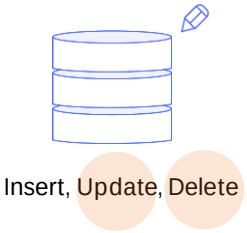
Database Update with Procedure-specific Commands

How to Delete

GeneXus™

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**



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

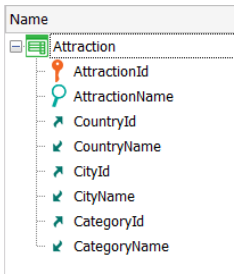
In the video on updating with For each in a procedure, we saw a case in which we needed to modify the value of the primary key of a record, for which we had to create a new one with the new key value and delete the old one.

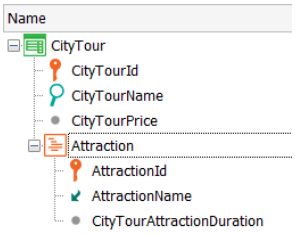
And we did this by selecting the record in question, creating a new one with New, and immediately executing the Delete command to delete the record of the For each in which we were positioned.

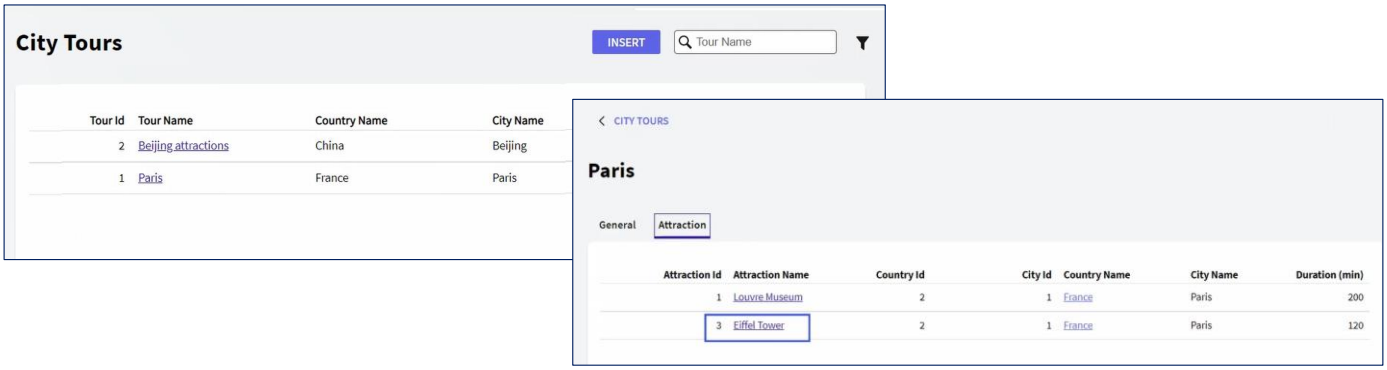
This is how deletion will be in general. Using For each to choose a record and executing Delete to delete it.

Delete

Let's study the details of the deletion process.

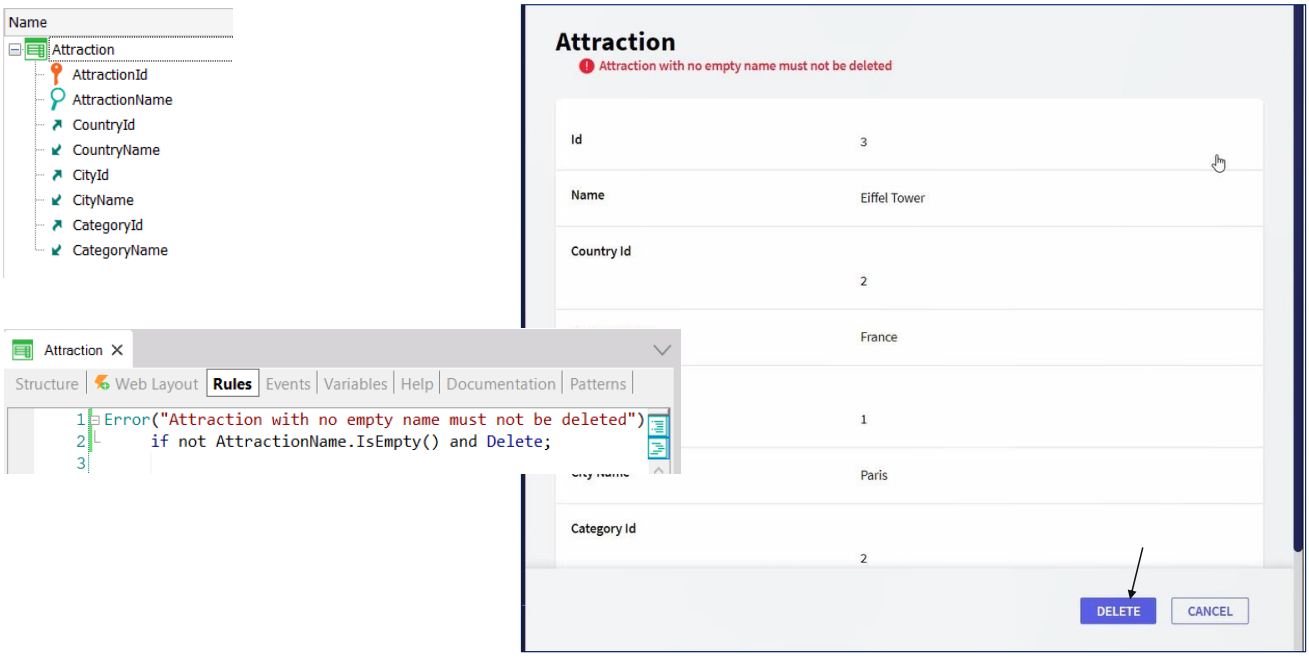






Let's remember the transactions that we have been using to study the insertion and update of the database through procedures.

Here we had CityTour, where we could specify the attractions to be visited on the current tour. For example, at runtime, we had this tour of Paris, which would visit the Louvre museum and Eiffel Tower attractions.



In addition, we had the transaction that records the attractions. And we have added this error rule to prevent the deletion of an attraction with an entered name.

So, let's see that if we try to delete the Eiffel Tower attraction through the transaction, we will not be allowed to do so.

Delete through Procedure?

Now, what happens if we try to delete it through a procedure?

We know that this attraction, Eiffel Tower, is part of a city tour, and it also has a name assigned to it, which is, precisely, Eiffel Tower. So, will it allow us to eliminate it?

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Delete in For each	✕	✕	✕

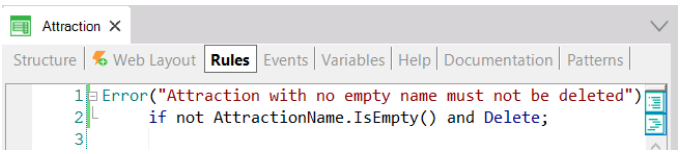
CityTourId	AttractionId	CityTourAttractionDuration
1	1	200
1	3	120
2	2	240
2	4	240

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 AttractionName = "Eiffel Tower"

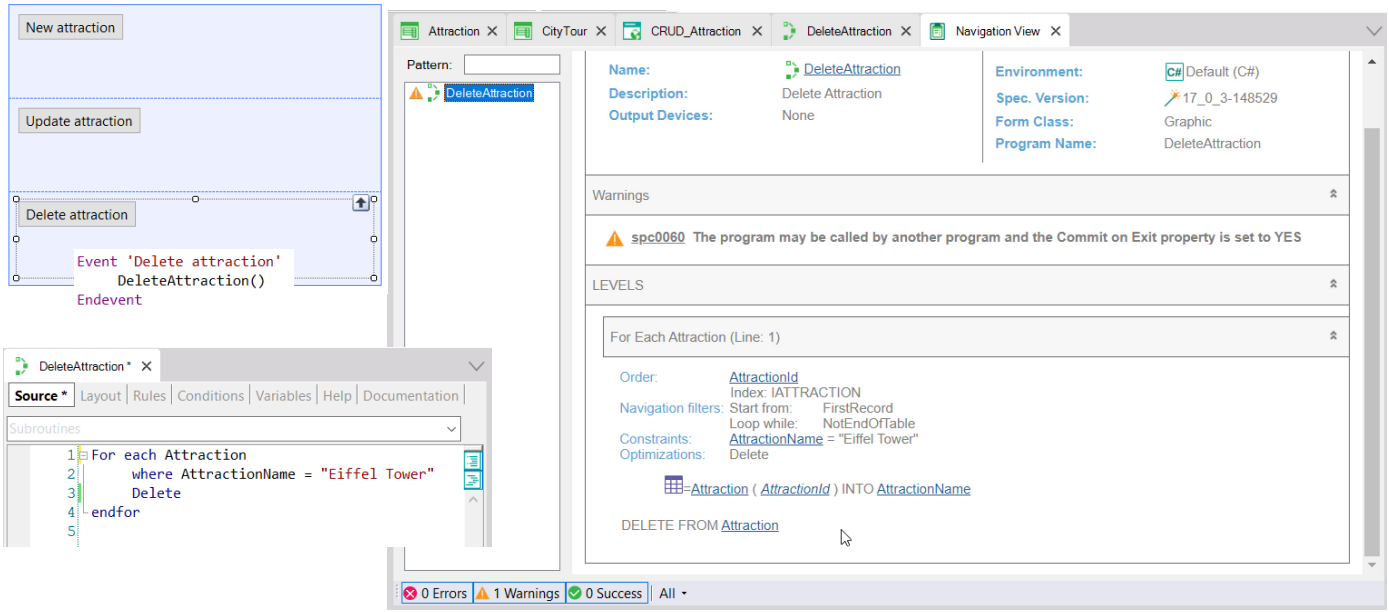
Delete

endfor

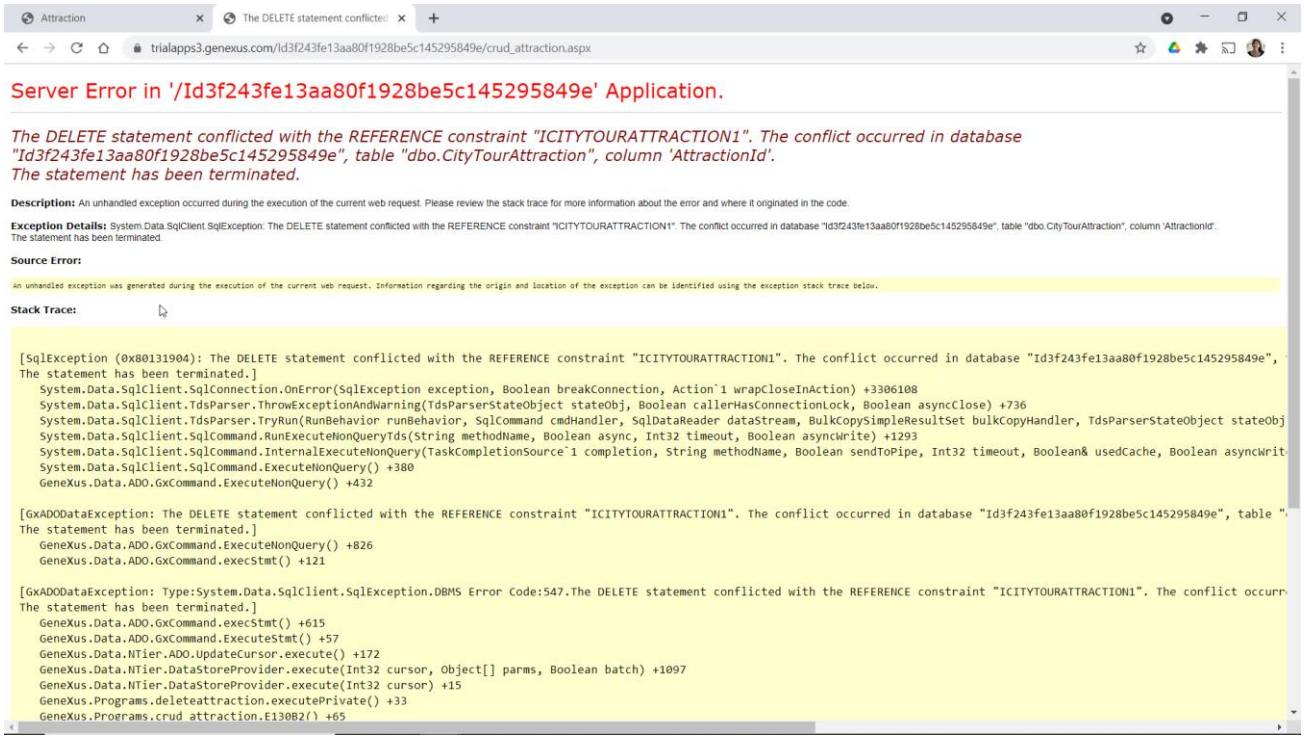


If we think about everything we have seen so far regarding the database update commands through procedures, we can say YES, we will be able to delete the attraction because:

- **No** programmatic referential integrity checks are performed, i.e., it will not check that there is no city tour that is referencing the attraction to be deleted.
- In addition, no rules are executed for the transaction associated with the table from which a record is being deleted.



So if we go to GeneXus, we see that we programmed this button that invokes this procedure... that runs through the attractions with a For each filtering by the name “Eiffel Tower” and for the records found (in this case it will be only one), it deletes them with the Delete command...



If we run it... The program fails. Why?

For the same reason we've seen before.

	Uniqueness check	Referential Integrity check	Rules/Events execution
Delete in For each	✕	✕	

CityTourId	AttractionId	CityTourAttractionDuration
1	1	200
1	3	120
2	2	240
2	4	240

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

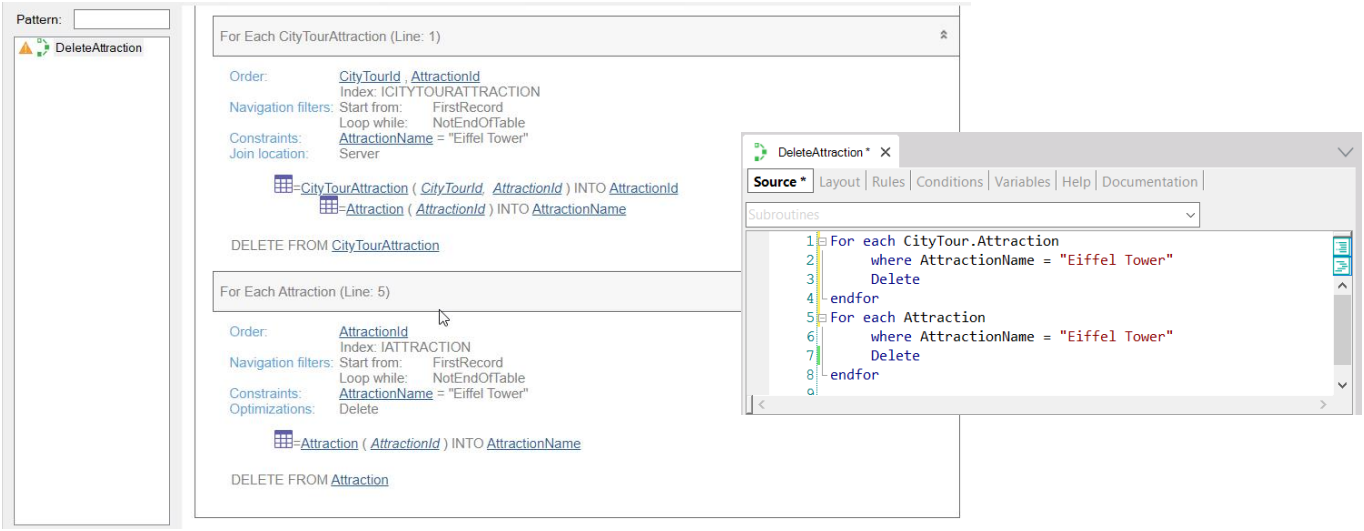
```
For each Attraction
    Where AttractionName = "Eiffel Tower"

    Delete

endfor
```

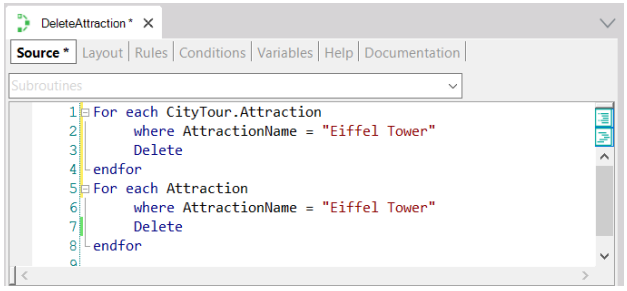
exception

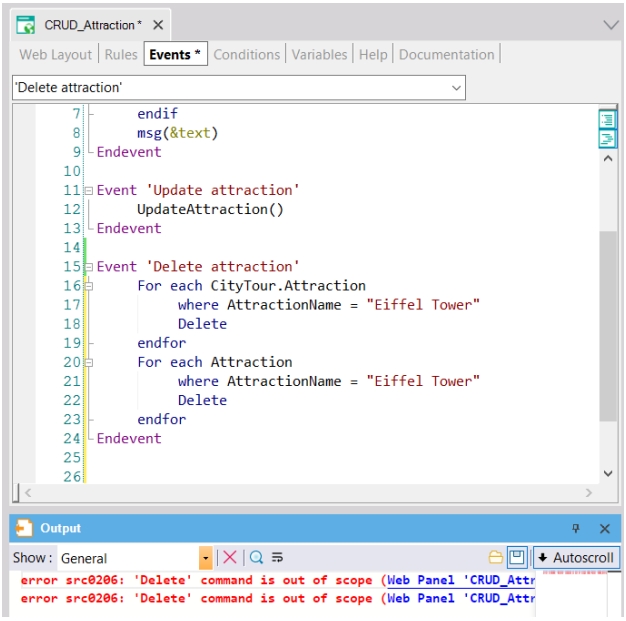
The Delete operation doesn't check referential integrity but the database does it by default, and we're not catching the exception it throws.



So, if we want to eliminate that attraction, we should first eliminate it from all the city tours where it is included. In this way...
We run it.

As an obvious observation, the Delete command only deletes the record of the base table of the For each in which we are positioned. It doesn't delete records from the extended table. In this sense, it works as the New command.





And another thing that we have already emphasized many times, but is important to emphasize again: the Delete command can only be used within a For each and in a procedure. We couldn't have programmed the deletion directly within the event, for example.

Unlike what happens when updating through a Business Component.

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Delete in For each	✕	✕	✕

CityTourId	AttractionId	CityTourAttractionDuration
1	1	200
1	3	120
2	2	240
2	4	240

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

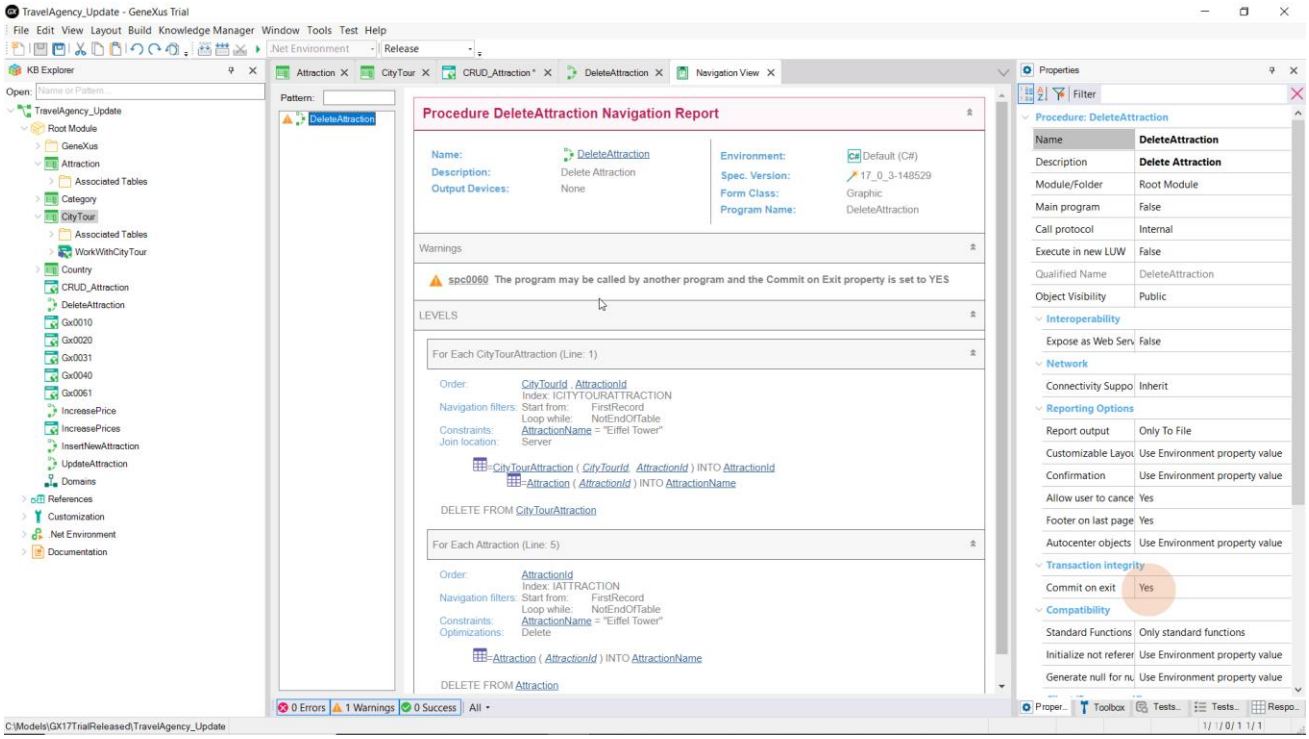
```
For each CityTour.Attraction
      Where AttractionName = "Eiffel Tower"
      Delete
endfor
For each Attraction
      Where AttractionName = "Eiffel Tower"
      Delete
endfor
Commit
```

COMMIT?

Transaction integrity	
Commit on exit	Yes

Lastly, in this procedure we're deleting two records. When is this operation committed in the database?

The same happens as we saw with New and update with For each. If the Commit on Exit property is left with its default value, which is Yes, since GeneXus realizes that you want to perform a deletion on the database, it automatically adds a Commit command at the end of the procedure Source.



And that's why the navigation list shows a warning to indicate this, so that we know that even if we have not explicitly specified a commit, GeneXus will add it.

	Uniqueness check	Referential Integrity check	Rule/Event Execution
Delete in For each	✕	✕	✕

CityTourId	AttractionId	CityTourAttractionDuration
1	1	200
1	3	120
2	2	240
2	4	240

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 CityTour.Attraction
    Where AttractionName = "Eiffel Tower"
    Delete
Endfor
Commit
For each Attraction
    Where AttractionName = "Eiffel Tower"
    Delete
Endfor
Commit
```

```
For each CityTour.Attraction
    Where AttractionName = "Eiffel Tower"
    Delete
    Commit
Endfor
For each Attraction
    Where AttractionName = "Eiffel Tower"
    Delete
    Commit
Endfor
```

Of course, we could program a commit after every For each.

Or even after every Delete (because in this case, the For each will retrieve a single record, but there could be several records).

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*
...
Delete
...
When duplicate
...
When none
...
endfor

	Uniqueness check	Referential Integrity check
Delete	✕	✕

COMMIT

Transaction integrity	
Commit on exit	Yes

In short, to delete records specifically through a procedure, we have the Delete command that must be used within the For each command.

The deletion will be of the record of the For each base table in which it is positioned in each iteration.

For a deletion there is no point in performing a key uniqueness check, because nothing is being inserted or updated. And as we saw with insertion and update, the Delete operation does not programmatically perform any referential integrity check. Again, this is for performance reasons. However, databases in general do make it, unless we turn off that feature; therefore, 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, i.e. permanently deleted, 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 deletions 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.

For insertions, updates and deletions, redundancies are not automatically maintained when done procedurally. It is the developer's responsibility to do so.

We suggest visiting our wiki for more information.

GeneXus™

training.genexus.com
wiki.genexus.com