desired changes; for example, delete the category (which will be allowed because it accepts nulls). After confirming, everything is done again on the server: the database record is loaded, the changes made in the client are applied, and field-by-field validation is performed, triggering the corresponding rules. If nothing fails, the table record will be updated –in this case, by removing the category–, and the structure will be updated, also removing the category name, which was inferred.

If no other rules in the transaction prevent it, the browser will show the updated information and a message indicating this. Note that the transaction is in Update mode. We could change this record again, for example, by adding the category again.

If now we wanted to delete it, it would be enough to press the Delete button, and in the server, with the structure already loaded it would be easy to look for attraction 7 in the table to delete it.

This makes it quite clear that if we manage to work with a structured variable like the one we imagine, which uses the transaction internally in the server, encapsulates the rules of the transaction, and also allows performing operations on the table, we could insert, change, and delete data from the database through code, complying with the logic declared in the transaction.

Business Component



This is none other than a Business Component.

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, its rules and some of its events), a kind of data type similar to a 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.

GeneXus™

training.genexus.com

wiki.genexus.com

training.genexus.com/certifications