

Database Update

Business Components vs. Procedure-specific Commands

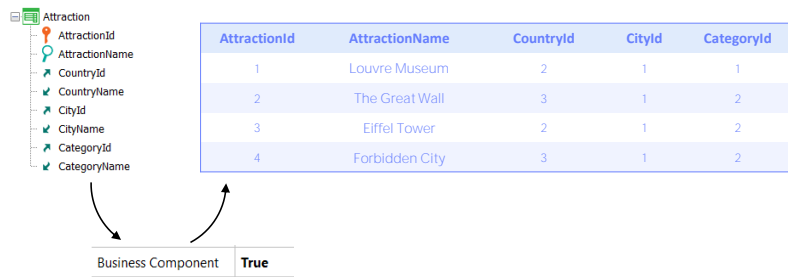
GeneXus™



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



Insert, Update, Delete



2. Procedure: New, Assignment in For each, Delete

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

Using the business component of the transaction, through its methods, or exclusively within a procedure, through the New, For Each commands with direct assignment of the attributes to be modified, and the Delete command.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in 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	Forbidden City	3	1	2

```
&attraction.Load(3)
&attraction.CategoryId = 100
&attraction.Update()
```

```
&attraction.GetMessages()
```

Name	Type	Description	Is Collection
Messages		Messages	<input checked="" type="checkbox"/>
Message			
• Id	VarChar(128)	Id	<input type="checkbox"/>
• Type	MessageTypes, GeneXus	Type	<input type="checkbox"/>
• Description	VarChar(256)	Description	<input type="checkbox"/>

As we saw, both options control the uniqueness of records; that is, no records with a repeated primary or candidate key are ever allowed in the database. In the example, it will never be possible to have two attractions with the same identifier. And if, for example, AttractionName were a candidate key, no two attractions with the same name would be allowed either.

This is automatically checked by the Business Component or by the command in the procedure before making the operation on the Database.

So far it is the same. Now let's look at the differences.

Regarding the referential integrity check, the insertion, update or deletion through a Business Component will check before making the operation that the referential integrity is not broken, just as the transaction does. For example, if we want to change the category of attraction 3 to a non-existent one, the Business Component will first check that the category 100 exists in the Category table and if it doesn't exist, **it will not try** to update it. It will also load an error message indicating it in the collection of messages returned if we ask it to do so.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in 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	Forbidden City	3	1	2

```

For each Attraction
Where AttractionId = 3
  CategoryId = 100
endfor

```

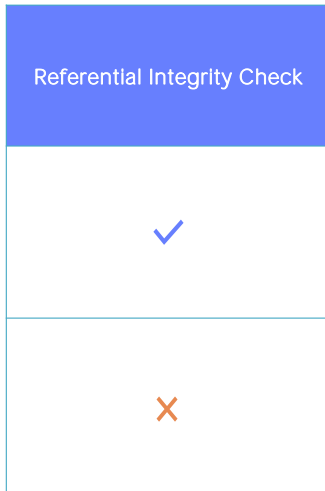


Exception!

On the other hand, the operations performed by procedure-specific commands will not perform any referential integrity check whatsoever. This means that if, for example, we want to update the category of attraction 3, now by assignment within a For Each, the Category table will not be queried to find out if a category 100 exists before the order is sent to the Database to update that record. Therefore, if the database doesn't check the referential integrity either, we will be left with inconsistent data. And if it does check it, as is the default behavior, it will throw an exception and the program being executed will cancel.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	
New, Assignment in For Each, Delete	✓	✗	

So, for the time being, the scale seems to be tipping towards the use of Business Components.



```

For each Attraction
  &attraction.Load(AttractionId)
  &attraction.CategoryId = 100
  &attraction.Update()
endfor

```

VS

```

For each Attraction
  CategoryId = 100
endfor

```



```

If find(CategoryId, CategoryId=100, 0) = 100
  For each Attraction
    CategoryId = 100
  endfor
endif

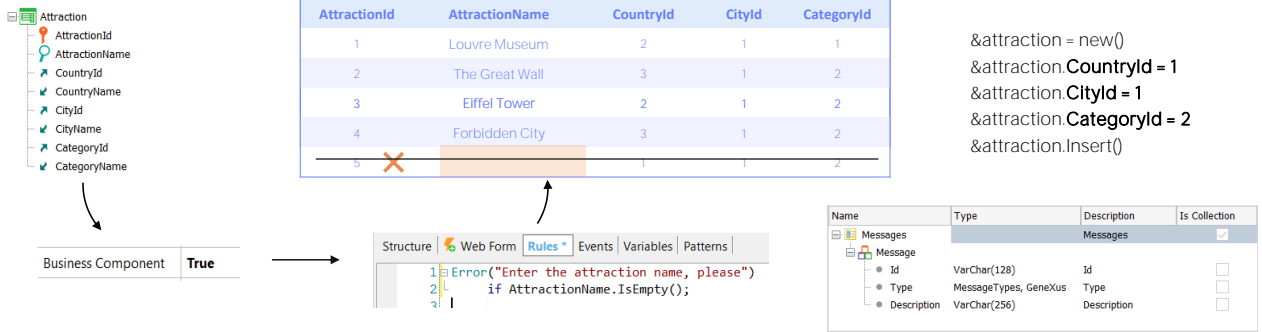
```

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
...

However, if the records to be updated are in the millions or billions, things change a bit. Checking the referential integrity of each record before performing the operation is a time-consuming task. It is negligible in the case of a few records, but very important in the case of millions. In that case the update with procedural commands is THE solution if performance becomes critical.

In the example we could run the For Each only if we are sure that category 100 exists. And we could even improve performance further by using the blocking clause of the For Each.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	✓
New, Assignment in For Each, Delete	✓	✗	

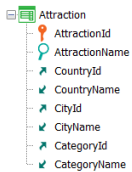


And finally, what about rules and events defined in the transaction?

We know that the update through Business Components will execute them (the corresponding ones, of course, that do not have to do with the transaction interface).

So, if we have an error rule that doesn't allow us to enter an attraction without a name, and we try, for example, to insert a new, auto-numbered attraction through the Business Component, will it be allowed? No, it won't be allowed. And the error will be captured in the message collection.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution
Business Component	✓	✓	✓
New, Assignment in 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	Forbidden City	3	1	2
5		✓ 1	1	2

```
New
CountryId = 1
CityId = 1
CategoryId = 2
endnew
```

```
Structure | Web Form | Rules * | Events | Variables | Patterns
1 | Error("Enter the attraction name, please")
2 |   if AttractionName.IsEmpty();
3 |
```

On the other hand, when it comes to database update commands in procedures, the transaction is not involved at all (or, it is only involved to the extent that it defines the database table, but only for that). So rules or events are not considered at all.

Therefore, in the example, even if there is an error rule, the insertion through the New command in a procedure will not even be aware of it, and will insert the record with an empty name without any problem.

Again, here the scale clearly tips towards using Business Components rather than update commands in procedures, since we ensure that all the data logic is executed.

We could repeat it in the procedure, but we know the problems of duplication.

In addition, executing rules and events obviously takes time. If performance is a problem, then we probably need to update the database with procedure-specific commands.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution	Scope
Business Component	✓	✓	✓	Any Object
New, Assignment in For Each, Delete	✓	✗	✗	Procedure only!

Commit?

Transaction integrity	
Commit on exit	Yes

Rollback?

One last advantage of Business Components over specific update commands is that, while the former can be used in virtually any object that supports code, for example, in Web Panel or Panel events, the latter can only be programmed in procedures.

The same applies to Commit and Rollback for both ways. That is to say, the developer must decide to use them where it is most convenient, regardless of whether procedures have the Commit on Exit property set to Yes by default.

Let's look at a special case that may cause confusion.

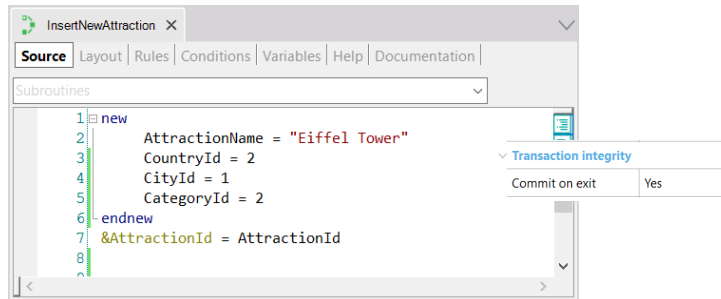
DEMO



```

Event 'New attraction'
  InsertNewAttraction(&AttractionId)
  If not &AttractionId.IsEmpty()
    &text = "The attraction " + &AttractionId.ToString() + " was inserted"
  else
    &text = "The attraction could not be inserted"
  endif
  msg(&text)
Endevent

```



Suppose that from this Web Panel we want to call this procedure that will try to insert an attraction in the database and will return its ID, since it is auto-numbered.

We have to decide whether to insert the record with the New command or with the Business Component. First let's do it with the command.

Since the Commit on Exit property is set to its default value, Yes, and clearly an operation is being performed on the database, then GeneXus places an implicit commit at the end of the procedure source code.

For this reason, we know that when returning from procedure execution, if it was possible to insert the record in this statement then it will have already been committed.

The screenshot displays the GeneXus IDE interface. On the left, there is a 'Pattern:' field and a tree view containing 'InsertNewAttraction'. The main window shows a report titled 'Procedure InsertNewAttraction Navigation Report'. The report details are as follows:

Name:	InsertNewAttraction	Environment:	C# Default (C#)
Description:	Insert New Attraction	Spec. Version:	17_0_3-148529
Output Devices:	None	Form Class:	Graphic
		Program Name:	InsertNewAttraction
		Parameters:	out: &AttractionId

Below the details, there is a 'Warnings' section with a warning icon and the message: **spc0060** The program may be called by another program and the Commit on Exit property is set to YES.

The 'LEVELS' section shows a table with one row:

LEVEL
New Attraction (Line: 1)

The SQL code for this level is:

```

=Attraction ( AttractionId ) INTO CategoryId CityId CountryId AttractionName AttractionId

INSERT INTO Attraction (AttractionName, CountryId, CityId, CategoryId)

```

And that is why the navigation list gives us this warning.

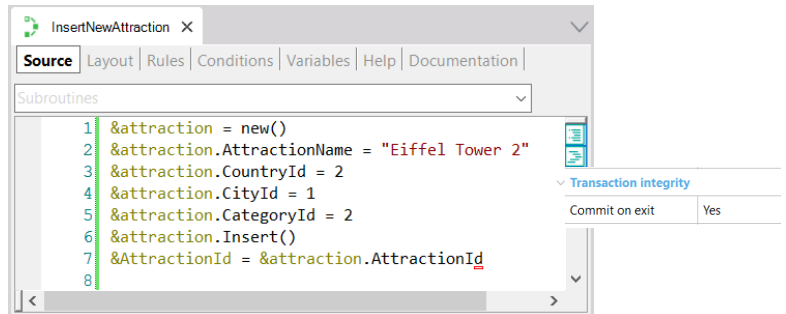
Let's take a look at the attractions in the database. Now we run it. Let's look at the attractions again. As we expected, here we found the new tourist attraction.



```

Event 'New attraction'
  InsertNewAttraction(&AttractionId)
  If not &AttractionId.IsEmpty()
    &text = "The attraction " + &AttractionId.ToString() + " was inserted"
  else
    &text = "The attraction could not be inserted"
  endif
  msg(&text)
Endevent

```



But now let's see what would have happened if instead of inserting with the New command, we had done it with the Business Component (let's put a 2 to the name Eiffel Tower) and not explicitly Commit, since we have the Commit on Exit property enabled.

The screenshot displays the 'Navigation View' window in GeneXus. On the left, a navigation list shows a single item 'InsertNewAttraction' with a green checkmark icon. The main area is titled 'Procedure InsertNewAttraction Navigation Report' and contains the following details:

Name:	InsertNewAttraction	Environment:	Default (C#)
Description:	Insert New Attraction	Spec. Version:	17_0_3-148529
Output Devices:	None	Form Class:	Graphic
		Program Name:	InsertNewAttraction
		Parameters:	out: &AttractionId

At the bottom of the window, a status bar shows: 0 Errors, 0 Warnings, 1 Success, and All.

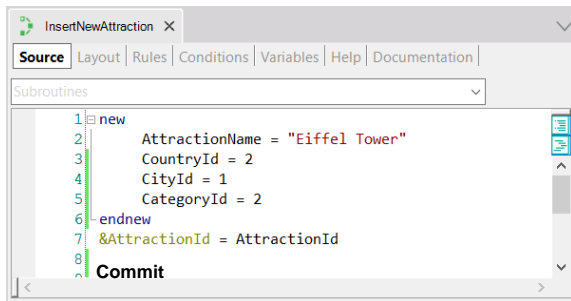
If we look at the navigation list, we can already see something odd: it is not giving us the same warning as in the other case, regarding the Commit on Exit property.

And then, if we run it, we see that it informs us of the next attraction number, which makes us think that it was correctly inserted and committed, but if we look for it in the transaction... it's not there!

What happened?

Commit on exit?

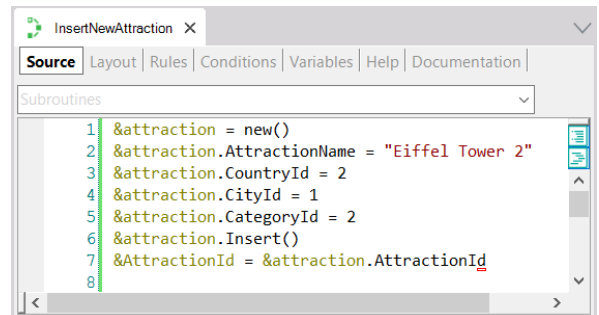
Transaction integrity	
Commit on exit	Yes



```

1 new
2   AttractionName = "Eiffel Tower"
3   CountryId = 2
4   CityId = 1
5   CategoryId = 2
6 -endnew
7 &AttractionId = AttractionId
8 Commit

```



```

1 &attraction = new()
2 &attraction.AttractionName = "Eiffel Tower 2"
3 &attraction.CountryId = 2
4 &attraction.CityId = 1
5 &attraction.CategoryId = 2
6 &attraction.Insert()
7 &AttractionId = &attraction.AttractionId
8

```

In previous videos we have repeatedly said that having the Commit on Exit property of a procedure set to Yes doesn't mean that GeneXus will always place a Commit at the end of the source code.

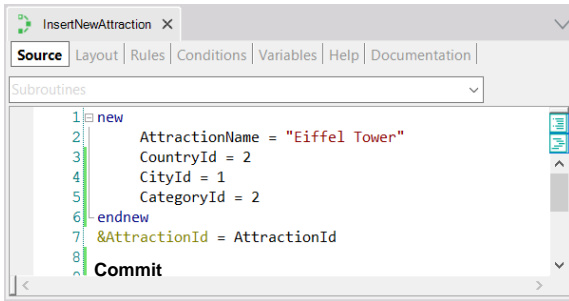
We said that it will only do so if it finds that in the procedure's Source, somewhere, the database is being accessed to do some CRUD operation (Create, Update, Delete).

Thus, in the first case it will clearly place a Commit, because the New command unequivocally represents a CRUD operation.

However, it doesn't recognize an operation of this type for the second case. It takes the Business Component by its structure, as an SDT, and fails to interpret the Insert method accordingly. So it doesn't understand that in the Source of this procedure you want to do any CRUD operation. And that's why it doesn't add the Commit at the end of the source code.

Commit on exit?

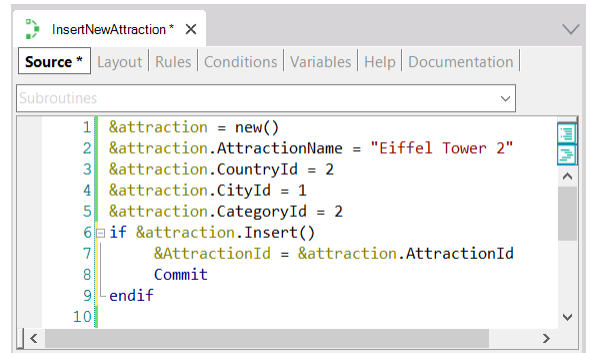
Transaction integrity	
Commit on exit	Yes



```

1 new
2   AttractionName = "Eiffel Tower"
3   CountryId = 2
4   CityId = 1
5   CategoryId = 2
6 -endnew
7 &AttractionId = AttractionId
8
9 Commit

```



```

1 &attraction = new()
2 &attraction.AttractionName = "Eiffel Tower 2"
3 &attraction.CountryId = 2
4 &attraction.CityId = 1
5 &attraction.CategoryId = 2
6 if &attraction.Insert()
7   &AttractionId = &attraction.AttractionId
8   Commit
9 endif
10

```

So in this case, for the time being, we will have no choice but to make it explicit.

If we try it now... it behaves as expected.

	Uniqueness Check	Referential Integrity Check	Rules/Events execution	Scope
Business Component	✓	✓	✓	Any Object
New, Assignment in For Each, Delete	✓	✗	✗	Procedure only!

Commit?

Transaction integrity	
Commit on exit	Yes

Rollback?

We have seen the main differences between performing CRUD operations with Business components and doing it directly in procedures through specific commands.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications