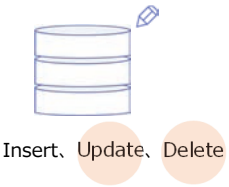
プロシージャー固有のコマンドによるデータベースの更新

削除方法

GeneXus™

For each コマンド

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	2
3	エッフェル塔	2	1	3
4	紫禁城	3	1	2
5	エッフェル塔	2	1	3



Insert、Update、Delete

```
For each Attraction
      Where AttractionName = “エッフェル塔”
   AttractionId = 5
   CategoryId = 3
endfor
```

```
For each Attraction
      Where AttractionName = “エッフェル塔”
   new
         AttractionId = 5
         CategoryId = 3
   endnew
   Delete
endfor
```

プロシージャーで For each を使用してデータを更新する方法を説明した章で、レコードの主キーの値を変更する方法について解説しました。新しいキー値を持つ新しい主キーを作成し、古い主キーを削除する必要がありました。

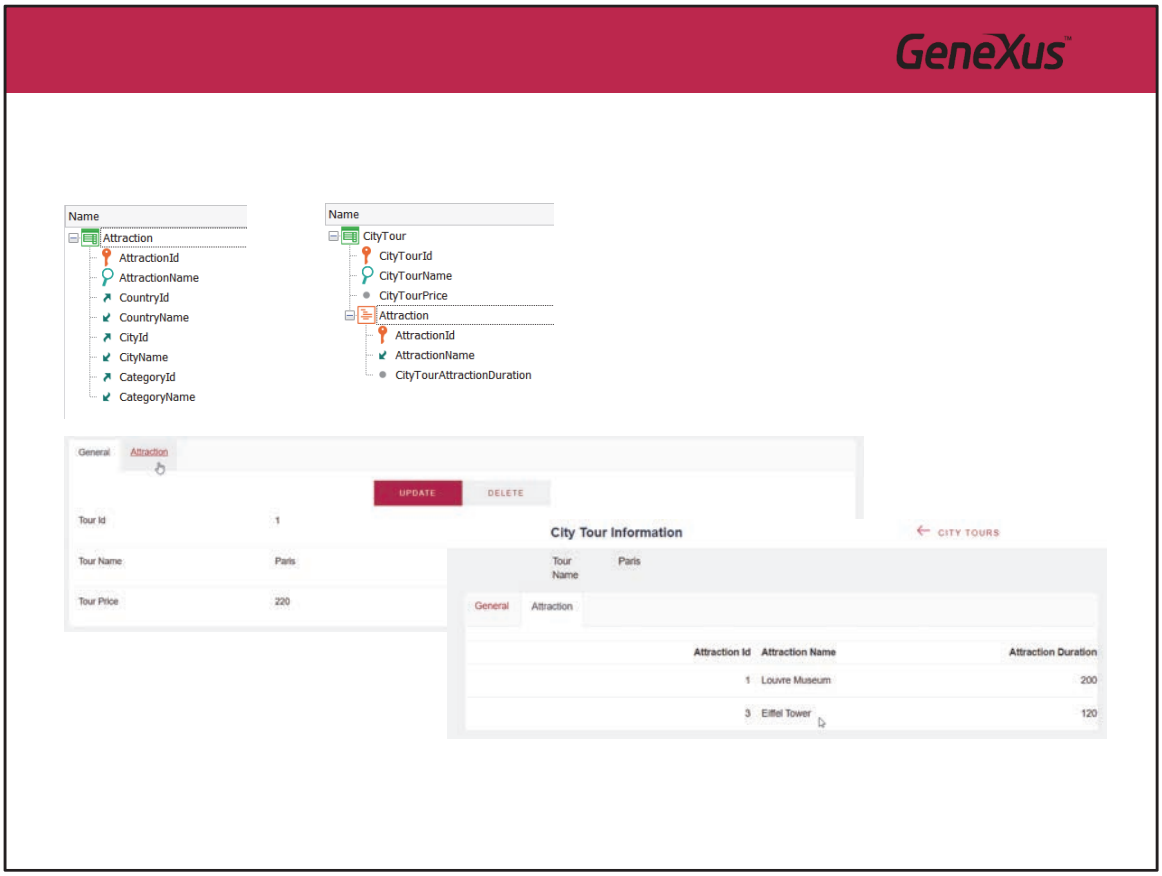
その場合、対象となるレコードを選択し、New コマンドで新しいレコードを作成し、すぐに Delete コマンドを実行して、そのときポイントしていた For each のレコードを削除しました。

これがデータの一般的な削除方法です。For each を使用してレコードを選択し、Delete を実行して削除します。

削除

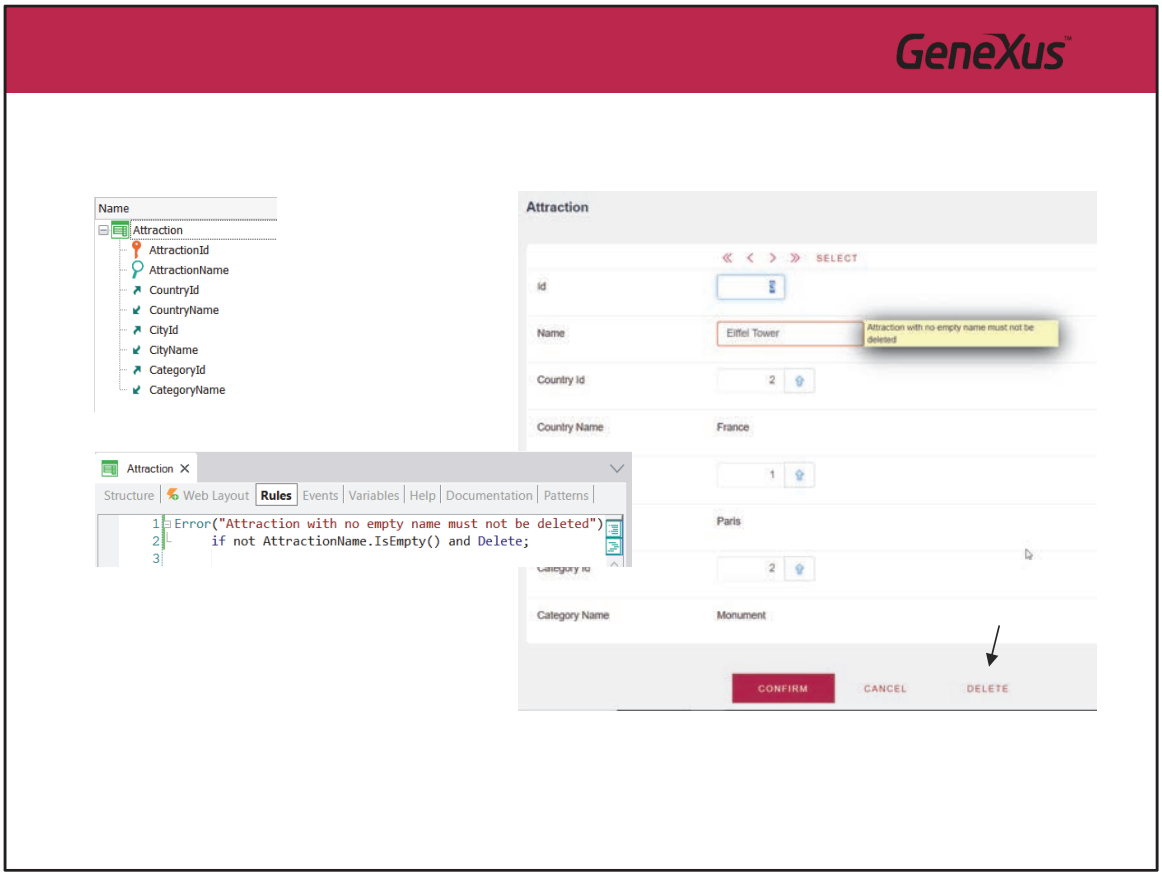
ここでは、削除プロセスについて詳しく学習しましょう。

デモ



プロシージャーを介したデータベースの挿入と更新の学習で使用したトランザクションを思い出してください。

この CityTour では、現在のツアーで訪問する観光名所を指定できました。たとえば、実行時に、観光名所としてルーブル美術館やエッフェル塔を訪問する「パリ」というツアーが表示されました。



また、観光名所を記録するトランザクションがありました。さらに、名前が入力された観光名所が削除されないようにするためのエラールールを追加しました。

そのため、トランザクションから観光名所「エッフェル塔」を削除しようとしても、削除することはできません。

プロシージャーを介した削除

プロシージャーを介して削除を試みると、どうなるでしょうか。

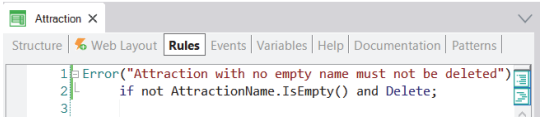
観光名所エッフェル塔は市内観光に含まれており、「エッフェル塔」と言う名前が割り当てられていることが分かっています。これを削除できるでしょうか。

	一意性チェック	参照整合性チェック	ルール/イベントの実行
For each で削除	×	×	×

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

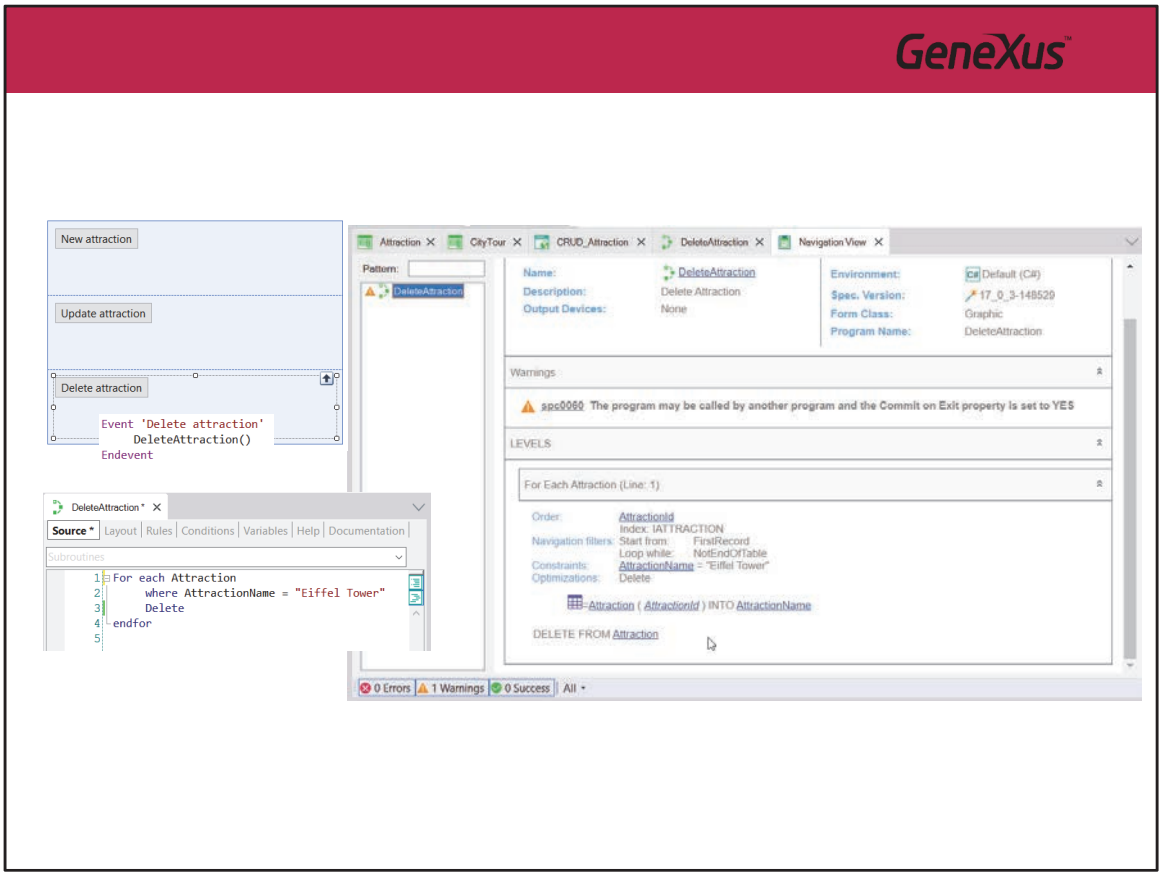
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	2
3	エッフェル塔	2	1	2
4	紫禁城	3	1	2

```
For each Attraction
      Where AttractionName = "エッフェル塔"

      Delete

endfor
```

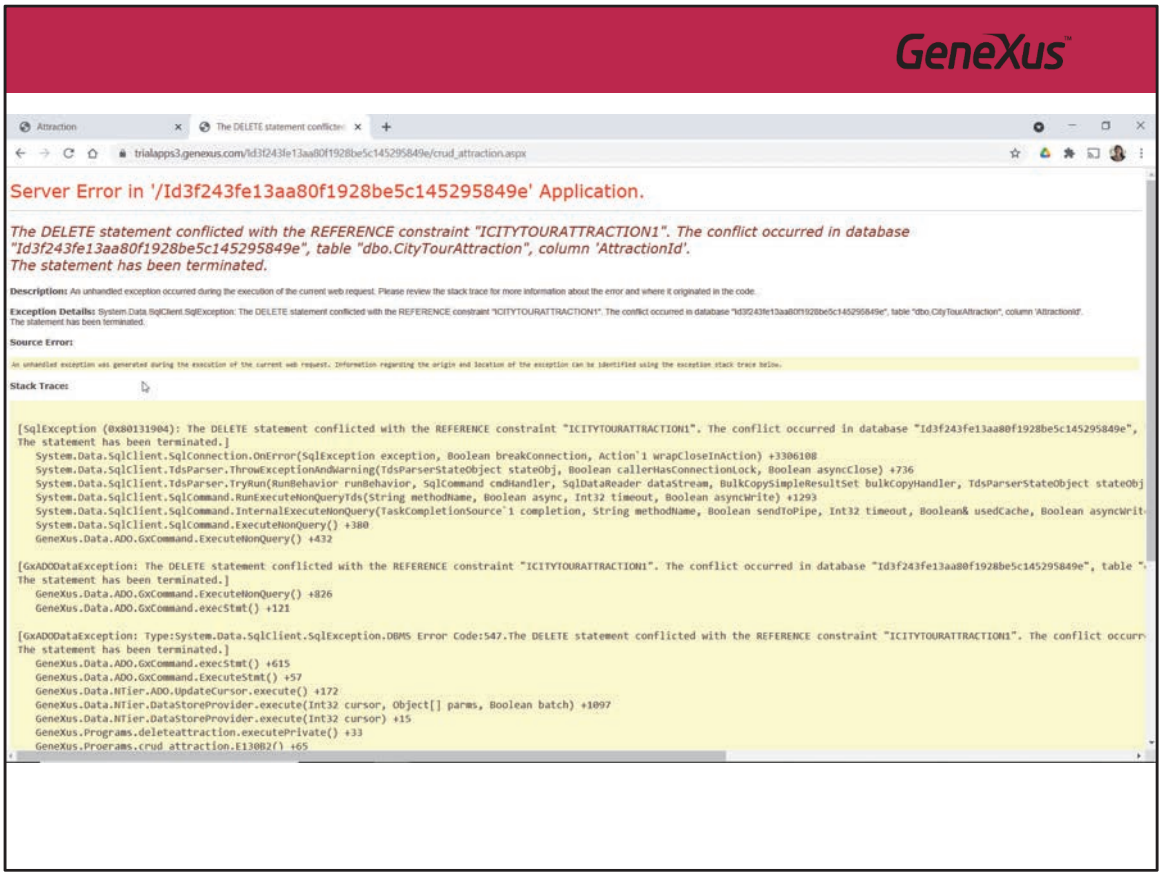


プロシージャーを介したデータベース更新コマンドについて、これまでに見てきたことを考えると、可能であると言えます。いくつかの理由から、観光名所を削除できます。

- プログラムの参照整合性チェックは実行**されません**。つまり、削除する観光名所を参照している市内観光があるかどうかはチェックされません。
- また、レコードが削除されるテーブルと関連付けられたトランザクションに対してルールは実行されません。



GeneXus で確認すると、このプロシージャーを呼び出すボタンがプログラミングされています。このプロシージャーは、For each を使用し、観光名所を参照して「エッフェル塔」と言う名前でフィルタし、見つかったレコード (この場合は 1 つのみ) を Delete コマンドで削除します。



これを実行してみると、プログラムが失敗します。なぜでしょうか。

前に説明したのと同じ理由です。

GeneXus

	一意性チェック	参照整合性チェック	ルール/イベントの実行
For each で削除	×	×	

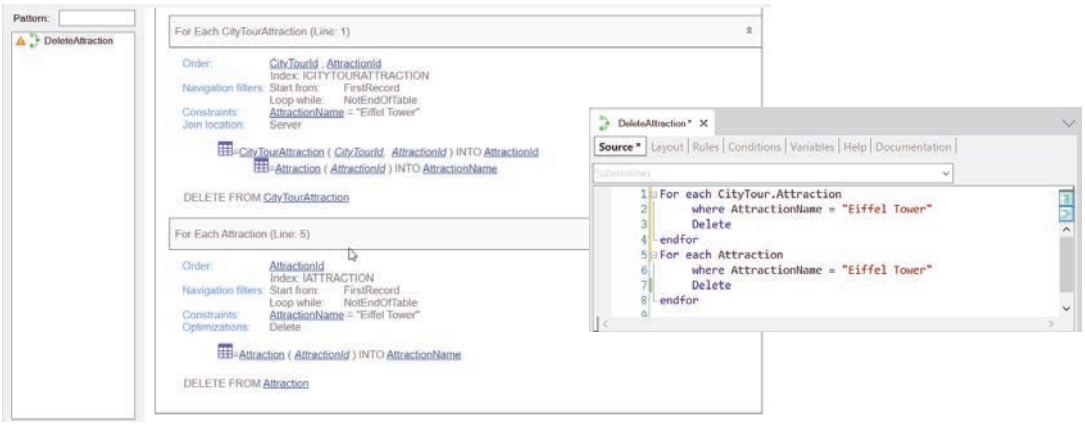
CityTourId	AttractionId	CityTourAttractionDuration
1	1	200
1	3	120
2	2	240
2	4	240

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	2
3	エッフェル塔	2	1	2
4	紫禁城	3	1	2

```
For each Attraction
      Where AttractionName = “エッフェル塔”

      Delete

endfor
```

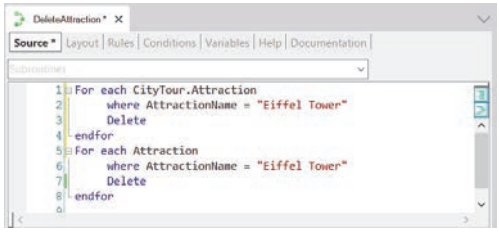
例外

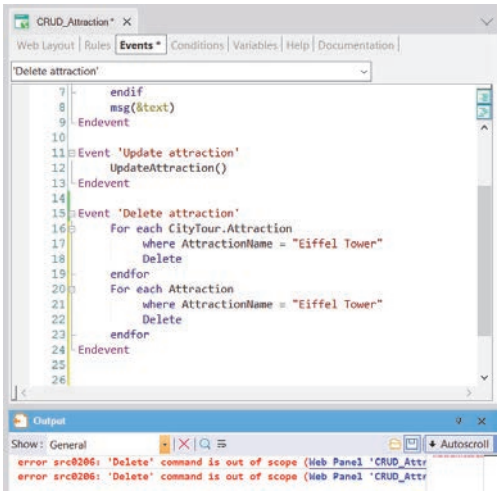
Delete オペレーションでは参照整合性チェックが実行されませんが、データベースでは既定で実行されます。また、データベースがスローした例外はキャッチされません。



そのため、この観光名所を削除するには、まずその観光名所が含まれているすべての市内観光から、その観光名所を削除する必要があります。
このようにして実行します。

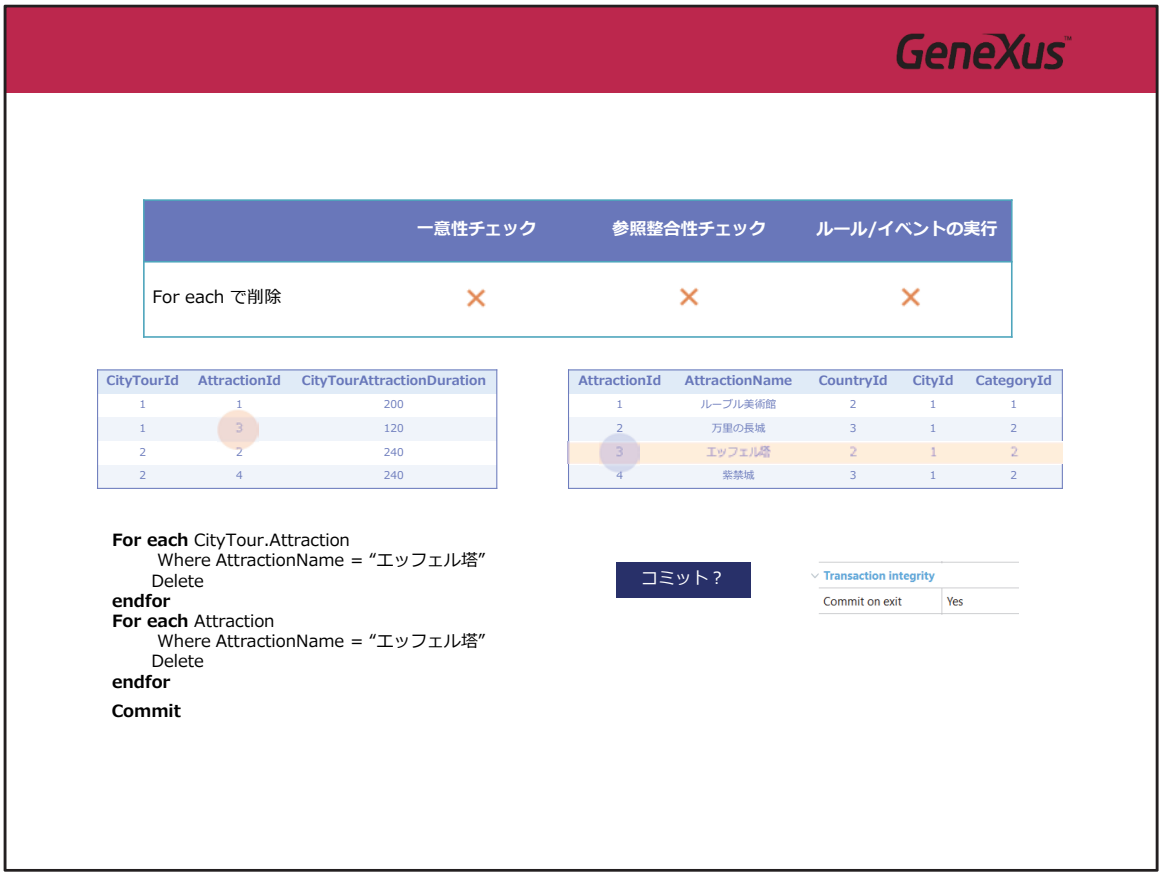
Delete コマンドは、そのときポイントしていた For each のベーステーブルのレコードを削除します。拡張テーブルのレコードは削除されません。その意味では、New コマンドと同じ処理を行います。





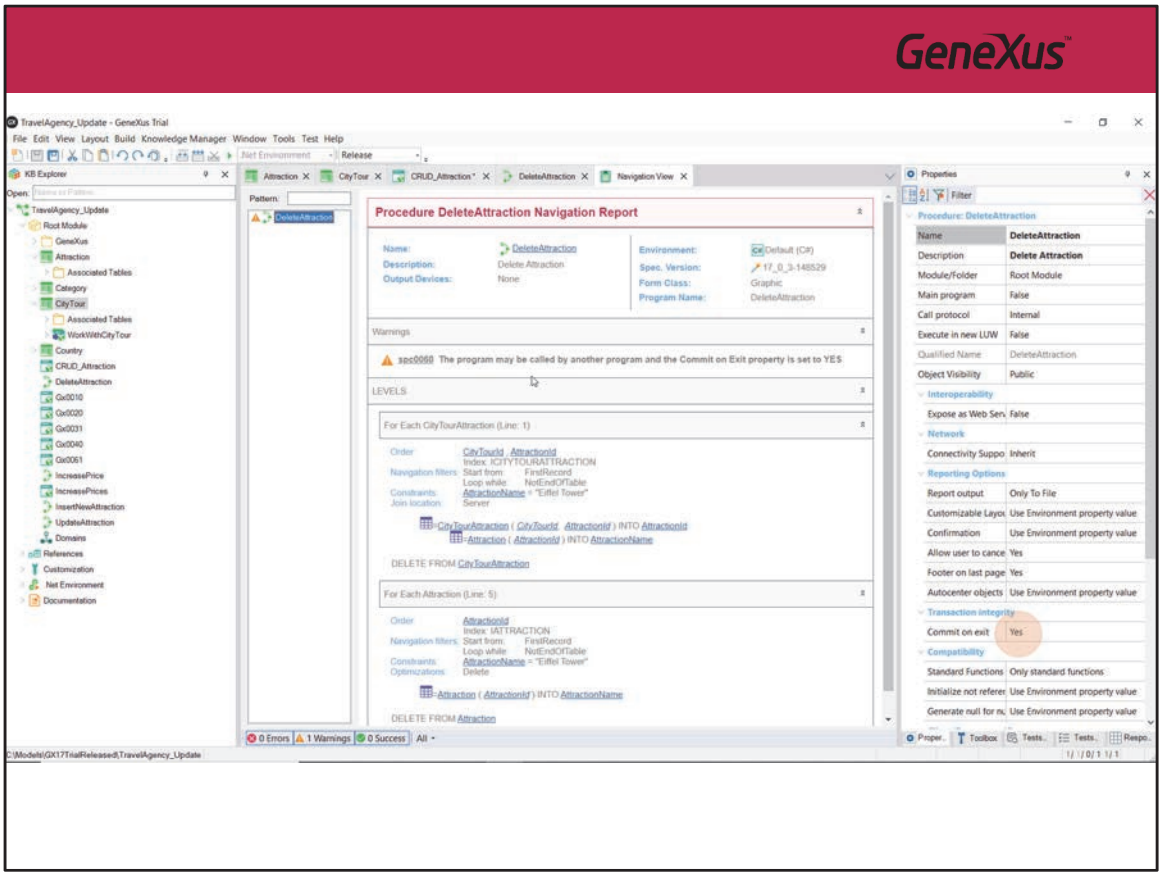
これまで何度か説明していますが、重要な点は、Delete コマンドは For each コマンド内とプロシージャーの中でのみ使用できるということです。たとえば、イベント内に削除の処理を直接プログラミングすることはできません。

ビジネスコンポーネントを介した更新の場合とは異なります。



最後に、このプロシージャーでは 2 つのレコードを削除しています。このオペレーションはいつデータベースにコミットされるのでしょうか。

New コマンドや For each で更新する場合と同じようになります。[Commit on exit] プロパティを既定値の Yes のままにすると、GeneXus ではデータベースに対する削除処理を実行すると認識され、プロシージャーのソースの最後に Commit コマンドが自動的に追加されます。



そのため、ナビゲーション表示に警告が示され、明示的にコミットを指定していなくても、GeneXus によってコミットが追加されることが分かります。

GeneXus

	一意性チェック	参照整合性チェック	ルール/イベントの実行
For each で削除	×	×	×

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

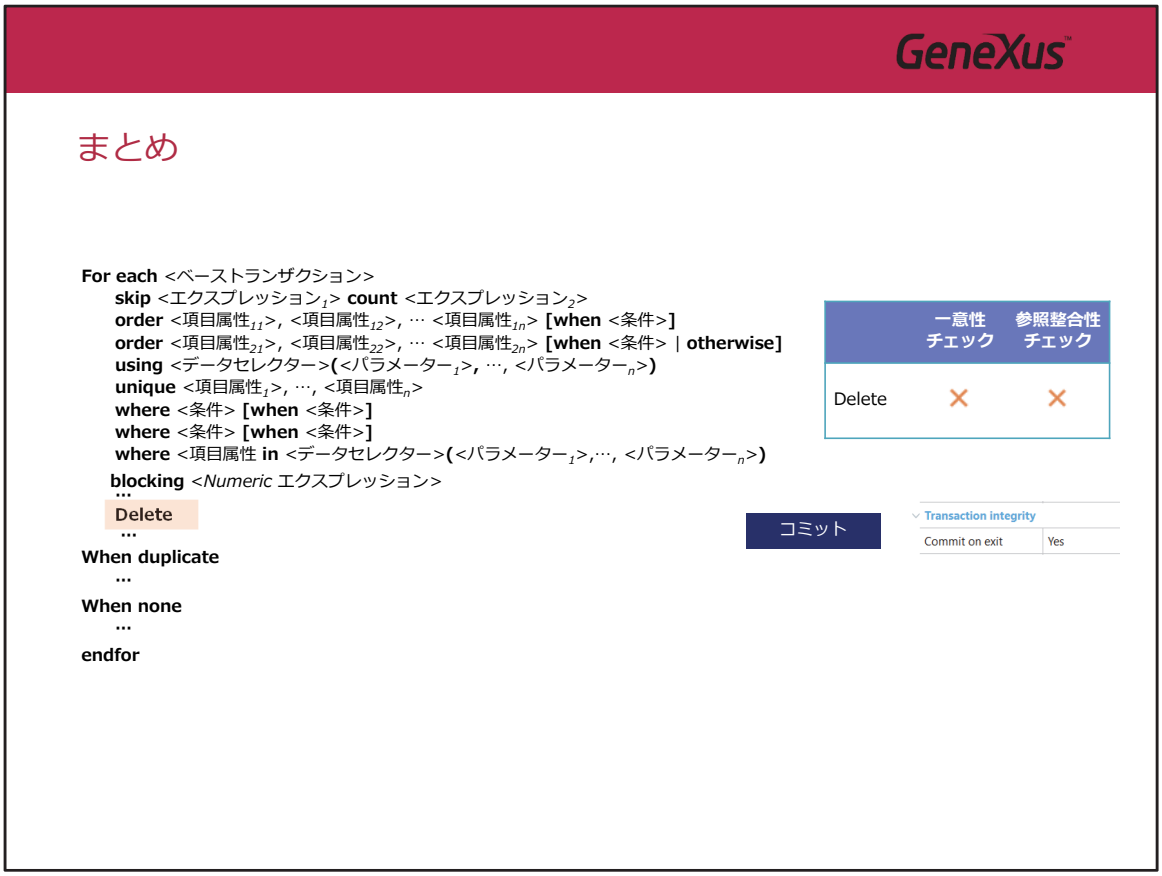
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	2
3	エッフェル塔	2	1	2
4	紫禁城	3	1	2

For each CityTour.Attraction
Where AttractionName = “エッフェル塔”
Delete
Endfor
→ **Commit**
For each Attraction
Where AttractionName = “エッフェル塔”
Delete
Endfor
→ **Commit**

For each CityTour.Attraction
Where AttractionName = “エッフェル塔”
Delete
→ **Commit**
Endfor
For each Attraction
Where AttractionName = “エッフェル塔”
Delete
→ **Commit**
Endfor

もちろん、それぞれの For each の後にコミットをプログラミングすることもできます。

また、それぞれの Delete の後に追加することも可能です (For each は 1 つのレコードを取得しますが、レコードが複数ある可能性もあるため)。



要約すると、プロシージャーを介してレコードを削除するには、For each コマンド内で Delete コマンドを使用する必要があります。

削除は、それぞれの反復処理でポイントされていた For each ベーステーブルのレコードに対して実行されます。

削除処理では、挿入や更新は行われないため、キーの一意性をチェックしても意味がありません。また、Delete オペレーションも、挿入や更新と同じようにプログラムによる参照整合性チェックは実行されません。これはパフォーマンス上の理由からです。しかし、チェック機能を無効にしないかぎり、通常はデータベースでチェックが実行され、整合性チェックに失敗すると例外がスローされます。

最後に、データベースにコミットする、つまり永久的に削除するレコードについて、Commit コマンドが実行されることを確認する必要があります。プロシージャーには、(ソースがデータベースのどこかにアクセスして更新することが理解されているかぎり) 暗黙的な Commit が既定で最後に置かれます。また、ソース内に明示的に Commit を記述することもできます。

ここでは確認しませんが、レコードごとにではなくブロックで削除を行うための、Blocking 節を指定することも可能です。その場合は、N 個のレコードで構成されるレコードブロックを処理するため、アクセス回数が減り、パフォーマンスが向上します。

挿入、更新、削除の各処理をプロシージャーを介して実行する場合、**冗長性**は自動的には維持されません。これは開発者が行う必要があります。

詳細については、GeneXus の Wiki を参照してください。

プロシージャーから行う CRUD オペレーションとビジネスコンポーネントを介した CRUD オペレーション

プロシージャー固有のコマンドを使用して CRUD (作成/更新/削除) オペレーションを行う場合と、ビジネスコンポーネントを使用して行う場合の比較については、別の章で確認します。