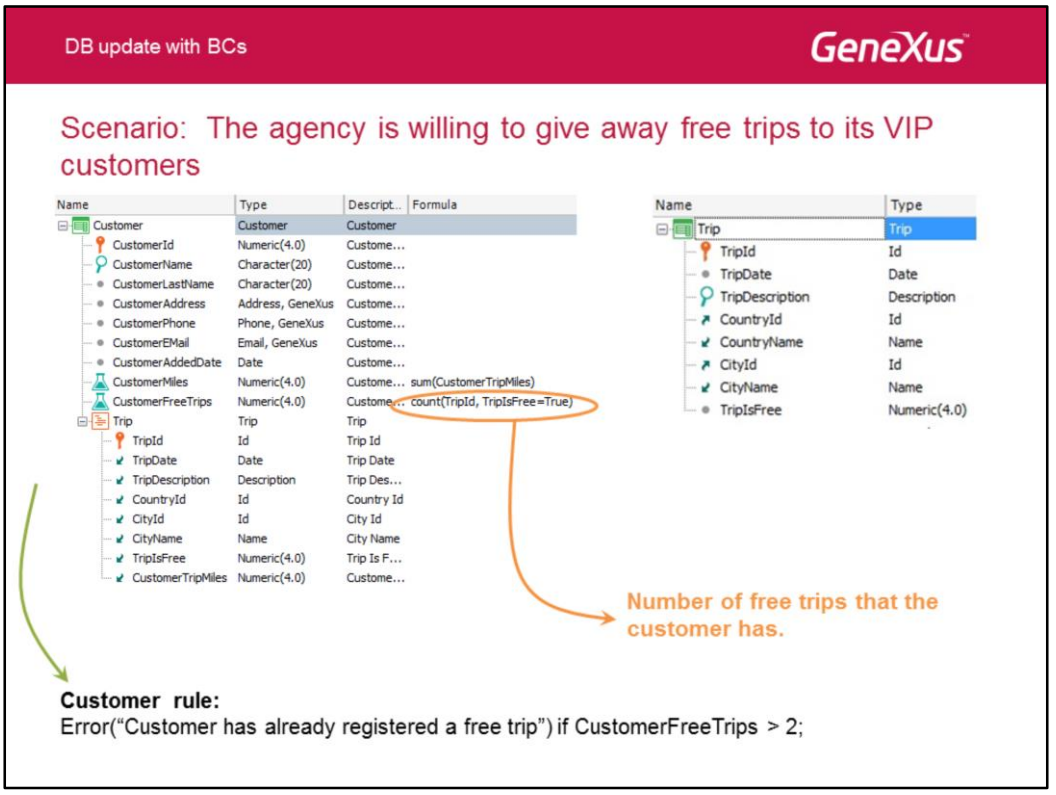
Já abordamos o conceito geral de Business Component e como se aplica a uma transação de nível único.

Agora vamos ver o que acontece quando criamos o Business Component para uma transação de dois níveis.



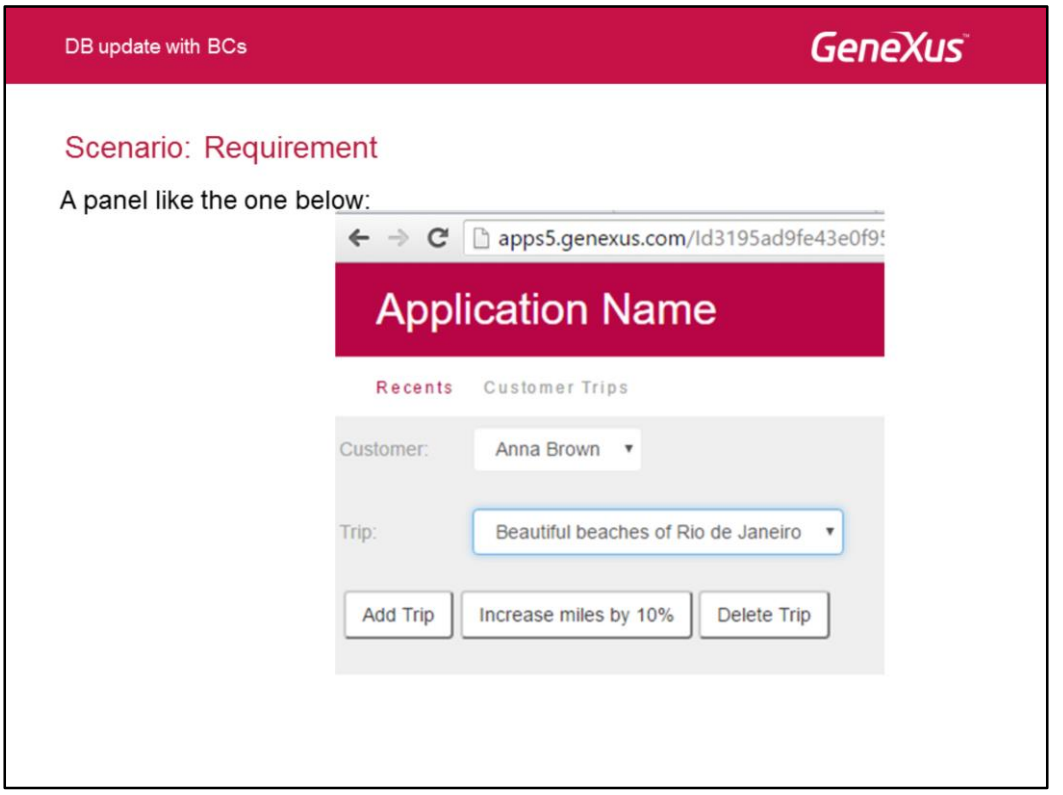
A Agência oferece a possibilidade de viagens a diferentes cidades oferecendo visitas a atrações turísticas. De vez em quando, a agência dá uma viagem gratuita aos seus clientes VIP.

Para cada cliente, o sistema deve gravar as viagens que o cliente tenha comprado, além disso, para as viagens gratuitas fornecidas como prêmio da agência. A agência dá até duas viagens por cliente.

Com finalidade de gravar viagens da Agência, nós usaremos uma transação Trip para inserir informações por meio do atributo Booleano TripIsFree para quando a viagem é grátis, ou não. Para gravar viagens do cliente, nós adicionamos um segundo nível para a transação Customer com TripId como identificador, os atributos inferidos na transação Trip e CustomerTripMiles como único atributo secundário, para gravar as milhas acumuladas pelo cliente nessa viagem.

Para controlar que um máximo de duas viagens grátis são concedidos a cada cliente, nós declaramos o atributo fórmula CustomerFreeTrips e a seguinte regra:

Error (“Cliente já tem registrado uma viagem gratuita") if CustomerFreeTrips > 2;



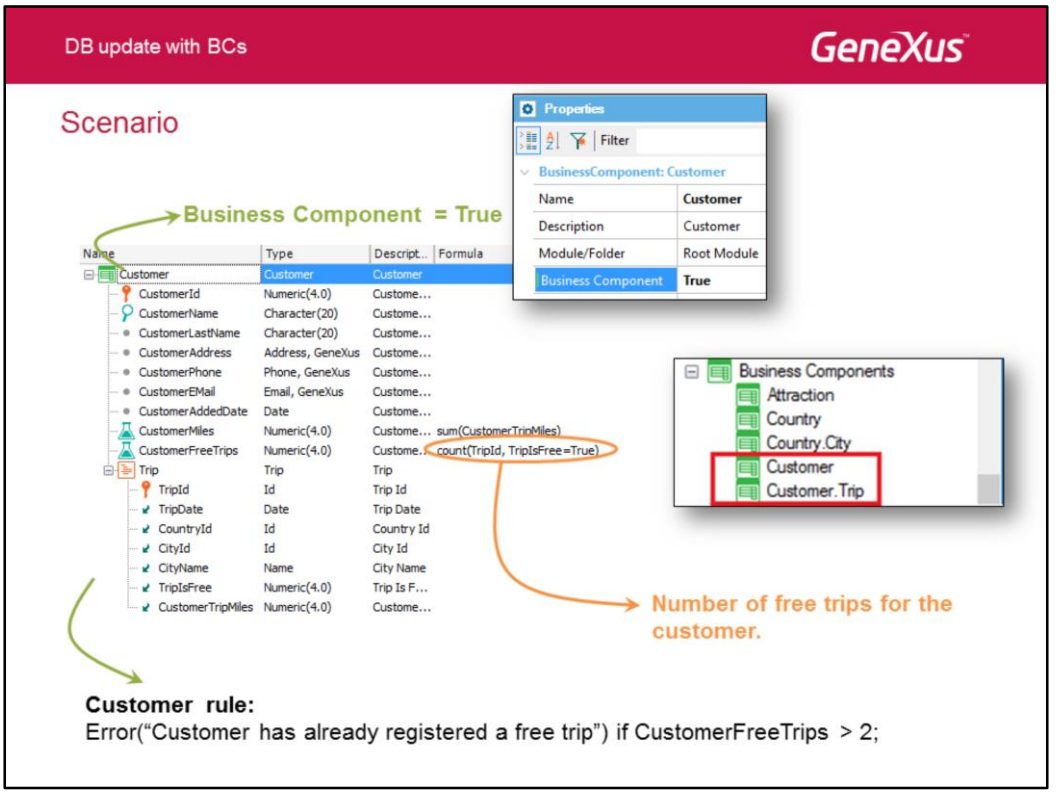
O usuário receberá um painel como aquele acima permitindo a seleção de um cliente através de um combo dinâmico, e a seleção de uma viagem através de um outro combo dinâmico.

Depois de selecionar o cliente e a viagem, teremos três operações possíveis:

- Adicionar viagem: para tentar adicionar a viagem selecionada para o cliente específico. Se a viagem é gratuita e 2 viagens grátis já tiverem sido atribuídas a esse cliente, então a inserção não pode ser permitida e deve haver uma mensagem de tela informando-nos sobre esse erro.
- Aumentar milhas em 10%: para tentar aumentar a milhagem em 10% quando a viagem selecionada for gravada para o cliente. Caso contrário, uma mensagem de tela vai nos dizer que a operação não poderá ser feita.
- Excluir viagem: tentar apagar a viagem selecionada a partir do registro do cliente. Quando a operação não pode ser executada (como no caso onde a viagem não foi encontrada para o cliente), então, uma mensagem deve informar o usuário em relação a este problema.

A tela mostrada será implementada com um web panel, um assunto que iremos estudar adiante, incluindo detalhes de funcionamento que podemos não estar considerando agora.

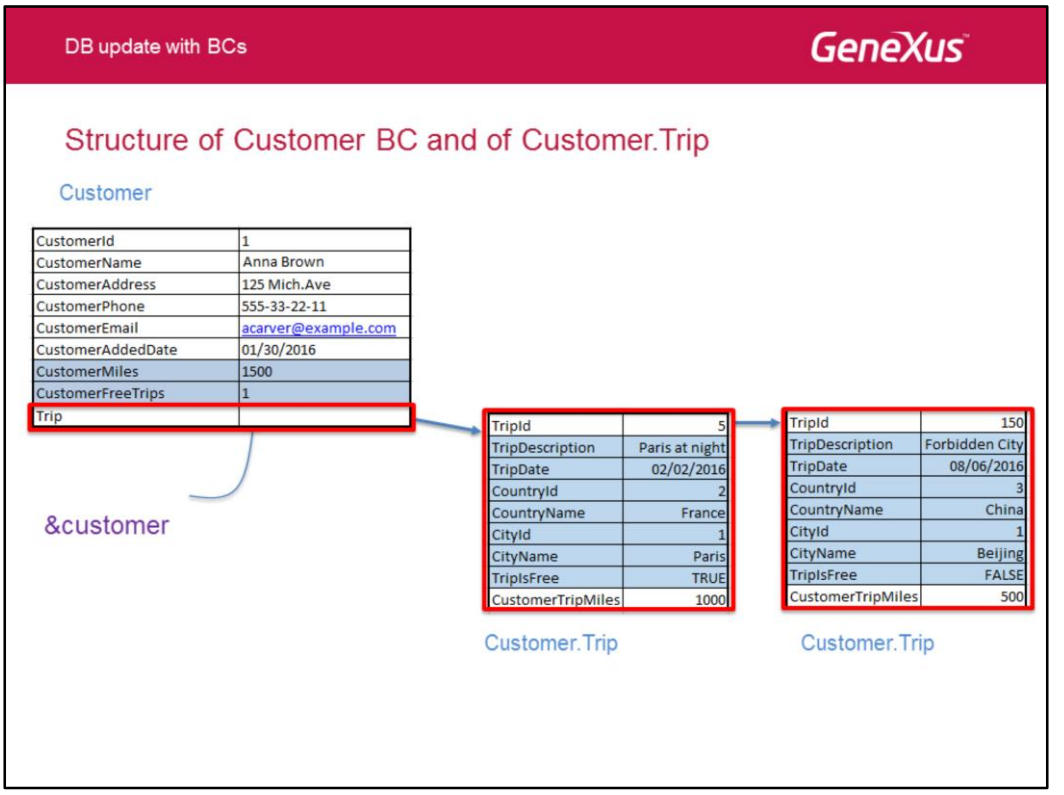
Neste momento, vamos nos concentrar em como programar o código correspondente a cada botão, ou seja, como realizar a inserção / modificação / exclusão aplicável a uma linha na transação Customer, todavia sem usar a transação.



Criamos o Business Component associado com a transação, declarando o valor da sua propriedade Business Component como True.

Sabemos que, após salvar, GeneXus vai criar o tipo de dados Customer associado com a transação. Neste caso, porque a transação tem dois níveis, ele também irá criar o tipo de dados de Customer.Trip especificamente associado com as linhas no segundo nível, que, no nosso exemplo, correspondem às viagens do cliente.

Vamos encontrá-lo na KB como qualquer tipo de dados business component. No entanto, a diferença com tipo de dados Business Component correspondente à transação (Customer neste caso) - o que nos permite executar operações como se fosse a transação-, um segundo apenas nos permite definir a estrutura de uma linha na transação, mas não está associado com nenhum método específico, tais como Insert, Update, Delete, Save, etc.



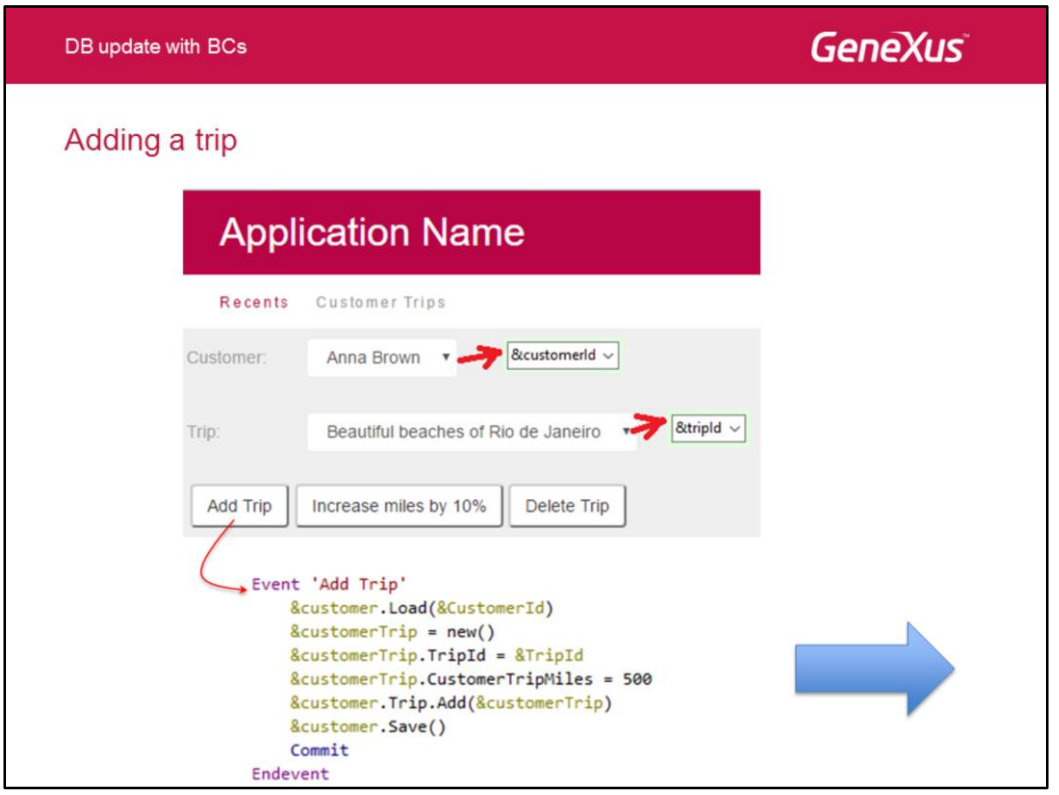
Aqui, podemos ver a estrutura do Business Component do Customer e do Customer.Trip. Se observarmos atentamente, veremos a variável &costumer , do tipo costumer, com um elemento para cada atributo no cabeçalho da transação e com um elemento para a coleção de linhas.
Assim, teremos o seguinte:

&customer.CustomerId baseado no atributo CustomerId, do tipo Id (numérico)
&customer.CustomerName baseado no atributo CustomerName, do tipo Name(caractere)
…
&customer.CustomerAddedDate baseado no atributo CustomerAddedDate, do tipo Data
&customer. CustomerMiles baseado no atributo CustomerMiles, do tipo Numérico
&customer. CustomerFreeTrips baseado no atributo CustomerFreeTrips, do tipo Numérico
&customer. Trip, do **tipo coleção**, de **Customer.Trip**

Observe que os atributos CustomerMiles e CustomerFreeTrips, que, na transação, são atributos fórmulas, são usados apenas para consultar informações sobre um cliente já carregado na variável &customer (como no exemplo mostrado acima).

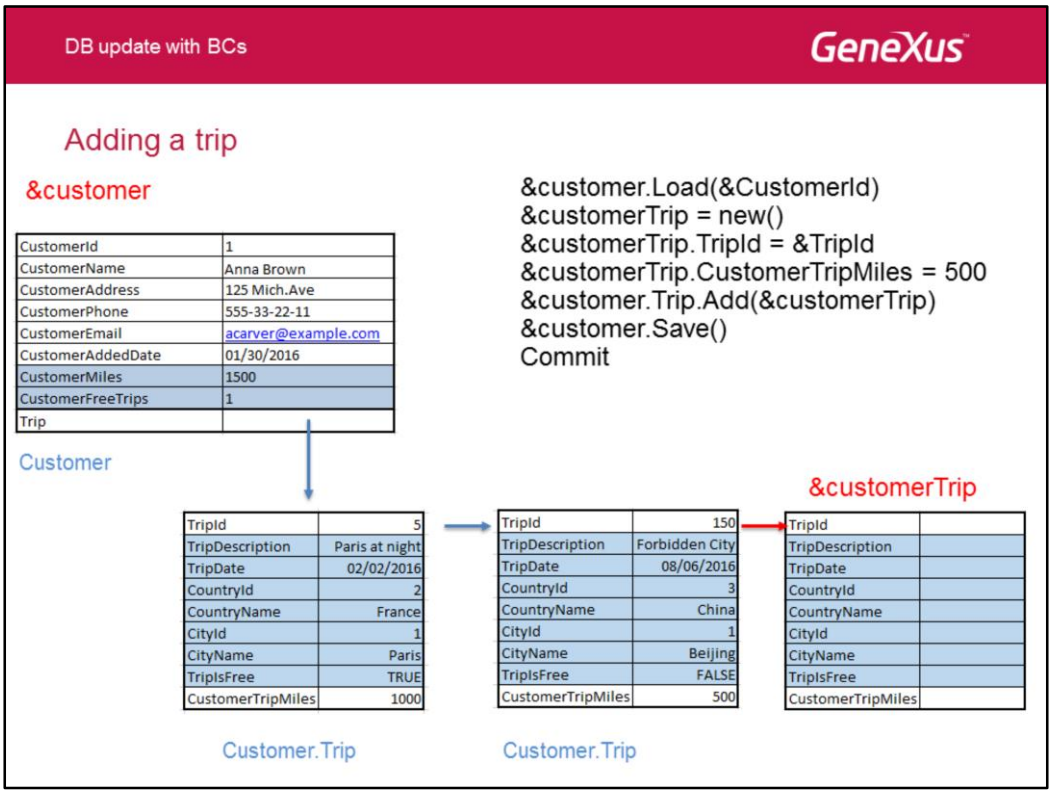
O tipo de dados Customer.Trip refere-se a uma estrutura com os elementos correspondentes aos atributos do nível Trip da transação. Novamente, os atributos inferidos naquele segundo nível da transação serão apenas atributos de consulta.

Como veremos adiante, a fim de inserir uma nova "linha", ou para modificar uma linha existente, teremos que usar uma variável do tipo de dados Customer.Trip.



Deixando de lado detalhes sobre o web panel, o combo perguntando o cliente ao usuário será uma variável &CustomerId baseada no tipo de dados do atributo CustomerId (que mostrará os dados do atributo CustomerFullName). E o que está pedindo as viagens será outra variável &TripId, baseada no tipo de dados do atributo TripId (que mostrará, em tempo de execução, os dados do atributo de TripDescription).

Vamos começar pela ação associada ao botão "Add Trip". Este será o código que iremos explicar agora.



Se trabalhamos com a transação, então, a primeira coisa a fazer seria encontrar o cliente com o qual vamos trabalhar. Para isso, vamos **carregar** (com o método Load) a variável customer(do tipo Customer Business Component) com o valor selecionado pelo usuário na variável &CustomerId na tela no web panel. No exemplo, o cliente de Id 1.

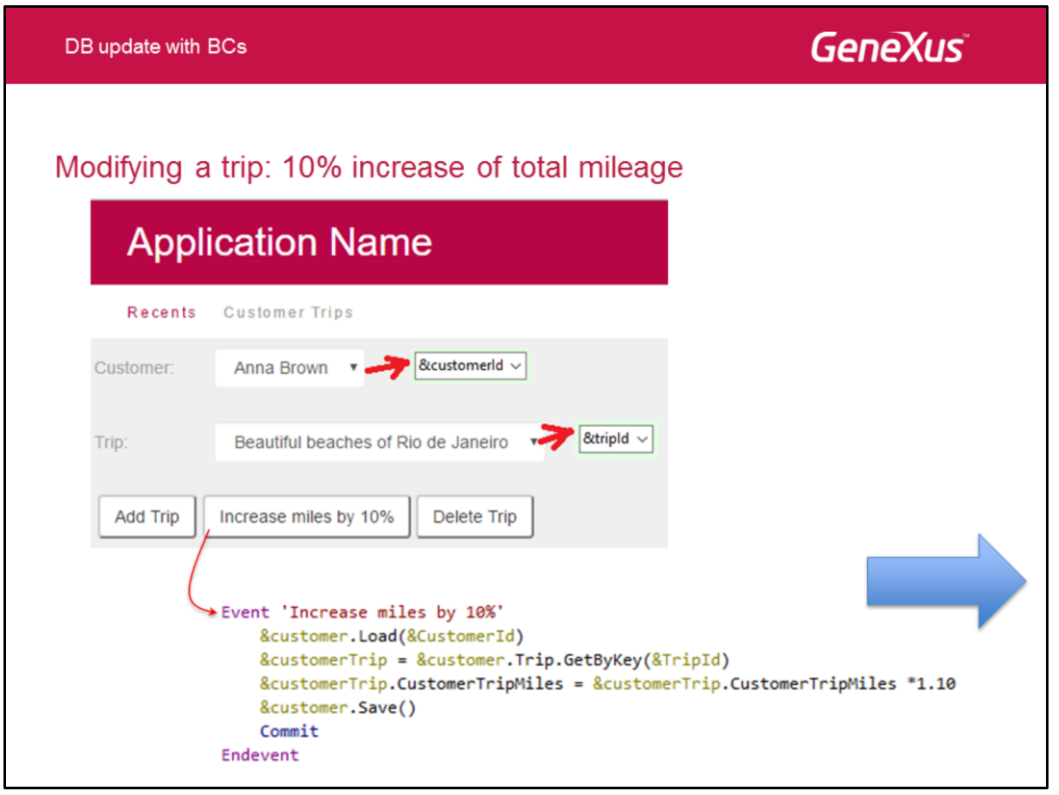
Se nós trabalhamos na transação Customer , para adicionar viagem temos de adicionar uma linha. Isso é feito com o método New que nos permite criar a variável &customerTrip com o tipo de dados dos itens na linha, que é: Customer.Trip. Nós a usamos para inserir a nova viagem para esse cliente. Preenchemos os valores de TripId com a viagem selecionada pelo usuário no painel, e salvo na variável &TripId, e podemos atribuir o valor de milhas para essa viagem -500 neste caso.

Ao contrário de quando trabalhamos com a transação onde temos as linhas disponíveis na tela, neste caso, temos de adicionar explicitamente esse item na coleção de linhas do cliente. Para fazer isso, usamos o método Add (de coleções). Devemos lembrar que & customer.Trip é uma coleção de itens do tipo Customer.Trip.

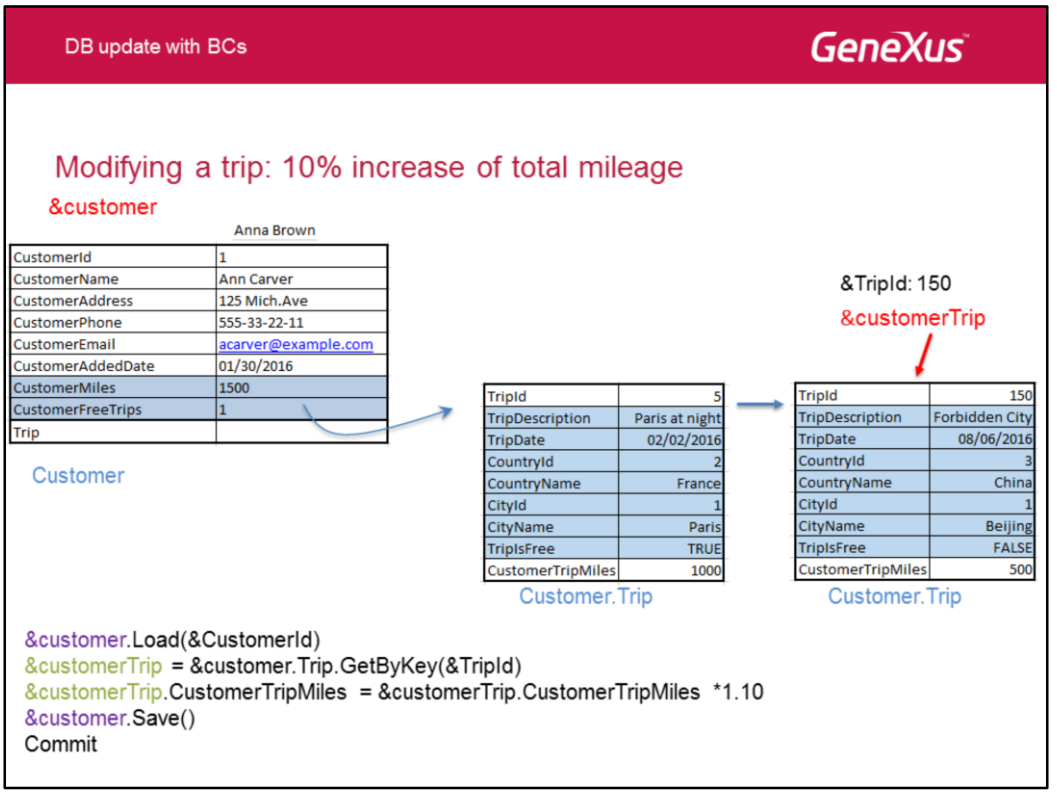
E por último, e como faríamos na transação, devemos confirmar a operação, para o que adicionamos o Save do Customer business component e confirmamos.

Nota: em vez do método Save, neste caso poderíamos ter usado o método Update , porque mesmo quando estamos inserindo uma nova linha, estamos na verdade fazendo uma atualização no nível do cliente. É por isso que o método Insert não seria o mais adequado, já que estes métodos estão sendo aplicados para a variável correspondente ao cabeçalho.

Se não for possível concluir a operação, temos que ver como gerenciar erros.



Agora vamos dar uma olhada na ação associada com o botão “Increase miles by 10%”. Aqui temos o código que será explicado na próxima página.

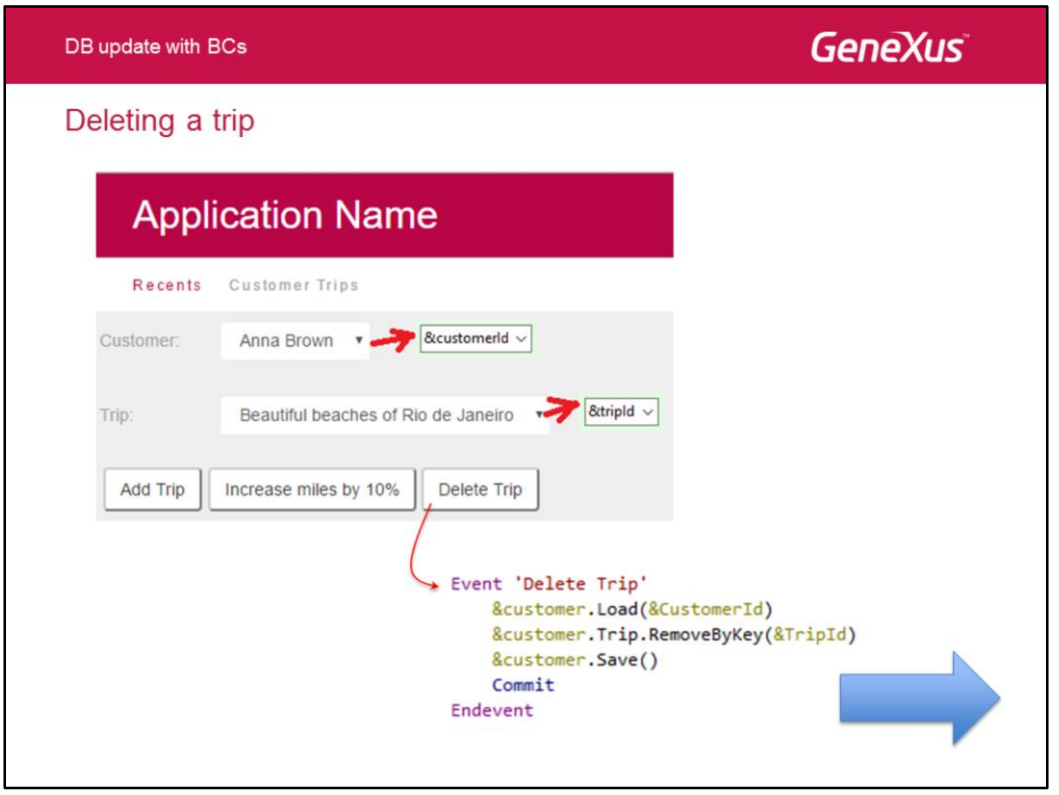


1. Como faríamos na transação, devemos carregar o cliente, a milhagem que desejamos atualizar para uma de suas viagens: Load.
2. Devemos acessar a viagem que queremos modificar usando um método da coleção de linhas chamado GetByKey, ao qual passamos o identificador de linha e o retorno é aquela linha na variável &customerTrip. Não é uma cópia da linha, mas a própria linha.
3. Então modificamos o valor desejado – milhas de viagem neste caso.
4. Salvamos, e
5. Confirmamos

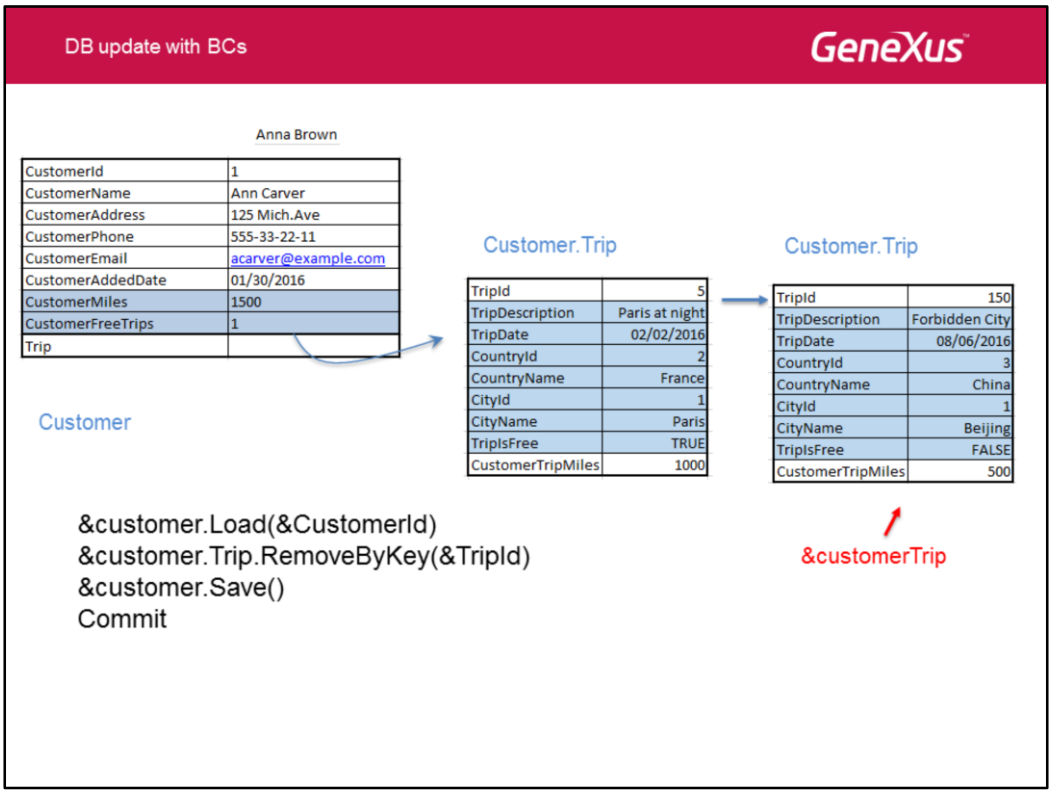
Nota: se tivéssemos qualquer outro atributo secundário, ao qual nós não atribuímos explicitamente um valor, então, ele manterá o valor que tinha anteriormente. Isto significa que os valores são atribuídos apenas aos elementos que desejamos modificar.

Os conceitos que consideramos antes em relação os métodos Insert e Update em vez de Save também se aplicam aqui.

Mais tarde veremos como os erros são gerenciados para o caso onde a operação não pode ser concluída

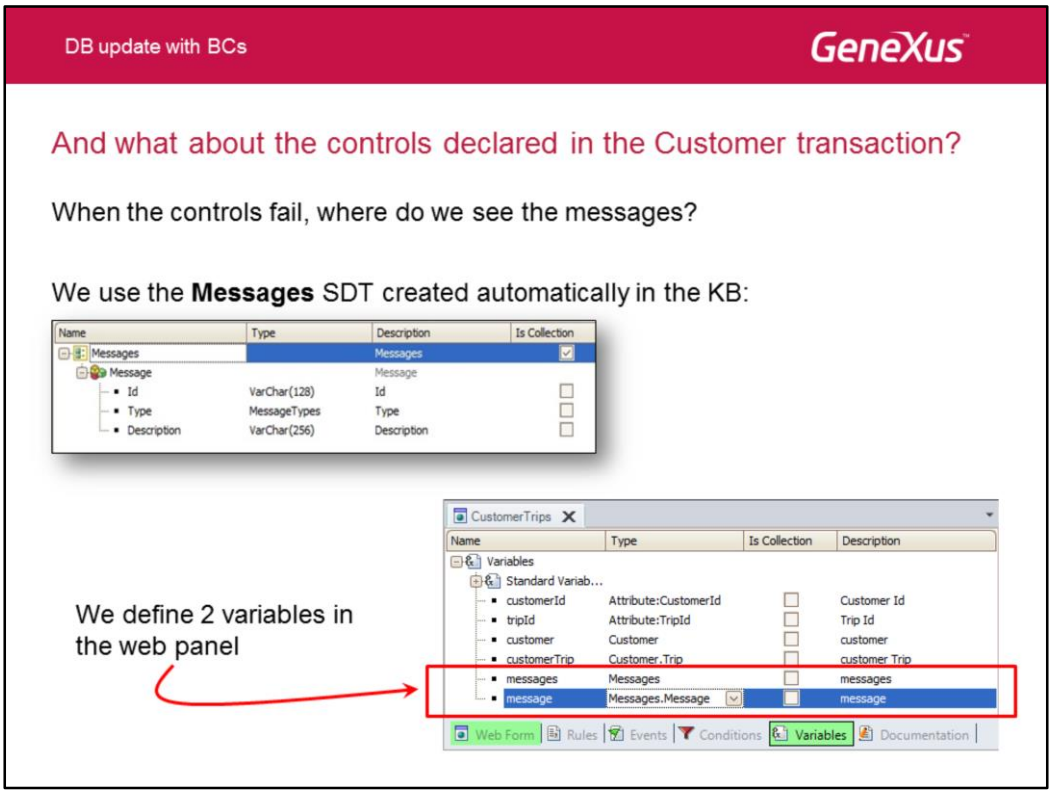


Vamos agora ver a ação associada com o botão “Delete Trip”. Aqui temos o código, que é analisado na próxima página.



Para excluir uma linha da transação posicionamos na linha e a marcamos para exclusão. No business component, isso é feito com o método RemoveByKey da coleção de linhas. Devemos passar para esse método, por parâmetro, o identificador da linha que deve ser excluída.
Para a linha ser finalmente eliminada na transação, Pressionamos Confirma. Aqui, executamos o método Save. Novamente, como já vimos, poderíamos substituir o Save com Update, embora não com Delete porque esses métodos se aplicam para o cabeçalho, que já existe.

Agora veremos como os erros são gerenciados para o caso onde a operação não pode ser concluída.



Na ação para a inclusão de uma viagem, o que aconteceria se a viagem que pretendemos inserir:

1) Já foi atribuído ao cliente?
2) É de graça e o cliente já tinha 2 viagens nessas condições?

Em ambos os casos a inserção falhará. In case 1) porque o BC irá verificar a unicidade da chave composta do registro que queremos inserir, e no caso 2) porque o BC irá disparar ao regra error que foi especificada na transação para controlar o número de viagens grátis.

Mesmo quando esse controle é acionado automaticamente quando aplicamos o método Save() para a variável no Business Component, se queremos que o usuário final esteja ciente disso, temos de obter e mostrar explicitamente as mensagens emitidas.

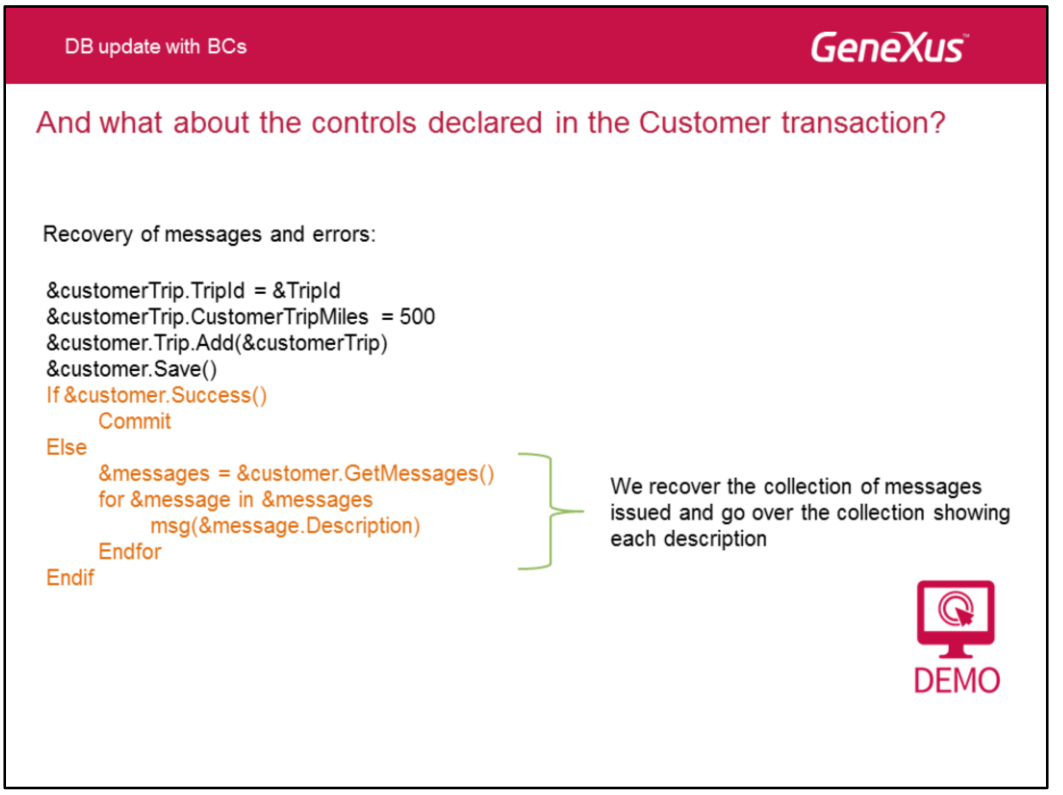
Que mensagens? Aquelas emitidas por GeneXus devido aos controles de consistência dos dados, unicidade em valores de chaves primárias, e as mensagens em relação às nossas próprias regras Error e Msg.

Então, como podemos recuperar as mensagens que ocorreram?

Nós usamos o tipo de dados estruturados Messages que GeneXus define automaticamente ao criar uma nova KB. A finalidade deste tipo de dados **collection** é permitir o nosso acesso às mensagens emitidas na execução de um Business Component.

Portanto, definimos duas variáveis:
&mensages (do tipo de dados Messages, e collection) e

&mensage (do tipo de dados Messages.Message, que representa um item na coleção).



Quando aplicamos o método GetMessages para a variável &Customer, obtemos a coleção de mensagens que ocorreram.

E por último, usando a estrutura "**For item in collection",** vamos percorrendo cada mensagem na variável coleção de mensages e publicamos sua descrição.

O mesmo código colorido poderia ser adicionado para os exemplos anteriores quando uma viagem é modificada ou excluída.

GeneXus

To learn more about using Business Components go to:

http://wiki.genexus.com/commwiki/servlet/wiki?5846,Toc%3ABusiness+Component,

GeneXus™

VIdeos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications