Database Update

Using Single-level Business Components (REVIEW)

GeneXus

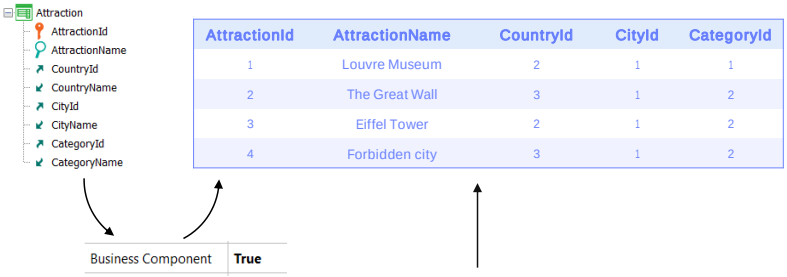
Database update

Insert, Update, Delete

Save()

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

InsertOrUpdate()



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

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

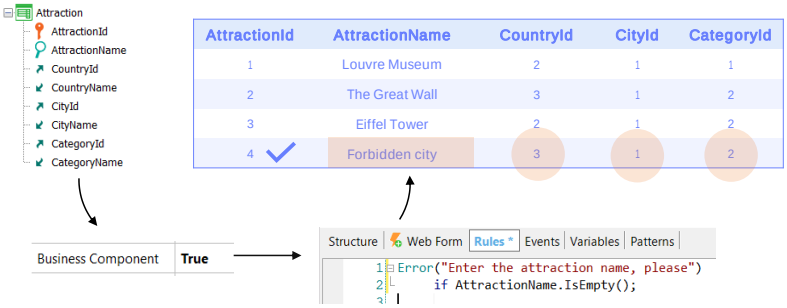
To update the database information using code, there are two possibilities:

Do so using the Business Component of the transaction, through its Insert, Update or Delete methods (or with Save or even InsertOrUpdate instead of Insert or Update), or do it exclusively within a procedure, through the New command, For each command with direct assignment of the attributes to be modified, and the Delete command within a For each to delete the record in which you are positioned.

Database update

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

Insert, Update, Delete



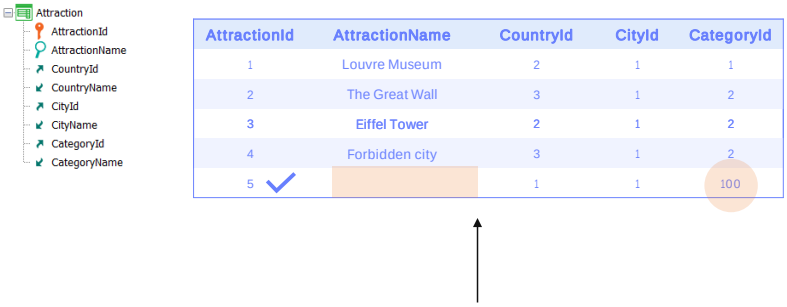
2. **Procedure**: New, For each, Delete

The huge difference between these two alternatives is that while the first one is strongly linked to the logic of the transaction, because the rules, including the control of duplicates and referential integrity, are triggered...

Database update

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

Insert, Update, Delete

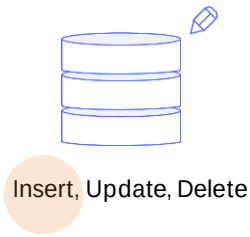


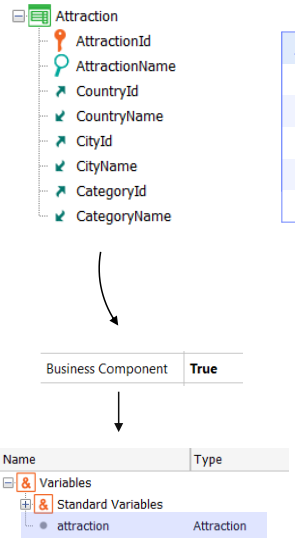
2. **Procedure**: New, For each, Delete

...in the second one the update is independent of the transaction, so no rule will be triggered, and the only control that will be performed is the duplicate control: in this way, you could assign a non-existent category, which the program will not check. However, the database will check it and the execution will be interrupted with an error screen in the user's browser.

So, let's continue to look into the first alternative, that of Business Components.

BC: Insert





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
5	Christ the Redeemer	1	2	2

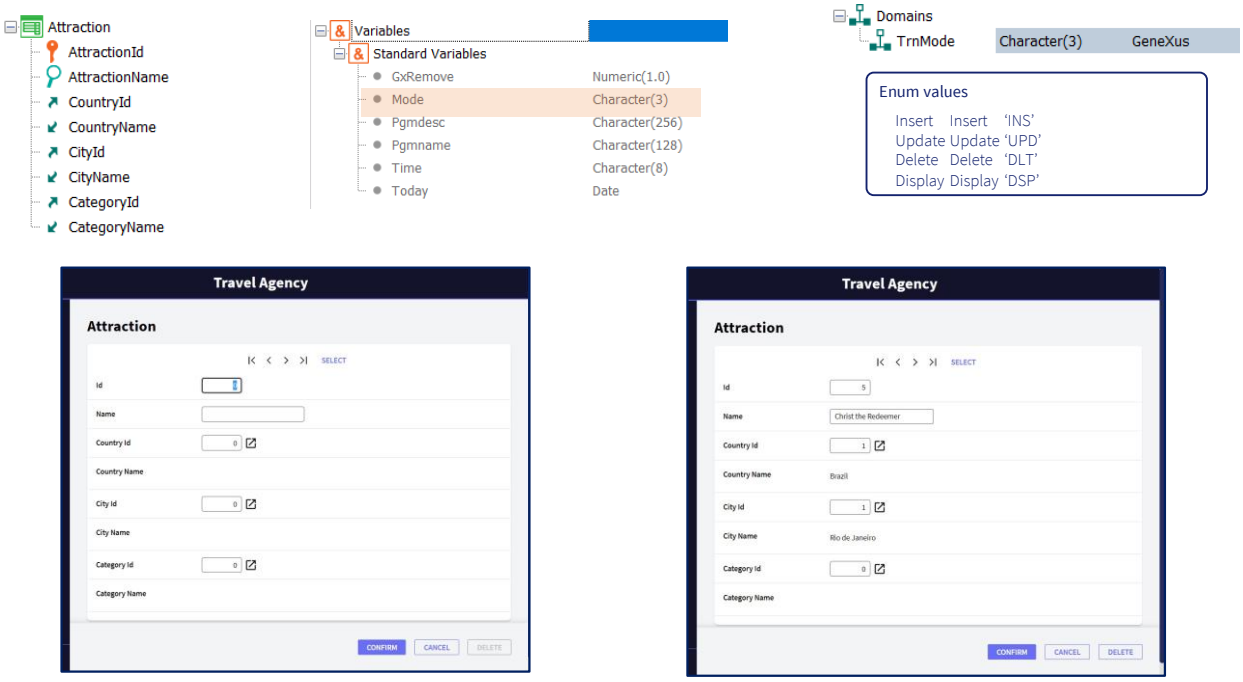
```
&attraction = new()

&attraction.AttractionId        = 5
&attraction.AttractionName  = "Christ the Redeemer"
&attraction.CountryId           = find( CountryId, CountryName = "Brazil" )
&attraction.CityId        = find( CityId, CityName = "Rio de Janeiro" )
&attraction.CategoryId          = find( CategoryId, CategoryName = "Monument" )

&attraction.Save()
if &attraction.Success()
   Commit
endif
```

To insert a new tourist attraction using the business component, it is enough to:

- Create a variable of this type,
- Assign a new memory space to it (it is not essential but it is a good practice to make sure that, no matter what has happened before, the variable at this time will be completely new).
- Assign a value to all the elements of the Business Component corresponding to attributes in the table:
    - if the identifier is autonumbered, there is no need to assign it a value: the database will do it when you insert it,
    - if no value is assigned to any attribute in the table, it will remain empty or null. This is not a problem in general, except for the case in which the attribute is a foreign key and does not admit nulls; there, if you try to insert it, you will get a reference integrity failure error.
- Finally, after loading the Business Component structure, the only thing left to do is to invoke the Insert method, and
- Commit if you want when the insertion is successful.

This is the equivalent to using the Save method; since in this case the Business Component variable is in Insert mode, it will try to insert (and not do an update).

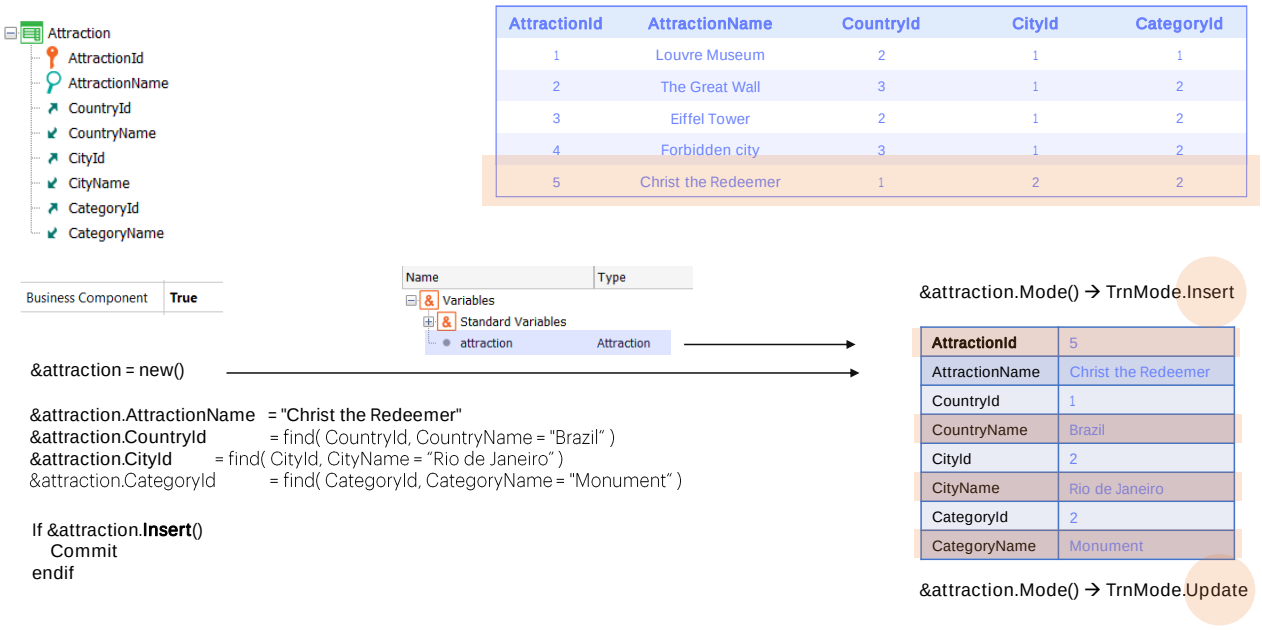
Mode



Remember that among the standard variables of every transaction, we will find the one named Mode. This variable contains at all times the mode in which you are executing the transaction.
In order to handle its values at a high level, the GeneXus module incorporates the TrnMode enumerated domain, which can take the 4 values: Insert, Update, Delete, and Display –which indicates that the information is being displayed but nothing will be done with it.

When the transaction is opened –if you don't specify it to receive mode and identifier as parameters– it does so in Insert mode. That's why the fields are empty. When you exit the identifier, a search is made for a record with the value you left it in –in this case 0– and since it can't be found, the transaction remains in Insert mode. If there were rules conditioned with If Insert they would be triggered. After saving. it informs that the data has been inserted. Also, the form has been emptied again, which means that the transaction has been left in Insert mode again. As you will see, this will not happen when you insert through the Business Component, which will remain in Update mode.
If you now choose an existing value in the database for the identifier, for example 5 which is the one that has just been inserted, when you leave the field the transaction brings its values loaded in the fields on screen and automatically remains in Update mode. You can change something, for example, remove the category, leaving it empty (assuming that you accept nulls in that foreign key). After confirming you are informed that the record was successfully updated, but also you are positioned on the same record, in Update mode. Here, the Business Component will behave in the same way.

Mode()



Let's see what happens when working with the Business Component.

The Mode() method allows you to check which mode it is in. It is read-only.

If you define an &attraction variable and check what mode it's in before you do anything else, you'll see that it's in Insert mode.

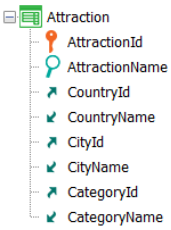
Every time you restart it with New it will be in that mode.

Then you have to enter the values you want to give to the attributes of the associated table (only those you don't want to leave empty). Throughout this time the business component variable will remain in Insert mode.

What happens once the Insert method is executed?
The insertion in the database will be attempted, executing the corresponding rules. If the process is successful, all the elements of the business component variable are loaded with the corresponding values. Since AttractionId is autonumbered, you get the value that was assigned to it in the database, and the attributes that are inferred in the transaction are also loaded here, so they may be queried.
In addition, the mode of the business component variable is changed to Update.

BC: Update

Insert, Update, Delete



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redeemer	1	2	2

Business Component	True

```
&attraction = Load(2)
&attraction.CategoryId  = find( CategoryId, CategoryName = "Tourist site" )

&attraction.Save()
if &attraction.Success()
  Commit
endif
```

&attraction.Mode() → TrnMode.Update

AttractionId	2
AttractionName	The Great Wall
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	3
CategoryName	Tourist site

On the other hand, if what we wanted was to modify the tourist attraction of ID 2 by changing its category to "Tourist site," it was enough to:

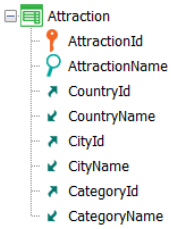
- Load in the &attraction Business component variable the values of the record with primary key 2 of the Attraction table (using the Load method, which automatically left the variable in Update mode if that record existed),
- Assign a value to all the elements of the Business Component to be modified (in this case only CategoryId, as we want the others to keep the value they had).
- Lastly, invoke the Update method to perform this update in the database, and
- Commit if the operation was successfully executed.

Once the Update is done, the CategoryName element, which is inferred in the transaction, remains in the variable with the correct value. The variable remains in Update mode.

Here too, it is equivalent to using the Save method; in this case, since the business component variable after the Load was left in Update mode, the Save will try to update and not insert.

BC: Update

Insert, Update, Delete

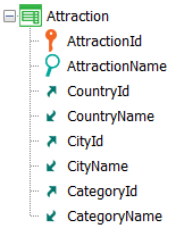


AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redeemer	1	2	2

```
For each Attraction
Where CountryName = "China" and CityName = "Beijing"
Where CategoryName = "Monument"

    &attraction.Load(AttractionId)
    &attraction.CategoryId = find( CategoryId, CategoryName = "Tourist site" )        st site" )

    If &attraction.Update()
       Commit
    endif

                endfor
```

If what we wanted was to change the category of all the tourist attractions in Beijing that were monuments, to assign them the "Tourist Site" category, then it was enough to place the previous code inside a For Each command that selects only the desired records, and the Load is made for each AttractionId of those records.

BC: Update



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redeemer	1	2	2

Insert, Update, Delete

&attraction.Mode() → TrnMode.Update

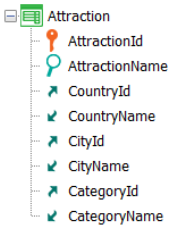
```
For each Attraction
Where CountryName = "China" and CityName = "Beijing"
Where CategoryName = "Monument"

   &attraction.Load(AttractionId)
   &attraction.CategoryId = find( CategoryId, CategoryName = "Tourist site" )

   If &attraction.Update()
      Commit
   endif

endfor
```

AttractionId	2
AttractionName	The Great Wall
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	3
CategoryName	Tourist site

Thus, first the Load of that of ID 2 is made, the variable remains in Update mode, then its CategoryId is modified, and when executing the Update method the record in the table is updated and the variable remains with the corresponding CategoryName.

BC: Update

Insert, Update, Delete



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redeemer	1	2	2

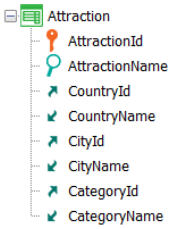
```
For each Attraction
Where CountryName = "China" and CityName = "Beijing"
Where CategoryName = "Monument"

   &attraction.Load(AttractionId)
   &attraction.CategoryId = find( CategoryId, CategoryName = "Tourist site" )

   If &attraction.Update()
      Commit
   endif

endfor
```

&attraction.Mode() → TrnMode.Update

AttractionId	4
AttractionName	Forbidden city
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	3
CategoryName	Tourist site

And then the same for the next record that meets the conditions; that is, the one with ID 4.

BC: Delete



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Christ the Redeemer	1	2	2
5	Christ the Redeemer	1	2	2

Insert, Update, Delete

```
&attraction.Load(2)
&attraction.Delete()

If &attraction.Success()
   Commit
endif
```

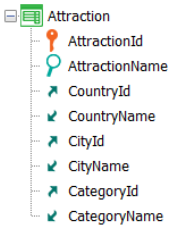
&attraction.Mode() → TrnMode.Update

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

&attraction.Mode() → TrnMode.Delete

On the other hand, if we wanted to delete a tourist attraction, for example, the one with the identifier 2, we first had to load it into the Business Component variable, after which it remains in Update mode and then use the Delete method, which deletes it from the table and leaves the variable loaded with the data but in Delete mode. Then it will be checked for Success so as to perform a Commit.

BC: Delete



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
3	Eiffel Tower	2	1	2
5	Christ the Redeemer	1	2	2
4	Forbidden city	3	1	2
5	Christ the Redeemer	1	2	2

Insert, Update, Delete

```
For each Attraction
Where CountryName = "China" and CityName = "Beijing"
Where CategoryName = "Monument"

    &attraction.Load(AttractionId)
    &attraction.Delete()

    If &attraction.Success()
       Commit
    endif

endfor
```

But what if we wanted to eliminate all the monument type attractions in Beijing? Again, we would place the attractions one by one with the For Each command and load them with the Load action and then use the Delete method to remove them.

GeneXus™

training.genexus.com
wiki.genexus.com