

# Database Update with Procedure-specific Commands

How to Delete

**GeneXus**<sup>™</sup>

## For each Command

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

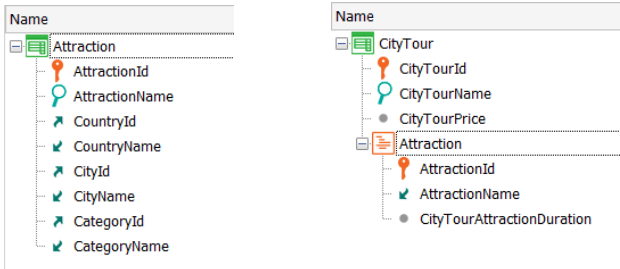
In the video on updating with For each in a procedure, we saw a case in which we needed to modify the value of the primary key of a record, for which we had to create a new one with the new key value and delete the old one.

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.



**City Tours** INSERT

Tour Id	Tour Name	Country Name	City Name
2	Beijing attractions	China	Beijing
1	Paris	France	Paris

< CITY TOURS

**Paris**

General **Attraction**

Attraction Id	Attraction Name	Country Id	City Id	Country Name	City Name	Duration (min)
1	Louvre Museum	2	1	France	Paris	200
3	Eiffel Tower	2	1	France	Paris	120

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.

The image shows the GeneXus IDE interface. On the left, a tree view under 'Name' shows the 'Attraction' entity with fields: AttractionId, AttractionName, CountryId, CountryName, CityId, CityName, CategoryId, and CategoryName. In the center, a 'Rules' window is open, displaying the following code:

```
1 Error("Attraction with no empty name must not be deleted")
2   if not AttractionName.IsEmpty() and Delete;
3
```

On the right, a data table titled 'Attraction' is displayed. At the top, a red error message reads: 'Attraction with no empty name must not be deleted'. The table contains the following data:

Id	Name	Country Id	City Name	Category Id
3	Eiffel Tower	2	Paris	2
2	France	1		
1				

At the bottom right of the table, there are 'DELETE' and 'CANCEL' buttons. An arrow points to the 'DELETE' button.

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

```

Attraction X
Structure | Web Layout | Rules | Events | Variables | Help | Documentation | Patterns
1 Error("Attraction with no empty name must not be deleted")
2   if not AttractionName.IsEmpty() and Delete;
3
  
```

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.

The screenshot displays the GeneXus IDE with the following components:

- Design View (Left):** Shows a form with three buttons: "New attraction", "Update attraction", and "Delete attraction". The "Delete attraction" button is connected to an event "Delete attraction" which triggers the "DeleteAttraction()" procedure.
- Source View (Bottom Left):** Shows the code for the "DeleteAttraction" procedure:

```
1 For each Attraction
2   where AttractionName = "Eiffel Tower"
3   Delete
4 endfor
```
- Properties View (Right):** Shows the configuration for the "DeleteAttraction" procedure:
  - Name:** DeleteAttraction
  - Description:** Delete Attraction
  - Output Devices:** None
  - Environment:** Default (C#)
  - Spec. Version:** 17\_0\_3-148529
  - Form Class:** Graphic
  - Program Name:** DeleteAttraction
- Warnings (Right):** A warning message: "spc0060 The program may be called by another program and the Commit on Exit property is set to YES".
- LEVELS (Right):** Shows the execution flow:
  - For Each Attraction (Line: 1)
  - Order: AttractionId
  - Index: IATTRACTION
  - Navigation filters: Start from: FirstRecord, Loop while: NotEndOfTable
  - Constraints: AttractionName = "Eiffel Tower"
  - Optimizations: Delete
  - SQL: `DELETE FROM Attraction`
- Status Bar (Bottom):** Shows "0 Errors", "1 Warnings", "0 Success", and "All".

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...



Attraction x The DELETE statement conflicted: x +

trialapps3.genexus.com/Id3f243fe13aa80f1928be5c145295849e/crud\_attraction.aspx

## Server Error in '/Id3f243fe13aa80f1928be5c145295849e' Application.

*The DELETE statement conflicted with the REFERENCE constraint "ICITYTOURATTRACTION1". The conflict occurred in database "Id3f243fe13aa80f1928be5c145295849e", table "dbo.CityTourAttraction", column 'AttractionId'. The statement has been terminated.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.Data.SqlClient.SqlException: The DELETE statement conflicted with the REFERENCE constraint "ICITYTOURATTRACTION1". The conflict occurred in database "Id3f243fe13aa80f1928be5c145295849e", table "dbo.CityTourAttraction", column "AttractionId". The statement has been terminated.

**Source Error:**

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

**Stack Trace:**

```
[SqlException (0x80131904): The DELETE statement conflicted with the REFERENCE constraint "ICITYTOURATTRACTION1". The conflict occurred in database "Id3f243fe13aa80f1928be5c145295849e", The statement has been terminated.]
  System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +3306108
  System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose) +736
  System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, TdsParserStateObject stateObj)
  System.Data.SqlClient.SqlCommand.RunExecuteNonQueryTds(String methodName, Boolean async, Int32 timeout, Boolean asyncWrite) +1293
  System.Data.SqlClient.SqlCommand.InternalExecuteNonQuery(TaskCompletionSource`1 completion, String methodName, Boolean sendToPipe, Int32 timeout, Boolean& usedCache, Boolean asyncWrite)
  System.Data.SqlClient.SqlCommand.ExecuteNonQuery() +380
  GeneXus.Data.ADO.GxCommand.ExecuteNonQuery() +432

[GxADODataException: The DELETE statement conflicted with the REFERENCE constraint "ICITYTOURATTRACTION1". The conflict occurred in database "Id3f243fe13aa80f1928be5c145295849e", table "dbo.CityTourAttraction". The statement has been terminated.]
  GeneXus.Data.ADO.GxCommand.ExecuteNonQuery() +826
  GeneXus.Data.ADO.GxCommand.execStmt() +121

[GxADODataException: Type: System.Data.SqlClient.SqlException, DBMS Error Code: 547. The DELETE statement conflicted with the REFERENCE constraint "ICITYTOURATTRACTION1". The conflict occurred in database "Id3f243fe13aa80f1928be5c145295849e", table "dbo.CityTourAttraction". The statement has been terminated.]
  GeneXus.Data.ADO.GxCommand.execStmt() +615
  GeneXus.Data.ADO.GxCommand.ExecutesStmt() +57
  GeneXus.Data.NTier.ADO.UpdateCursor.execute() +172
  GeneXus.Data.NTier.DataStoreProvider.execute(Int32 cursor, object[] parms, Boolean batch) +1097
  GeneXus.Data.NTier.DataStoreProvider.execute(Int32 cursor) +15
  GeneXus.Programs.deleteattraction.executePrivate() +33
  GeneXus.Programs.crud_attraction.F130B2() +65
```

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.

The screenshot displays the GeneXus IDE interface for a pattern named 'DeleteAttraction'. On the left, a sidebar shows the pattern name. The main workspace is divided into two sections, each representing a 'For Each' loop.

**For Each CityTourAttraction (Line: 1)**

- Order: `CityTourId, AttractionId`
- Index: `ICITYTOURATTRACTION`
- Navigation filters: Start from: `FirstRecord`, Loop while: `NotEndOfTable`
- Constraints: `AttractionName = "Eiffel Tower"`
- Join location: `Server`

Below the configuration, the code for this loop is shown:

```

CityTourAttraction ( CityTourId, AttractionId ) INTO AttractionId
Attraction ( AttractionId ) INTO AttractionName

DELETE FROM CityTourAttraction

```

**For Each Attraction (Line: 5)**

- Order: `AttractionId`
- Index: `IATTRACTION`
- Navigation filters: Start from: `FirstRecord`, Loop while: `NotEndOfTable`
- Constraints: `AttractionName = "Eiffel Tower"`
- Optimizations: `Delete`

The code for this loop is:

```

Attraction ( AttractionId ) INTO AttractionName

DELETE FROM Attraction

```

On the right, a 'Source' window shows the compiled source code for the 'DeleteAttraction' pattern:

```

1 For each CityTour.Attraction
2   where AttractionName = "Eiffel Tower"
3   Delete
4 -endfor
5 For each Attraction
6   where AttractionName = "Eiffel Tower"
7   Delete
8 -endfor
9

```

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.

```

DeleteAttraction * X
Source * Layout Rules Conditions Variables Help Documentation
Subroutines
1 For each CityTour.Attraction
2   where AttractionName = "Eiffel Tower"
3   Delete
4   endfor
5 For each Attraction
6   where AttractionName = "Eiffel Tower"
7   Delete
8   endfor
9

```

```

CRUD_Attraction * X
Web Layout Rules Events * Conditions Variables Help Documentation
'Delete attraction'
7   endif
8   msg(&text)
9   Endevent
10
11 Event 'Update attraction'
12   UpdateAttraction()
13   Endevent
14
15 Event 'Delete attraction'
16   For each CityTour.Attraction
17     where AttractionName = "Eiffel Tower"
18     Delete
19   endfor
20   For each Attraction
21     where AttractionName = "Eiffel Tower"
22     Delete
23   endfor
24   Endevent
25
26

```

Output

Show: General

error src0206: 'Delete' command is out of scope (Web Panel 'CRUD\_Attr

error src0206: 'Delete' command is out of scope (Web Panel 'CRUD\_Attr

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

```

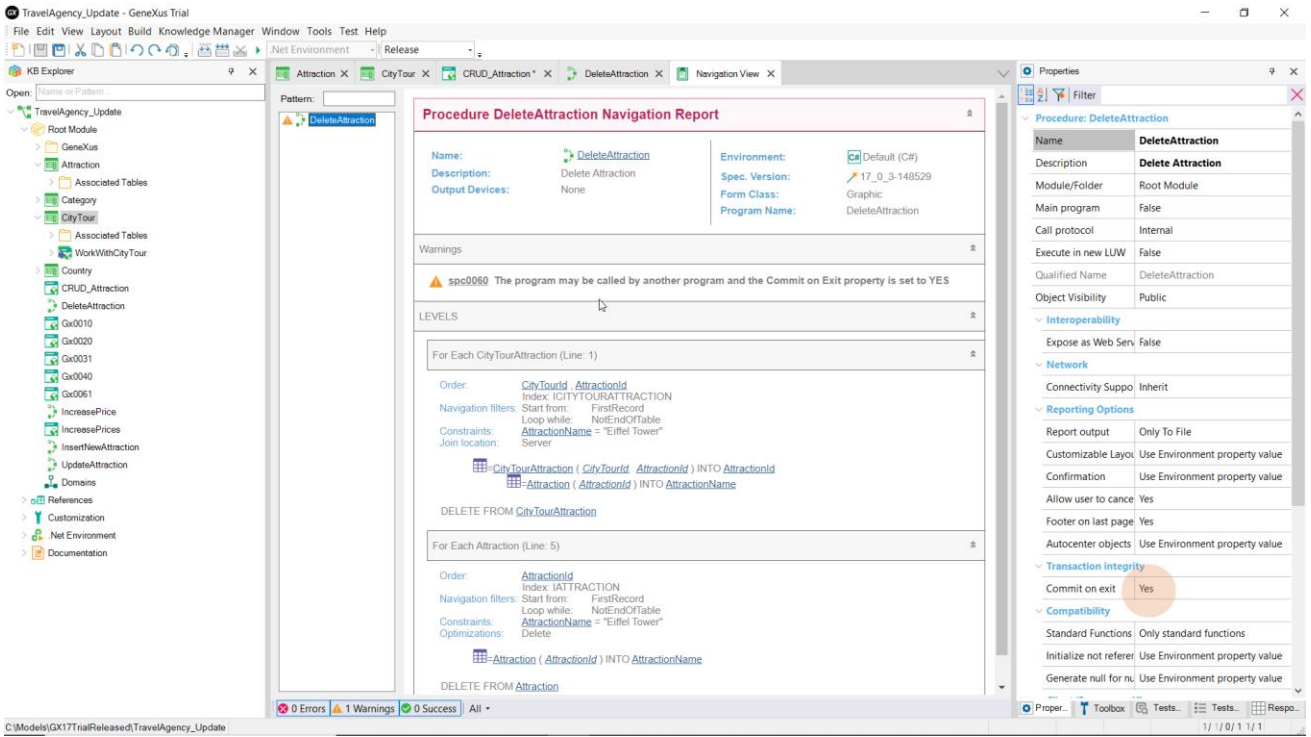
COMMIT?

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 expression1 count expression2
order att11, att12, ... att1n [when condition]
order att21, att22, ... att2n [when condition | otherwise]
using DataSelector(parm1, ..., parmn)
unique att1, ..., attn
where condition [when condition]
where condition [when condition]
where att in DataSelector(parm1, ..., parmn)
blocking NumericExpression
...
Delete
...

```

	Uniqueness check	Referential Integrity check
Delete	✗	✗

**COMMIT**

Transaction integrity	
Commit on exit	Yes

When duplicate

...

When none

...

endfor

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*<sup>TM</sup>

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