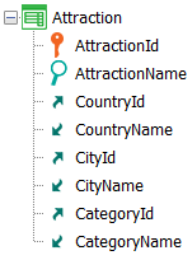
Database Update with Procedure-specific Commands

How to Update Data

GeneXus™

Previously: New Command

	Uniqueness check
New command	✓



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

3

New

AttractionId = 3
AttractionName = "Eiffel Tower"
CountryId = 2
CityId = 1
CategoryId = 3

When duplicate

for each Attraction
CategoryId = 3
endfor

endnew

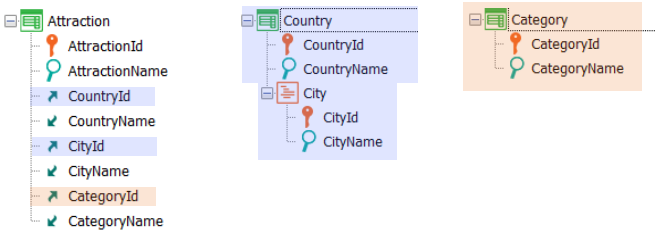
In the video where we studied the New command to insert a record in a table through a procedure, we saw that if the record we wanted to insert was duplicated by primary key or candidate key, then we could choose to modify it, writing a For each within the When duplicate clause.

But what if we already know that the record exists and we precisely want to update it?
In the example, changing the category of tourist attraction 3, Eiffel Tower, which we know exists.

Update

Is there a special Update command in procedures?

For each Command



Insert, Update, 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	Forbidden City	3	1	2

3

For each Attraction
Where AttractionName = "Eiffel Tower"

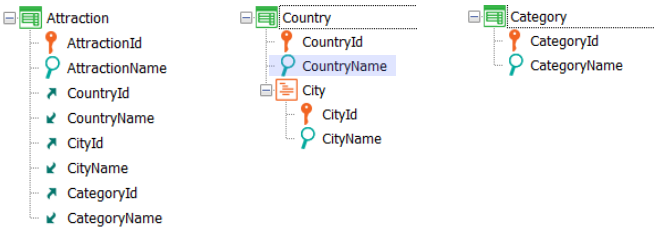
CategoryId = 3

endfor

The answer is NO. The well-known For each command is used to update data. By this we are saying that the For each is not only used to query the database, but also to update it.

In the example, if we want to change the category of the record corresponding to the Eiffel Tower, then it will be enough to write this For each. We are running through the Attraction table, filtering by AttractionName "Eiffel Tower" and for the record found, directly assigning a value to the attribute we want to modify. Of course, we could modify the value of almost every attribute in the record. And not only for it, but also for all related records in the extended table. Here, many records can be updated in a single operation, unlike what happened with the New command.

For each Command



Insert, Update, 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	Forbidden City	3	1	2

CountryId	CountryName
1	Brazil
2	France
3	China

Francia

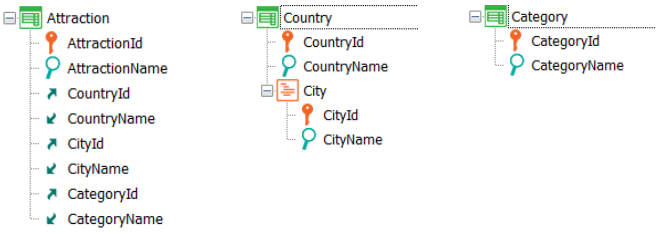
For each Attraction
Where AttractionName = "Eiffel Tower"

CategoryId = 3
CountryName = "Francia"

endfor

For example, we could modify the name of the country (writing it in Spanish – we see that in the table it is in English). Then the For each is positioned in the AttractionName record "Eiffel Tower," modifies its CategoryId attribute, and accesses the related country record, by the extended table, and in it modifies its CountryName attribute, placing "Francia" in Spanish.

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	3
4	Forbidden City	3	1	2

CountryId	CountryName
1	Brazil
2	Francia
3	China

For each Attraction
Where AttractionName = "Eiffel Tower"

AttractionId = 5

CategoryId = 3
CountryName = "Francia"

endfor

The next question is whether all attributes of the extended table can be updated or there are restrictions.

For example, can we update attributes of the primary key? Could we change the attraction ID to 5? No, all the attributes of the record and the related attributes by extended table can be updated except for the primary key. The navigation list will indicate this error.

Therefore, if we need to change the primary key of a record, we will have no choice but to create a new record with the new key and delete the old one.

For each Command

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	Eiffel Tower	2	1	3

Insert, Update, Delete

For each Attraction
Where AttractionName = "Eiffel Tower"

AttractionId = 5
CategoryId = 3

endfor

For each Attraction
Where AttractionName = "Eiffel Tower"

new
AttractionId = 5
CategoryId = 3
endnew

Delete

endfor

So, for example, if we want to change the ID of the attraction named "Eiffel Tower" to 5 and the category to 3, we execute a New command, which we want to have the same base table, Attraction. There we specify the new primary key value, and for the rest of the attributes, we want it to take on the same values as the For each record we are in, except for CategoryId, because we wanted to take the opportunity and change it to 3.
Then, what we will do is to execute the Delete command, which deletes the record in which we were positioned in the For each, that is to say, in our case the one with ID 3.

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Assignment in For each	✓		

Unique Index

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

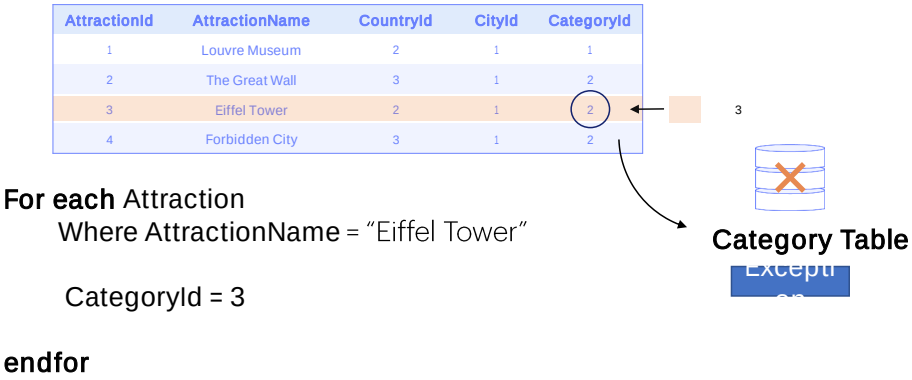
```
For each Attraction
    Where AttractionId = 3

    AttractionName = "Louvre Museum"

endfor
```

```
For each Attraction
    Where AttractionId = 3

    AttractionName = "Louvre Museum"

When duplicate
    ....
endfor
```

As we saw in the case of the New command, the update through For each will perform record uniqueness checks. In this case, it will only apply when there are candidate keys; that is, when there is a unique index defined over some attribute. Suppose this is the case of AttractionName. And that we want to modify in the For each the name of attraction 3, so that it becomes "Louvre Museum."
Before attempting this update, the For each will check, using the unique index, that a record with that value does not already exist. Since in this case it does exist, then it will do nothing. The same thing that happened with the New command.

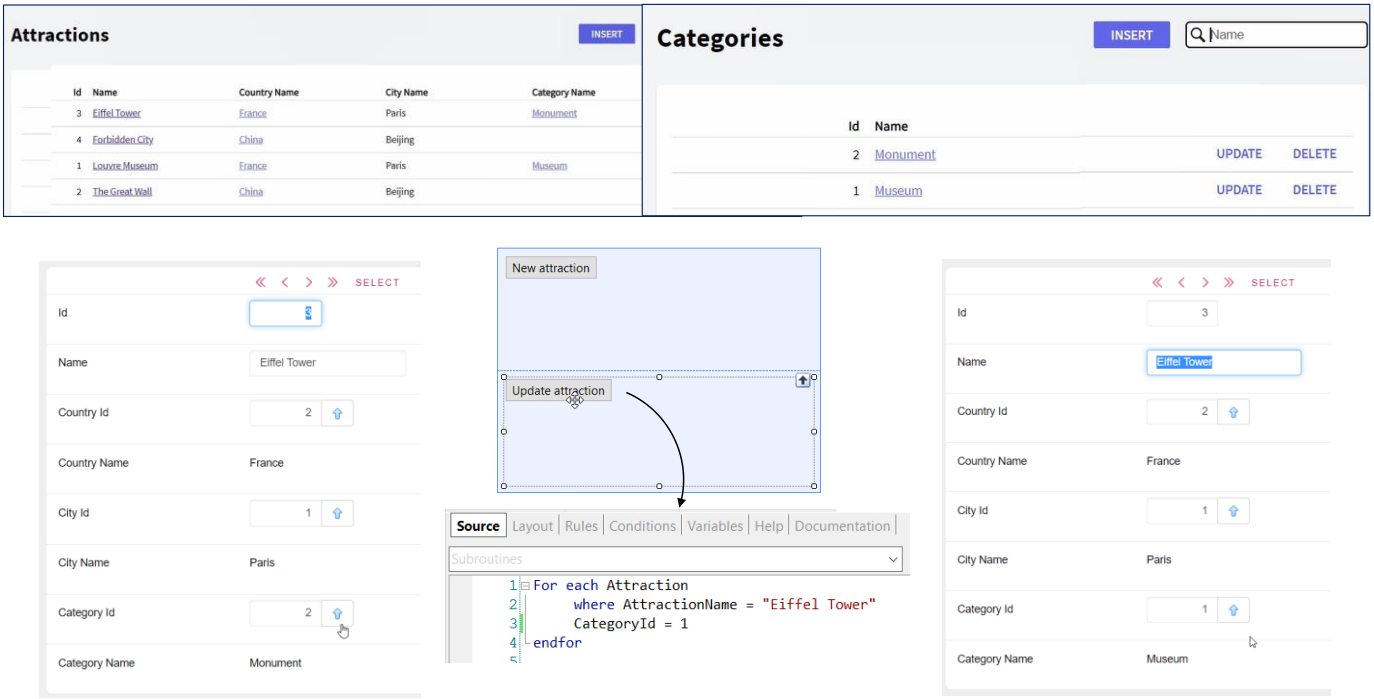
But also, as in the New command, it will do nothing unless... we have programmed a When duplicate clause. In this case, its contents will be executed. We'll go back to this later.

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Assignment in For each	✓	✗	



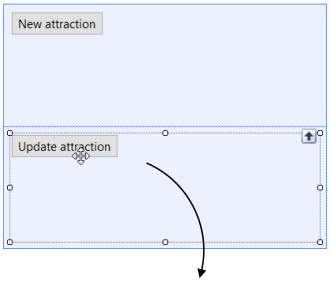
On the other hand, as we saw with the case of the New command, the update through For each will not perform referential integrity checks, so if category 3 does not exist in the Category table, the program will not check it. Again, if the database has referential integrity declared, it will check it, so it will throw an exception and the program will cancel.

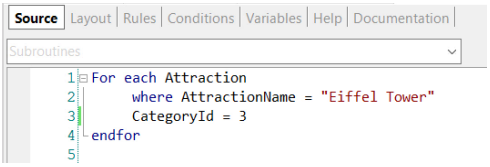
Let's see it in practice.

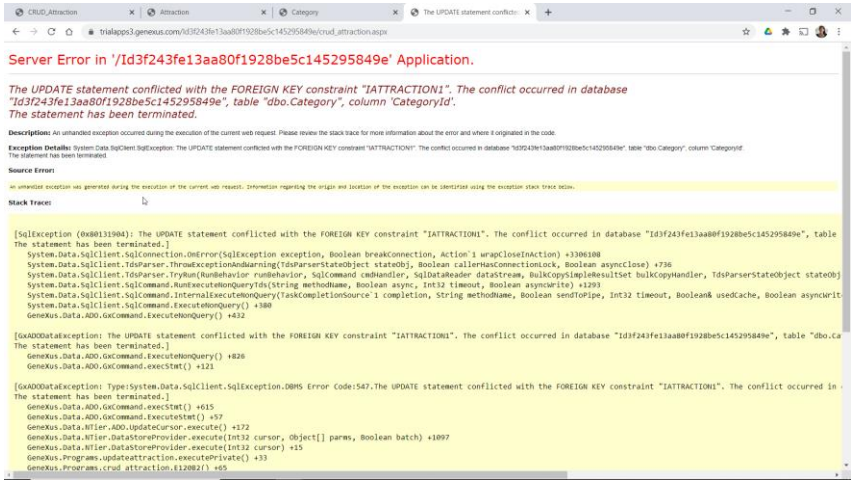


We have the four attractions, and the third one is the Eiffel Tower, which is category 2.
And if we look at the category data we have only 2 categories: 1 and 2.

Now let's see it in GeneXus. We have this web panel, in which we have programmed, in the event associated with the button, the invocation to the procedure UpdateAttraction, in which we have this For each which tries to change the category of the Eiffel Tower to 1, which we know that exists.







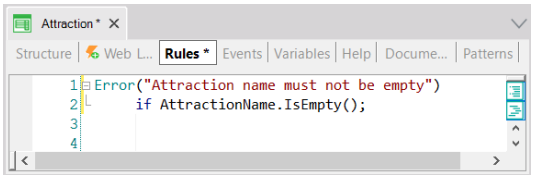
Now let's see what happens if instead of assigning category 1, which we know exists, we assign category 3, which does not exist. Let's try it.

The program crashed because the For each did not perform an integrity check, and tried to perform the update, but the database did perform the check. So this exception was thrown and was not caught by our procedure so as to do something about it. This can be done and will be discussed in another video.

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Assignment in For each	✓	✗	✗

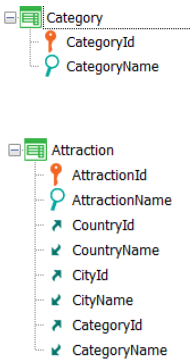
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3		2	1	2
4	Forbidden City	3	1	2

```
For each Attraction
    Where AttractionId = 3

    AttractionName = ""

endfor
```



Of course, just as we said for the case of the New command, the update by assignment within the For each of a procedure will not execute absolutely any transaction rule or event either. Unlike what happens when updating through a Business Component.

For each Command



CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist site

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

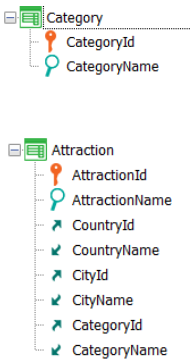
```
New
    CategoryName = "Tourist Site"
endnew

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist Site"
    COMMIT
endfor
```

Here is an example where we insert a new category, Tourist Site, in the Category table, and immediately run through with a For each the tourist attractions of the country China and city Beijing, of category "Monument," changing that category for the new one, "Tourist Site."

Again, this code is only valid in the Source of a procedure. And what happens with the Commit? When are the record inserted in Category and the two records modified in Attraction committed?

If we don't write a Commit command explicitly in the Source, and if we don't modify the default value of the Commit on Exit property of the procedure, then GeneXus will add a Commit at the end. This is because it will have already figured out that you are trying to update the database from the Source. In a procedure where GeneXus doesn't find that the database is to be updated, it doesn't add it, even if the Commit on Exit property is set to Yes.

For each Command

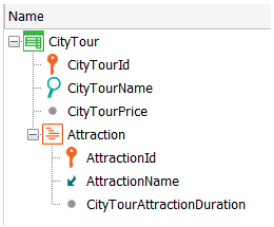


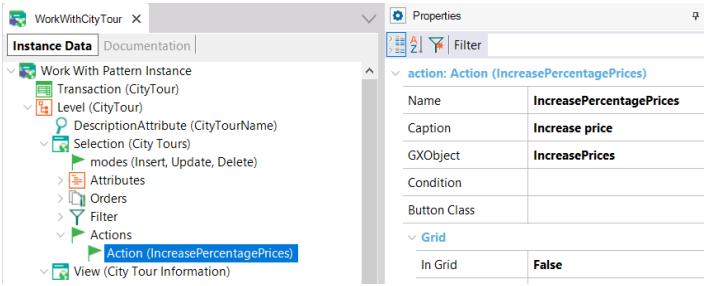
CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist site

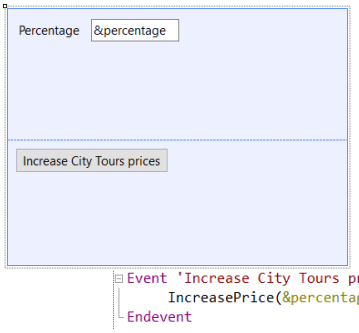
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden City	3	1	3
5	Christ the Redeemer	1	2	2

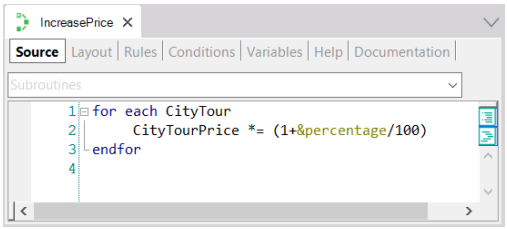
```
New
    CategoryName = "Tourist Site"
Endnew
Commit

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist Site" )
Endfor
Commit
```

Of course, if we wanted a Commit to be made after inserting the category, and then after modifying the attractions, it would be enough to make it explicit.









We have added a CityTour transaction to represent the city tours offered to the public. Each city tour will cover a set of tourist attractions, and will have a given price.
We have applied the Work With pattern and added an action that will call a web panel that we have created to ask the user for a percentage, and call a procedure that will increase the price of all the city tours based on that percentage.

To this end, we run through the CityTour table, and assign to the CityTourPrice attribute as a new value the value it had for this factor here, which is the one that applies the increase. We could also have used the "equal" operator directly to avoid repeating the attribute.

Note that the navigation list informs us which attribute is being updated in the CityTour table.

If we now run it... we see that we have two city tours entered: one with a value of 300, and another one with a value of 200; what we are going to do is run that web panel, and we are going to tell it that we want it to increase by 10%... And now we see how it has indeed done so.

Summary

For each *BaseTransaction*
skip $expression_1$ **count** $expression_2$
order att_{11}, att_{12}, ... att_{1n} **[when** *condition***]**
order att_{21}, att_{22}, ... att_{2n} **[when** *condition* **| otherwise]**
using *DataSelector*(**$parm_1$**, ..., **$parm_n$**)
unique att_1, ..., att_n
where *condition* **[when** *condition***]**
where *condition* **[when** *condition***]**
where *att* **in** *DataSelector*(**$parm_1$**,..., **$parm_n$**)
blocking *NumericExpression*
...
$Attribute_1$ = $expression_1$
$Attribute_2$ = $expression_2$
...
$Attribute_N$ = $expression_N$
...
When duplicate
...
When none ...
endfor

	Uniqueness check	Referential Integrity check
Assignment	✓	✗

COMMIT

In short, to update records specifically by procedure we have the For each command.

In its body, in addition to being able to do other things, both attributes of the base table and the extended table can be updated. The only restriction is that the primary key of the For each base table cannot be updated.

On the other hand, we had seen that the only programmatic control performed is the uniqueness control. In the case of the For each, it will be checked that no candidate key is repeated. If the For each finds that performing the assignment will repeat that key, then it does nothing, unless the When duplicate clause is programmed.
In the code of this clause a New command can be programmed to insert a new record, for example.

We know that the For each doesn't make a referential integrity check, so it will try to perform the update that has been specified without taking into account whether the referenced record exists or not. This is for performance reasons. But databases in general do perform the check, unless we turn off that feature; so, if we don't turn it off, and integrity fails, they will throw an exception.

Finally: for the record to be committed in the database we must make sure that the Commit command is executed. In a

procedure, by default, an implicit Commit is placed at the end (as long as it is understood that the Source is accessing the database somewhere to update it). But we can explicitly write Commits in the Source, where it is convenient for us.

We will not see it here, but optionally you can specify a Blocking clause, which allows you to make updates in blocks, instead of record by record. That is, it will process records in blocks of N records to reduce the number of accesses and improve performance.

Finally, something we haven't said so far is that redundancies are not automatically maintained when updates are made using procedures. It is the developer's responsibility to do so. This means that if a redundancy depends on an attribute that is being updated, GeneXus will not look for or calculate the new value to store in the redundant attribute. The developer has to do it.

More information about all this can be found in our wiki.

GeneXus™

training.genexus.com
wiki.genexus.com