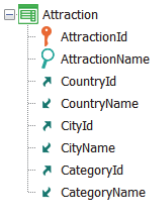
プロシージャー固有のコマンドによるデータベースの更新

データの更新方法

GeneXus™

これまで: New コマンド

	一意性チェック
New 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	2
4	紫禁城	3	1	2

3

New

AttractionId = 3
AttractionName = “エッフェル塔"
CountryId = 2
CityId = 1
CategoryId = 3

When duplicate

for each Attraction
CategoryId = 3
endfor

endnew

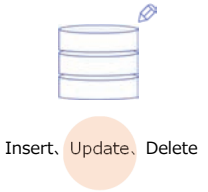
前の資料で、New コマンドを使用し、プロシージャーを介してテーブルにレコードを挿入する方法を学習しました。挿入するレコードが主キーまたは候補キーで重複していた場合は、When duplicate 節内に For each を記述し、それを修正できることを確認しました。

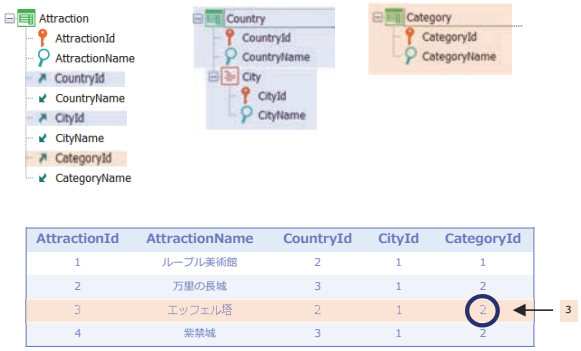
ところで、そのレコードが存在することが既に分かっており、それを更新したい場合はどうでしょうか。
この例では、レコードが存在していることが分かっている観光名所 3 「エッフェル塔」のカテゴリを変更します。

更新

プロシージャーに、特別な Update コマンドはあるでしょうか。

For each コマンド



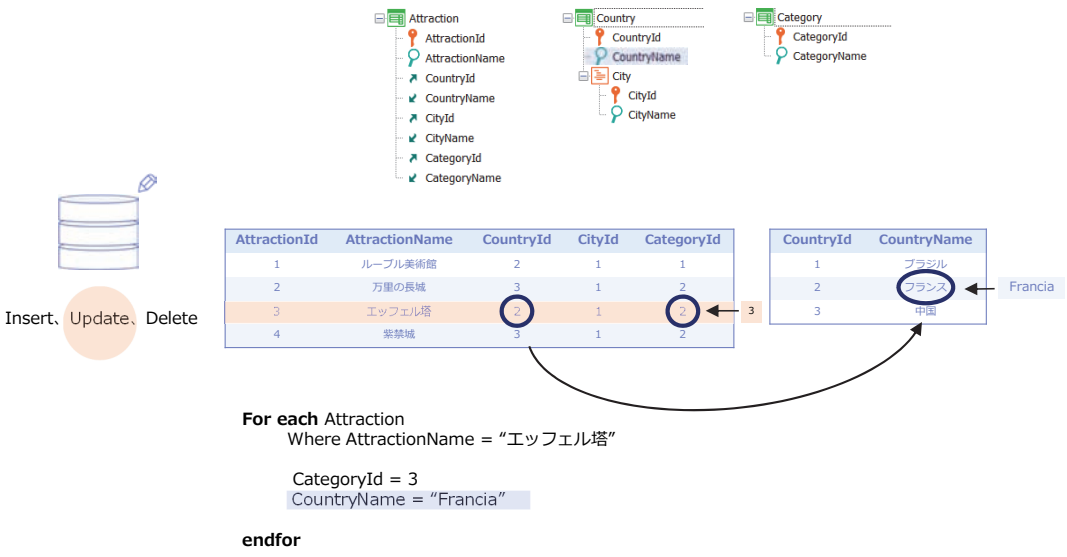



```
For each Attraction
      Where AttractionName = “エッフェル塔”

          CategoryId = 3

endfor
```

そのようなものはありません。For each コマンドを使用してデータを更新します。つまり、For each はデータベースの照会だけでなく、更新にも使用されます。

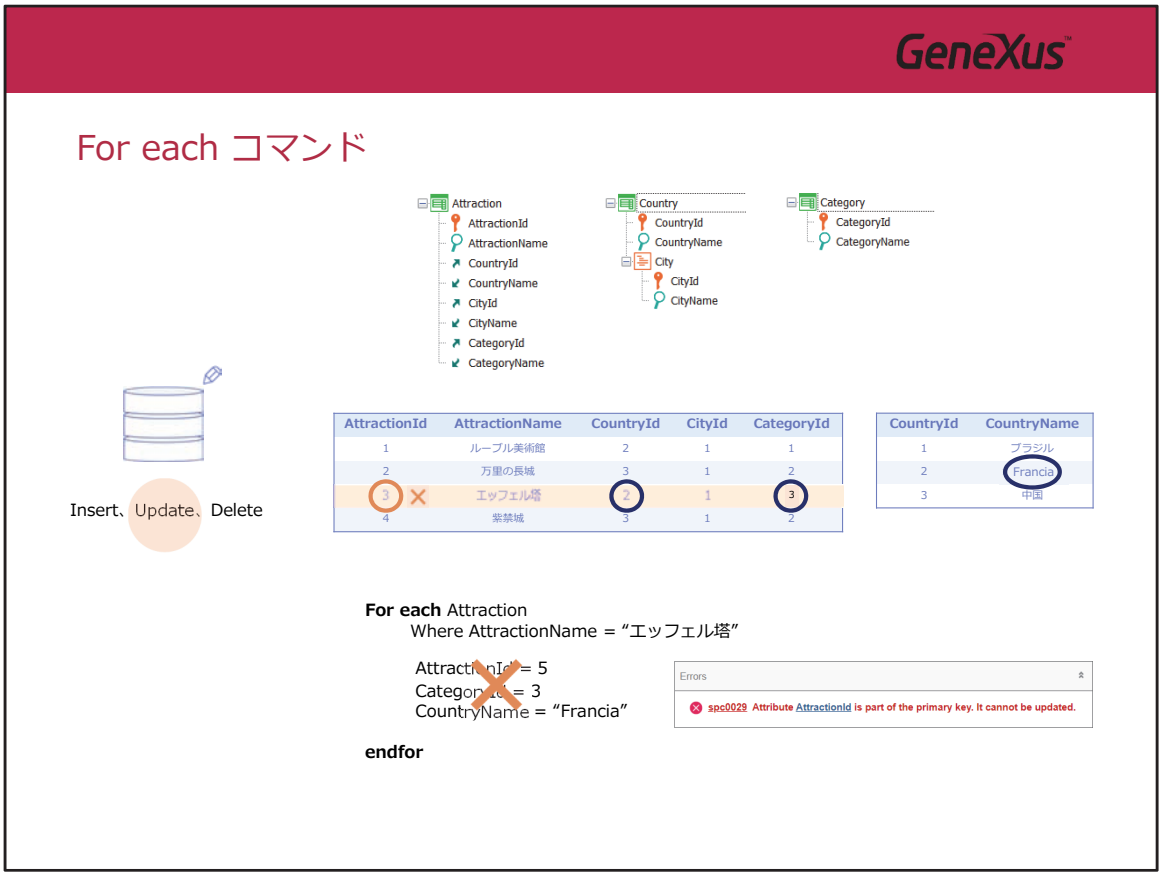
この例で、エッフェル塔に対応するレコードのカテゴリを変更したい場合は、この For each を記述するだけで十分です。Attraction テーブルを参照し、AttractionName = “エッフェル塔” でフィルタします。見つかったレコードについては、変更する項目属性に直接値を割り当てます。対象となるレコードのほぼすべての項目属性の値を変更することもできます。また、そのレコードだけでなく、拡張テーブル内のすべての関連レコードに対しても同じようにできます。New コマンドの場合とは異なり、1 つのオペレーションで多くのレコードを更新することができます。

For each コマンド



たとえば、国の名前をスペイン語に変更するとします。For each は AttractionName が「エッフェル塔」のレコードをポイントし、その CategoryId 項目属性を変更します。また、拡張テーブルで関連する国のレコードにアクセスし、CountryName 項目属性を変更してスペイン語の「Francia」にします。

一方、New コマンドの場合は、Attraction レコードに対する操作のみ行うことができます。

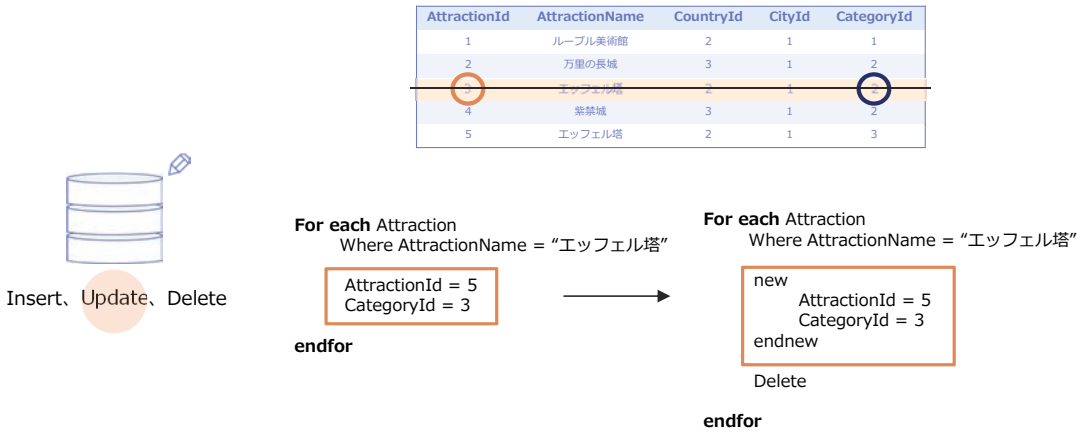


次に確認したいことは、拡張テーブルのすべての項目属性を更新することはできるか、または何らかの制約があるかです。

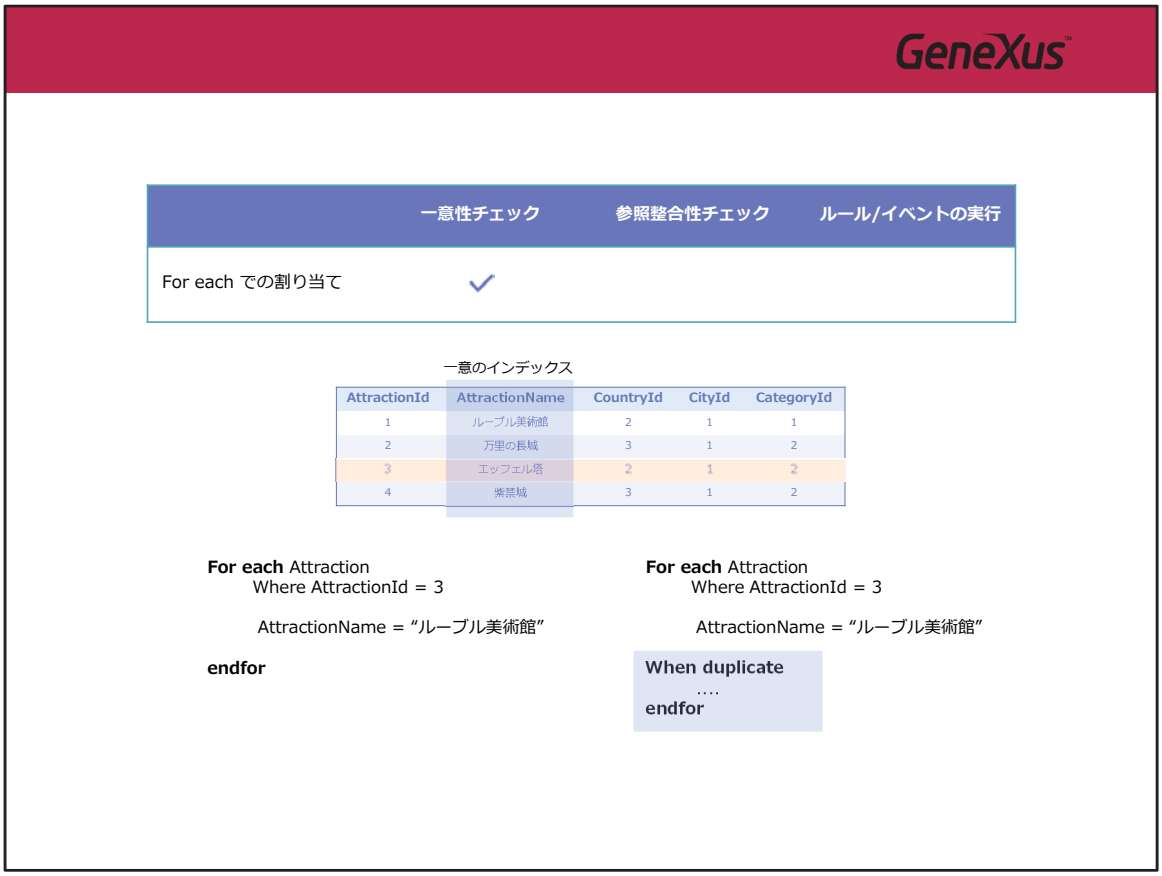
たとえば、主キーの項目属性の変更は可能でしょうか。観光名所の ID を 5 に変更することはできるでしょうか。できません。レコードのすべての項目属性と、拡張テーブルの関連項目属性は、主キー**以外を**更新できます。ナビゲーション表示には、ここに示すエラーが表示されます。

そのため、レコードの主キーを変更する必要が生じた場合は、新しいキーを持つレコードを新たに作成し、古いレコードを削除するしかありません。

For each コマンド



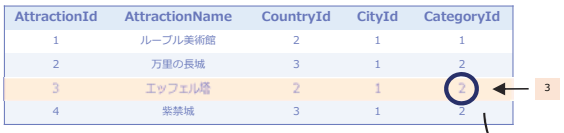
たとえば、観光名所「エッフェル塔」の ID を 5 に、カテゴリを 3 に変更する場合、New コマンドを実行します。ベーステーブルは、同じ Attraction にします。ここで、新しい主キーの値を指定します。残りの項目属性については、現在の For each レコードと同じ値を使用しますが、カテゴリを 3 に変更するため、CategoryId は変わります。
次に Delete コマンドを実行します。このコマンドは、For each でポイントしていたレコードを削除します。この場合は ID 3 のレコードになります。



New コマンドの場合で見たように、For each による更新ではレコードの一意性がチェックされます。この場合は、候補キーがある場合にのみチェックされます。つまり、一部の項目属性に対して一意のインデックスが定義されている場合です。AttractionName の場合を考えてみましょう。For each で、観光名所 3 の名前を変更して「ルーブル美術館」にする必要があるとします。
この更新を行う前に、For each は一意のインデックスを使用して、その値を持つレコードが既存していないことを確認します。この場合は存在するため、何の処理も行われません。New コマンドの場合と同じ動作になります。

ただし、New コマンドの場合と同様に、**When duplicate 節**をプログラミングしていないかぎり、何の処理も行われません。この場合は、コンテンツが実行されます。これについては後で確認します。

	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 での割り当て	✓	✗	



Category テーブル

例外

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

CategoryId = 3

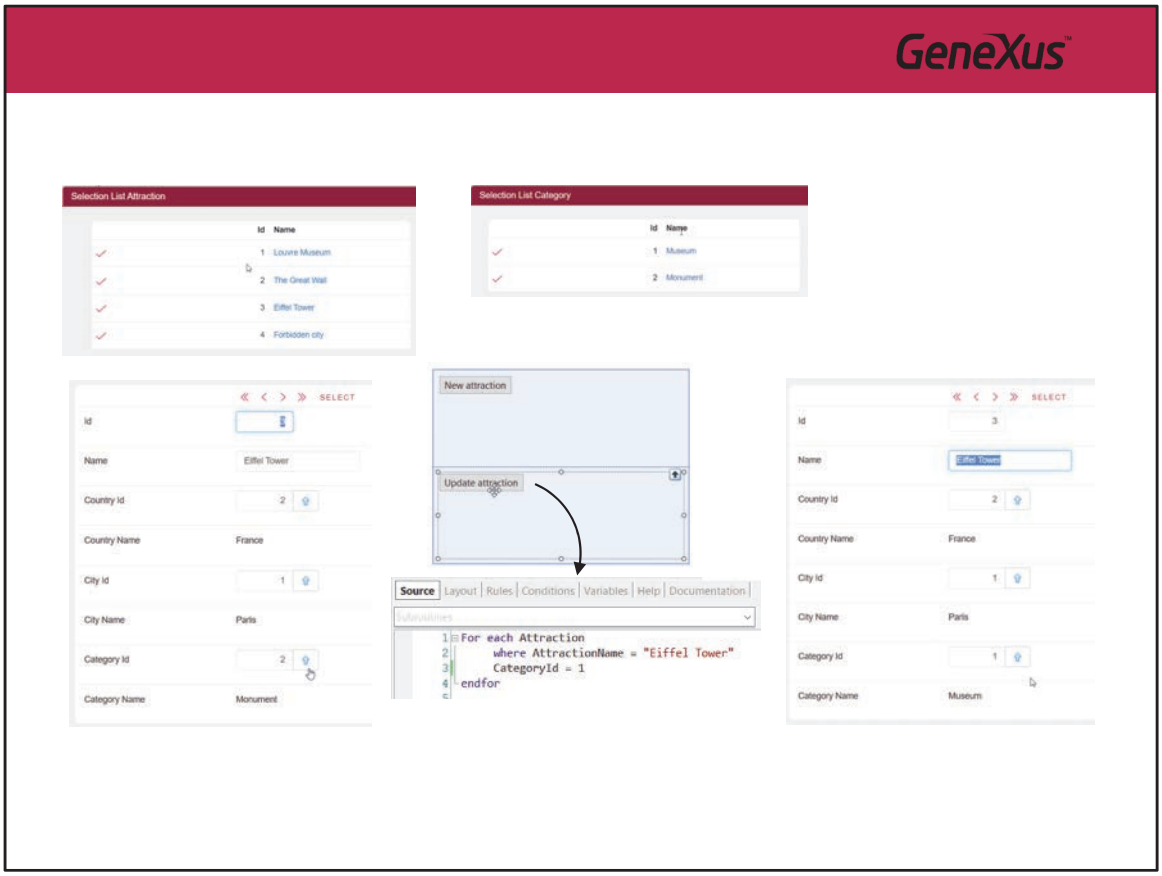
endfor

一方、New コマンドの場合で見たように、For each による更新では参照整合性チェックが行われません。そのため、Category テーブルにカテゴリ 3 が存在しなくても、プログラムでチェックされません。データベースで参照整合性が宣言されている場合は、チェックが行われ、例外がスローされ、プログラムがキャンセルされます。

実際に確認してみましょう。

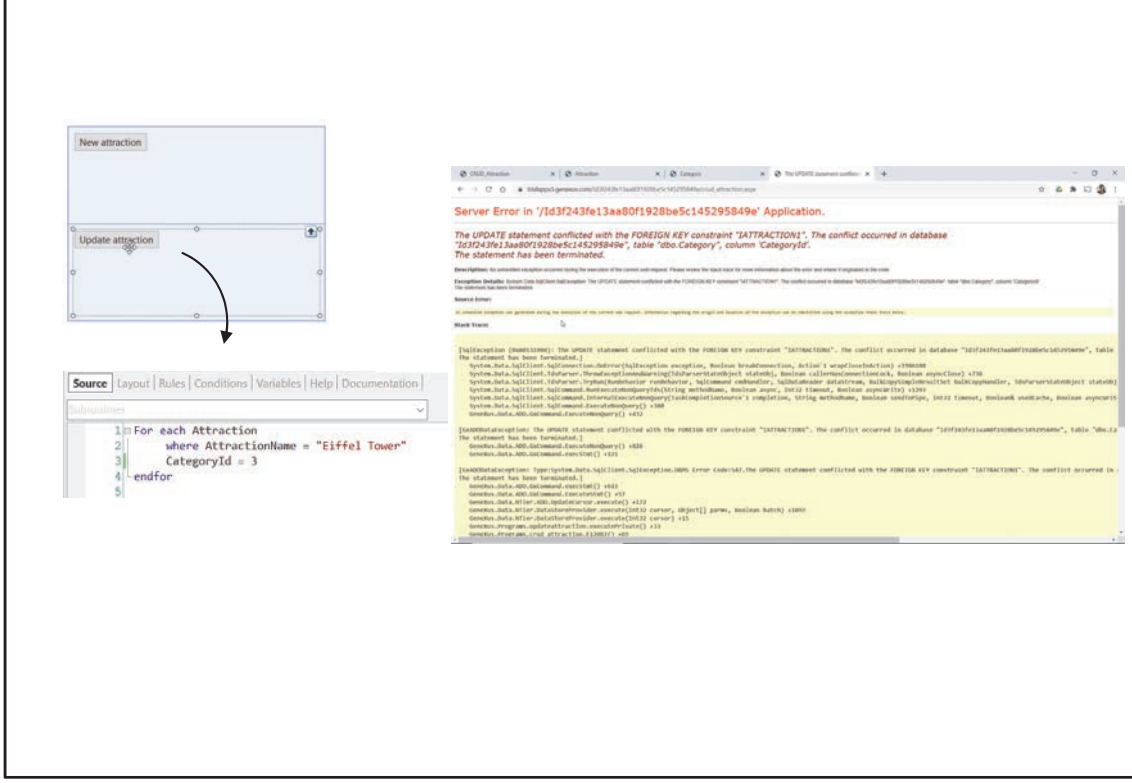
デモ

GeneXus に移動する前に、実行時のデータを確認しましょう。



4 つの観光名所があります。3 番目の「エッフェル塔」はカテゴリ 2 です。
カテゴリのデータを見ると、カテゴリは 2 つだけあります: 1 と 2 です。

これを GeneXus で見てみましょう。この Web パネルでは、ボタンに関連付けられたイベントに、UpdateAttraction プロシージャーへの呼び出しをプログラミングしています。このプロシージャーには、「エッフェル塔」のカテゴリを (存在していることが分かっている) 1 に変更しようと試みる For each があります。



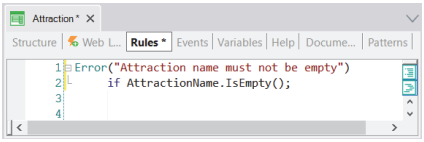
存在することが分かっているカテゴリ 1 ではなく、**存在していない**カテゴリ 3 を割り当てたらどうなるでしょうか。試してみましょう。

プログラムはクラッシュします。For each は整合性チェックを行わずに更新しようとしましたが、データベースでチェックが実行されました。そのため、例外がスローされ、何らかの処理を行うプロシージャーにキャッチされませんでした。これについては別の章で説明します。

	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 での割り当て	✓	✗	✗

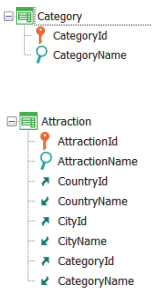
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	2
3		2	1	2
4	紫禁城	3	1	2

```
For each Attraction
      Where AttractionId = 3

      AttractionName = ""

endfor
```



もちろん、New コマンドの場合に説明したように、プロシージャーの For each 内での割り当てによる更新では、どのトランザクションルールも、どのイベントも実行されません。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。

For each コマンド



CategoryId	CategoryName
1	美術館
2	遺跡
3	観光地

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	3
3	エッフェル塔	2	1	2
4	紫禁城	3	1	3
5	コルコバードのキリスト像	1	2	2

```
New
    CategoryName = “観光地”
endnew

For each Attraction
    Where CountryName = “中国” and CityName = “北京”
    Where CategoryName = “遺跡”
    CategoryId = find( CategoryId, CategoryName = “観光地” )
endfor
```

コミット

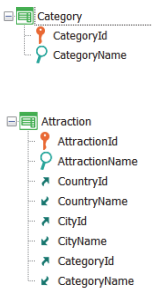
Transaction integrity	
Commit on exit	Yes

この例では、新しいカテゴリ「観光地」を Category テーブルに挿入し、すぐに For each を実行して、国が「中国」、都市が「北京」、カテゴリが「遺跡」の観光名所を参照し、そのカテゴリを「観光地」に変更します。

このコードはプロシージャーのソースでのみ有効です。コミットするとどうなるでしょうか。Category に挿入されたレコードと、Attraction で変更された 2 つのレコードは、いつコミットされるでしょうか。

Commit コマンドをソースに明示的に記述せず、プロシージャーの [Commit on exit] プロパティの既定値を変更しない場合、GeneXus は Commit を最後に追加します。これは、ソースからデータベースを更新しようとしているということを理解しているためです。プロシージャーで、データベースが更新されることを GeneXus が確認できない場合、[Commit on exit] プロパティが Yes に設定されていても Commit は追加されません。

For each コマンド



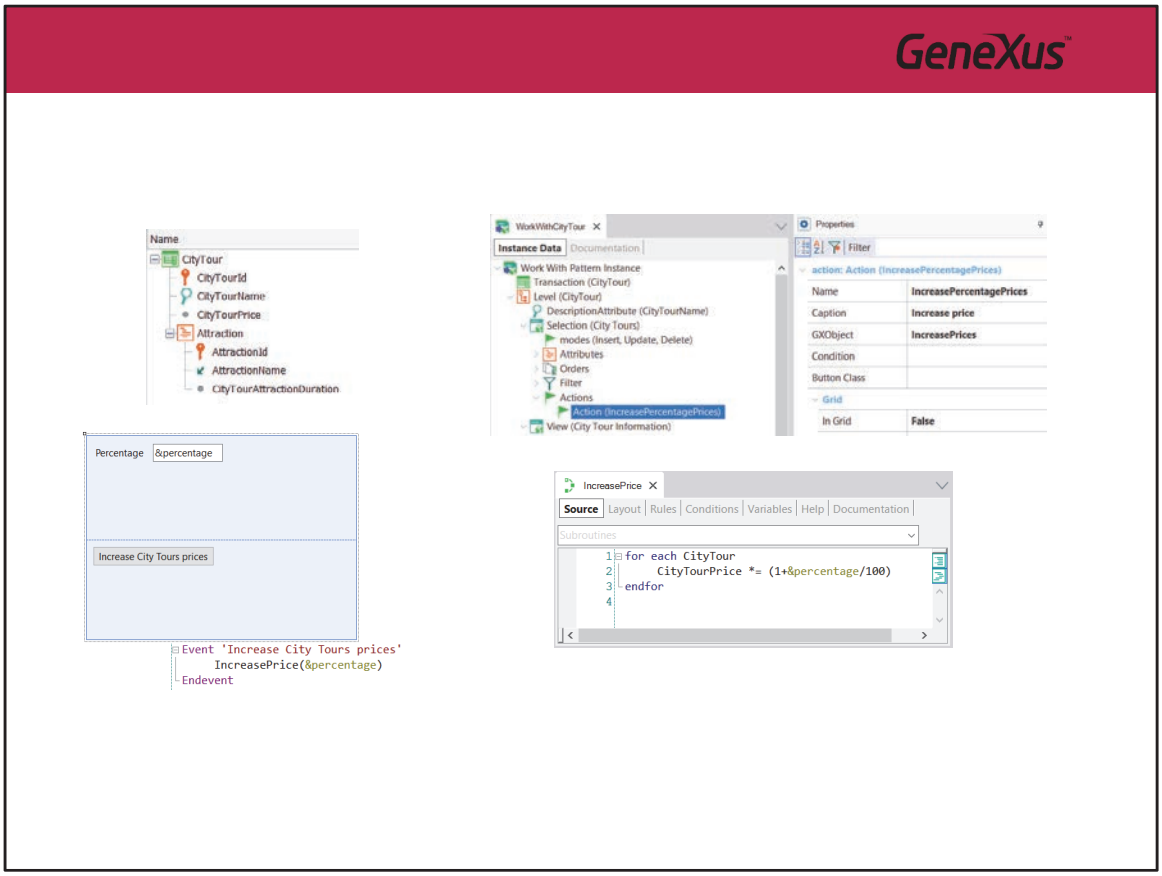
CategoryId	CategoryName
1	美術館
2	遺跡
3	観光地

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	ルーブル美術館	2	1	1
2	万里の長城	3	1	3
3	エッフェル塔	2	1	2
4	紫禁城	3	1	3
5	コルコバードのキリスト像	1	2	2

```
New
    CategoryName = “観光地”
Endnew
Commit

For each Attraction
    Where CountryName = “中国” and CityName = “北京”
    Where CategoryName = “遺跡”
    CategoryId = find( CategoryId, CategoryName = “観光地” )
Endfor
Commit
```

もちろん、カテゴリを挿入した後および観光名所を変更した後でコミットを実行したい場合は、明示的に実行できます。

別の例を見てみましょう。

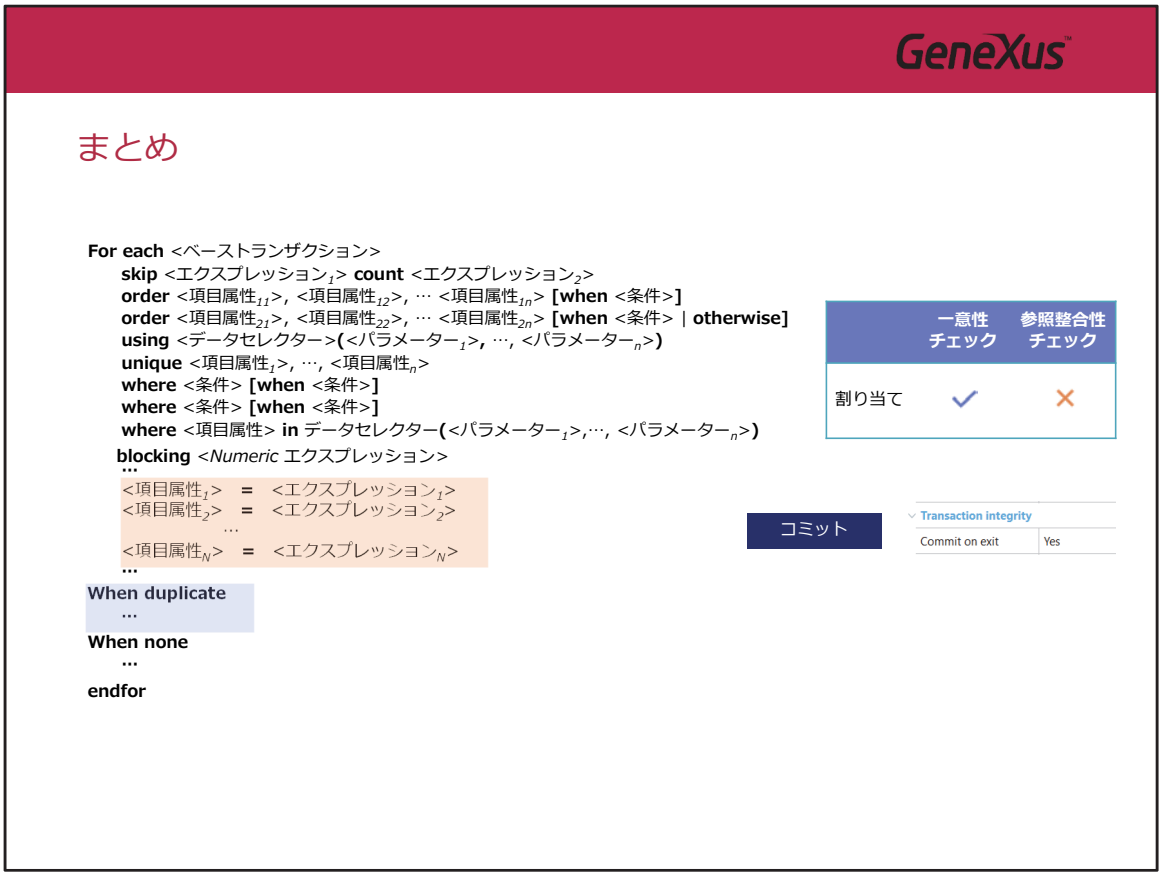


提供する市内観光を表す CityTour トランザクションを追加しました。各市内観光には一連の観光名所が含まれており、それぞれに料金が設定されます。
Work With パターンを適用し、作成した Web パネルを呼び出してユーザーにパーセンテージを尋ね、そのパーセンテージに基づいてすべての市内観光の料金を引き上げるプロシージャーを呼び出すアクションを追加しました。

CityTour テーブルを参照し、CityTourPrice 項目属性に、料金の引き上げを適用する値を新しい値として割り当てます。複合代入演算子を使用して、記述する項目属性の重複を回避することもできます。

ナビゲーション表示に、CityTour テーブルのどの項目属性が更新されているかが示されます。

実行すると、2 つの市内観光が入力されていることが分かります。1 つは値が 300、もう 1 つは値が 200 のものです。次にこの Web パネルを実行して、料金を 10% 引き上げるように指示します。これが実際にどう行われたのかが分かります。



本章の説明を要約すると、プロシージャーでレコードを更新する場合は、For each コマンドを使用します。

その本文で、他の処理に加えて、ベーステーブルと拡張テーブルの両方の項目属性を更新できます。唯一の制約は、For each のベーステーブルの主キーは**更新できない**ということです。

一方、実行されるプログラムによるコントロールは、一意性のコントロールだけでした。For each の場合は、候補キーが重複していないことをチェックします。For each で、割り当ての実行によってキーが重複することが分かった場合、When duplicate 節がプログラミングされていないかぎり、何の処理も実行されません。たとえば、この節のコードでは、新しいレコードを挿入するように New コマンドをプログラミングできます。

For each では**参照整合性チェックが実行されない**ため、参照先のレコードが存在するかどうかを考慮せずに、指定されている更新の実行を試みます。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更新を行うための 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。

ここまでの説明では触れませんでしたが、プロシージャーを使用して更新を行う場合、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。つまり、更新される項目属性によって冗長性が左右される場合、GeneXus は、冗長項目属性に格納する新しい値を探したり計算したりしません。開発者がこの処理を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。

削除

次の資料では、この資料で少し触れた、プロシージャーを使用してデータを削除する方
法を説明します。