Database Update

Business Components: Differences Between Methods

Methods to operate on the database

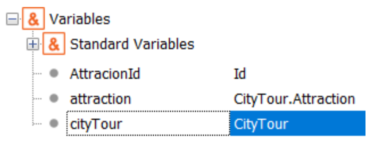
- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

```
&cityTour ...
...
&cityTour.Save()
```

```
&cityTour ...
...
&cityTour.Insert()
```

```
&cityTour ...
...
&cityTour.Update()
```

TrnMode.Insert



Variables	
Standard Variables	
AttracionId	Id
attraction	CityTour.Attraction
cityTour	CityTour

```
&cityTour = new()
```

TrnMode.Udpate

```
&cityTour.Load(2)

&cityTour.Insert()

&cityTour.Update()
```

TrnMode.Delete

```
&cityTour.Delete()
```

Let's go back to the end of the video where one- and two-level business components were compared and continue.

“Methods have been used repeatedly: the Load method to load existing information into a BC, combined with the Update() or Delete() methods to update or delete data, and the Insert() method to insert it.

But other ways can be used. You just have to be clear about the differences and use cases.

For example, the Save method avoids specifying the operation in question. It's more like the Confirm button of the transaction. It will depend on the mode the variable is in, whether it will try to insert or update data: if it is in Insert mode it will try to Insert, and if it is in Update mode it will try to update.

For example, every BC variable defined in an object is in Insert mode, and the same happens when new memory space is allocated to it with new().

If the variable is loaded with Load(), it immediately goes into Update mode.
After performing some operation on it, if it is successful, it will be in Update mode if it was inserted or modified, and in Delete mode if the Delete() method was applied.

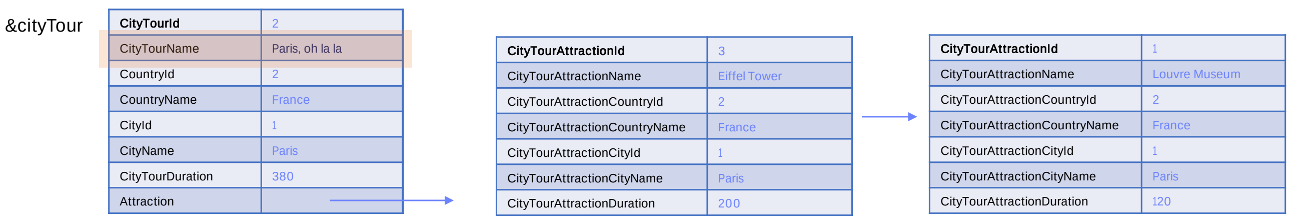
Then, in order to know what operation was attempted when this Save was found, we must know the context of this variable. With the Insert and Update methods, however, this is not necessary. Regardless of the mode, you want to insert or update information in the database.”

GeneXus

Methods to operate on the database

- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

```
&cityTour.Load(2)
&cityTour.CityTourName = “Paris, oh la la”
...
&cityTour.Save()
```



&cityTour

CityTourId	2
CityTourName	Paris, oh la la
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	

CityTourAttractionId	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	200

CityTourAttractionId	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

This example will make it quite clear.

Note that after the Load operation, if there is a city tour 2 the variable will load the information from the database and will be in Update mode, so the Save will want to update. It will only update the elements that are different, in this case, CityTourName.

Methods to operate on the database

- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

```
&cityTour.Load(2)
&cityTour.CityTourName = “Paris, oh la
...
&cityTour.Save()
```

```
&cityTour.CityTourId = 2
&cityTour.CityTourName = “Paris, oh la la
...                    ✗
&cityTour.Save()
```

```
&cityTour.CityTourId = 2
&cityTour.CityTourName = “Paris,
...                    ✓
&cityTour.Update()
```

&cityTour

CityTourId	2
CityTourName	Paris, oh la la
CountryId	
CountryName	
CityId	
CityName	
CityTourDuration	
Attraction	

&cityTour_aux

CityTourId	2
CityTourName	Paris, oh la la
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	→

CityTourAttractionId	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	200

→

CityTourAttractionId	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

But what would happen if instead of using the Load method, a value was directly assigned to the primary key?
It will depend on the mode of the variable. If it was not in Update mode, for example because it was only defined in the object and this is the first thing done with it, then the Save operation will try to insert a city tour 2, which will fail because of a duplicate primary key.

However, if the Update method is used instead of the Save method, there will be no problem. At first glance it might seem so, because the &cityTour variable will have only two non-empty elements. The two lines we know CityTour 2 has are not even there. We can then think that this will make a mess in the database by running the Update operation.
However, Update is a smart operation, and it knows that since the variable is in Insert mode, that means that it doesn’t have all the data loaded (because if a Load operation had been executed, the variable would be in Update and not Insert mode); so, in the database it should only change the data actually assigned in the BC and no other.
Internally, it loads the data into an auxiliary BC variable, which of course remains in Update mode. It changes the data that has been modified in the variable that is in Insert mode –in this case, only the CityTourName–, and performs a Save() operation; since the auxiliary variable is in Update mode because it has been loaded... it performs an update.

In general, it is not good practice to work in this way, directly assigning the primary key of the BC variable without performing a Load operation.

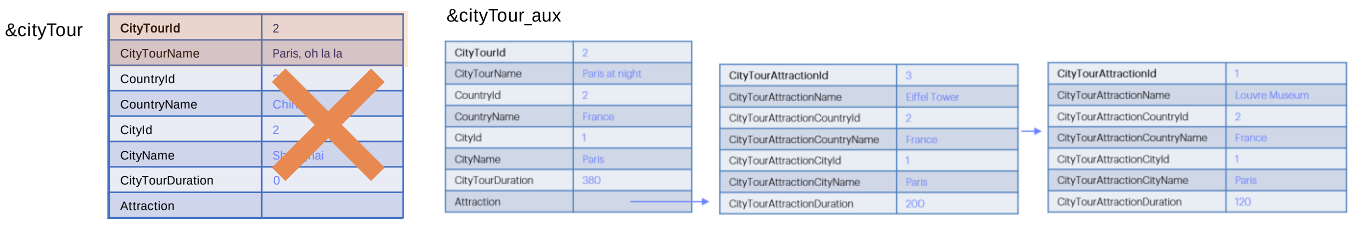
Methods to operate on the database

- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

```
&cityTour.Load(2)
&cityTour.CityTourName = “Paris, oh la
...
&cityTour.Save()
```

```
&cityTour.CityTourId = 2
&cityTour.CityTourName = “Paris, oh la la
...
&cityTour.Save()
```
✗

```
&cityTour.Load(5)
&cityTour.CityTourId = 2
&cityTour.CityTourName = “Paris, 
...
&cityTour.Update()
```
✓



&cityTour

CityTourId	2
CityTourName	Paris, oh la la
CountryId	2
CountryName	Chi
CityId	2
CityName	Sh hai
CityTourDuration	0
Attraction	

&cityTour_aux

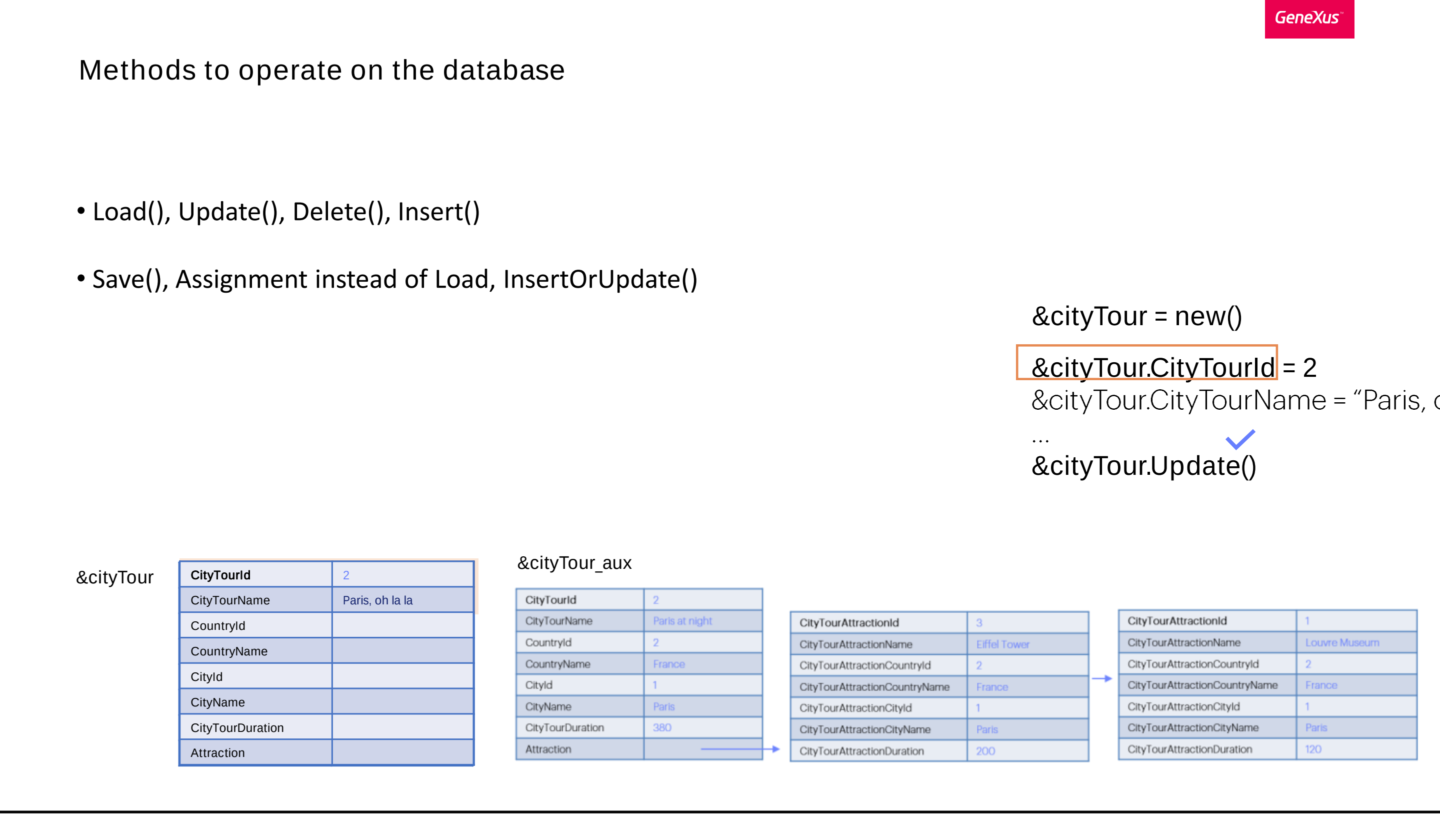
CityTourId	2
CityTourName	Paris at night
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	

CityTourAttractionId	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	200

CityTourAttractionId	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

The reason is that if the variable was in Update mode because, for example, it had been used before (suppose that it had been loaded with the city tour 5 that only has a header), then, if you are not careful, you can leave junk values that come from that previous use.

The assignment is the only way when the Business Component is loaded in a Data Provider, as you will see below, while studying the InsertOrUpdate method.



GeneXus

Methods to operate on the database

- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

```
&cityTour = new()
&cityTour.CityTourId = 2
&cityTour.CityTourName = “Paris, o
...
&cityTour.Update()
```

&cityTour

CityTourId	2
CityTourName	Paris, oh la la
CountryId	
CountryName	
CityId	
CityName	
CityTourDuration	
Attraction	

&cityTour_aux

CityTourId	2
CityTourName	Paris at night
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	

CityTourAttractionId	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	200

CityTourAttractionId	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

If you are going to use the assignment, it is strongly recommended that you first ask for new memory space for the variable, so that it is clean and in Insert mode.

The assignment is the only way when the Business Component is loaded in a Data Provider, as you will see below, while studying the InsertOrUpdate method.

Methods to operate on the database

- Load(), Update(), Delete(), Insert()
- Save(), Assignment instead of Load, InsertOrUpdate()

Insert()
Update()
Delete()
InsertOrUpdate()

FOR A COLLECTION OF BCs TOO!

```
Parm( in: &cityTour )

If &cityTour.InsertOrUpdate()
   Commit
else
   Rollback
endif
```

```
If not &cityTour.Insert()
   for &message in &cityTour.GetMessages()
  if &message.Id = "DuplicatePrimaryKey"
     &cityTour.Update()
  endif
  endfor
endif
If &cityTour.Sucess
   Commit
else
   Rollback
```

This method is used when you don't know if the information you are handling in the BC exists in the database.

For example, in a procedure data is received in a business component variable. We have no idea what information will be loaded in the code. However, we do know that the variable will be in Insert mode, as it is new to this object, even if it is a parameter. It doesn't receive the mode it was in the calling object. It only receives the value of its elements.

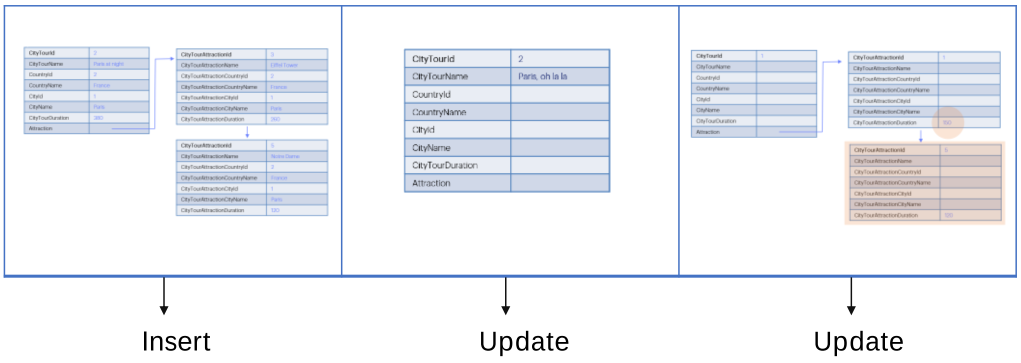
So, if you try to make a Save(), it will only be successful if the BC's primary key is not in the database. Otherwise, it will fail due to a duplicate key (since the variable will necessarily be in Insert mode, the Save operation will try to insert). Therefore, if the object that called the procedure did so to have an update made, we are in trouble.

The InsertOrUpdate method saves us from having to program both possibilities ourselves. That is, try to insert, and if that fails because of a duplicate key, and not something else, then try the Update. This is exactly what InsertOrUpdate() does.

Also, remember that Insert, Update, Delete, and InsertOrUpdate can be used on variables of the business component collection type, which is the other case where this last method will be very useful.

Methods to operate on the database

&cityTours



```
&cityTours = GetCityTours(...) ⟶ Data Provider Source?
If &cityTours.InsertOrUpdate()
    Commit
else
    Rollback
endif
```

Here is a variable of Business Component collection type, which could be loaded based on what a Data Provider returns, or through code, in this same program. In this case, the Data Provider will return a collection of items of the CityTour Business Component type and we will want the database to be impacted by what has been done in each Business Component of the collection.

It could be that the first item of the collection corresponds to information to be inserted (in this case, a header and its two lines); the second item is information to be changed (for example, only the CityTourName of the header); and the last one is also information to be changed (in the example, one line is changed and another one is added).

In this case, the operation will not depend on the mode of each BC variable, because that mode will be Insert in all cases, do you see why?
For this reason, you cannot run through the collection and save (it will only work well for the first element, the one you want to insert, but it will fail for the others, because it will try to insert and will find a duplicate key).

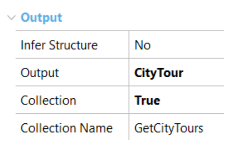
The operation should therefore not depend on the mode of the variable, but on the existence or absence of the primary key in the database. Here the right method, therefore, is again

InsertOrUpdate. When applied to the collection, it will be applied to each item. For the first one, it will try to insert and it will succeed. For the second one, it will try to insert but it will find a duplicate key, so it will try to update, and it will succeed, and the same for the last one.

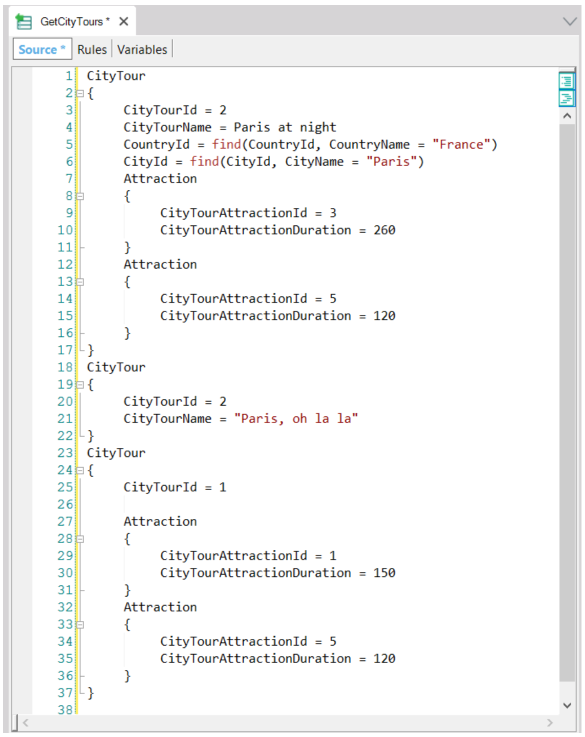
You can ask for the result of the operation, and if it was successful for the entire collection, then you can commit; otherwise, you can roll back if you don't want some records to be changed but not others.

Based on what you have seen here, how would you declare the Data Provider to load the data as shown?

DP Source



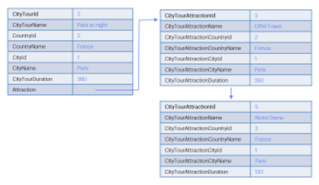
Infer Structure	No
Output	**CityTour**
Collection	**True**
Collection Name	GetCityTours

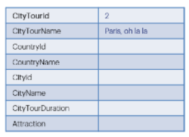


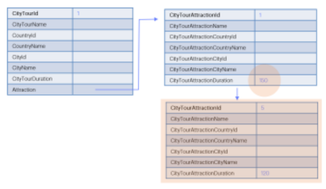
```
CityTour
{
     CityTourId = 2
     CityTourName = Paris at night
     CountryId = find(CountryId, CountryName = "France")
     CityId = find(CityId, CityName = "Paris")
     Attraction
     {
          CityTourAttractionId = 3
          CityTourAttractionDuration = 260
     }
     Attraction
     {
          CityTourAttractionId = 5
          CityTourAttractionDuration = 120
     }
}
CityTour
{
     CityTourId = 2
     CityTourName = "Paris, oh la la"
}
CityTour
{
     CityTourId = 1

     Attraction
     {
          CityTourAttractionId = 1
          CityTourAttractionDuration = 150
     }
     Attraction
     {
          CityTourAttractionId = 5
          CityTourAttractionDuration = 120
     }
}

```







Here it is shown, without going into detail. The Data Provider returns a collection of the CityTour Business Component data type. A Collection!

It is assumed that CityTourId is not auto-numbered, and that 2 does not exist. Note that there are 3 CityTour groups, for the three items in the collection that will be returned. The first one loads all the header data and the two lines.

The second one assumes that the Insert operation on the first one has already been done, so for the same CityTour it will try to change its name.

The third one changes the value of the line of ID 1, and adds the line of ID 5 (here you don't notice the difference, because the stored attributes of the Attraction level are only those two). But if the table had more attributes, they would all appear here (or all those you want to give value to), while here only those you want to change.

GeneXus™

training.genexus.com
wiki.genexus.com