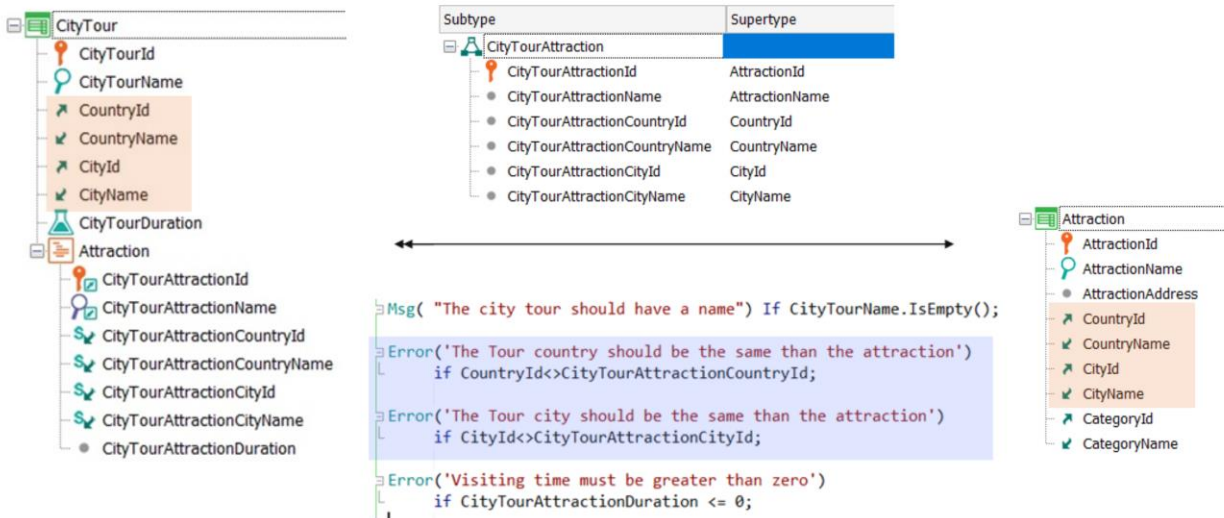


# Database Update

Using Two-level Business Components

*GeneXus*<sup>™</sup>

## Two-level BC



Suppose a CityTour transaction has been created to represent the tours that are offered to the travel agent's customers for visiting the different tourist attractions of a given city.

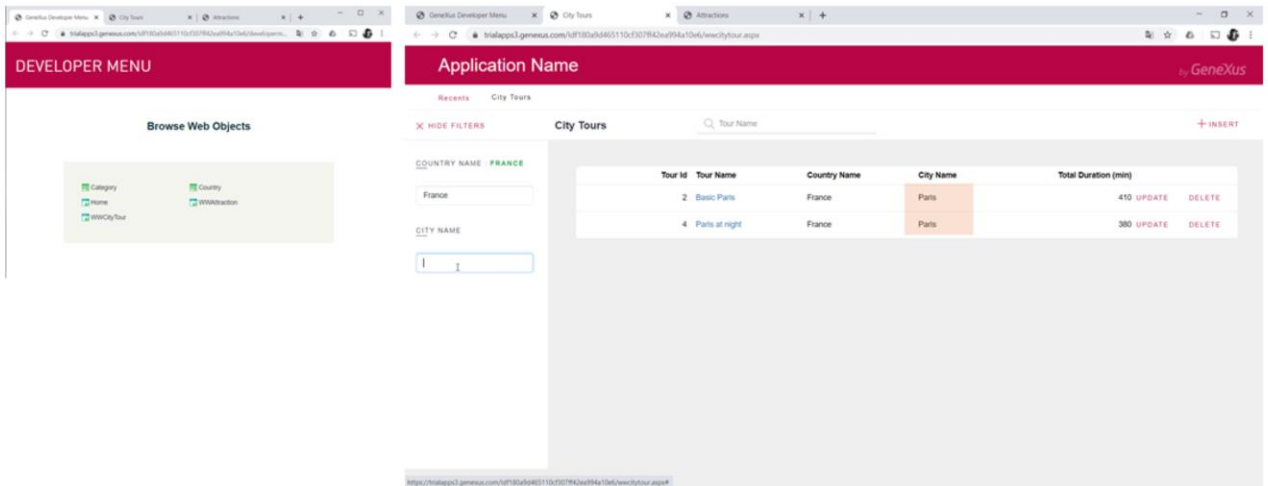
It is a two-level transaction: in the first level, in addition to the name of the tour, you specify its country and city, and the estimated duration of the entire tour, which is the sum of the estimated visiting times for each attraction.

The sublevel indicates the tourist attractions included in the tour, for which it was necessary to define a group of subtypes, since in the first level you need to specify the country and city of the city tour. In addition, attractions have a country and city, which must be checked to make sure they match those of the tour.

Note that CityTourAttractionId is a subtype of AttractionId, so it is like the same attribute, and therefore will be a foreign key in the second level table of CityTour.

That table will also have the secondary attribute CityTourAttractionDuration, to specify how long the visit to that attraction is estimated to last.

## Insert many lines



The pattern Work With a CityTour has been applied, and also to the attractions, in order to work more comfortably. And some data has been loaded. For example, note that for Paris only two city tours have been created. The first one has the Louvre and the Eiffel Tower, and the second one only the Eiffel Tower.

Suppose that when the back-office user is working with the tourist attractions to enter a new one into the system –for example, Notre Dame cathedral– this one should be automatically added to all the city tours corresponding to the city of the attraction –in this case, Paris–, with a fixed duration for the visit of 120 minutes, which can be later modified. How do we get this behavior?

## Insert many lines

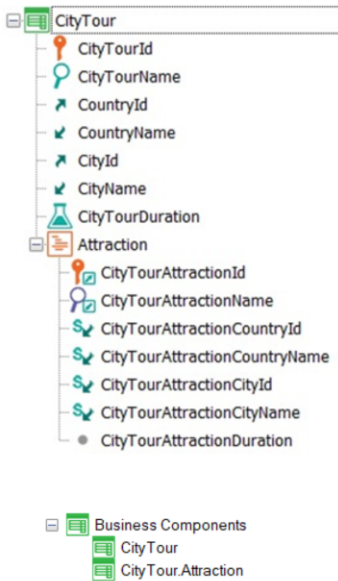
```

Attraction X
Structure Web Form Rules Events Variables Patterns
6 AttractionId = &AttractionId if not &AttractionId.IsEmpty();
7 noaccept(AttractionId);
8 noprompt(AttractionId);
9
10 CountryId = &Insert_CountryId if &Mode = TrnMode.Insert and not &Insert_CountryId.IsEmpty();
11 noaccept(CountryId) if &Mode = TrnMode.Insert and not &Insert_CountryId.IsEmpty();
12 CityId = &Insert_CityId if &Mode = TrnMode.Insert and not &Insert_CityId.IsEmpty();
13 noaccept(CityId) if &Mode = TrnMode.Insert and not &Insert_CityId.IsEmpty();
14 CategoryId = &Insert_CategoryId if &Mode = TrnMode.Insert and not &Insert_CategoryId.IsEmpty();
15 noaccept(CategoryId) if &Mode = TrnMode.Insert and not &Insert_CategoryId.IsEmpty();
16 /* Generated by Work With Pattern [End] - Do not change */
17
18 Error("The attraction name should not be empty")
19     if AttractionName.IsEmpty();
20
21 AddToCityTour(AttractionId)
22     on AfterInsert;
23

```

In short, as soon as a new attraction is inserted through this transaction, it is required that a new line with that attraction is automatically inserted for all city tours that have the same country and city, with a duration of 120 minutes.

Then, on AfterInsert –that is, immediately after the new attraction has been inserted in the Attraction table–, we will call a procedure to which we will pass the identifier of that entered attraction, and that will be in charge of inserting the line for the city tours that it finds with the same country and city.



&amp;cityTour

<b>CityTourId</b>	4
CityTourName	Basic Paris
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	410
Attraction	

&amp;cityTour.Insert()

&amp;cityTour.Update()

&amp;cityTour.Delete()

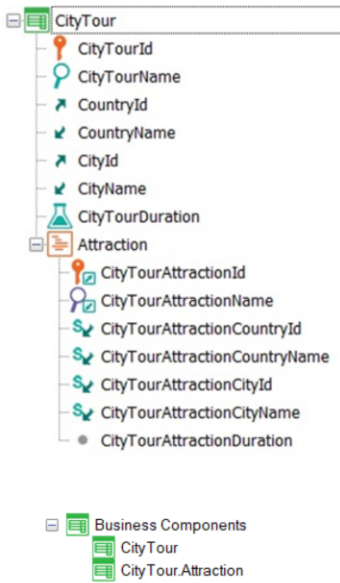
<b>CityTourAttractionId</b>	
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	150

<b>CityTourAttractionId</b>	
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

When you turn on the Business Component property of the transaction, two business components will be automatically created in the KB and not just one, as you might think.

The first is the expected one, with a structure like the one displayed. So, if city tour 4 has two lines, the structure of a variable of the CityTour Business Component data type will be like the one represented here, where the last element, Attraction, is a collection of lines. Each line will correspond, in turn, to a structured data item. With what structure? That of the Business Component CityTour.Attraction, which is the one for each line of the transaction.

That is, while the Business Component corresponding to the transaction as a whole is the one that will allow you to perform the Insert, Update and Delete operations on the database, the one created for the lines is only to be used as a data structure.



&cityTour

<b>CityTourId</b>	4
CityTourName	Basic Paris
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	410
Attraction	

<b>CityTourAttractionId</b>	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	150

&cityTourAttraction

~~&cityTourAttraction.Insert()~~  
~~&cityTourAttraction.Update()~~  
~~&cityTourAttraction.Delete()~~

<b>CityTourAttractionId</b>	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

These other operations will not be allowed. And this may seem confusing at first.

&cityTour

<b>CityTourId</b>	4
CityTourName	Basic Paris
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	410
Attraction	

<b>CityTourAttractionId</b>	1
CityTourAttractionName	Louvre Museum
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	150

<b>CityTourAttractionId</b>	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

&cityTour.Update()

But just as when you want to insert a new line through the CityTour transaction screen you must first instantiate the header, then insert the line and finally select the global Confirm, the same will happen with the Business Components. To work with their lines, for example by adding one, you will have to work with the **header and its lines**, adding the line and then doing the desired operation on everything, that is, on the transaction's BC variable.

## Insert a line



```
&cityTour.Load(2)
```

<b>CityTourId</b>	2
CityTourName	Paris at night
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	

<b>CityTourAttractionId</b>	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

```
&attraction = new()
```

```
&attraction.CityTourAttractionId = 5
```

```
&attraction.CityTourAttractionDuration = 120
```

```
&cityTour.Attraction.add(&attraction)
```

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



<b>CityTourAttractionId</b>	5
CityTourAttractionName	Notre Dame
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

So, to insert a new attraction for the city tour 2, you will need a `&cityTour` variable of the Business Component data type of the `CityTour` transaction. After applying the **Load** operation, we obtain the variable loaded with the data from the database: in this case, the header and its single line.

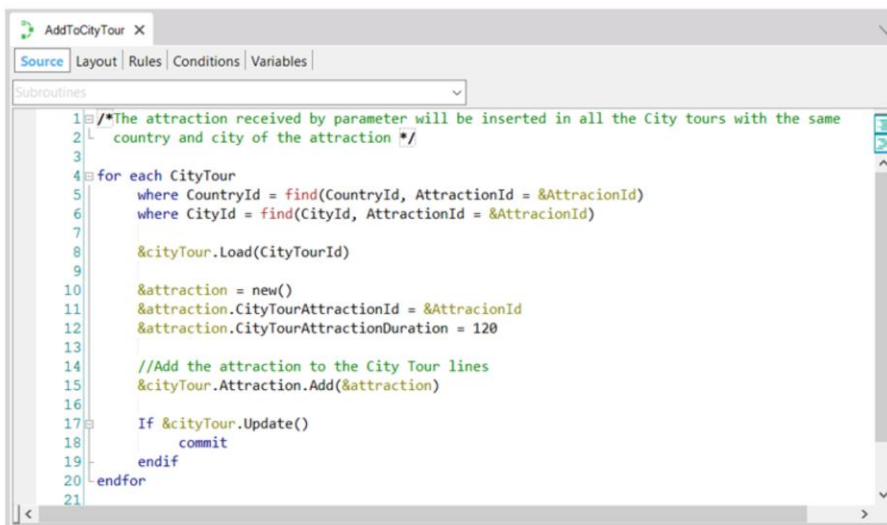
Then, as we will need to add a new line, we define an `&attraction` variable of the Business Component data type corresponding to the lines. After loading the values of its elements, we only have to add this structure to the collection of lines. This is done with the **add method** applied to the `Attraction` collection, no less.

Once the information is as we want it in the `&cityTour` variable, we perform the **Update operation** on it, because the `CityTour` of ID 2 already existed, and we just want to modify it by adding the line. This operation will be the one that actually inserts the line into the database.



## Insert many lines

```
parm(in:&AttracionId);
```



```

1  /*The attraction received by parameter will be inserted in all the City tours with the same
2  country and city of the attraction */
3
4  for each CityTour
5      where CountryId = find(CountryId, AttractionId = &AttracionId)
6      where CityId = find(CityId, AttractionId = &AttracionId)
7
8      &cityTour.Load(CityTourId)
9
10     &attraction = new()
11     &attraction.CityTourAttractionId = &AttracionId
12     &attraction.CityTourAttractionDuration = 120
13
14     //Add the attraction to the City Tour lines
15     &cityTour.Attraction.Add(&attraction)
16
17     If &cityTour.Update()
18         commit
19     endif
20 endfor
21

```

Looking at the procedure implemented, you'll understand everything. The attraction ID is received in the variable &AttracionId.

We are iterating in the For Each command for all city tours in the country and city of the attraction received. Here it is clear why the procedure has to be called after the attraction has been inserted in the Attraction table: otherwise, you will not find its country and city to use this filter.

&cityTour is a variable of the CityTour Business Component data type. It is loaded from the city tour ID found in the For Each command. You just have to add the line with the attraction.

To do so, define a variable of the Business Component data type corresponding to the lines. Create new memory space for this variable, and assign the storable elements: this is the attraction ID and the duration of the visit.

And then you just have to add that line to the CityTour line collection. Finally, perform the Update operation that will return True if it was successful, and in that case select commit.

Then it goes on to the next iteration.

## Delete many lines

The screenshot shows a web browser window with the URL `trialapps3.genexus.com/Id180a9d465110cf307f42ea994a10e6/wwattraction.aspx`. The application header is red with the text "Application Name" and "by GeneXus". Below the header, there is a breadcrumb trail: "Recents Paris at night — City Tours — Basic Paris — Attraction — Attractions". The main content area is titled "Attractions" and contains a table with the following data:

Id	Name	Country Name	City Name	Category Name	Address	UPDATE	DELETE
3	Eiffel Tower	France	Paris	Monument	Champ de Mars, 5 Avenue Anatole France	UPDATE	DELETE
4	Forbidden city	China	Beijing	Monument	4 Jingshan Front St, Dongcheng, Beijing, China	UPDATE	DELETE
1	Louvre Museum	France	Paris	Museum	Rue de Rivoli	UPDATE	DELETE
19	Notre Dame Cathedral	France	Paris	Tourist site	6 Paris Notre-Dame - Pl. Jean-Paul II, 75004 Paris, France	UPDATE	DELETE
2	The Great Wall	China	Beijing	Monument		UPDATE	DELETE

Now let's look at how to remove lines through the Business Component.

The Work with Attractions contains the Notre Dame cathedral that was previously inserted. If you go to the city tours, you'll see that there are two in Paris that contain it: this one and this one. So, what we want now is to be able to delete the attraction. Normally this would not be allowed because of the referential integrity check –to confirm that there are no city tours that contain this attraction– to allow its deletion. So, when pressing Delete, we want to first delete all those lines of city tours where the attraction is located, and then go on to effectively delete the attraction.

I confirm, and we can see now that it was deleted from this city tour, and also from the one we had seen before. The question is how to implement it.

## Delete many lines

```

6 AttractionId = &AttractionId if not &AttractionId.IsEmpty();
7 noaccept(AttractionId);
8 noprompt(AttractionId);
9
10 CountryId = &Insert_CountryId if &Mode = TrnMode.Insert and not &Insert_CountryId.IsEmpty();
11 noaccept(CountryId) if &Mode = TrnMode.Insert and not &Insert_CountryId.IsEmpty();
12 CityId = &Insert_CityId if &Mode = TrnMode.Insert and not &Insert_CityId.IsEmpty();
13 noaccept(CityId) if &Mode = TrnMode.Insert and not &Insert_CityId.IsEmpty();
14 CategoryId = &Insert_CategoryId if &Mode = TrnMode.Insert and not &Insert_CategoryId.IsEmpty();
15 noaccept(CategoryId) if &Mode = TrnMode.Insert and not &Insert_CategoryId.IsEmpty();
16 /* Generated by Work With Pattern [End] - Do not change */
17
18 Error("The attraction name should not be empty")
19     if AttractionName.IsEmpty();
20
21 AddToCityTour(AttractionId)
22     on AfterInsert;
23
24 DeleteFromCityTours( AttractionId )
25     If Delete
26     on BeforeValidate;
27

```

From the Attraction transaction, in Delete mode. We want to delete the attraction and before the data is validated, that is, before the referential integrity checks are made, we call a procedure, passing it the attraction identifier; that procedure is the one that will be in charge of deleting all the lines of city tours that contain it.

How do we do this?

To delete a line using the transaction you have to first load the city tour, which will be in Update mode, then delete the line and finally press Confirm so that the deletion is actually carried out in the database. It will be similar here.

## Delete a line



```
&cityTour.Load(2)
```

<b>CityTourId</b>	2
CityTourName	Paris at night
CountryId	2
CountryName	France
CityId	1
CityName	Paris
CityTourDuration	380
Attraction	

<b>CityTourAttractionId</b>	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

```
&cityTour.Attraction.RemoveByKey(5)
```

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



<b>CityTourAttractionId</b>	5
CityTourAttractionName	Notre Dame
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

To delete the Notre Dame cathedral from the city tour 2, in a variable of the CityTour Business Component data type we load the structure using the **Load method**, and then we delete the line from the Attraction collection. How? With the **RemoveByKey method**. As its name indicates, this method will search the collection for the item corresponding to the indicated key value. In this case, that of ID 5, which is from the Notre Dame cathedral. This deletion has been done only in memory. Now it has to be impacted on the database. To do so, you have to update the city tour, and that's why the Update method is used. Of course, **Save()** could have been used as well. All this is equivalent to having pressed the Confirm button on the transaction.

## Delete many lines

```
parm( in: &AttractionId );
```

```

1 For each CityTour.Attraction
2   where CityTourAttractionId = &AttractionId
3
4   &cityTour.Load(CityTourId)
5   &cityTour.Attraction.RemoveByKey(&AttractionId)
6
7   if &cityTour.Update()
8     commit
9   endif
10
11 -endfor
12
13

```

```

14 CategoryId = &Insert_CategoryId if &Mode = TrnMode.Insert and
15 noaccept(CategoryId) if &Mode = TrnMode.Insert and not &Inse
16 /* Generated by Work With Pattern [End] - Do not change */
17
18 Error("The attraction name should not be empty")
19 if AttractionName.IsEmpty();
20
21 AddToCityTour(AttractionId)
22 on AfterInsert;
23
24 DeleteFromCityTours(AttractionId)
25 If Delete
26 on BeforeValidate;
27

```

If we now go to see how the procedure is implemented in GeneXus: the variable `&AttractionId` receives the attraction identifier to be deleted. A For Each command is used to search in the city tour lines for one corresponding to that attraction ID. If it finds it, in a city tour variable of the Business Component data type corresponding to the transaction we load the `CityTourId` corresponding to that line, and then apply the **RemoveByKey** method to the collection of attractions of that city tour to delete that of the attraction ID received by parameter. Then we perform the Update; if it was successful, we commit.

In this way we make sure that we have deleted all the city tour lines that had that attraction among their data. Then when the execution of this procedure is finished, the control returns here, the validation of the corresponding attraction is made –now it will not have any related data–, and it is deleted without giving any kind of integrity failure.

## Update a line

Application Name

City Tour

Tour ID: 2

Tour Name: Eiffel Tower

Country ID: 2

Country Name: France

City ID: 1

City Name: Paris

Tour Duration: 480

Attraction ID	Attraction Name	Country ID	Country Name	City ID	City Name	Attraction Duration
1	Louvre Museum	2	France	1	Paris	180

City ID: 1

City Name: Paris

Tour Duration: 480

Attraction ID	Attraction Name	Country ID	Country Name	City ID	City Name	Attraction Duration
1	Louvre Museum	2	France	1	Paris	180
2	Eiffel Tower	2	France	1	Paris	200

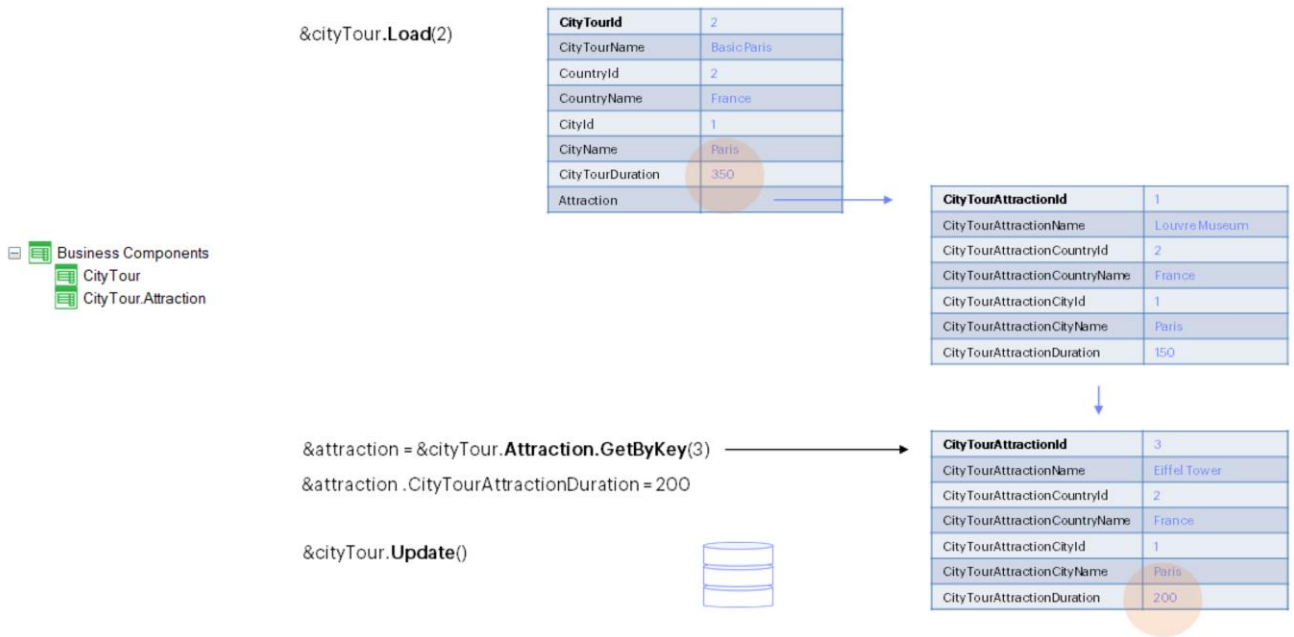
Update

This is the last case to be studied, which is how we update lines of a transaction through the Business Component.

If we did it through the transaction in this case we would edit the TourId 2. Suppose we are interested in changing the duration of the visit to the Eiffel Tower from 260 minutes to 200, and once we do this all that remains is to confirm, so that the update can take place.

We will have to do something like that through the Business Component: load it, modify the line and update.

## Update a line



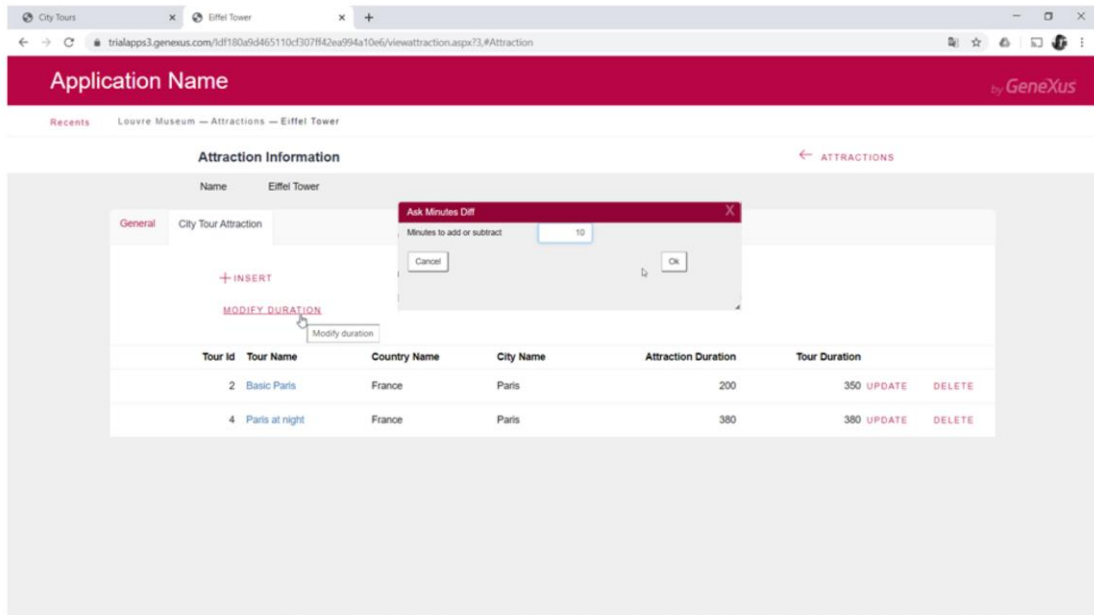
We then load the `&cityTour` variable with the city tour of ID 2.

We must access the item in the collection that corresponds to attraction 3. To this end, the **GetByKey method**, is used. When applied to the Attraction collection, that is, to the transaction lines, it will return a direct reference to that memory position. Therefore it is assigned to the `&attraction` variable of the Business Component data type corresponding to the lines. In this way, the `&attraction` variable will not be a copy of that line, but will be exactly **that line**.

Now the only thing left to do is to change the `CityTourAttractionDuration` element of that variable, and this will have a direct effect on the `&cityTour` variable.

This has to be impacted in the database, and for this purpose we perform the Business Component **Update** operation. Note that, as a consequence, not only is the record corresponding to that line updated in the database, but the `&cityTour` variable is also updated, and the `CityTourDuration` element corresponding to a formula in the transaction header that added up the visiting times of the lines is **also refreshed in the variable**.

## Update many lines



Let's now see how to implement the modification of lines through Business Components in the application.

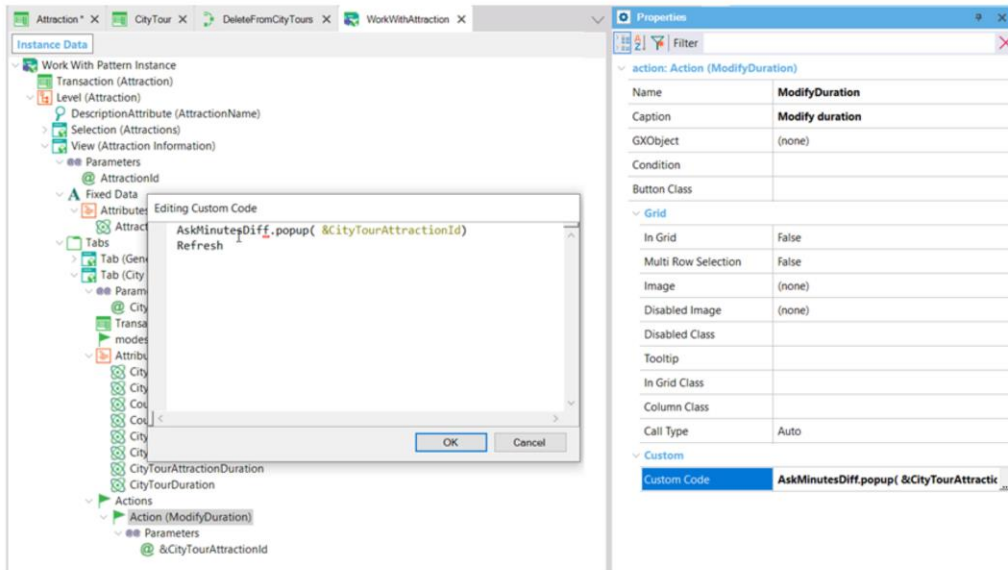
Positioned on Work With Attractions, if you edit a certain attraction, for example the Eiffel Tower, and select the tab that shows all the city tours in which that attraction is located, you'll see that it shows, among other data, the duration in minutes of the visit to that attraction in each of the city tours.

We wanted to make it possible to change that duration from here, by adding or subtracting minutes in all the city tours it is on. So, let's say we want to add 10 minutes to each one. This screen is shown to the user where he/she indicates that he/she wants to add 10 minutes and see that this addition has been made.

What was done in the background? Each of these two city tours was accessed, starting with the first one, accessing the line corresponding to this attraction and adding 10 minutes to the estimated time. And the same thing was done for this one. That is, one line of each of these two city tours had to be modified.



## Update many lines



If you want to see how to implement it in GeneXus, in the Work with Attraction we added an action, `ModifyDuration`, to the tab corresponding to the city tours of the attraction. Its code is the invocation to the web panel of that name, passing it the attraction identifier, and when this web panel is finished running, a refresh is done to show in the grid the refreshed data.

## Update many lines

The image shows the GeneXus IDE interface. On the left, a web form is displayed with a text input field labeled 'Minutes to add or subtract' containing '&mins', and two buttons: 'Cancel' and 'Ok'. Below the form, the 'Events' tab is active, showing an event 'ok' that calls the procedure 'ModifyMinsToCityTours(&AttractionId, &mins)'. On the right, the 'Source' tab is active, showing the code for the 'ModifyMinsToCityTours' procedure:

```

param( in: &AttractionId, in: &minutesToUpd);

1 for each CityTour.Attraction
2   where CityTourAttractionId = &AttractionId
3
4   &cityTour.Load(CityTourId)
5   &Attraction = &cityTour.Attraction.GetByKey(&AttractionId)
6   &Attraction.CityTourAttractionDuration = &Attraction.CityTourAttractionDuration + &minutesToUpd
7   &cityTour.Update()
8   commit
9 -endfor

```

Let's see that web panel, which is the one that asks the user in **this variable** to change the number of minutes and what it does is invoke a procedure that actually changes the corresponding data. The procedure needs to receive the attraction ID and the minutes to add or subtract to that attraction in the various city tours it is on.

Then let's see how to schedule that procedure.

You can see it receives the information in these two variables, and in a For Each command it runs through the attractions of the city tours, filtering by the attraction received by parameter. Then, for each of the records it finds, what it does is load the city tour in question in the &cityTour business component variable; in the &Attraction variable of the type CityTour.Attraction –that is, the business component corresponding to the lines– the item of the Attraction collection corresponding to the &AttractionId key is obtained. Then the duration is changed: the CityTourAttractionDuration element for that item, adding the number of minutes received by parameter to the value it had. Next, an Update is performed and committed.

## Summary

## Insert a line

```

&BC.Load(PKAttribute)

&lineBC = new()

&lineBC.PKLineAtt = ...
&lineBC.LineAtt2 = ...
...
&lineBC.LineAttn = ...

&BC.Lines.Add(&lineBC)

&BC.Update()

```

## Delete a line

```

&BC.Load(PKAttribute)

&BC.Lines.RemoveByKey(PKLineAtt)

&BC.Update()

```

## Update a line

```

&BC.Load(PKAttribute)

&lineBC = &BC.Lines.GetKey(PKLineAtt)
&lineBC.LineAtt2 = ...
&lineBC.LineAttn = ...

&BC.Update()

```

Here is an overview of what has been seen regarding how to insert, delete and change a line in a two-level Business Component.

In all cases the operation used is Update, since it was assumed that the header already existed.

So, as a first step, we always load the business component variable we want to work on. For inserting a line, a variable of the business component type of the lines is created; new memory space is reserved; a value is assigned to all the elements of the line corresponding to attributes of the table to which we want to give a value –those not assigned will be null or empty– and then the **line is added** to the collection of lines. This addition is also for the sake of reference, so we must be very careful to use the new() to create new memory before working with each line.

The **RemoveByKey method** is available for the deletion, which receives by parameter the identifier of the line to be deleted.

And finally, to be able to modify some attribute or attributes of a line, we have the **GetByKey method** of the line collection, which also receives by parameter the ID of the line you want to obtain. This method returns a reference to the business component corresponding to that line, so we must assign the method to a variable of that type. Then, when changing the variable we will already be changing that item in the collection.

## Insert a new header + lines

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

```
&cityTour.CityTourId = 2
&cityTour.CityTourName = 'Paris at night'
&cityTour.CountryId = 2
&cityTour.CityId = 1
```

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



```
&attraction = new()
&attraction.CityTourAttractionId = find(AttractionId, AttractionName = "Eiffel Tower")
&attraction.CityTourAttractionDuration = 260
&cityTour.Attraction.add(&attraction)
```

CityTourAttractionId	3
CityTourAttractionName	Eiffel Tower
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	260

```
&attraction = new()
&attraction.CityTourAttractionId = find(AttractionId, AttractionName = "Notre Dame")
&attraction.CityTourAttractionDuration = 120
```

```
&cityTour.Attraction.add(&attraction)
```



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

CityTourAttractionId	5
CityTourAttractionName	Notre Dame
CityTourAttractionCountryId	2
CityTourAttractionCountryName	France
CityTourAttractionCityId	1
CityTourAttractionCityName	Paris
CityTourAttractionDuration	120

To insert a new header and its lines, for example, the city tour 2, assuming it did not exist, we start by defining the BC variable and assigning it new memory.

Then we assign values for all the elements that correspond to non-inferred header attributes or formulas.

Then we create a new memory space for an &attraction variable of the Business Component type of the lines. We assign the values for the non-inferred attributes and **add** that variable to the **collection of the lines** of the global Business Component.

We do the same to enter a second line. Note that here it is **mandatory** to reserve new memory space, because otherwise, when making the assignments you will be overwriting the previous line.

We add the new variable to the collection of Attractions of the BC and finally we execute the **Insert operation**, so that the header and the two lines are inserted in the database.

If there were no errors, the variable will be loaded with all the data, the ones we entered and the inferred data and formula.

But... is it the only way to insert, update, delete through a BC?

We've seen how to insert, update, and delete lines in a Business Component by directly manipulating the collection of lines in the structure. Is it the only way? The answer is no. Remember that data providers are used to return structured data.



[training.genexus.com](http://training.genexus.com)  
[wiki.genexus.com](http://wiki.genexus.com)  
[training.genexus.com/certifications](http://training.genexus.com/certifications)