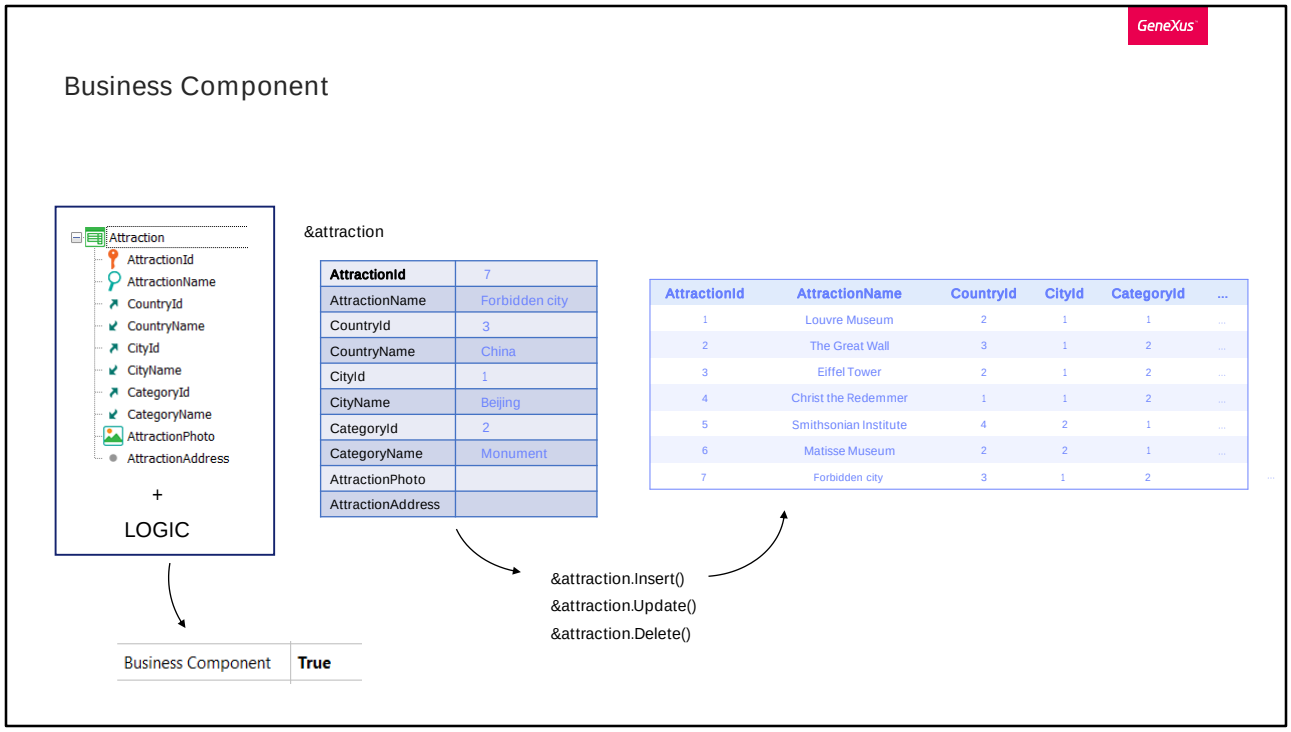
Atualização da base de dados

Usando Business Components

GeneXus



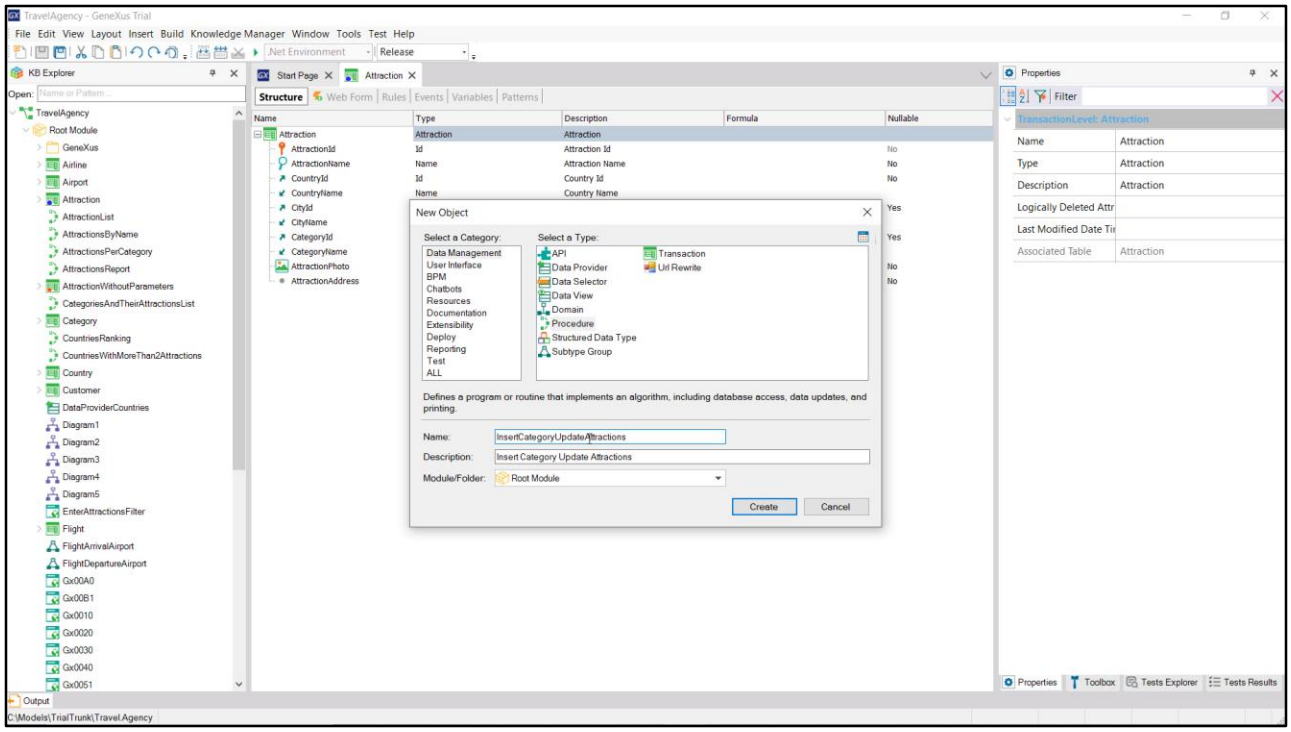
No vídeo anterior, fizemos um percurso parando em lugares conhecidos para preparar o caminho da compreensão e o uso dos business components.

Da estrutura da transação com sua lógica (e por lógica, entendemos os controles de duplicados –não só de chave primária, mas também de chaves candidatas–, de integridade referencial, a maioria de suas regras e alguns de seus eventos), obtém-se uma espécie de tipo de dados semelhante a um SDT, mas muito mais potente.

Então, conseguirá definindo em quase qualquer programa uma variável desse tipo de dados e manipulá-la, que é o que veremos a seguir.

Essa variável, em sua estrutura, será manejada de modo similar a um SDT. Mas também oferecerá métodos para realizar o que é específico de uma transação: carregar da base de dados, inserir, modificar, apagar; tudo isso executando a lógica da transação, e depois obter as mensagens geradas e os resultados de sucesso ou fracasso.

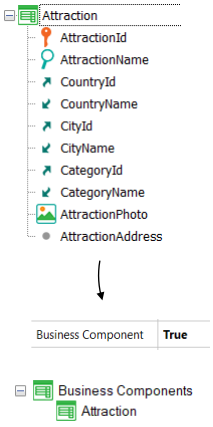
A forma de obter esse tipo de dados na base de conhecimento é simplesmente ativando a propriedade Business Component da transação.

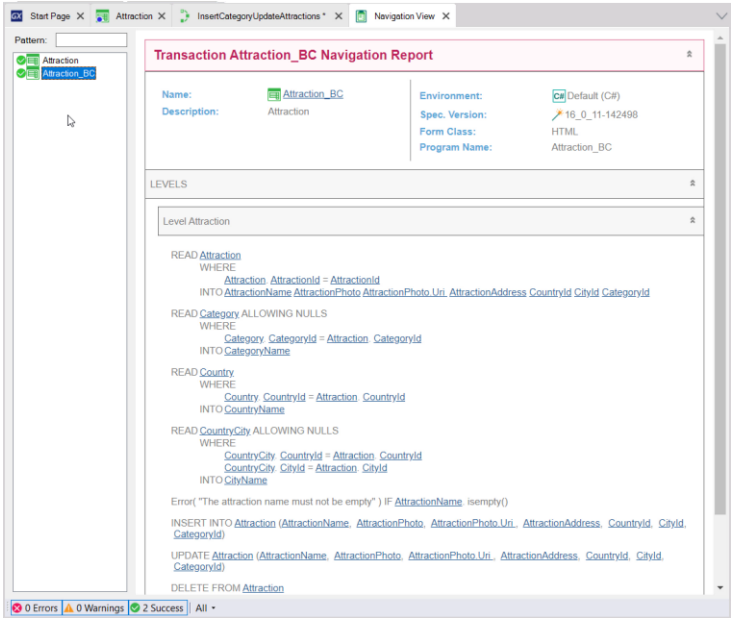


Vamos começar criando um procedimento simples que iremos modificando para implementar um requisito mais completo mais adiante.
Não prestemos atenção ao nome agora.
Comecemos por algo bem simples: suponhamos que queremos inserir uma nova atração turística.

GeneXus

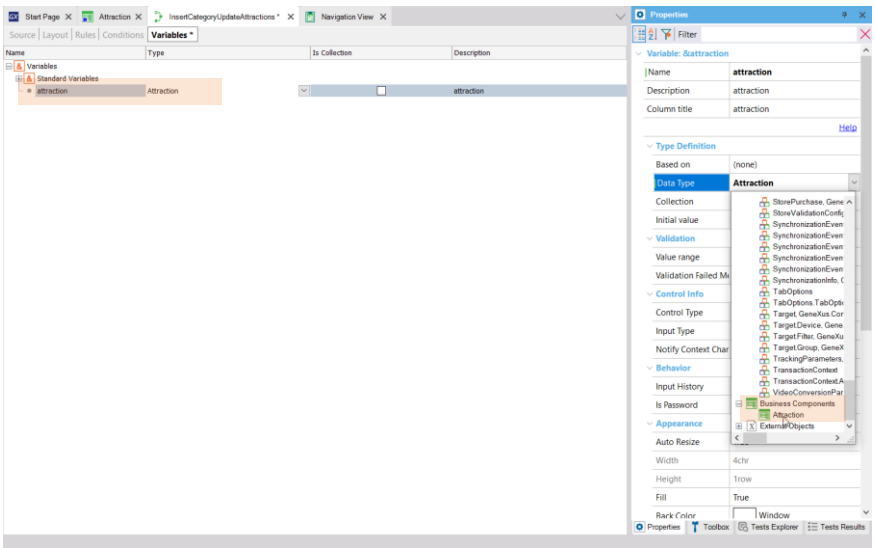
Insert through BC





A primeira coisa que devemos fazer é criar o Business Component da transação Attraction, para que possa ser utilizado em qualquer objeto da base de conhecimento. Portanto, vamos à transação e entre suas propriedades localizamos a de nome Business Component. Como vemos, por padrão está em False. Mudamos para True. A princípio, não vemos nenhum efeito. No entanto, se gravamos... vemos isto no mapa de navegação.

Insert through BC



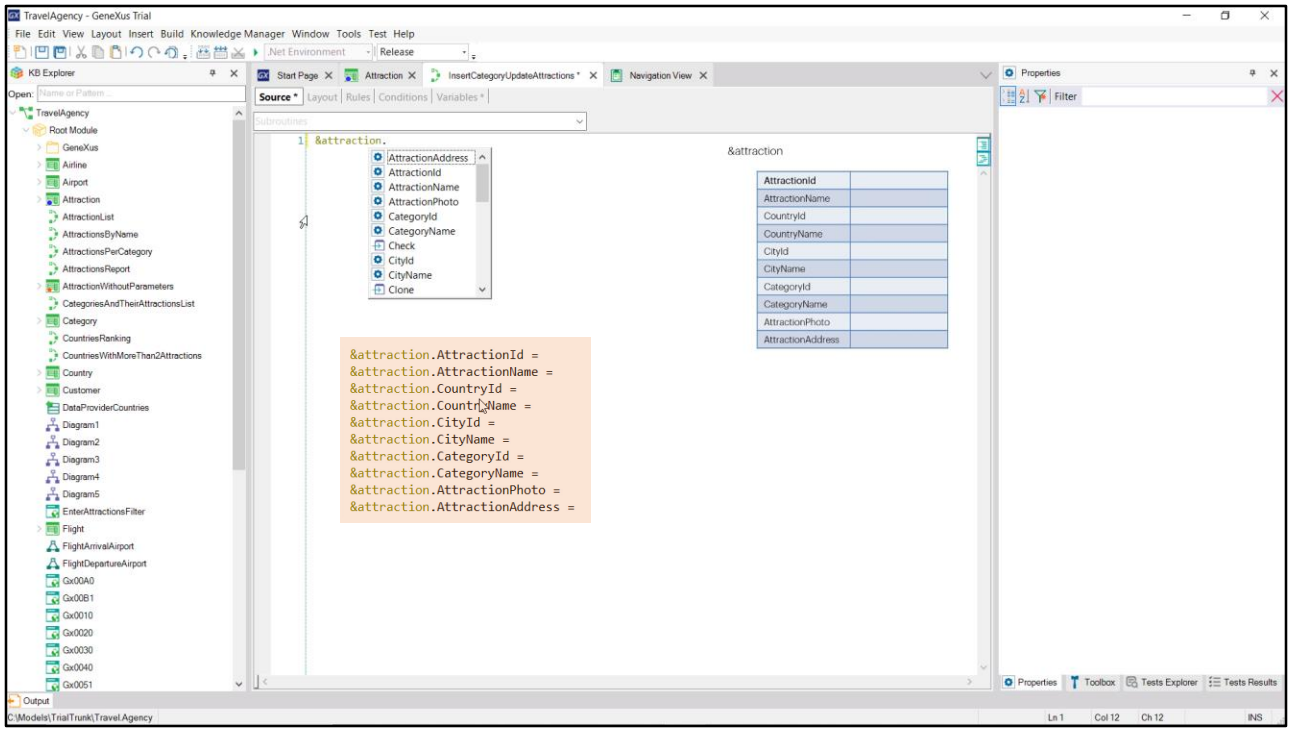
No vídeo anterior, fizemos um percurso parando em lugares conhecidos para preparar o caminho da compreensão e o uso dos business components.

Da estrutura da transação com sua lógica (e por lógica, entendemos os controles de duplicados –não só de chave primária, mas também de chaves candidatas–, de integridade referencial, a maioria de suas regras e alguns de seus eventos), obtém-se uma espécie de tipo de dados semelhante a um SDT, mas muito mais potente.

Então, conseguirá definindo em quase qualquer programa uma variável desse tipo de dados e manipulá-la, que é o que veremos a seguir.

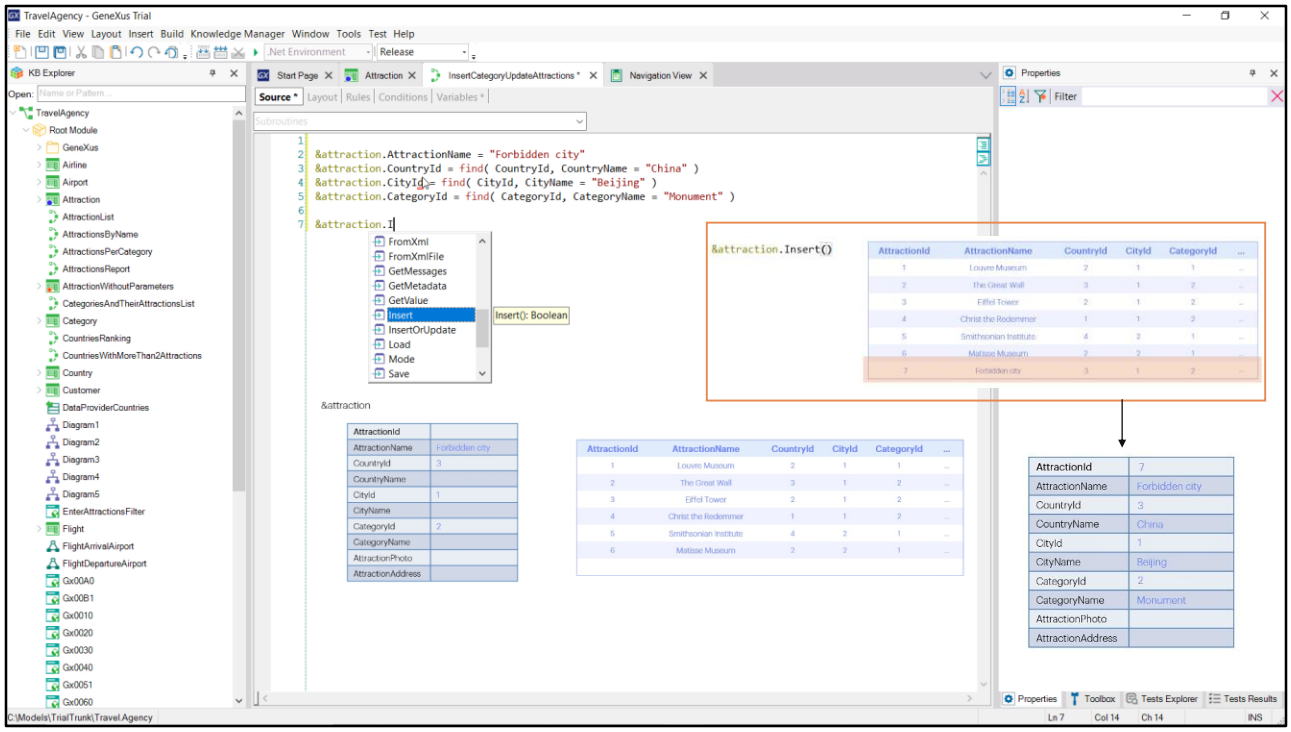
Essa variável, em sua estrutura, será manejada de modo similar a um SDT. Mas também oferecerá métodos para realizar o que é específico de uma transação: carregar da base de dados, inserir, modificar, apagar; tudo isso executando a lógica da transação, e depois obter as mensagens geradas e os resultados de sucesso ou fracasso.

A forma de obter esse tipo de dados na base de conhecimento é simplesmente ativando a propriedade Business Component da transação.



Ao ter definido esta variável no procedimento, automaticamente, se reserva o espaço de memória para poder carregar como um SDT todos seus elementos.

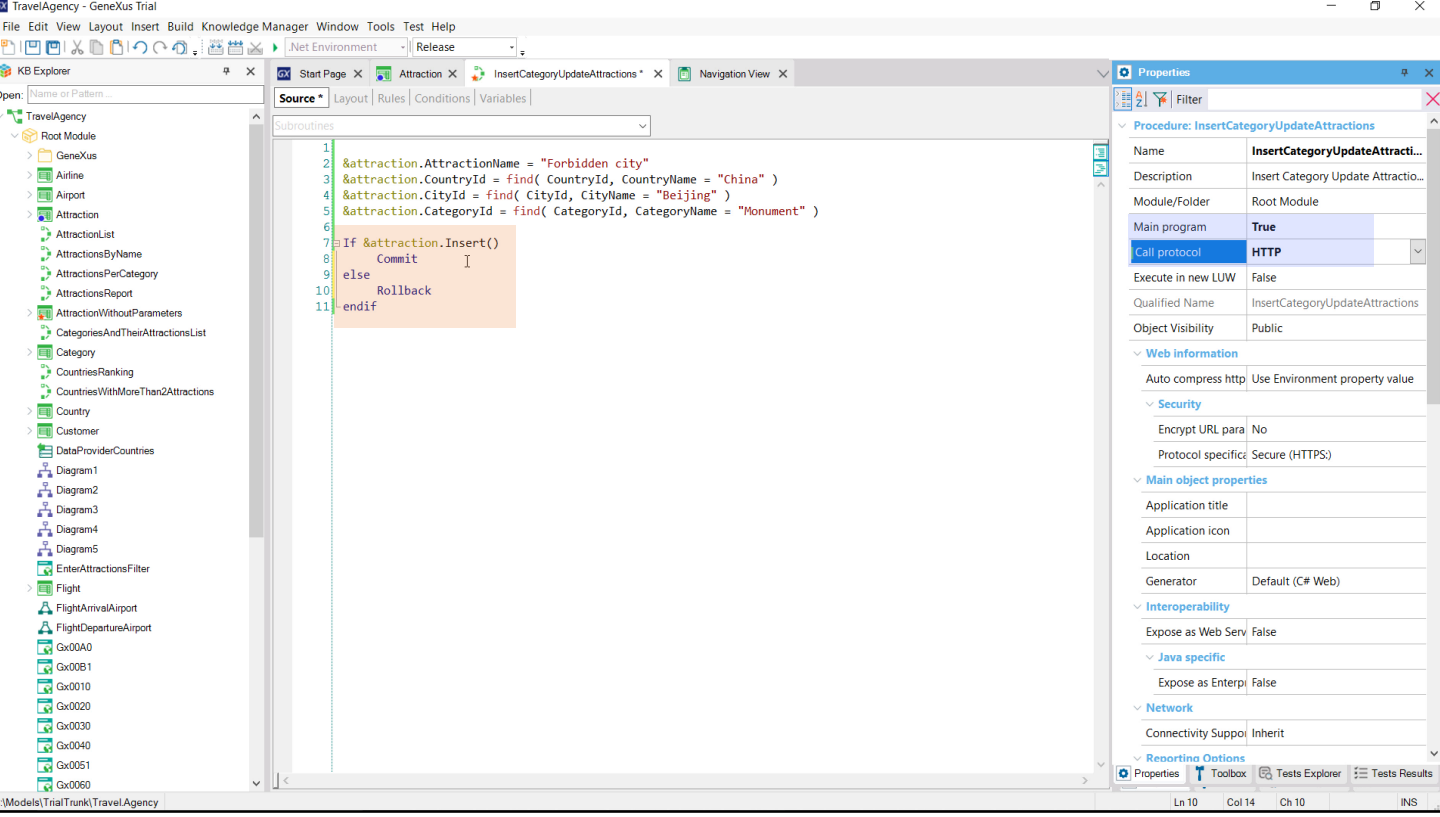
Assim, se inserimos a variável e digitamos ponto, vemos que nesta janela nos mostra os nomes de todos os atributos da transação, entre outras coisas, para que possamos utilizá-los na estrutura, por exemplo atribuindo-lhes valor. Assim, poderíamos fazer...



Se queremos inserir a atração turística a cidade proibida, como vimos antes quando usamos a transação, teremos que completar os dados.
Ao identificador não precisamos atribuir valor, dado que será autonumerado.
O nome de atração será... que sabemos estar na China, que tem o identificador 3, mas e se estivermos errados? Melhor procurar esse identificador com a fórmula find, porque do que estamos seguros é do nome do país, China.
Esse nome é um atributo inferido na transação, então não temos que atribuir valor aqui para inserir a atração.
A cidade sabemos que é Beijing, assim procuramos seu identificador e o atribuímos ao elemento do business component e removemos também o elemento inferido.
Atribuímos valor para o id de categoria, procurando pelo nome "Monument".
E à foto e endereço, não atribuiremos valor.

Temos então a variável carregada com os dados da atração que queremos inserir na tabela. Resta-nos dar a ordem para inserir. Para isso, contamos com o método Insert da variável Business component, que, como vemos, devolverá um booleano: True se pôde inserir, False em caso contrário.
Podemos simplesmente invocá-la e tentará inserir um novo registro

exatamente como a transação faz, ou seja, executando todas as regras e validações. Se conseguir, então a variável ficará carregada com o id que lhe deu a base de dados e todos os atributos próprios e inferidos.

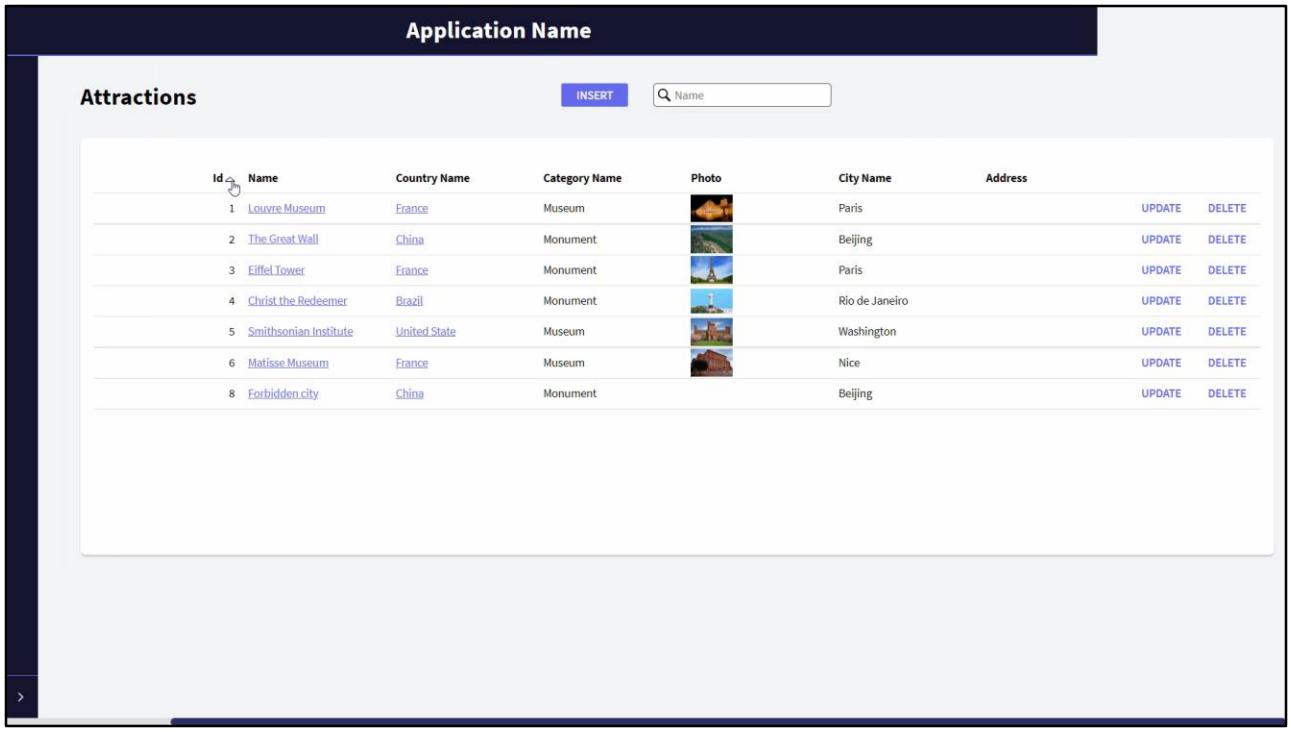


No entanto, devemos saber que, embora o registo já esteja na tabela, está num estado instável. Se houver um apagão ou o sistema cair por qualquer outra razão, quando a base de dados for restabelecida esse registro já não estará lá. Pois, as bases de dados nos permitem ir inserindo, modificando, excluindo registros como se fosse a um nível lógico e quando quisermos, no momento em que nos parecer conveniente, devemos indicar que todas essas operações que estivemos realizando já podem ser efetivadas. Isto é feito com o comando Commit. Portanto, se a inserção foi bem-sucedida, commitamos. Chamamos assim à ação de realizar um commit sobre a base de dados, ou seja, indicar-lhe que realmente impacte todas essas operações, que as deixe fixas, permanentes, na base de dados. Quando se executa um Commit, todas as operações que tenham sido realizadas entre o commit anterior e o atual serão efetivadas.

Assim como temos o Commit para efetivar todas essas operações, temos o oposto, o Rollback, para desfazer tudo o que se tenha feito depois do último Commit. Aqui poderíamos especificar um Rollback se a operação não foi bem-sucedida... Mas não terá muito sentido, já que se a operação não foi bem-sucedida, podemos supor que o registro não chegou sequer a ser inserido, portanto não haverá nada para desfazer.

Para executar este procedimento rapidamente a partir do Developer Menu, o especificamos como o objeto principal e com protocolo de invocação HTTP.

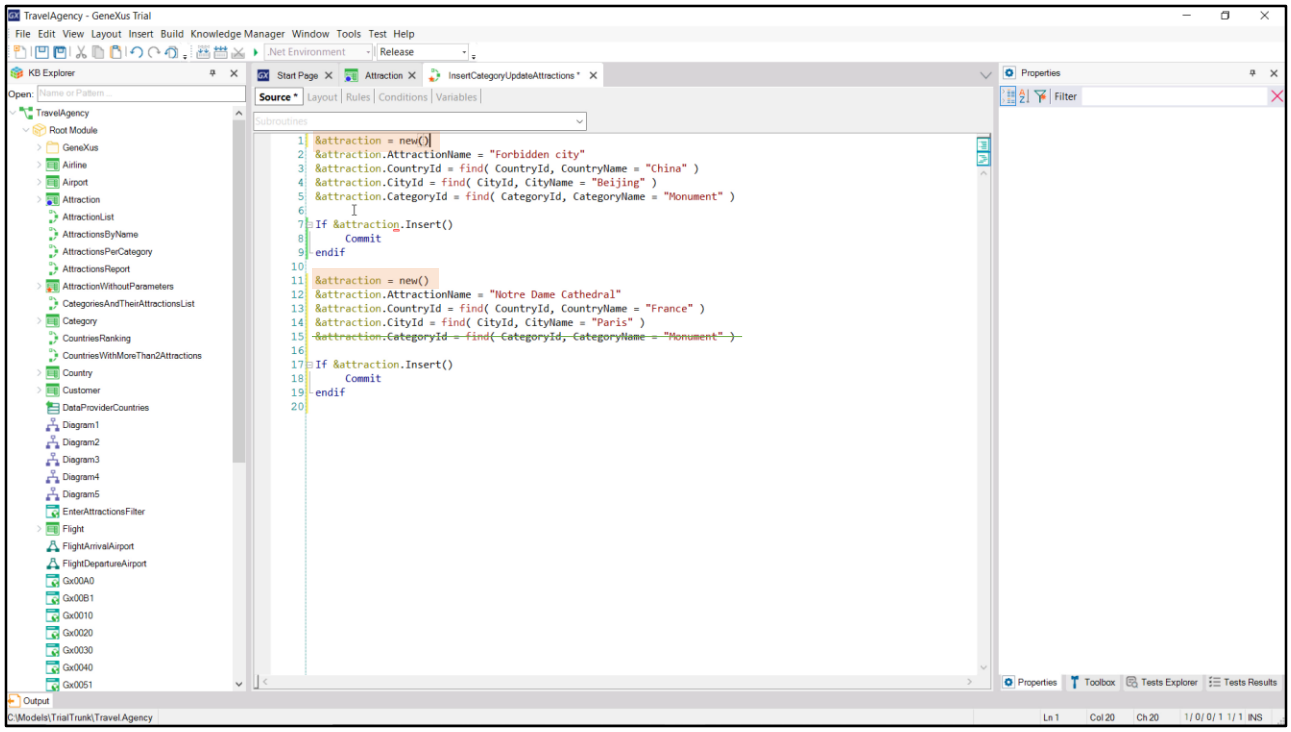
Pressionemos F5.



Se formos trabalhar com atrações, veremos que já tínhamos a atração 7 para a cidade proibida, que tínhamos inserido antes, através da transação. Vamos eliminá-la (o valor 7 do identificador será perdido, não será dado novamente para a próxima atração que se insira, que ficará com 8, como veremos agora, se executamos nosso procedimento).

Como não tem nenhuma saída, não vemos nada no navegador, mas se voltarmos a trabalhar com atrações... vemos que inseriu efetivamente a cidade proibida.

Vamos apagá-la novamente, para voltar a executar o procedimento, mas, ao qual adicionamos outra atração além dessa. Por exemplo, a catedral de Notre Dame.

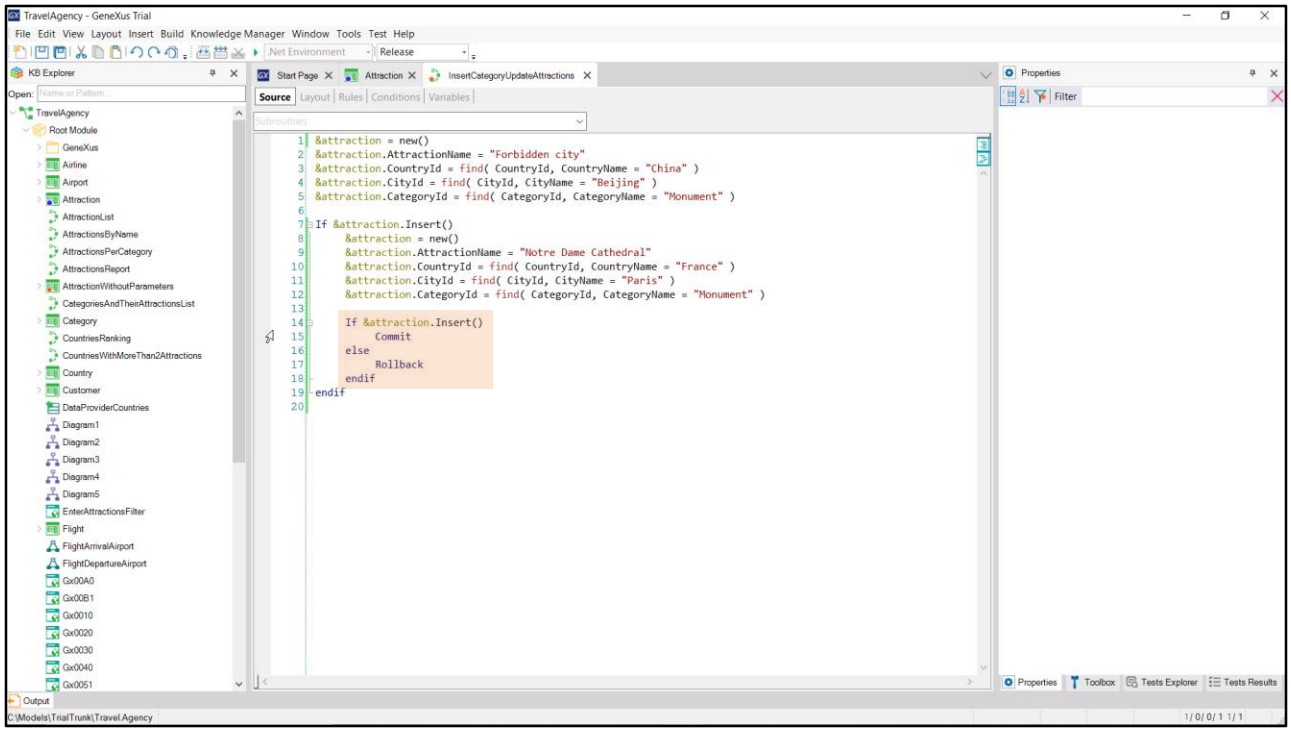


Devemos inseri-la então, aqui. Já não necessitamos da informação que a variável tinha. Na verdade, precisamos pelo menos limpá-la, para ter certeza de que não haverá nada velho quando carregarmos os dados da catedral. Uma maneira, é pedir memória nova para a variável, e isto é conseguido assim...

Agora sim. Depois, pedimos outra vez que insira.

Pedir memória nova é importante, dado que, por exemplo, se tivéssemos omitido atribuir valor ao identificador de categoria, porque a queremos deixar vazia para a Catedral de Notre Dame, teria ficado carregado com o valor que atribuímos mais acima. Portanto, sempre é boa prática antes de preencher os dados de um business component que vamos inserir, pedir memória para ele. Inclusive aqui em cima.

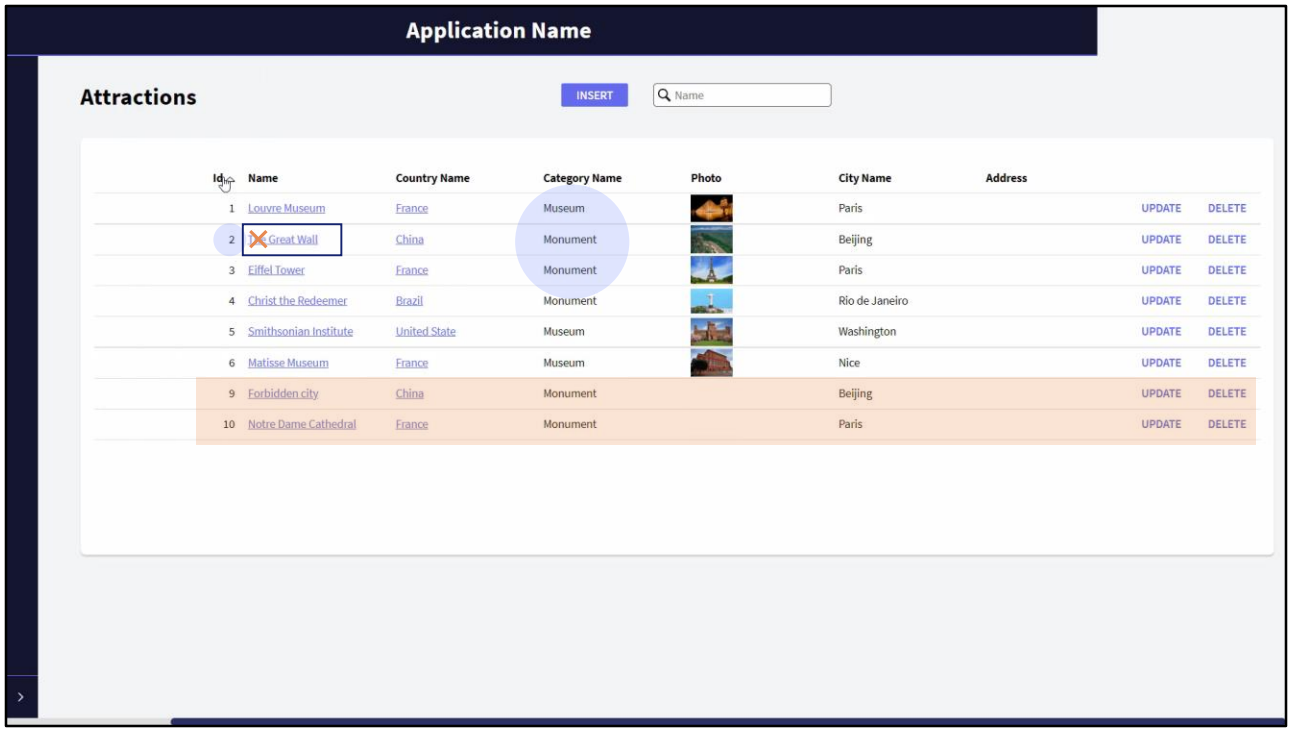
Por último, antes de executar, vemos que neste caso inserimos a primeira atração e commitamos se essa inserção foi bem-sucedida. E fazemos o mesmo com a segunda, e commitamos.



Mas poderíamos querer commitar apenas uma vez, depois de ter inserido as duas atrações. Ou, podemos inclusive tentar inserir a segunda atração, apenas se a primeira foi bem-sucedida. Por exemplo, nesse caso, seria assim.

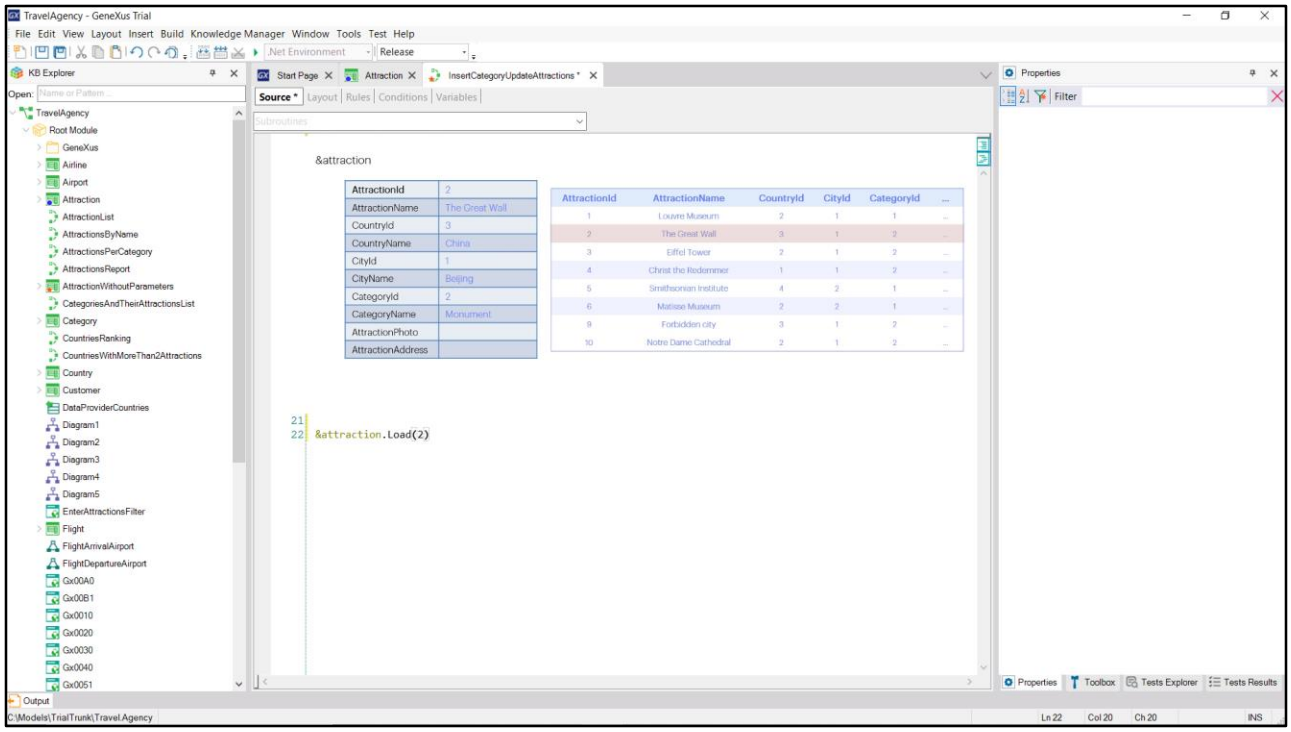
Aqui somente commitamos, quando a primeira inserção foi bem-sucedida e a segunda também. Por isso, poderia acontecer que a primeira tenha sido bem-sucedida e a segunda não. Então ficará gravado o registro da primeira na base de dados, mas não ficará commitado. Portanto, talvez nesse caso, quando a segunda não é bem-sucedida, não queremos que o primeiro registro fique na base de dados; e esse é um caso onde colocaríamos o comando Rollback.

Executemos para testar. F5.



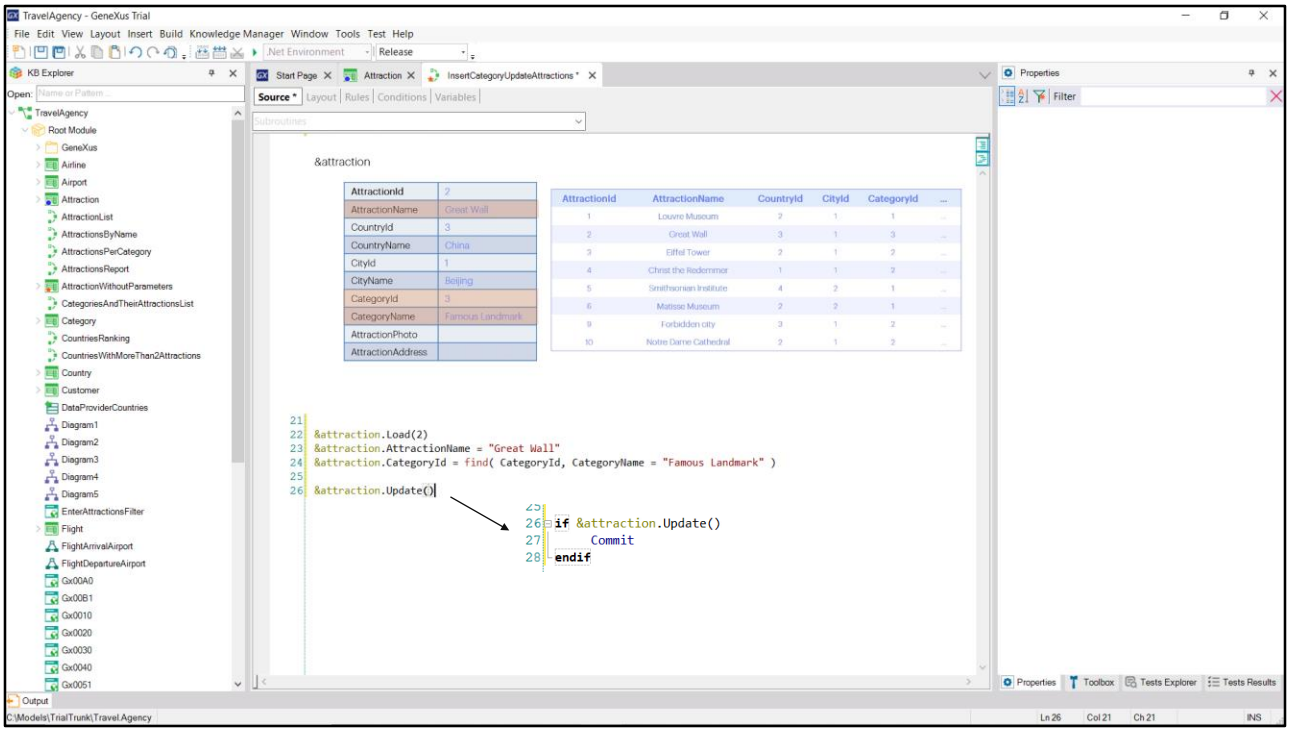
Vejamos as atrações. Executemos o procedimento. E voltemos a ver o work with de atrações. Agora temos a 9 e a 10.

E se, agora, quiséssemos mudar a categoria da Muralha da China para que não seja mais “Monument” e passe a ser “Famous Landmark”, e por exemplo, quiséssemos também mudar o nome, para tirar este “The”?



Deixemos o código anterior comentado para que não volte a inserir novamente estas duas atrações (se seus identificadores não fossem autonumerados, então as inserções falhariam por chave duplicada, mas não é o caso)

Precisamos fazer um Update como faríamos com a transação. Primeiramente, devemos carregar a variável com os valores da atração que queremos modificar. Sabemos que seu id é o 2. Então, o que fazemos é invocar o método Load da variável, passando por parâmetro a chave, 2. Ao executar isto, irá para a tabela para procurar se existe um registro 2, e em caso afirmativo, será carregada a variável BC com todos os valores: os próprios, inferidos e fórmulas, tal como acontece ao executar a transação.

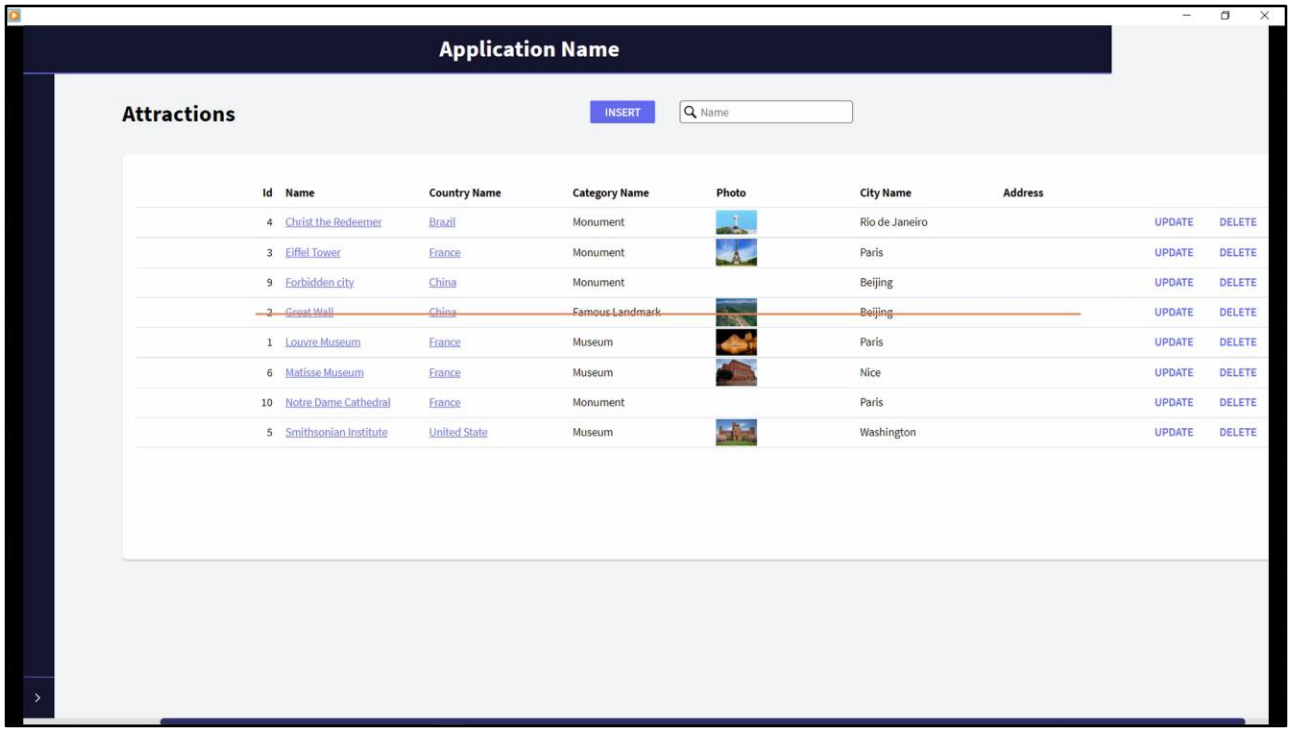


De todos estes valores, precisamos alterar dois: o nome da atração... E sua categoria...

Resta-nos apenas pedir à variável que atualize esses dados na tabela. E para isso, contamos com o método Update.
Ao invocá-lo, exatamente como acontece na transação, serão executadas todas as regras, fórmulas, controles de unicidade de chaves candidatas e integridade referencial; e se tudo correr bem, se atualizará o registro na tabela. E, claro, a variável ficará carregada com os dados atuais.
Mais uma vez, poderíamos perguntar sobre o resultado da atualização para, por exemplo, já realizar um Commit.

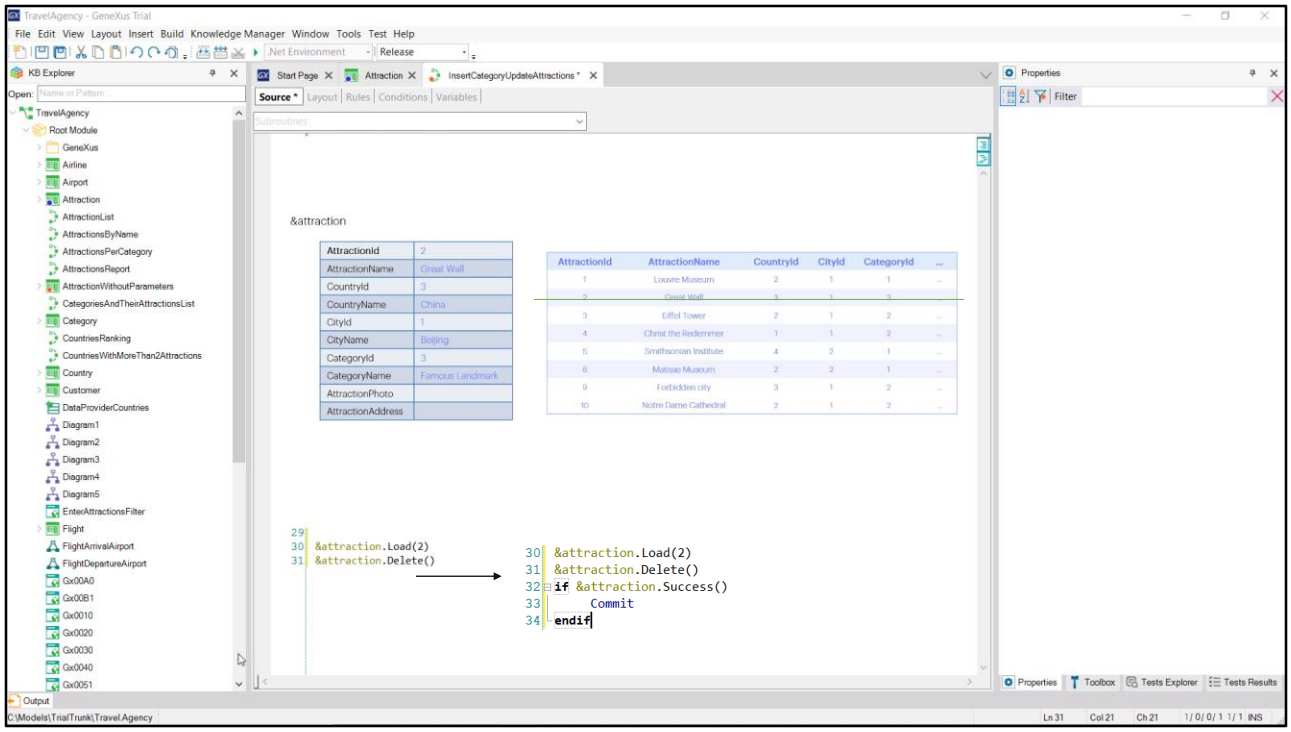
Executemos para testar.

Executamos o procedimento, e vamos ver as atrações. Vemos que ficou modificada.



Para completar as operações, e se agora quiséssemos eliminar uma atração por código?

Por exemplo, eliminar a Grande Muralha.



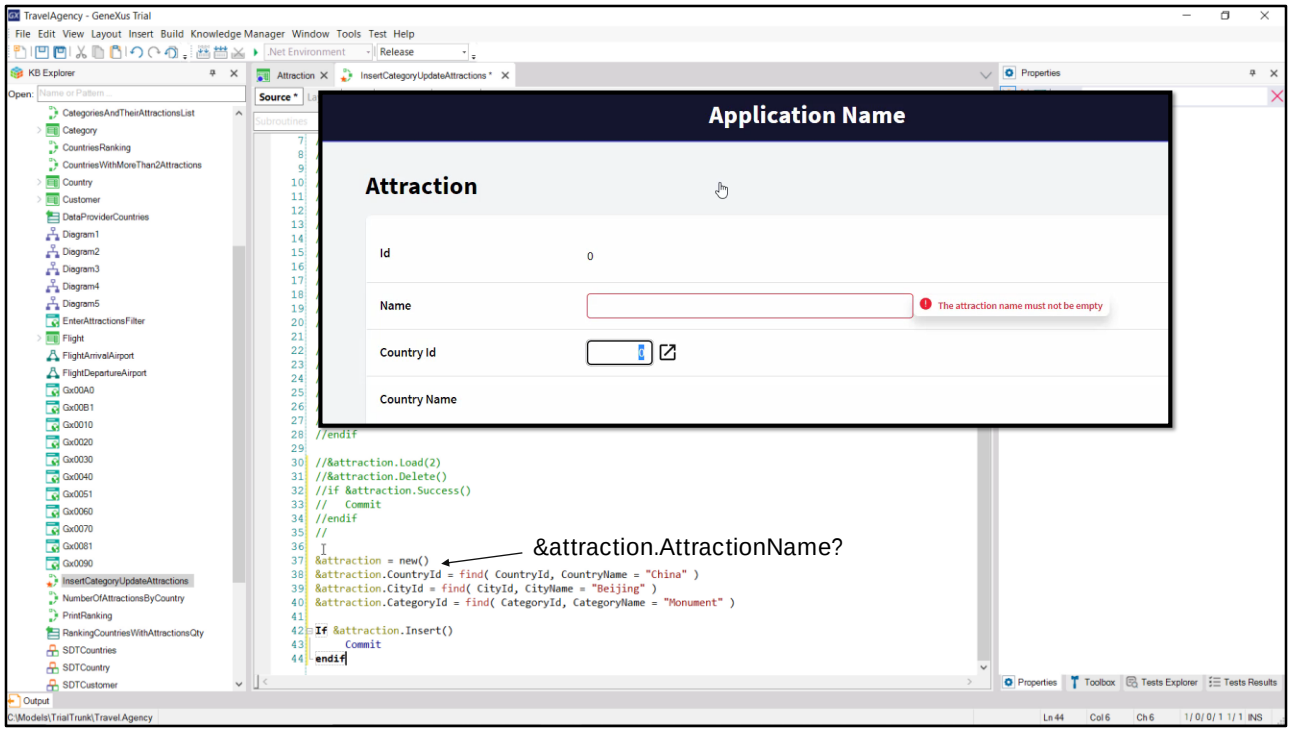
Assim como fazemos na transação, primeiro temos que carregar a estrutura da variável com a atração 2 e então, simplesmente dar a ordem para remover.

Isto fazemos com o método Delete. Não nos cansamos de repetir: ao executar este método será executada toda a lógica da transação em modo Delete, incluindo os controles de integridade referencial. Portanto, por exemplo, se houvesse informação relacionada, imaginemos que houvesse city tours que contivessem esta atração que desejamos eliminar entre seus dados, então, o controle de integridade referencial que realiza a transação –e portanto também o Business Component–, o impedirá. Será gerado um erro e o registro não será removido da tabela.
Em vez disso, se não houver nenhum erro, o registro será removido. A variável, de qualquer maneira, ficará carregada com os dados, se quisermos fazer algo com eles.

O método Delete não retorna o resultado de sua operação, por isso, para saber se foi ou não bem-sucedida, deveremos consultar de forma explícita, com o método Success (seu oposto é o método Fail). E, por exemplo, ali realizar o Commit da eliminação, ou seja, deixar permanentemente removido o registro. Este método pode ser utilizado depois de qualquer operação, por exemplo, depois do Load, para saber se

encontrou na tabela o registro pedido.

Testemos o que foi visto. Executamos o procedimento e vemos que efetivamente eliminou o registro 2.



E o que acontecerá se agora voltarmos a executar o procedimento? Nada aconteceu. Por quê?

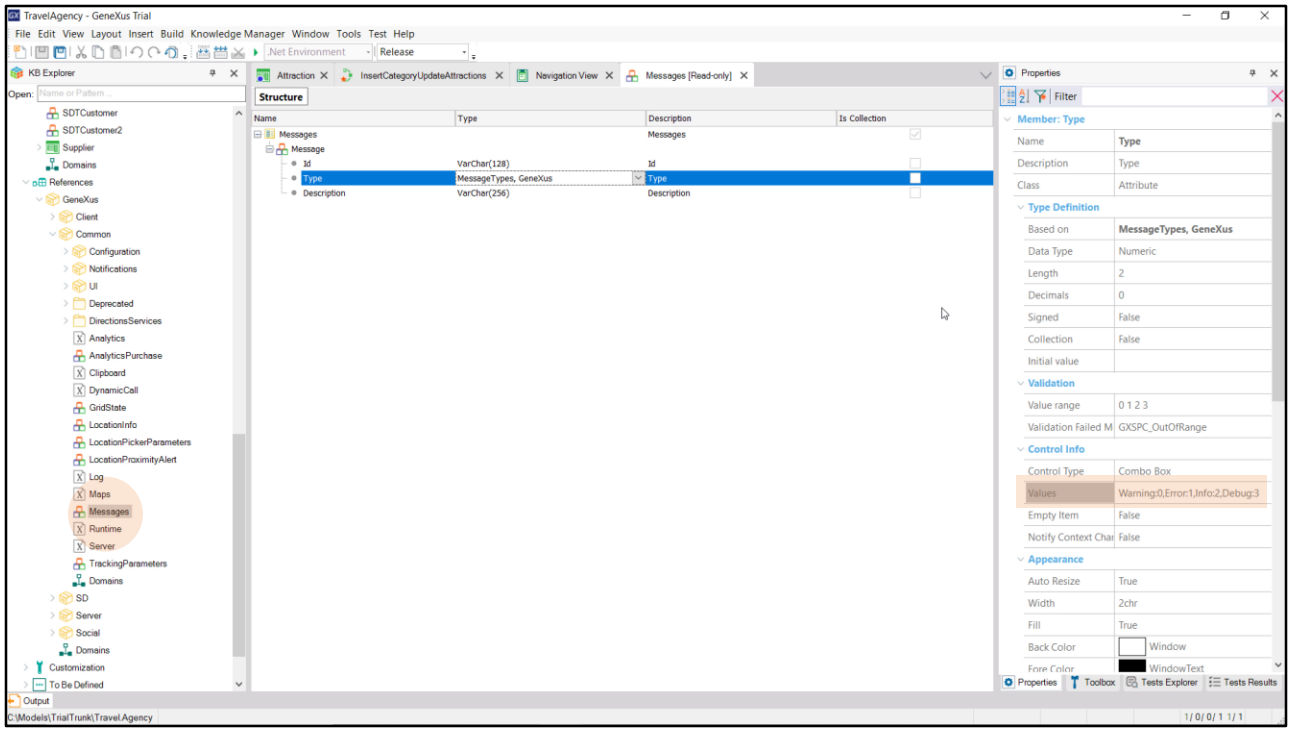
Nesta segunda execução a atração 2 já não existe. Portanto, não pôde nem carregá-la, nem eliminá-la.

E se agora tentarmos inserir novamente a Muralha da China, mas esquecermos de colocar o nome? Vamos testar.

Executo o procedimento. Vamos ao Work with de Attraction, e não vemos a nova atração inserida. Claro, esquecemos de colocar o nome e tínhamos uma regra de erro que está sendo disparada e está impedindo que se insira esse registro na tabela!

Se tivéssemos feito isso através da transação, ao usuário teria aparecido a mensagem que especificamos na regra de erro.

Onde ficou essa mensagem ao tentar a inserção através do business component?



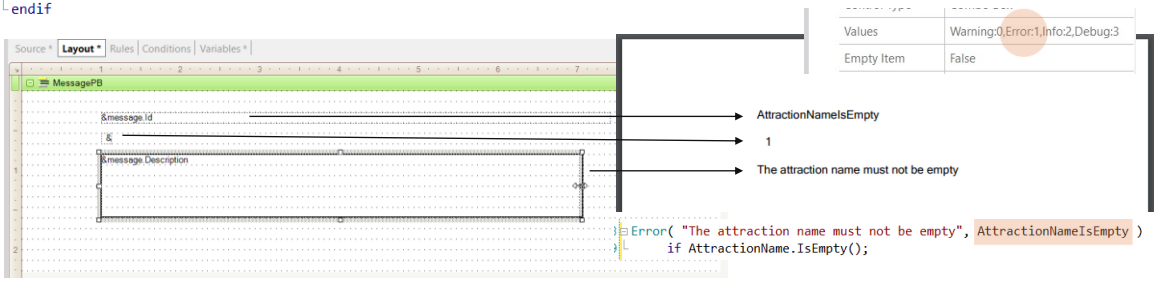
A transação tem uma tela para mostrar os erros ao usuário, mas o Business component não tem. Portanto, a manipulação de erros também fica a cargo do desenvolvedor.

Como os obtemos?

No KB Explorer contamos com um grupo References, do qual falaremos em outra oportunidade, mas que sempre conterá um módulo Genexus, com funcionalidades já implementadas para poder utilizar em nossa KB. Em particular, lá encontraremos um SDT de nome Messages. Podemos ver que é uma coleção de itens formados por um Id, um tipo e uma descrição. O tipo também vem predefinido no módulo. Trata-se de um enumerado que indica o tipo de mensagem de que se tratará em cada caso: advertência, erro, informação, debug.

Error handling

```
&attraction = new()
&attraction.CountryId = find( CountryId, CountryName = "China" )
&attraction.CityId = find( CityId, CityName = "Beijing" )
&attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )

If &attraction.Insert()
    Commit
else
    &messages = &attraction.GetMessages()
    for &message in &messages
        print MessagePB
    endfor
endif
```



Cada vez que se executa uma operação sobre uma variável Business Component, pode-se obter a coleção de mensagens que essa execução tenha produzido, com o método GetMessages. Teremos que definir uma variável do tipo de dados devolvidos por esse método, para poder manipular seu resultado.

Por exemplo, agora, só por praticidade, vamos mostrar o resultado em um pdf. Percorreremos a coleção de mensagens geradas com o comando for in que permite percorrer, entre outras coisas, coleções. Definimos para isso, uma variável do tipo de dados dos itens da coleção de mensagens. E para cada mensagem dessa coleção, o imprimimos na saída.

Vamos especificar a regra outputfile e testemos.

Executamos o procedimento. E aqui vemos sua saída...

Somente um item foi obtido na coleção de mensagens, com id vazio, tipo 1 –que é erro–, e como descrição, exatamente o que especificamos na regra de erro da transação.

Se vamos ver a sintaxe da regra de erro, vemos que permite um segundo parâmetro opcional. Esse parâmetro permitirá especificar o Id do erro

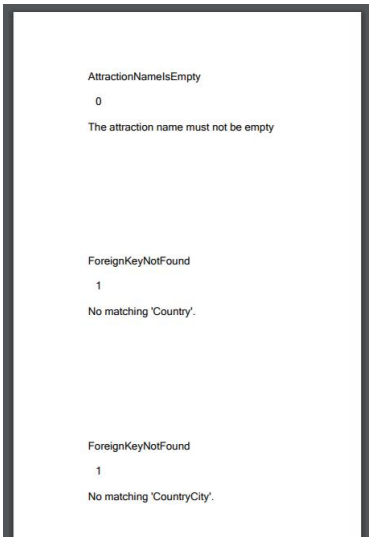
para a mensagem do business component. Por exemplo, lhe daremos este valor. Se agora testamos... ali o vemos.

GeneXus

Error handling

```
Msg( "The attraction name must not be empty", AttractionNameIsEmpty )
    if AttractionName.IsEmpty();
```

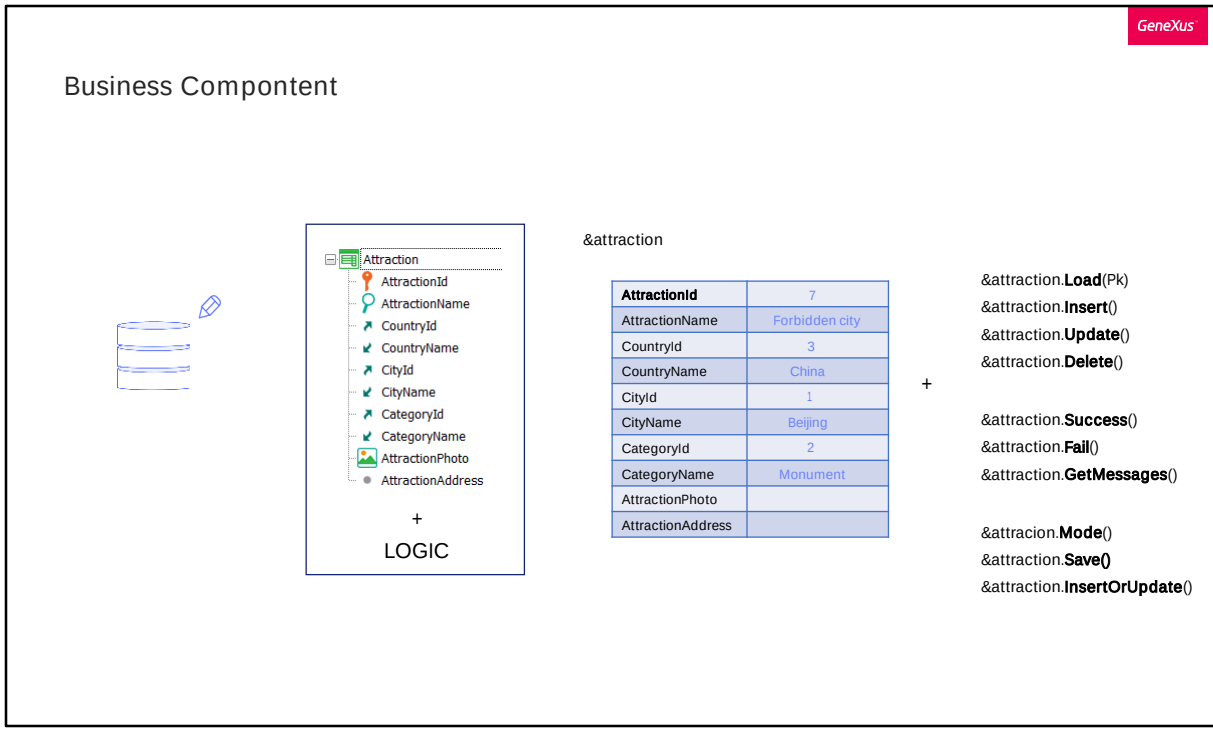
```
&attraction = new()
&attraction.CountryId = 22
//&attraction.CountryId = find( CountryId, CountryName = "China" )
&attraction.CityId = find( CityId, CityName = "Beijing" )
&attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )

If &attraction.Insert()
    Commit
else
    &messages = &attraction.GetMessages()
    for &message in &messages
        print MessagePB
    endfor
endif
```



Façamos um último teste: modifiquemos a regra, para que já não seja de erro, mas uma mensagem.
E também especifiquemos como CountryId um identificador de país inexistente, para ver como falha a inserção por controle de integridade referencial. Vamos testar.

Ao tentar inserir se produziram três mensagens: a primeira é do tipo Warning, e não teria provocado por si só a falha da inserção. Em vez disso a segunda e a terceira, sim, pois são do tipo Erro. O Id interno é este, e a mensagem que o usuário da transação vê na tela quando falha a integridade por país, é, NÃO CASUALMENTE, No matching 'Country'.

Também vai lançar um erro de integridade referencial ao tentar inserir a cidade, que depende do país.



Com isto, podemos dizer que vimos o mais relevante relacionado à atualização da base de dados com Business Components.

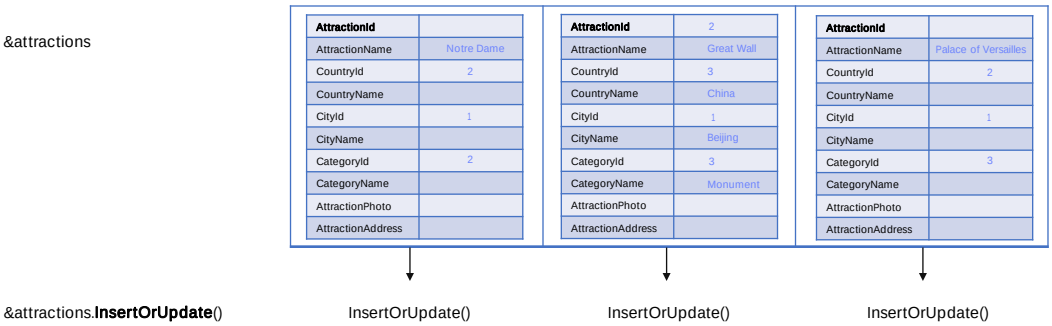
Assim, vimos que a partir da transação é possível criar um tipo de dados que é como um SDT, mas que permite realizar operações na base de dados a partir de métodos.
Estas execuções conservam a lógica da transação. Se bem que não entramos nisso, claramente nem todas as regras e eventos da transação serão incorporadas ao business component. As que têm a ver com a interface, obviamente não. A regra Parm também não é levada em conta.
Além de nos permitir estas operações sobre a base de dados, podemos consultar o resultado da última operação realizada, assim como obter as mensagens produzidas, em uma coleção.

Há outros métodos para estudar. Por exemplo, existe o método Mode que devolve o modo no qual se encontra o business component: se é Insert, Update, Delete.
Um método Save que, dependendo do modo da variável tentará Inserir se o modo é Insert ou atualizar se o modo é Update.
E o método InsertOrUpdate que sempre tentará inserir, e se falhar por chave duplicada, então tentará atualizar.

Os métodos Insert, Update, Delete e InsertOrUpdate podem ser aplicados também a coleções de Business Components.

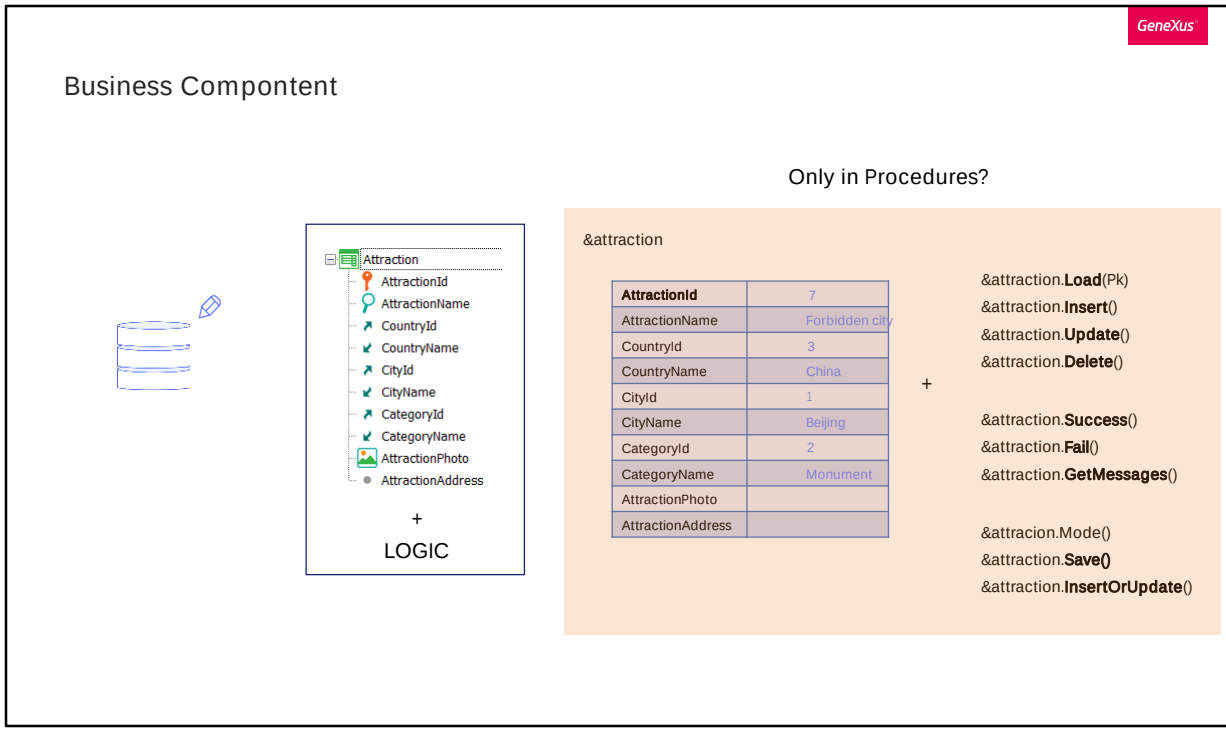
GeneXus

Business Compontent



Assim, se a variável &attractions fosse uma coleção do Business Component Attraction, –neste exemplo uma coleção de três itens Attraction–, aplicar-lhe, por exemplo, o método InsertOrUpdate, é equivalente a percorrer a coleção e ir lhe aplicando o método individualmente a cada item.

O resultado será True se os resultados individuais foram todos True.



A última coisa importante que mencionaremos, é que uma variável de tipo Business Component pode ser utilizada em qualquer objeto que tenha alguma seção de código, não apenas procedimentos. Isto significa que podemos atualizar a base de dados, por exemplo, a partir do evento de um painel que pede ou mostra dados aos usuários, como veremos.

Também poderá fazer a partir de eventos em transações, embora, nesse caso, existem algumas limitações que não veremos aqui.

GeneXus

More

- InsertCategoryUpdateAttractions proc

Insert "Tourist site" category
Update Attractions of Beijing

- CategoriesAndAttractions web panel

Do:
Insert "Tourist site" category
Update Attractions of Beijing

Undo:
Delete "Tourist site" category
Update Attractions of Beijing

- MassiveInsertRemove panel

Remove Data:
From Category
From Attraction

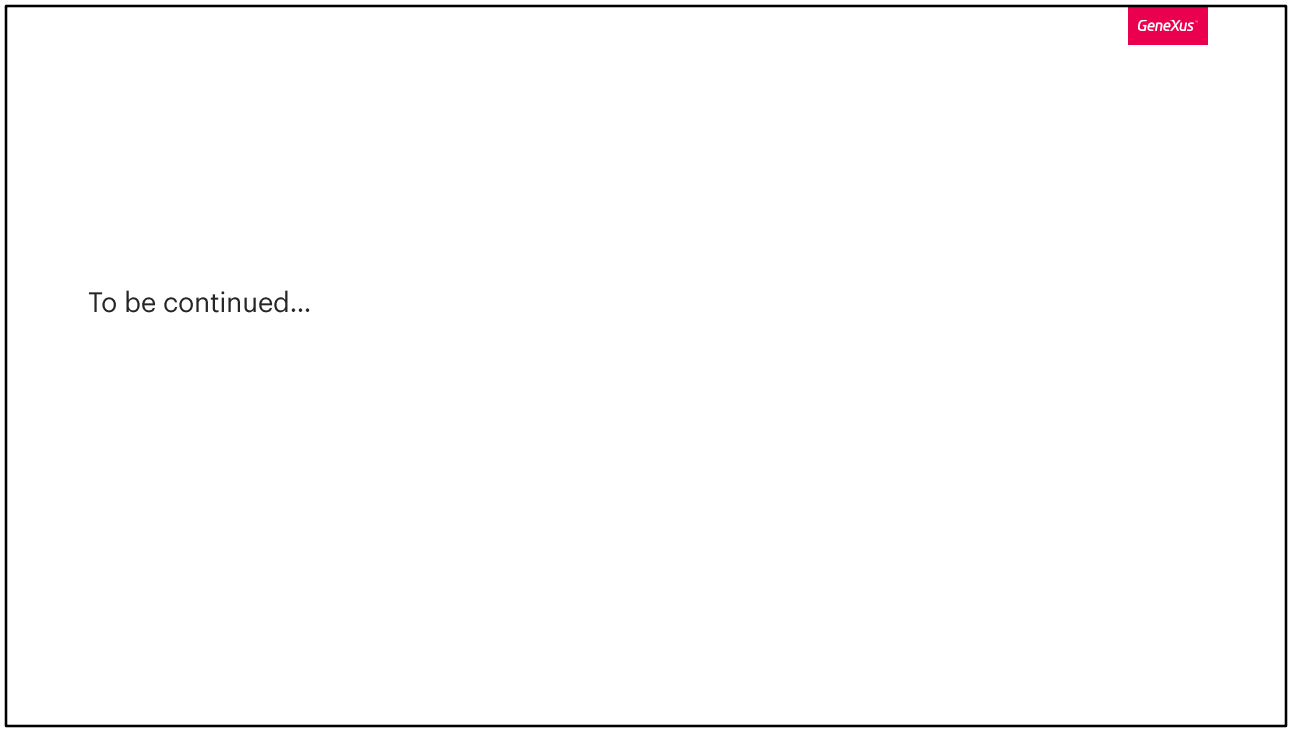
No próximo vídeo, aplicaremos em um exemplo, tudo o que foi visto aqui. Poderia pulá-lo, exceto a parte final.

Lá:

Implementaremos novamente o procedimento que aqui criamos, para que insira uma nova categoria de atração turística: "Tourist Site" e modificaremos todas as atrações de Beijing que tinham a categoria "monument", passando a atribuir-lhe a nova.

Veremos como fazer isto mesmo através de um web panel interativo, que ofereça fazer o anterior, mas também desfazer o feito, deixando tudo como estava.

E, por último, criaremos outro web panel que nos permita remover as informações de ambas as tabelas, Category e Attraction. Isto será retomado depois, por isso, recomendamos que veja pelo menos esta parte final.



Claro que há muito mais para ver, mas deixamos para o próximo nível. Se estiver interessado, pode buscar os vídeos relacionados a este tema no curso de Analista GeneXus.

GeneXus™

training.genexus.com
wiki.genexus.com