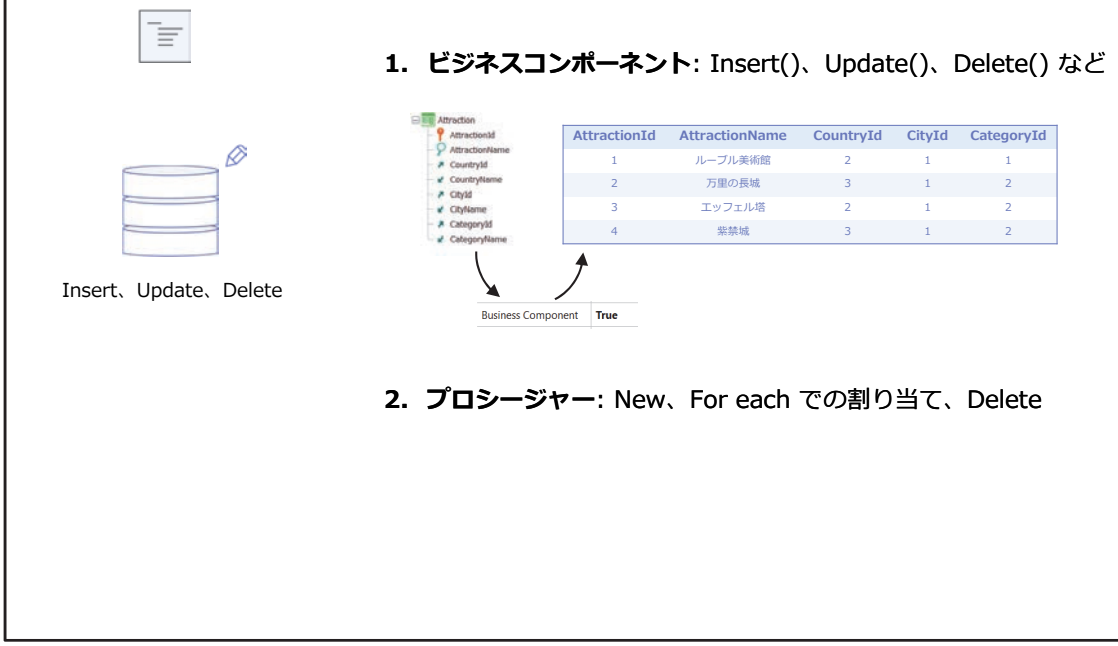
データベースの更新

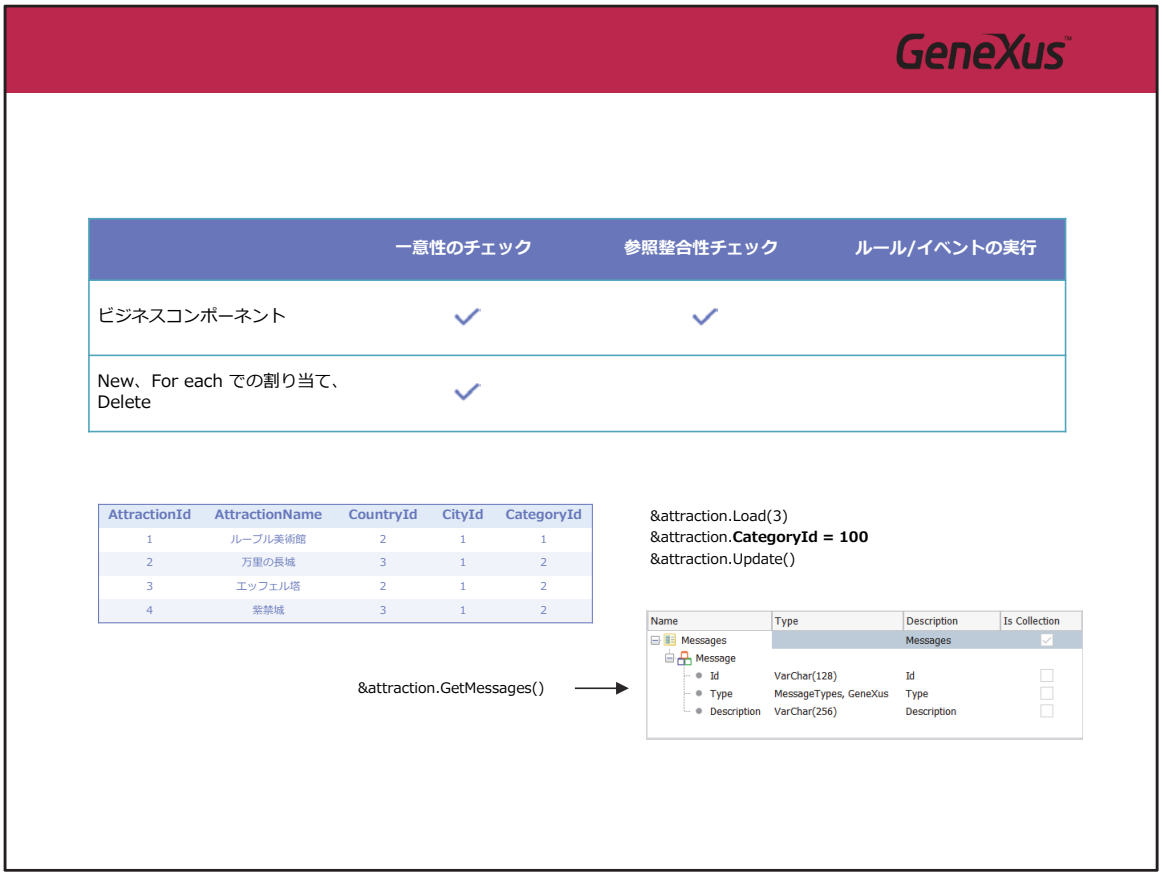
ビジネスコンポーネントを使用する場合と
プロシージャー固有のコマンドを使用する場合

GeneXus™



コードを使用してデータベースを更新する場合、2 つの方法があります。

1 つは、トランザクションのビジネスコンポーネントを使用し、そのメソッドを介して更新する方法です。もう 1 つは、プロシージャー内で New コマンド、For each コマンド内で変更する項目属性に値を直接割り当て、Delete コマンドを実行する方法です。



これまで確認したように、どちらを選択してもレコードの一意性がコントロールされます。つまり、主キーや候補キーが重複しているレコードは、データベースで許可されません。この例では、同じ ID を持つ 2 つの観光名所を設定することはできません。また、たとえば AttractionName が候補キーであった場合、同じ名前を持つ 2 つの観光名所を設定することもできません。
これは、データベースに対してオペレーションを実行する前に、ビジネスコンポーネントまたはプロシージャー内のコマンドによって自動的に重複がチェックされます。

ここまでは同じです。次に、相違点を確認しましょう。

ビジネスコンポーネントを介して挿入、更新、または削除を行う場合は、オペレーションを実行する前に、参照整合性に違反していないかがチェックされます。これはトランザクションでの処理と同じです。たとえば、観光名所 3 のカテゴリを存在しないカテゴリ 100 に変更しようとした場合、ビジネスコンポーネントではまずカテゴリ 100 が Category テーブルに存在するかどうかがチェックされ、存在しない場合は更新を**行いません**。また、エラーメッセージがロードされ、返されるメッセージのコレクションでそれが示されます (指定した場合)。

GeneXus

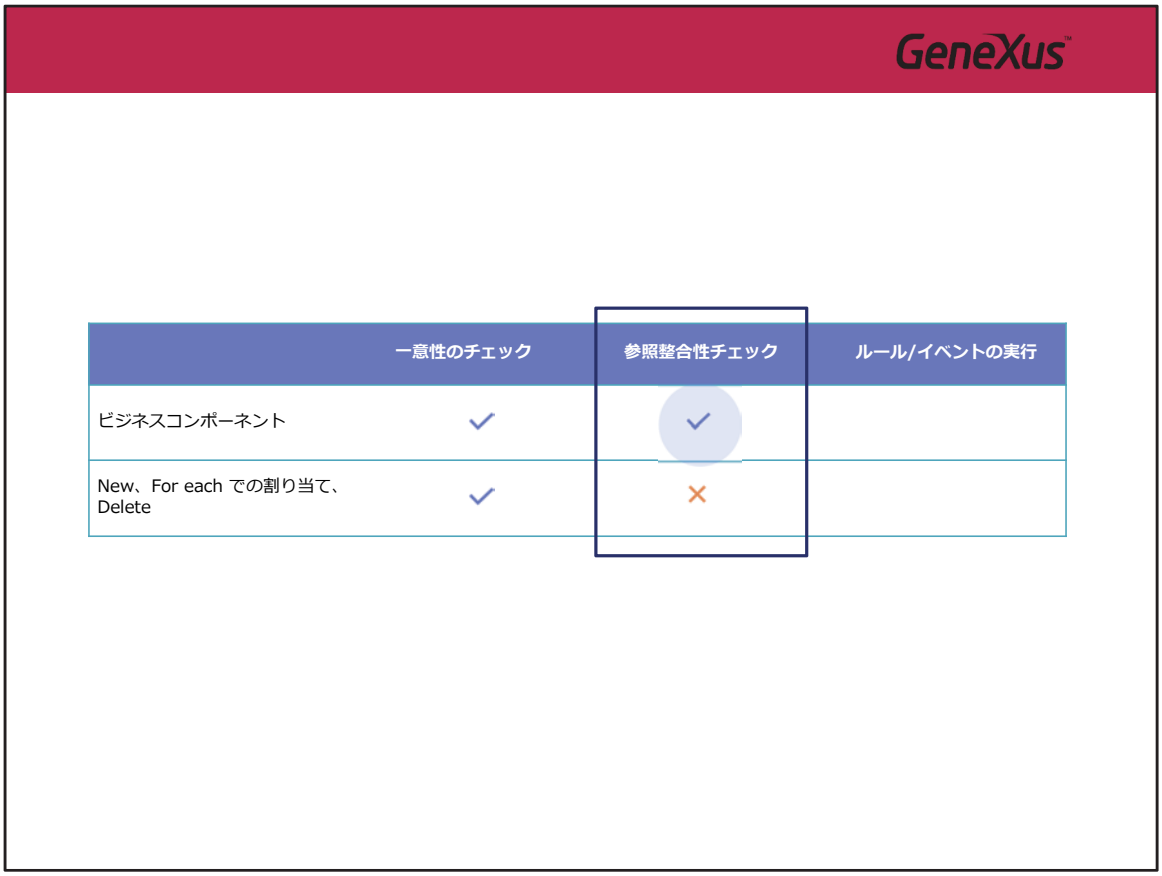
	一意性のチェック	参照整合性チェック	ルール/イベントの実行
ビジネスコンポーネント	✓	✓	
New、For each での割り当て、Delete	✓	✗	

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

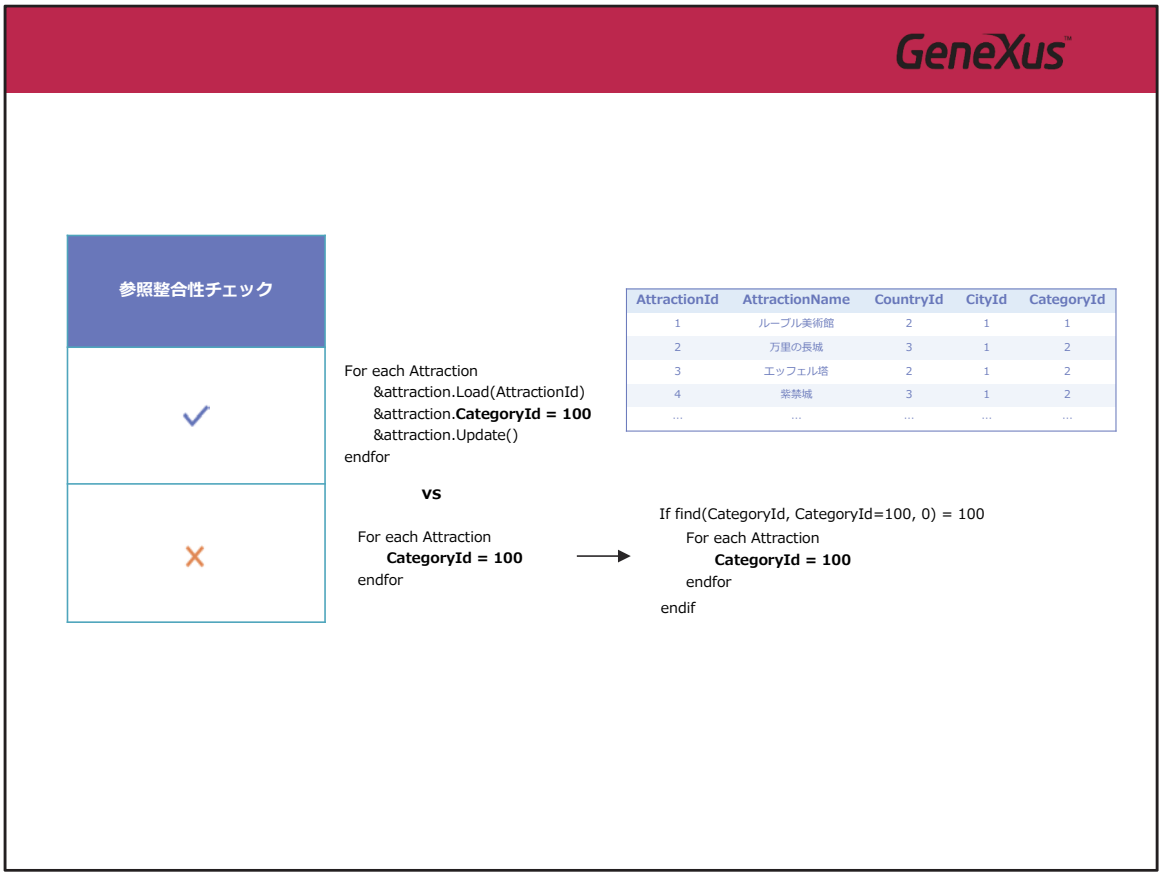
```
For each Attraction
Where AttractionId = 3
    CategoryId = 100
endfor
```

例外

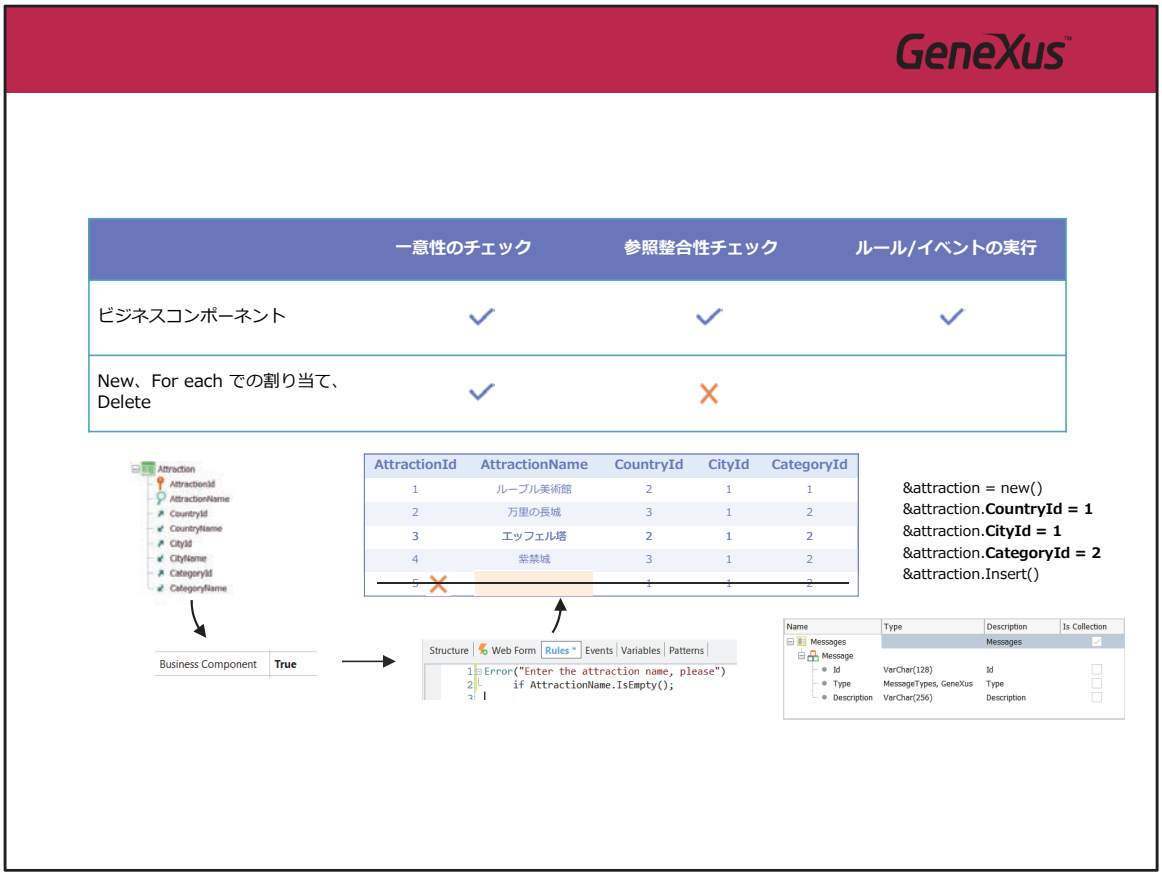
一方、プロシージャー固有のコマンドによって実行されるオペレーションでは、参照整合性チェックは行われません。
たとえば、観光名所 3 のカテゴリを For Each 内での割り当てによって更新する場合、レコードを更新するためにデータベースに命令を送る前に、カテゴリ 100 が存在するかどうかを Category テーブルに照会しません。そのため、データベースでも参照整合性がチェックされない場合は、データの不整合が生じてしまいます。
また、既定の動作でチェックが行われると、例外がスローされ、実行中のプログラムがキャンセルされます。



ここまでの説明では、ビジネスコンポーネントを使用するほうがメリットがあるように思われます。



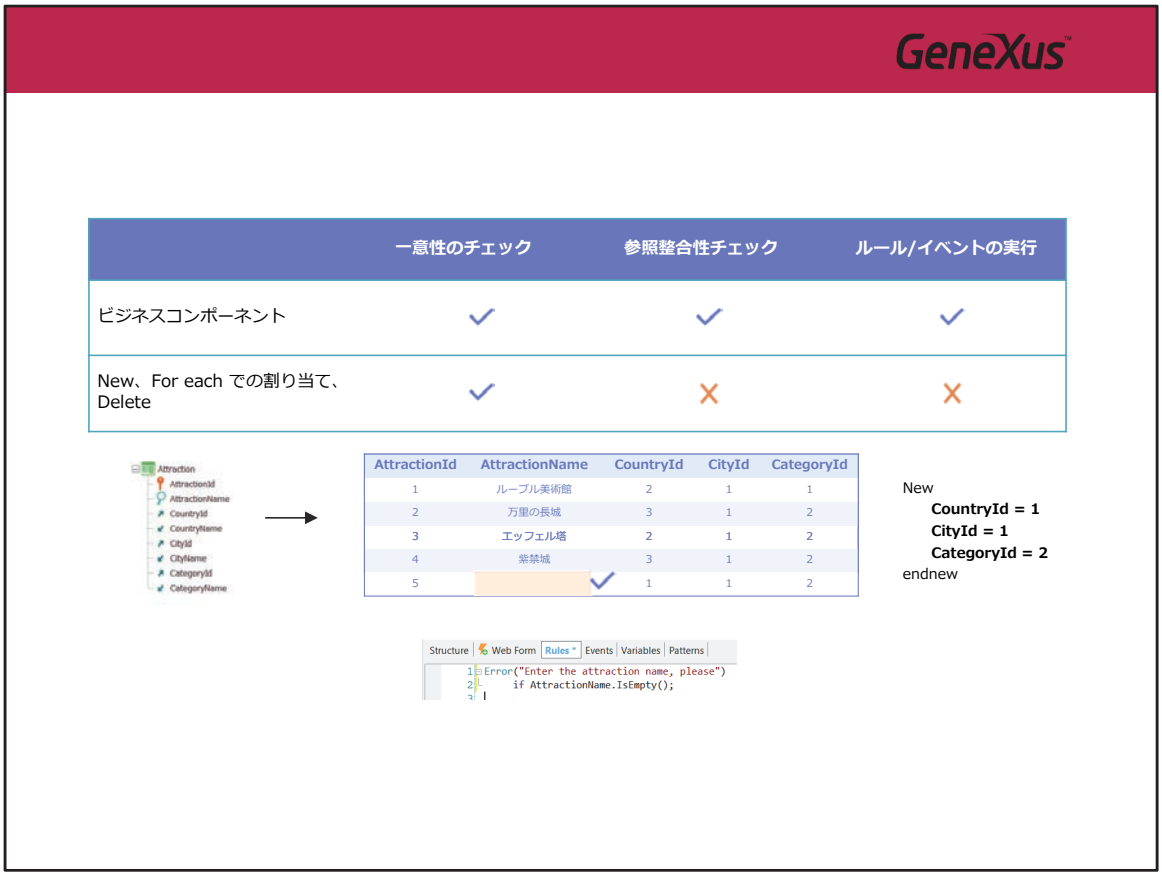
しかし、更新するレコードが大量にある場合は、少し状況が変わります。オペレーションを実行する前にそれぞれのレコードの参照整合性をチェックすると、時間がかかってしまいます。レコードの数が少ない場合はあまり問題になりませんが、レコードの数が百万単位の場合は大きな問題になります。その場合、パフォーマンスが重要であれば、プロシージャー型のコマンドが解決策になります。
この例では、カテゴリ 100 が存在する場合にのみ For Each を実行できます。また、For Each のブロッキング節を使用することで、さらにパフォーマンスを向上させることも可能です。



最後に、トランザクションで定義されているルールやイベントについて見てみましょう。

ビジネスコンポーネントを使用した更新では、ルールやイベント (トランザクションインターフェースとは関係がない、対応するルールやイベント) が実行されます。

たとえば、名前を付けずに観光名所を入力することは許可しないエラールールが設定されている場合に、ビジネスコンポーネントを使用して自動採番された新しい観光名所を入力することはできるでしょうか。そのような処理は許可されません。また、メッセージコレクションでエラーがキャプチャされます。



一方、プロシージャー内でデータベース更新コマンドを使用する場合、トランザクションはまったく関与しません (または、データベーステーブルの定義についてのみ関与)。つまり、ルールやイベントはまったく考慮されません。

そのため、この例でエラールールが設定されていたとしても、プロシージャー内の New コマンドを介した挿入処理ではそれが認識されず、名前が空のレコードも問題なく挿入されます。

ここでも、すべてのデータロジックが実行されるため、ビジネスコンポーネントを使うほうが、プロシージャー内で更新コマンドを使うよりもメリットがあります。

プロシージャー内で繰り返すこともできますが、重複の問題があります。

また、ルールやイベントを実行するのに時間がかかります。パフォーマンスが問題になる場合は、プロシージャー固有のコマンドを使用してデータベースを更新する必要があります。

GeneXus

	一意性のチェック	参照整合性チェック	ルール/イベントの実行	対象
ビジネスコンポーネント	✓	✓	✓	任意のオブジェクト
New、For each での割り当て、Delete	✓	✗	✗	プロシージャーのみ

コミット？

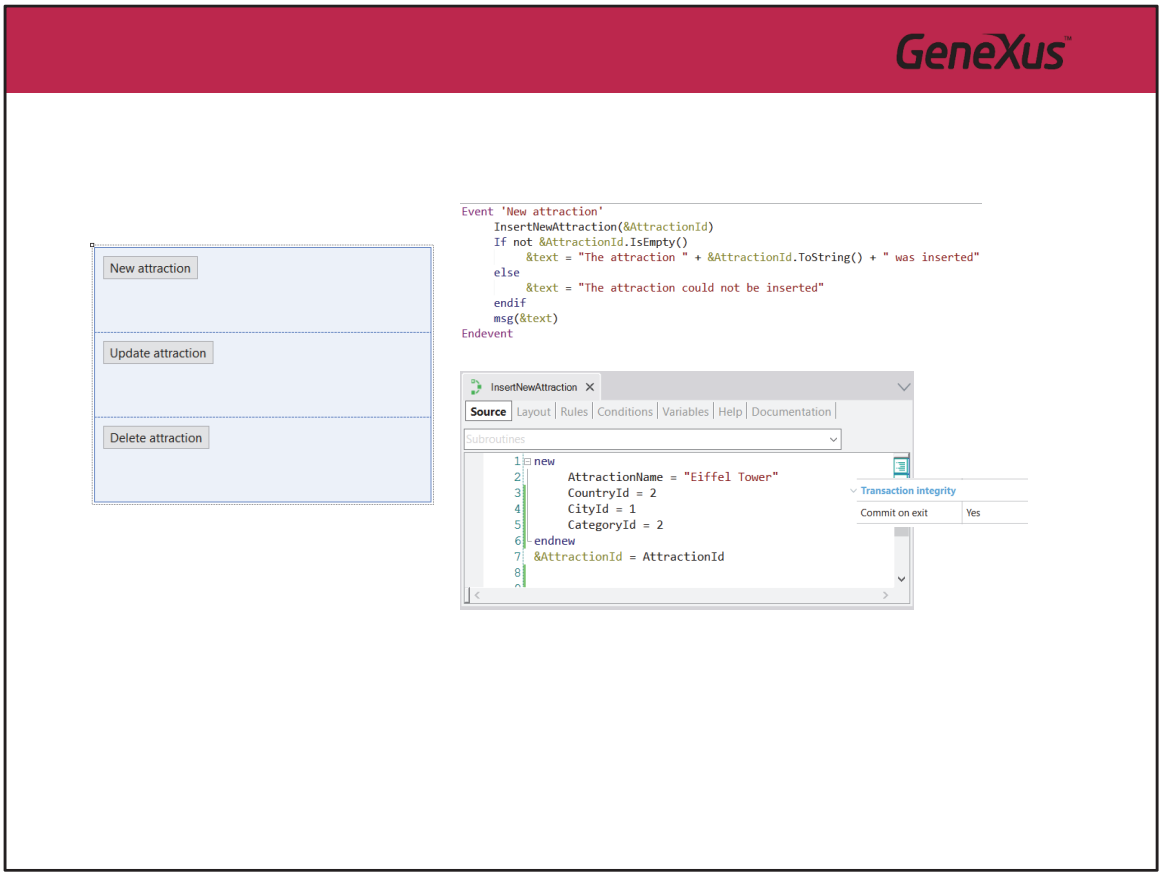
ロールバック

更新コマンドと比較したビジネスコンポーネントのメリットの最後の 1 つは、ビジネスコンポーネントは、Web パネルやパネルイベントなど、コードをサポートするほぼすべてのオブジェクトに対して使用できることです。これに対して、更新コマンドはプロシージャーの中でのみプログラミングできます。

コミットとロールバックについても同じことが言えます。プロシージャーで [Commit on exit] プロパティが既定で Yes に設定されているかどうかに関係なく、開発者はどこで使用するのが最も適切か判断する必要があります。

混乱を招く可能性がある特殊なケースを見てみましょう。

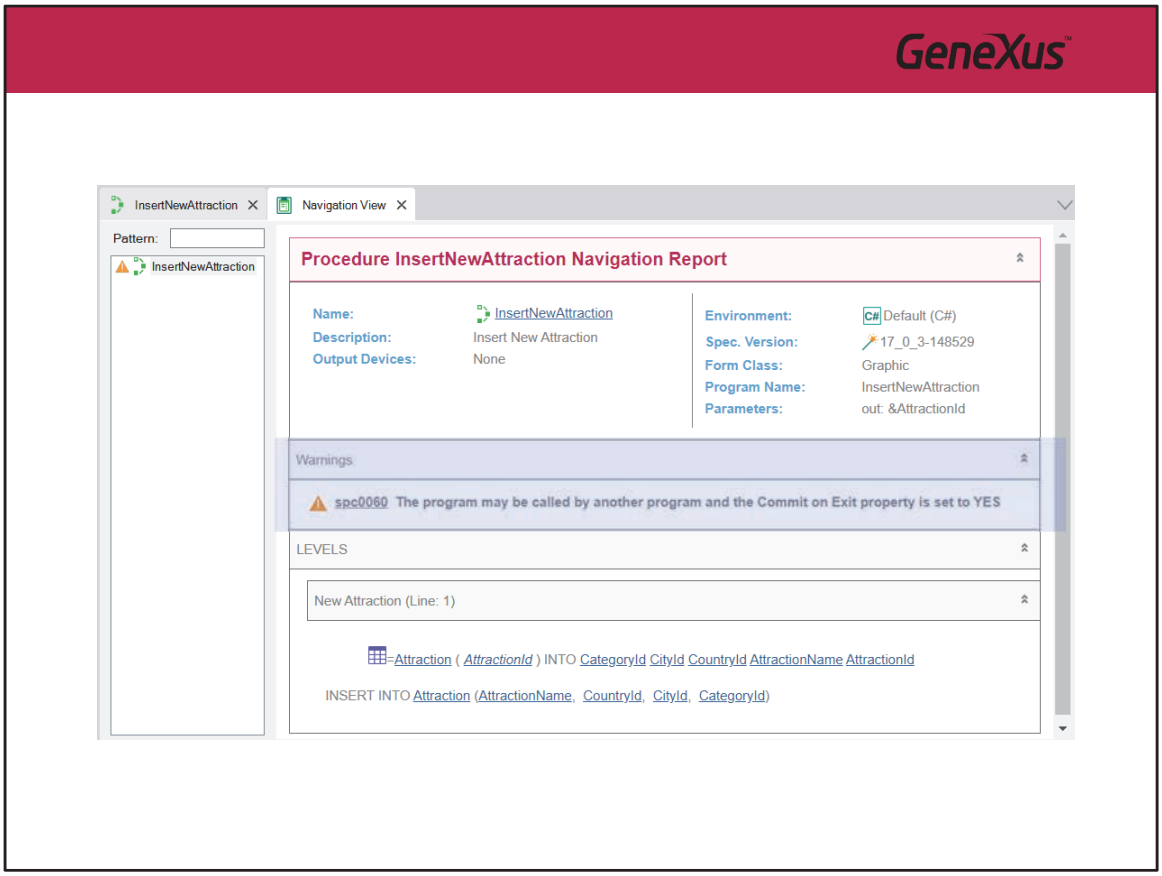
デモ



この Web パネルから、データベースに観光名所を挿入し、自動採番された ID を返すプロシージャーを呼び出すとします。
レコードを New コマンドを使用して挿入するか、ビジネスコンポーネントを使用して挿入するかを決める必要があります。まずコマンドを使って実行してみましょう。

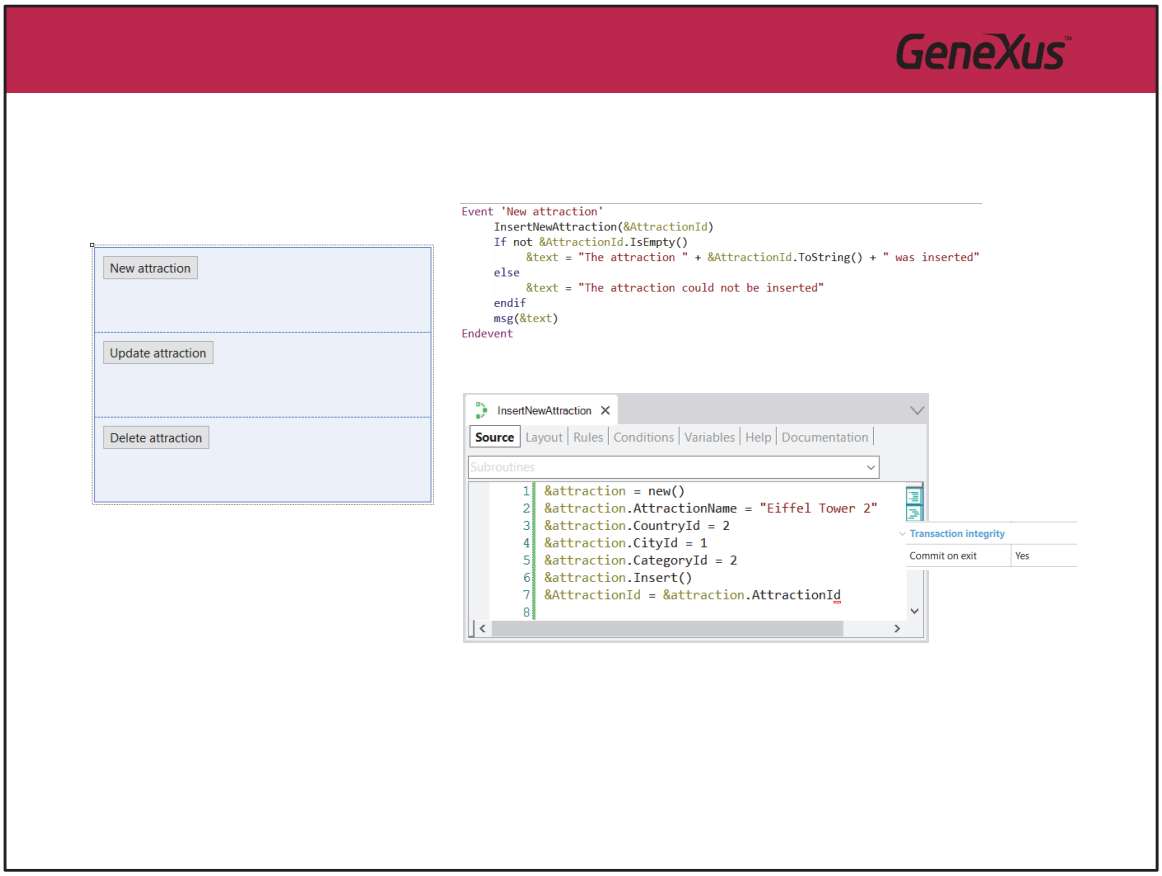
[Commit on exit] プロパティは既定値の Yes に設定されており、オペレーションはデータベースに対して実行されるため、GeneXus はプロシージャーのソースコードの最後に暗黙的なコミットを追加します。

そのため、プロシージャーの実行から戻る際に、このステートメントにレコードを挿入することが可能であった場合、すでにコミットが実行されているはずです。

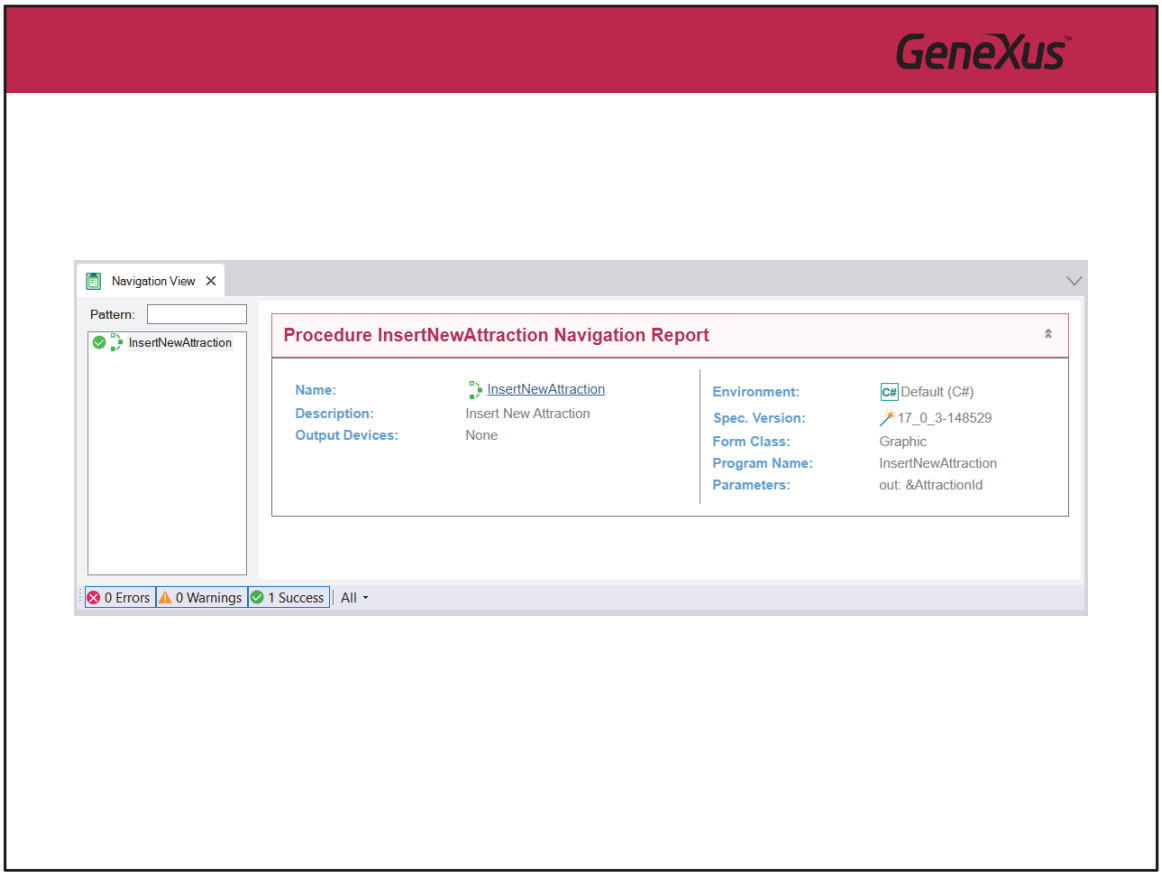


ナビゲーション表示に警告が表示されるのはそのためです。

データベース内の観光名所を見てみましょう。これを実行します。もう一度観光名所を見てみましょう。想定したとおり、新しい観光名所が見つかります。



ここで、New コマンドを使用して更新するのではなく、ビジネスコンポーネントを使用し (「エッフェル塔 2」を指定)、Commit を明示的に指定しないとどうなるか見てみましょう。明示的に指定しないのは、[Commit on exit] プロパティが有効になっているためです。

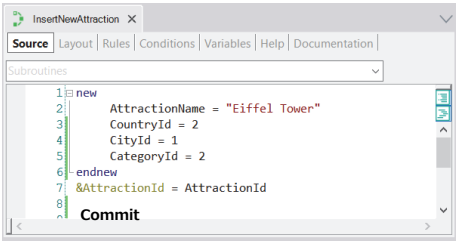


ナビゲーション表示を見ると、何かおかしいことに気付きます。別のケースで表示された、[Commit on exit] プロパティに関する警告が表示されていません。

また、これを実行すると、次の観光名所の番号が示されます。そのため、適切に挿入され、コミットされたと考えられますが、トランザクションでは見つかりません。

何が起こったのでしょうか。

[Commit on exit] プロパティ



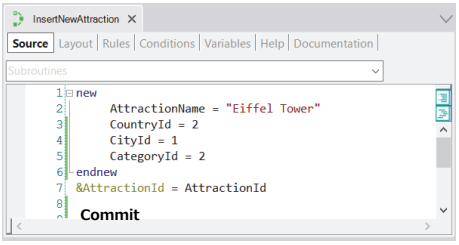
以前の章で、プロシージャーの [Commit on exit] プロパティを Yes に設定しても、GeneXus で常にソースコードの最後に Commit が追加されるわけではないことを繰り返し説明しました。

プロシージャーのソースのどこかでデータベースにアクセスし、CRUD (作成、更新、削除) オペレーションのいずれかを実行すると分かった場合にのみ Commit が追加されます。

最初のケースでは、New コマンドが明らかに CRUD オペレーションを表すため、Commit が追加されます。

しかし、2 番目のケースでは、そのようなオペレーションは認識されません。ビジネスコンポーネントを、その構造から SDT として捉えるため、Insert メソッドを適切に解釈することができません。このプロシージャーのソースでは、CRUD オペレーションのいずれかが行われることが理解できません。そのため、ソースコードの最後に Commit が追加されません。

[Commit on exit] プロパティ



このケースでは、これまでのところ、明示的に指定する以外の方法はありません。

ここで実行してみると、予想したとおりに動作します。

GeneXus

	一意性のチェック	参照整合性チェック	ルール/イベントの実行	対象
ビジネスコンポーネント	✓	✓	✓	任意のオブジェクト
New、For each での割り当て、Delete	✓	✗	✗	プロシージャーのみ

コミット？

ロールバック

この章では、CRUD オペレーションをビジネスコンポーネントを使用して実行する場合と、特定のコマンドを介してプロシージャー内で直接実行する場合の違いについて学習しました。