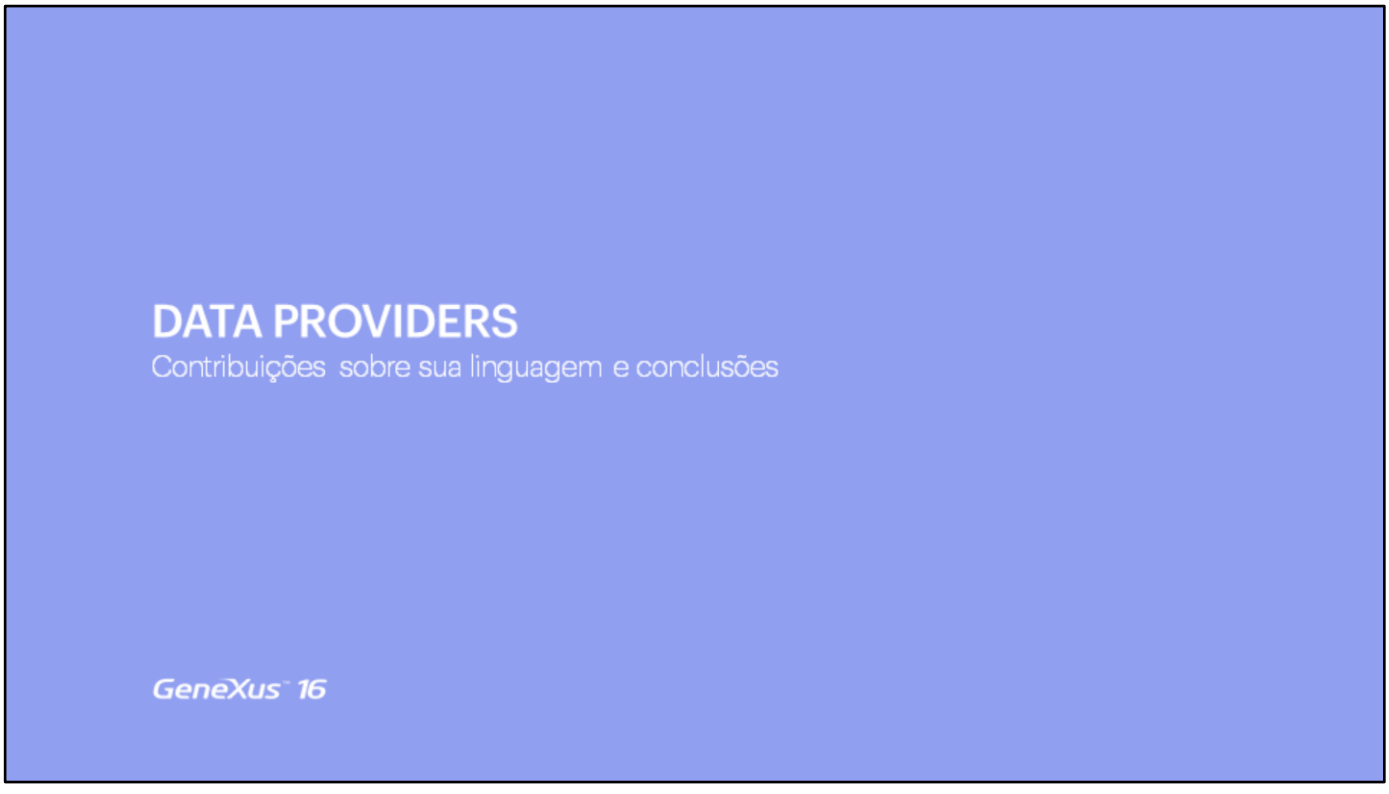
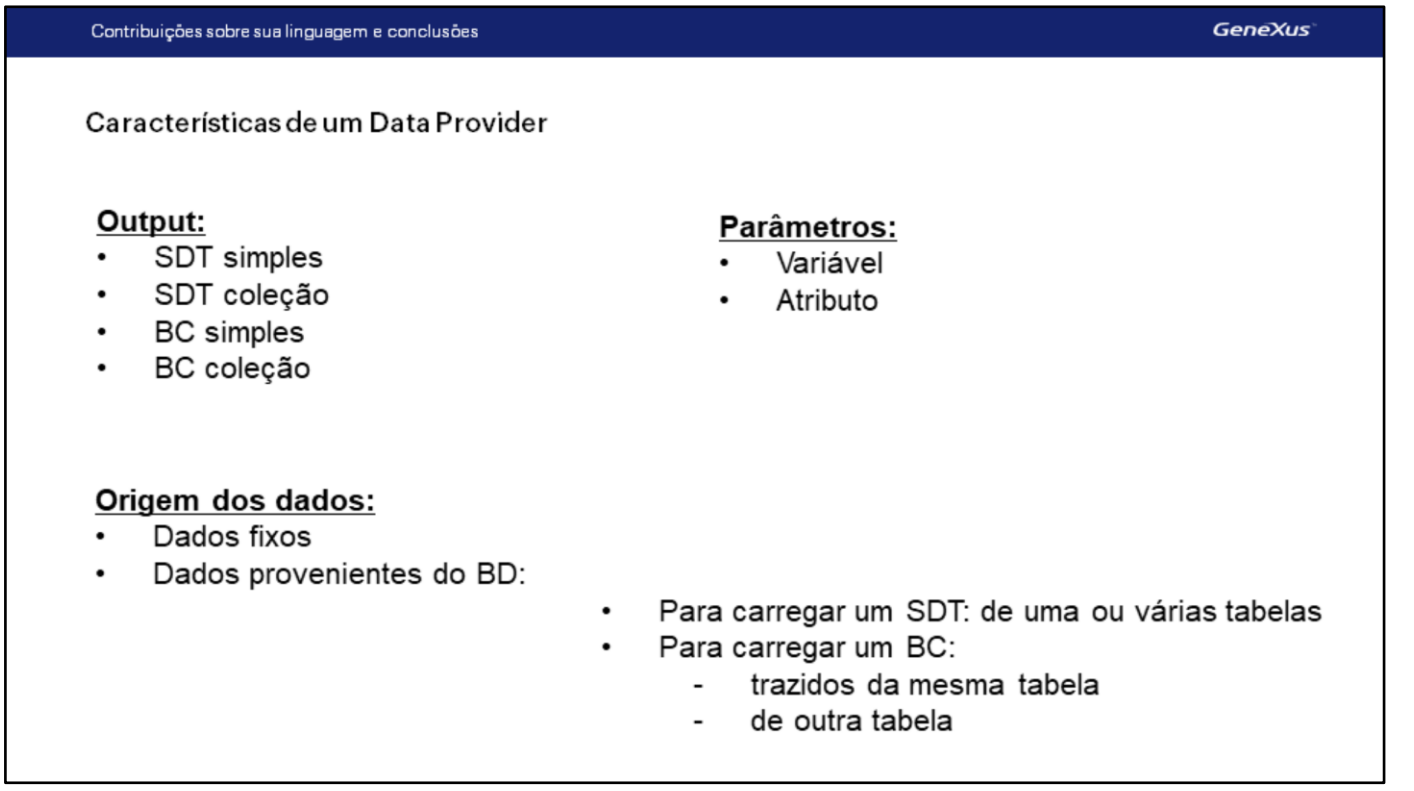


Aqui vamos introduzir novos conhecimentos sobre o uso de Data Providers.



Devemos lembrar que a finalidade de um Data Provider é devolver uma estrutura de dados carregada na memória (que pode ser coleção ou não). Para conseguir isso, ele nos fornece uma linguagem declarativa focada na estrutura de saída, de modo que basicamente precisamos indicar como obter cada um desses elementos de informação.

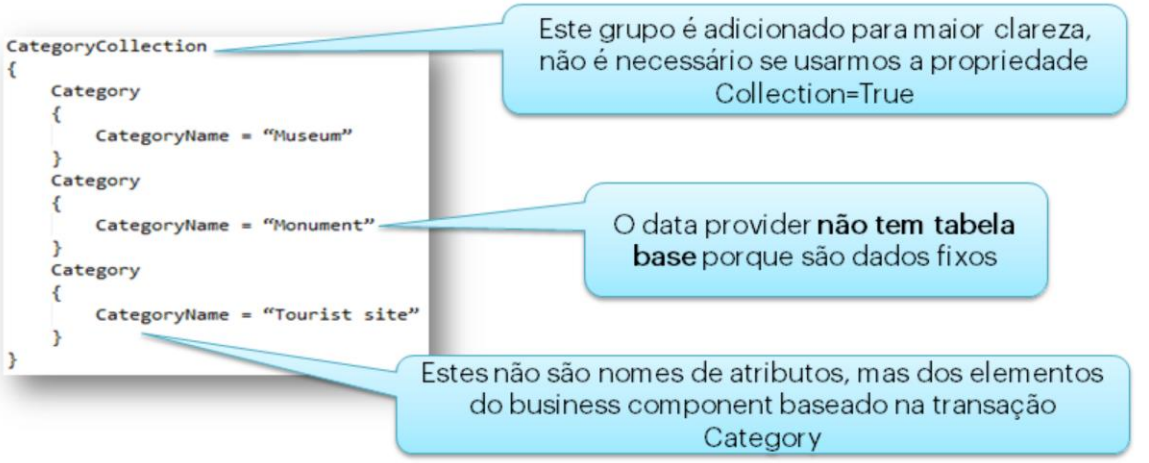
Para carregar, podemos usar um SDT simples ou de coleção, ou uma estrutura de Business Component, simples ou coleção.

Como qualquer outro objeto, um Data Provider pode receber parâmetros (variáveis e atributos). No entanto, ao contrário de todos os outros objetos, e precisamente por causa do objetivo de permitir que os desenvolvedores se concentrem na saída, ele não é declarado na regra parm, mas sim explicitamente como propriedade Output do objeto.

Vimos que os dados usados para carregar as estruturas podem ser dados fixos, ou variáveis, tirados de uma ou de várias tabelas da base de dados. Especificamente, quando estamos carregando uma estrutura do tipo business component, os dados podem ser da tabela associada à transação na qual o BC foi definido ou de outra tabela no banco de dados.

Inicializar uma tabela com dados fixos

- Criar registros na tabela CATEGORY

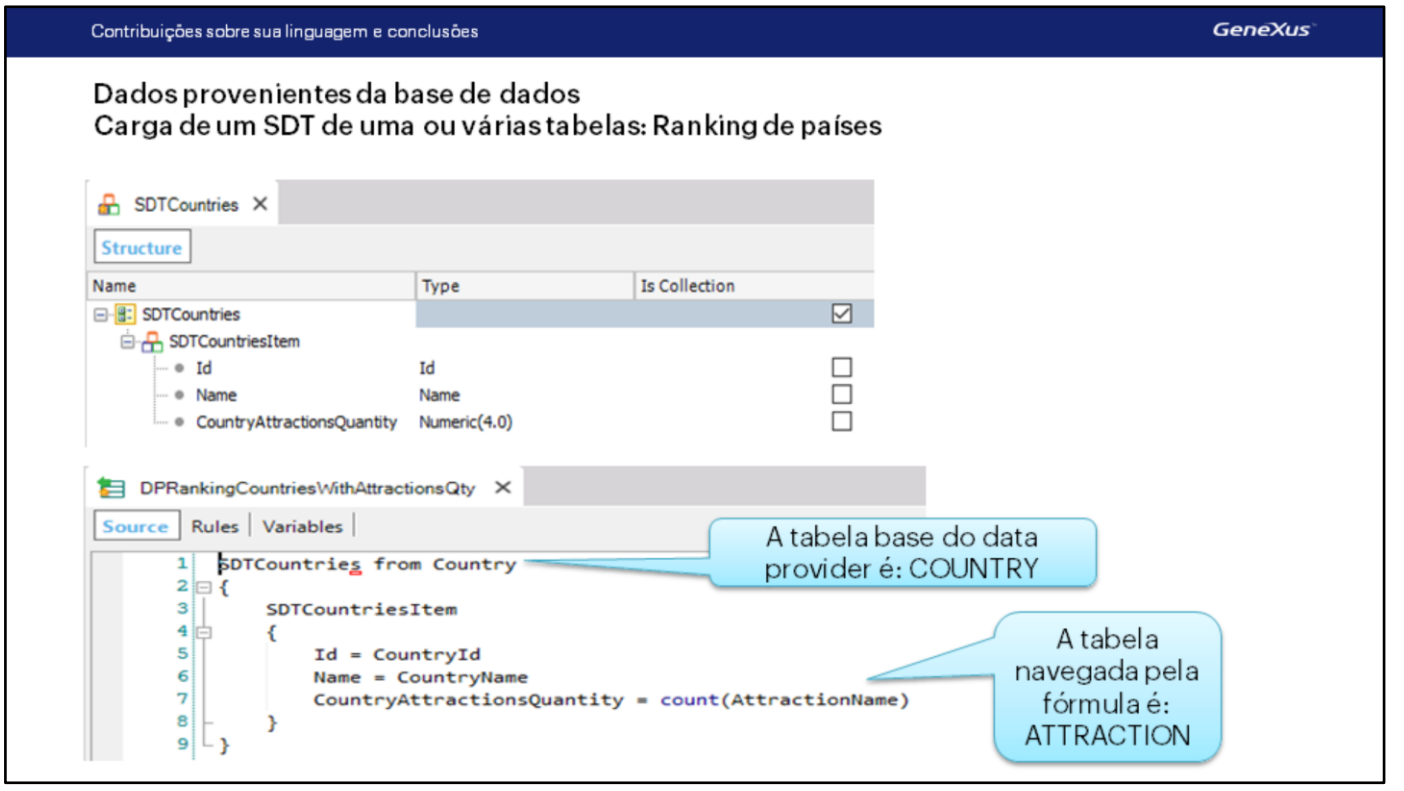


Aqui estamos inicializando uma estrutura de business components de Category, para a qual estamos usando um data provider.

Note que repetimos os grupos, um para cada categoria a ser criada. Poderíamos deixar de fora a definição do grupo CategoryCollection porque nosso objetivo é retornar uma coleção de elementos de Category e já configuramos a propriedade Collection do data provider com valor True.

Outra coisa que devemos notar é que, uma vez que o data provider não terá que ir a qualquer tabela para obter os dados, porque estamos fornecendo os dados na forma de valores fixos, então este data provider não terá tabela de base e não temos que usar uma cláusula "from".

E por último, também é importante notar que os elementos CategoryName à esquerda das atribuições não são os atributos da transação Category , mas sim os elementos do business component baseado na transação Category, que estamos carregando através do data provider.



Neste exemplo, vemos como podemos carregar um SDT com dados de várias tabelas. Nosso objetivo é construir um ranking de países pelo número de atrações em cada país.

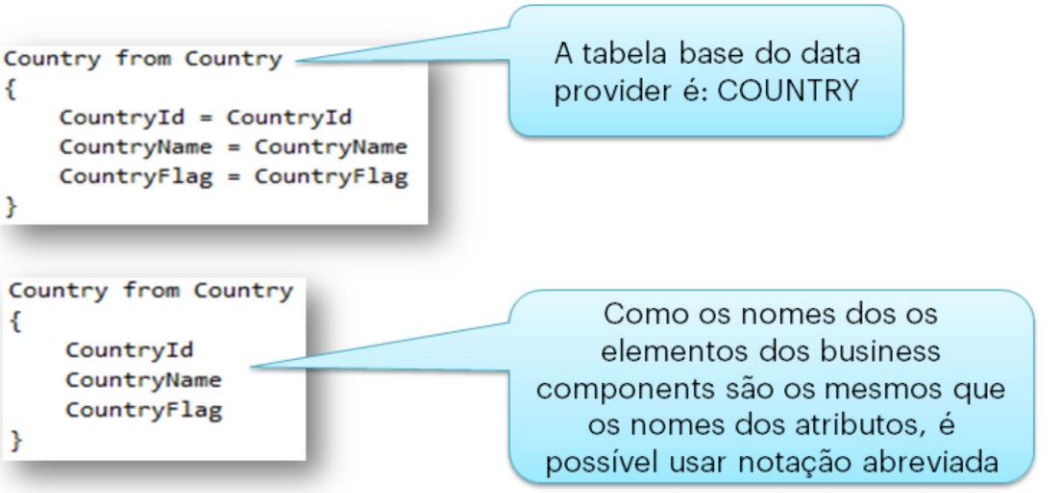
Para fazer isso, definimos uma coleção SDT para armazenar o identificador, nome e número de atrações em cada país. Para carregar este SDT usaremos um data provider.

Para obter os dados sobre os países, o data provider percorre a tabela COUNTRY e, para cada país, a fórmula de contagem navega através da tabela ATTRACTION para contar as atrações nesse país.

Uma vez que a coleção é obtida, podemos ordená-lo em ordem decrescente pelo número de atrações.

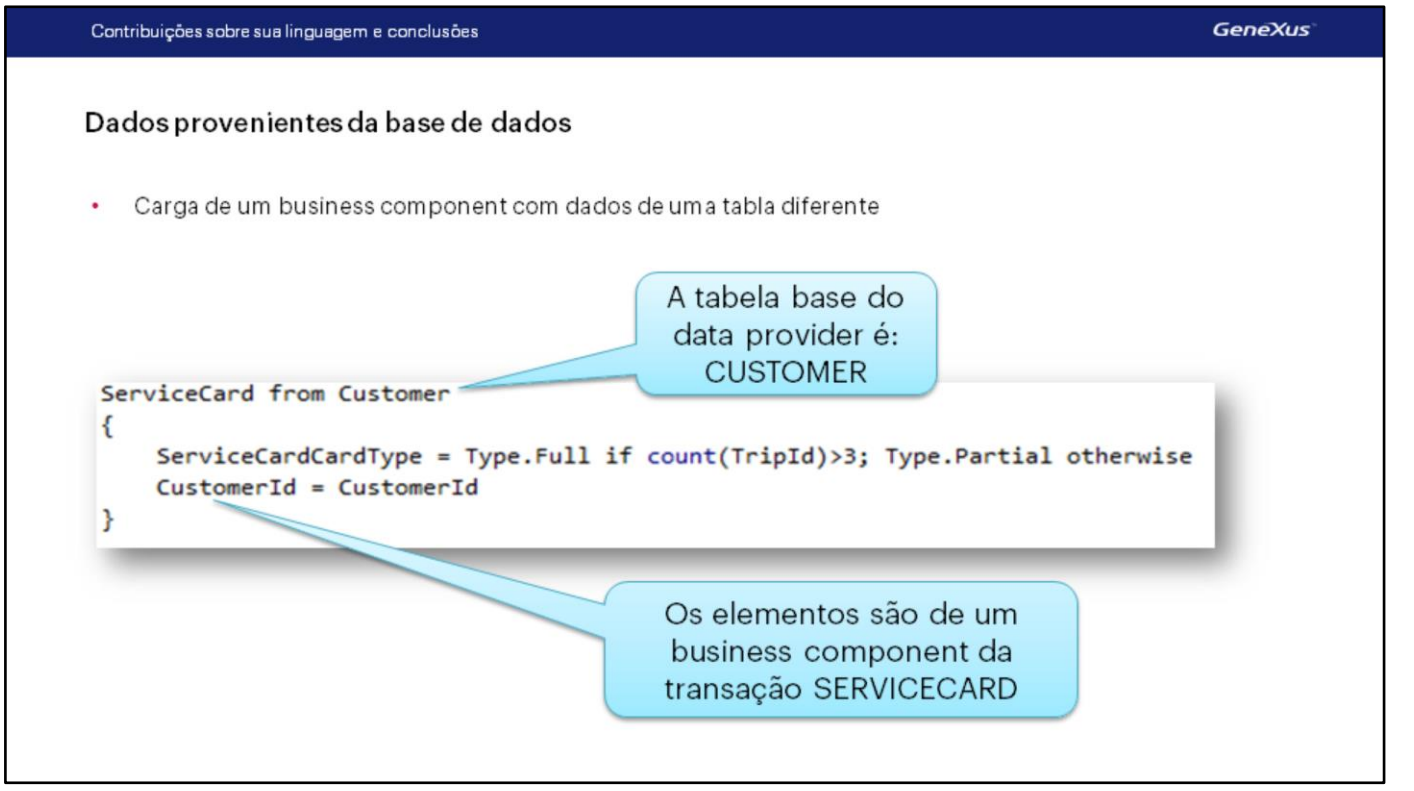
Dados provenientes da base de dados

- Carga de um business component de uma única tabela



Neste exemplo, estamos carregando os dados dos países na memória.

.

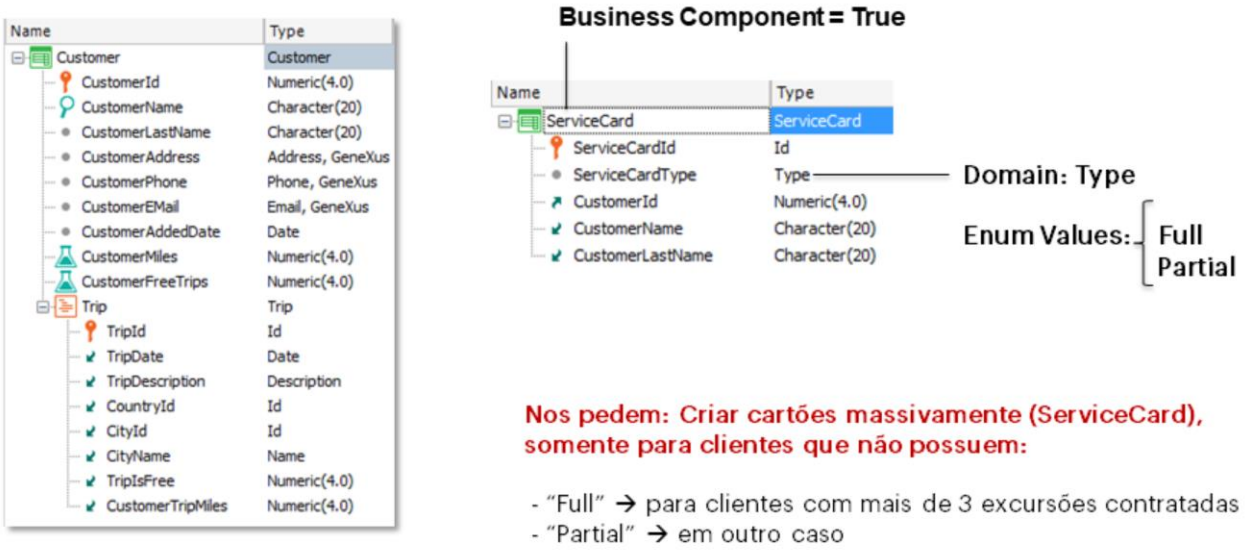


Neste exemplo, um cartão especial (tipo total ou parcial) deve ser concedido aos clientes que compraram mais de 3 viagens. Para fazer isso vamos aos clientes, a fim de contar o número de atrações e atribuir o cartão correspondente.

Fazemos isso por meio de um data provider com o qual carregamos uma estrutura de business component da transação SERVICECARD, para depois ir a essa coleção e salvar essas informações no banco de dados.

Vamos agora entrar nos detalhes deste exemplo.

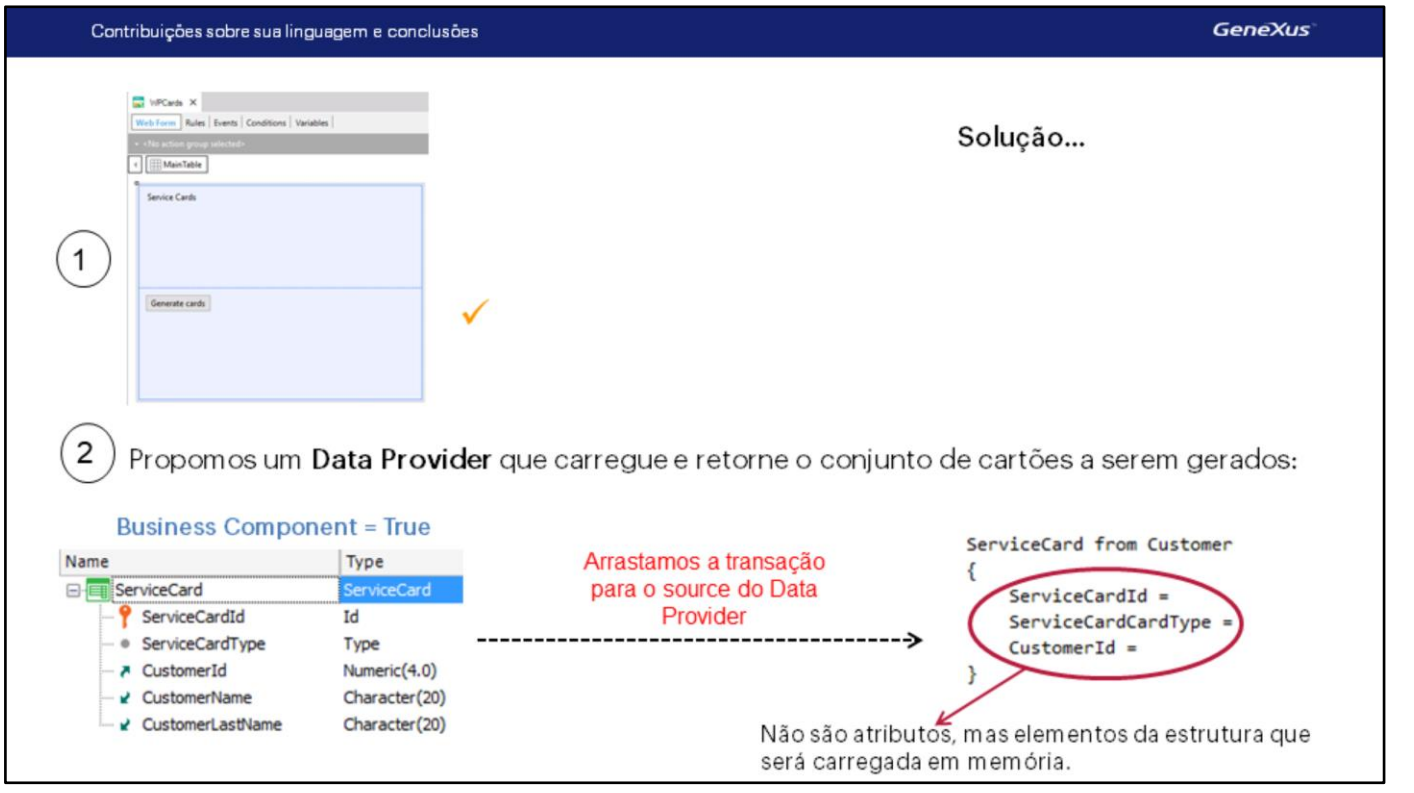
Exemplo



Agora suponhamos que a agência de viagens decide que todos os clientes que compraram mais de 3 viagens receberão um cartão especial do tipo "Full Services" que lhes permitam desfrutar todos os serviços de graça. Para casos onde o cliente tenha comprado menos de 3 viagens, será emitido um cartão do tipo "Partial Services".

As transações que temos são: a transação "Customer" e a transação "ServiceCard" que define os cartões.

Cada cartão tem um identificador de numeração automática, um cliente, e definimos o atributo de ServiceCardType com base no domínio enumerado Type (que só permite os valores "Full" ou "Partial").



Definimos o web panel WPCards que apenas oferece um botão para acionar o processo automático de geração e visualização de novos cartões.

O que deve acontecer quando o botão é pressionado?

Um cartão do tipo correspondente deve ser criado para cada cliente que não tem cartão emitido, tendo sempre em mente o número de viagens compradas na agência.

Vamos propor uma solução onde vamos **usar um Data Provider** para carregar e retornar a coleção de cartões a serem gerados**. E depois vamos à coleção para gravar os cartões no banco de dados.**

Como vamos fazer isso?

Primeiro, configuramos a transação SERVICECARD como Business Component, a fim de gravar os cartões, para o qual aplicamos o conceito de Business Component.

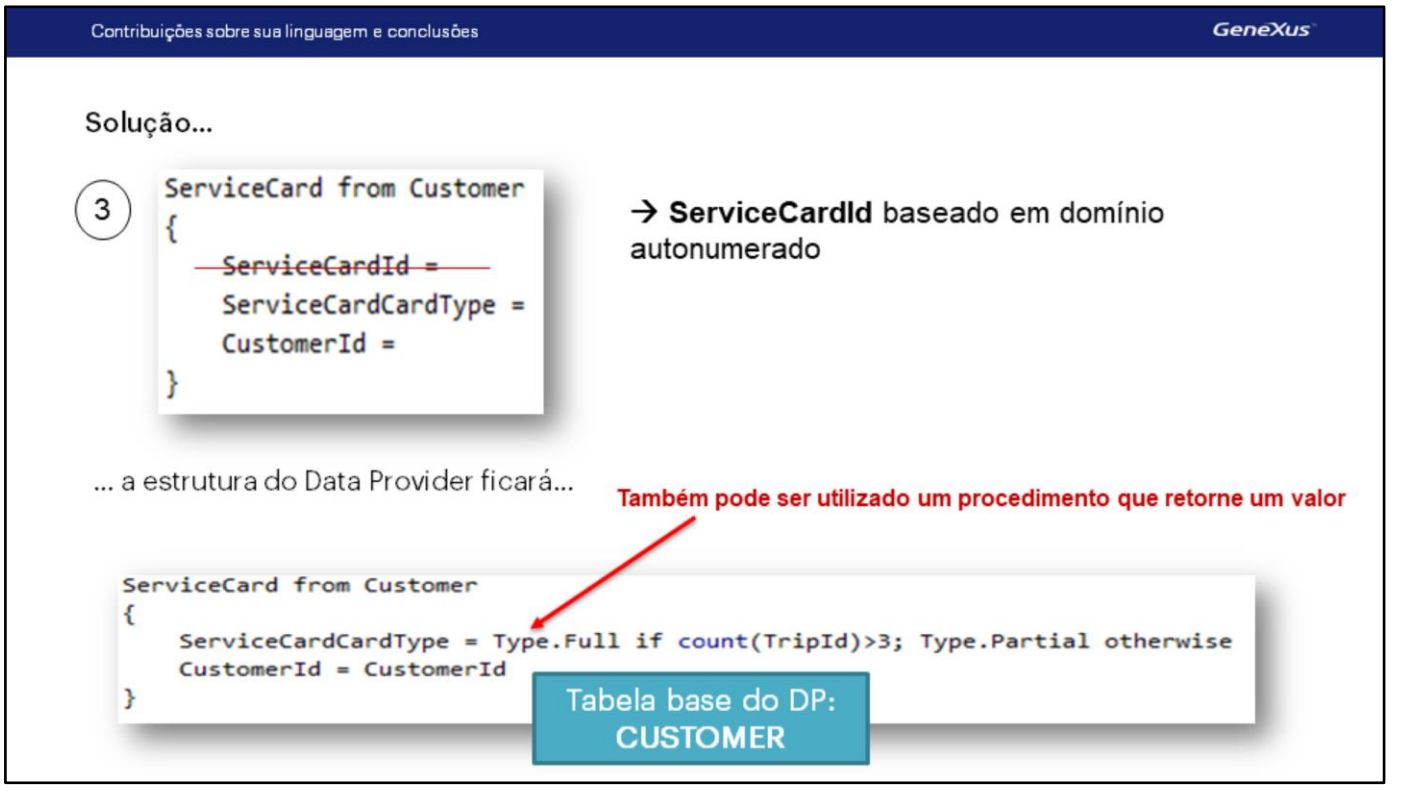
Em seguida, criamos um objeto Data Provider chamado DPCards e arrastamos a transação SERVICECARD para o source do data provider.

Como vimos em vídeos anteriores, isso definirá uma estrutura na memória com itens cujo nome e tipo são iguais aos dos atributos da transação definida como Business Components.

Então, devemos ir à tabela CUSTOMER, em seguida, filtrar os clientes para quem os cartões ainda não foram emitidos, **e adicionar para eles** um cartão na coleção.

A tabela que irá navegar o Data Provider é determinada com a transação base definida na cláusula From, que neste caso é CUSTOMER.

Observe que a tabela que iremos para obter os dados não é a mesma tabela associada à transação na qual definimos o Business Components, isto é, SERVICECARD.



É claro que iremos atribuir o atributo CustomerId ao item CustomerId.

Não precisamos atribuir um valor específico ao item ServiceCardId porque, como devemos lembrar, a estrutura a ser carregada foi arrastada de uma transação declarada como Business Component e, portanto, esse item é baseado no atributo ServiceCardId que pertence ao domínio Id, e sua propriedade Autonumber foi definida como Yes.

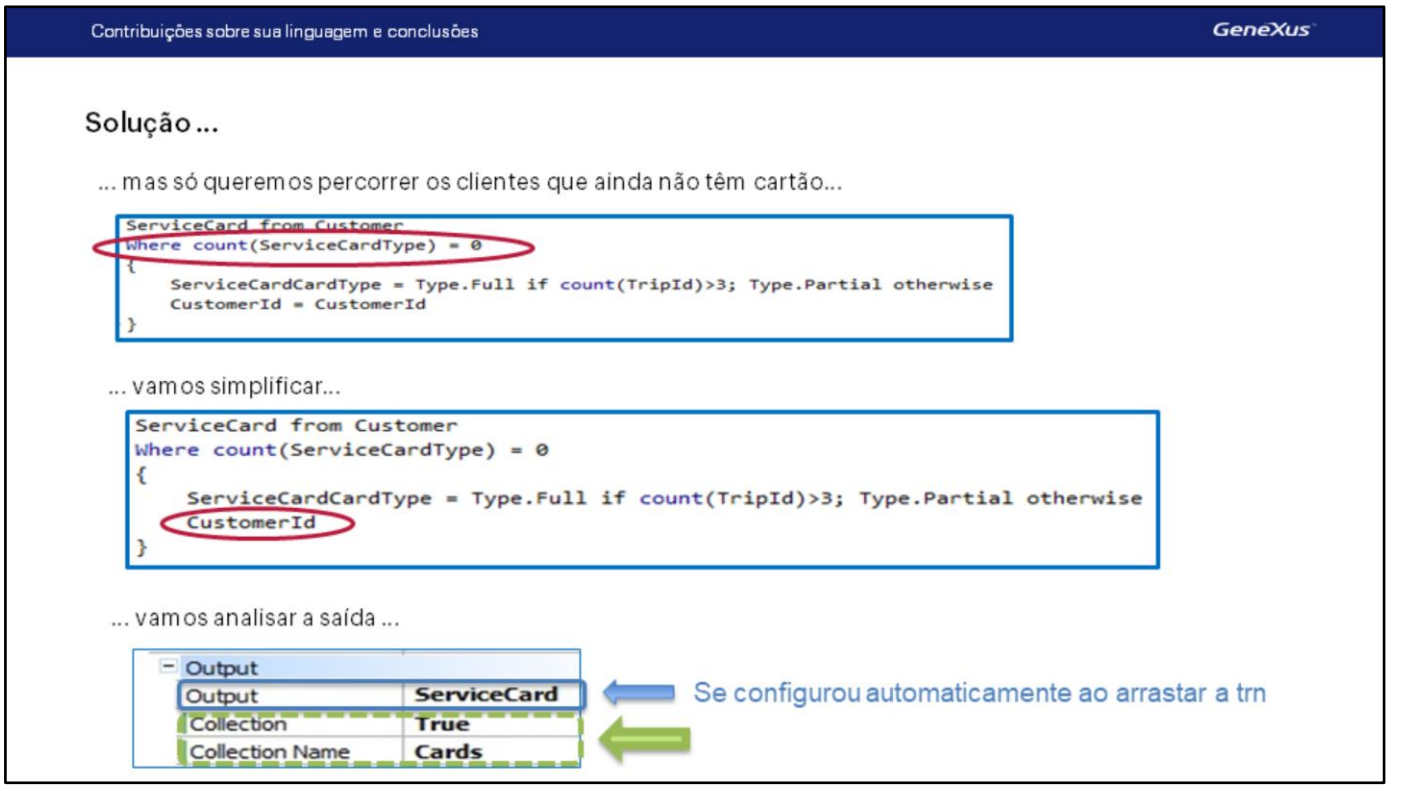
Para ServiceCardType podemos atribuir o valor retornado por uma fórmula condicional. Isso significa que a fórmula retornará o tipo "Full" ou "Partial" de acordo com o número de viagens que o cliente comprou. Poderíamos também ter usado um procedimento para calcular e retornar esse valor.

Para definir a tabela de base que irá navegar data provider, GeneXus vai para a transação base que adicionamos com a cláusula From, então a tabela à qual irá é a associada a essa transação, que neste caso é a tabela CUSTOMER.

GeneXus também verificará se os atributos que adicionamos à direita dos sinais de atribuição fazem parte da tabela estendida de CUSTOMER, caso contrário, ocorrerá um erro e o veremos reportado na lista de navegação do data provider.

Observe que os atributos dentro das fórmulas não são considerados para esta verificação, porque eles são usados somente para definir a tabela a ser navegada pela fórmula.

Não é necessário atribuir um valor ao elemento correspondente ao atributo autonumerado ServiceCardId porque ele recebe automaticamente devido à autonumeração do atributo. Da mesma forma, podemos deixar não atribuído qualquer elemento no business component para o qual não desejamos carregar um valor, então quando o registro é criado na tabela, o valor do atributo ficará vazio.
Neste caso, as restrições se aplicam às chaves estrangeiras, às quais devemos atribuir um valor, a menos que tenhamos definido o atributo como nullable.



Lembre-se que nós não queremos navegar todos os clientes e carregar um cartão para cada um. Nós queremos navegar somente aqueles clientes que ainda não têm cartões.

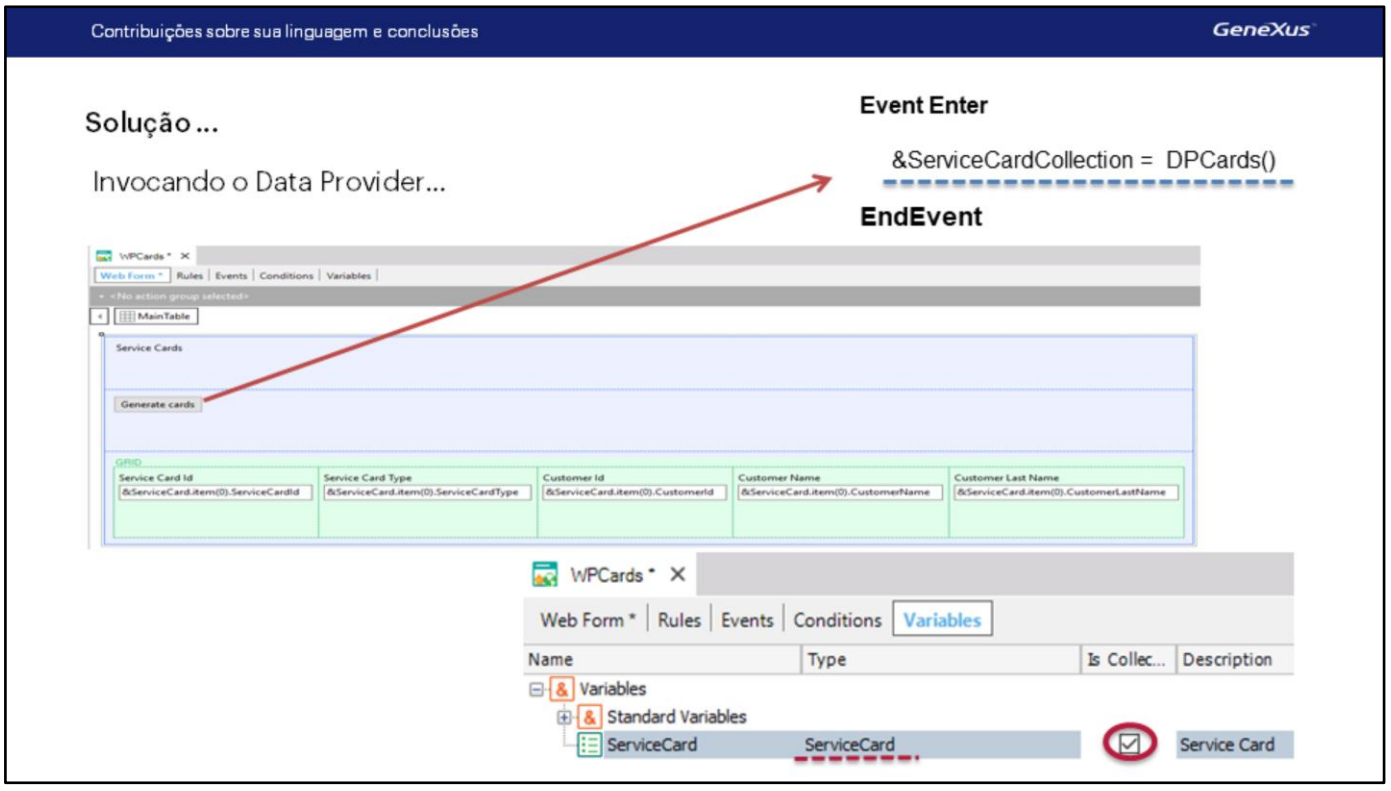
Data providers permitem-nos incluir, em sua sintaxe, todas as cláusulas permitidas no For each, então nós adicionamos a cláusula Where exibida no slide.

Como o único atributo referenciado no Where está dentro de uma fórmula, como disse, ele não será incluído na verificação se pertence à tabela estendida de Customer ou não.

Nota que, em vez de CustomerId = Customer Id, nós simplesmente usamos CustomerId. Lembre-se que o CustomerId à esquerda da atribuição é o elemento da estrutura que será carregado na memória, e o da direita é o atributo que vai definir o seu valor. Como eles têm o mesmo nome, podemos usar a **notação abreviada** e escrever apenas CustomerId.

Vamos agora analisar a saída, em outras palavras, o que retorna o data provider . Como a transação SERVICECARD foi arrastada para o source, então a propriedade Output foi automaticamente associada com o business component ServiceCard associado com a transação.

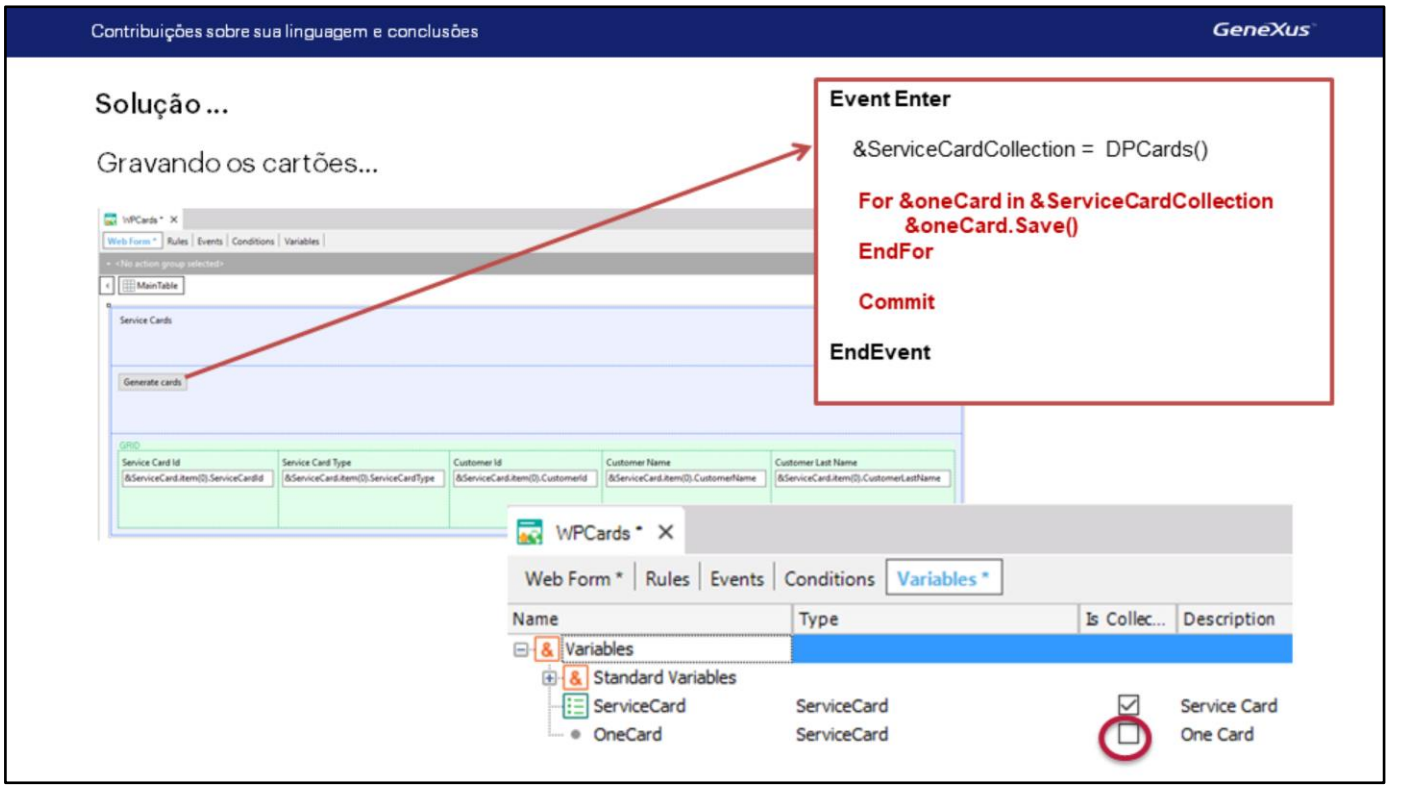
E sobre a propriedade Collection? A estrutura que estamos carregando não representa uma coleção. Ele apenas representa uma instância em memória, com a estrutura da transação SERVICECARD. No entanto, devemos obter uma **coleção de cartões gerados**, então configuramos a propriedade Collection com valor True... e podemos indicar um nome para a coleção que esse data provider irá retornar carregado.



Agora vamos voltar ao web panel para chamar o provedor de dados.

No evento Enter associado ao botão, atribuímos o que retorna o data provider para uma variável (&ServiceCardCollection) definido como uma coleção de cartões (ServiceCard).

No form do web panel, inserimos a variável &ServiceCardCollection. Porque é uma coleção, GeneXus automaticamente vai entender que ele deve mostrar o conteúdo em um grid.

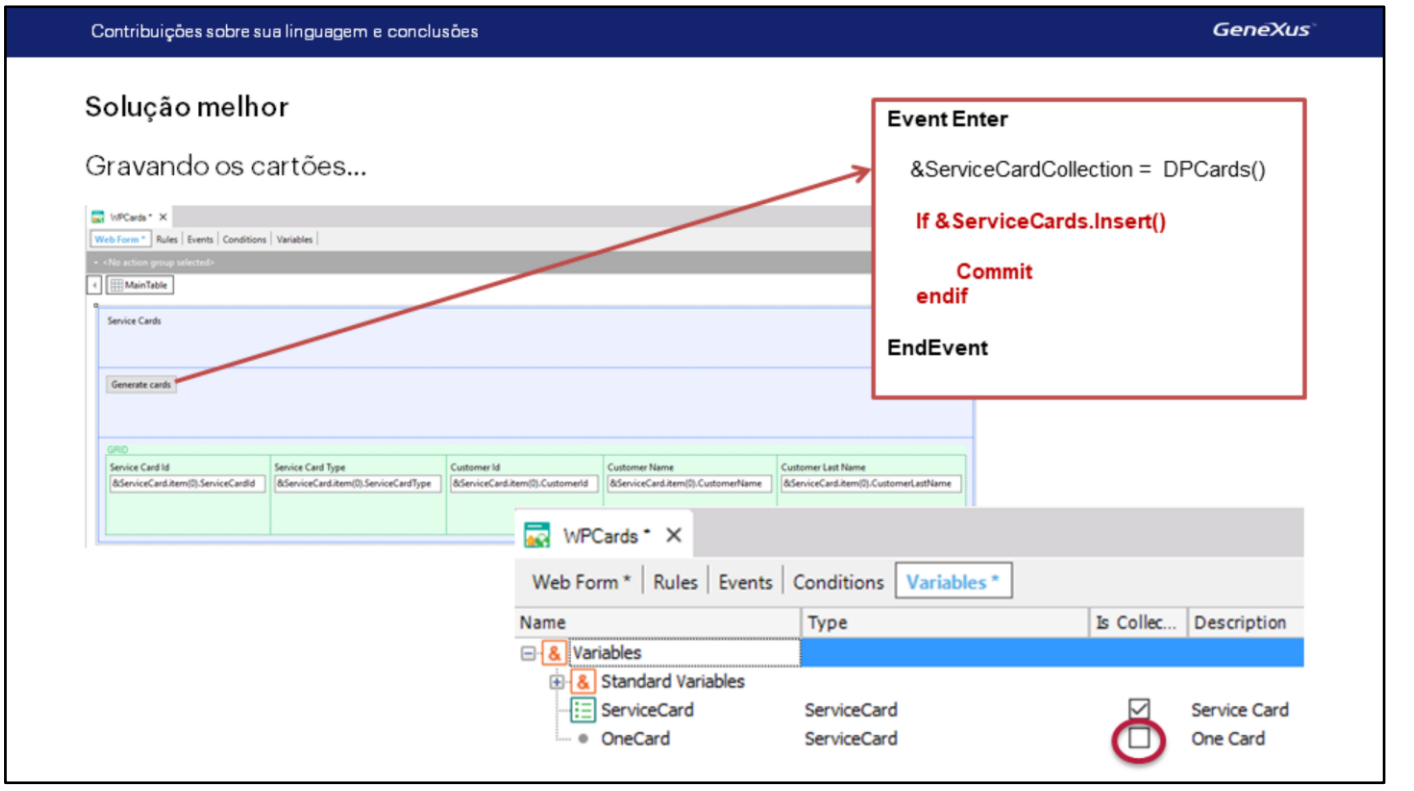


Agora, isso é suficiente para os cartões retornados pelo data provider estarem realmente gravados na tabela SERVICECARD associada com a transação SERVICECARD?

Não, não é suficiente. Por enquanto, os cartões estão carregados na memória e mostramos o conteúdo da coleção.

Quando estudamos o uso de business components, vimos que, para salvar devemos usar o método Save e então executar o Commit. Então, ainda temos que ir à coleção retornada pelo data provider e em seguida salvar cada elemento da coleção como um registro na tabela física. E após o salvamento de todos os cartões, nós declaramos o comando Commit.

Para executar através da coleção de cartões retornada pelo data provider, usamos o comando **For element in collection.** Esta variável &oneCard deve ser definida como o tipo business component ServiceCard, ela representa cada elemento da coleção que é iterada.

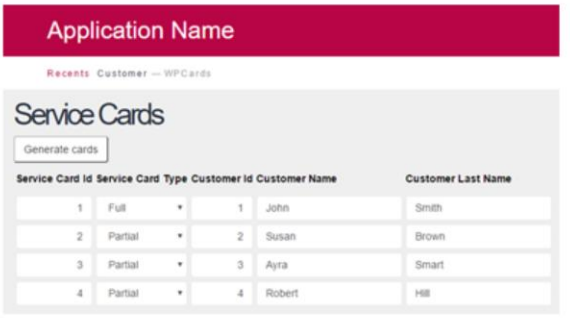


No entanto, lembremos que temos o método Insert de uma variável coleção de Business Components, que já faz automaticamente o que fizemos antes manualmente.

E não apenas isso, mas também retorna True se todas as inserções da coleção forem bem-sucedidas e False caso contrário. Desta forma, podemos fazer o commit se tudo for bem-sucedido. Se não, devemos proceder a consulta às mensagens de erro e tomar as ações que consideramos pertinentes.

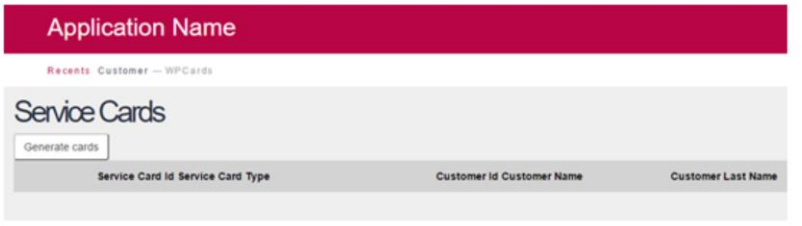
Solução..

Em execução...



1) Pressiona-se o botão.
Resultado: São exibidos os cartões gerados.

2) Pressiona-se novamente o botão.
Resultado: Não são gerados cartões.



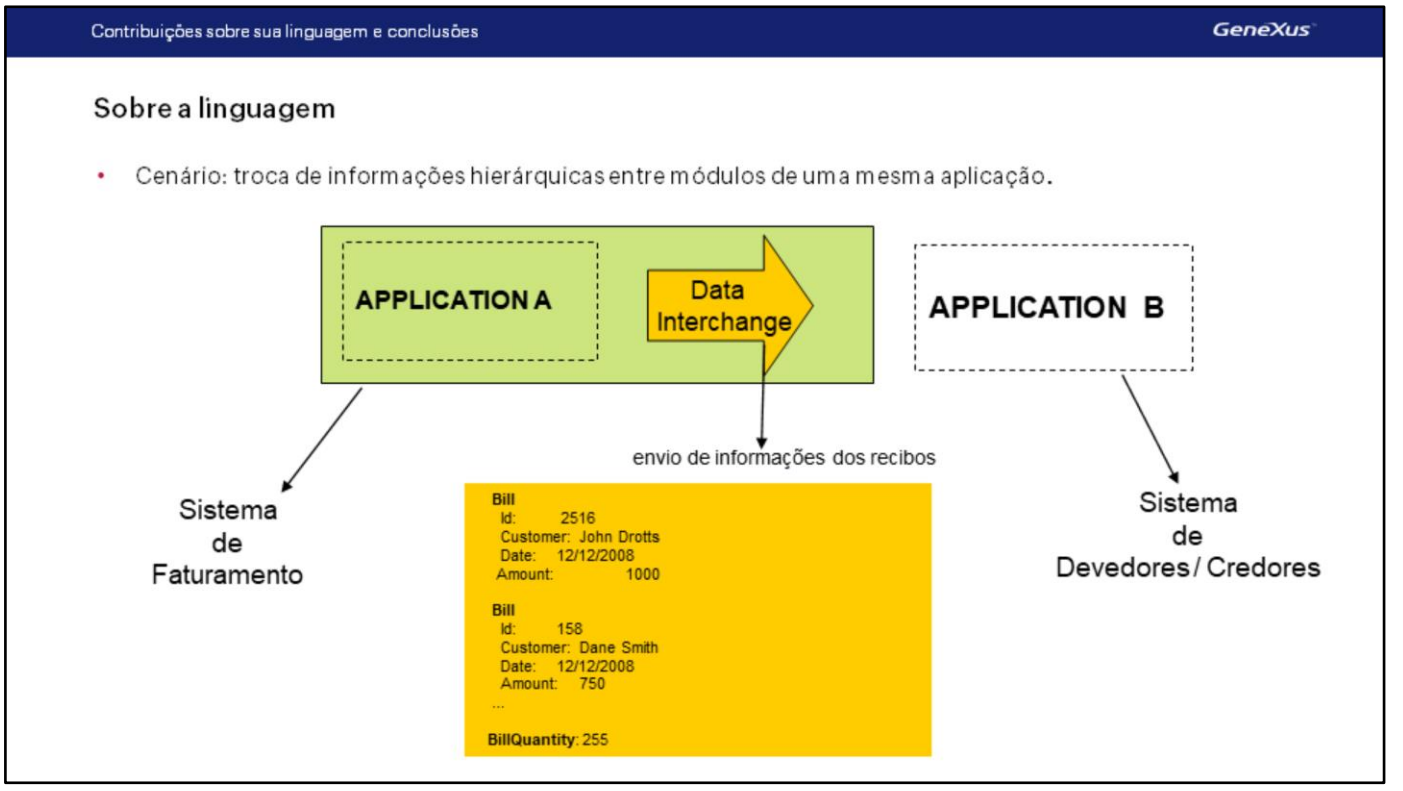
Agora, o desenvolvimento do que nos foi pedido finalmente está completo. Executamos o web panel e pressionamos o botão. Na grade podemos ver a lista de cartões gerados.

Poderíamos nos perguntar o que aconteceria se pressionássemos o botão "Generate Cards" mais uma vez ...?
Os cartões serão criados novamente para os mesmos clientes?

Eles não serão, porque, no data provider, filtramos que só queríamos navegar clientes sem cartões.

Devemos mencionar que existem outras soluções para resolver o mesmo requisito no GeneXus. **Com esta implementação utilizamos o conceito de Business Component para atualizar o banco de dados e combinamos seu uso com o carregamento prévio de uma estrutura de coleção em memória, com os dados a serem gravados.**

O uso de um Data Provider para este propósito é bastante simples e nos poupa de ter que escrever código explícito.

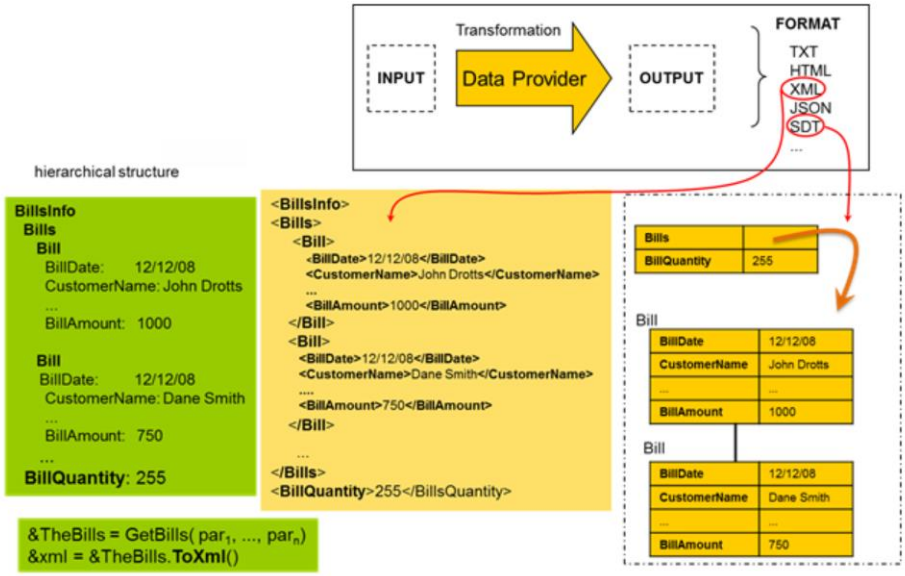


Suponha que o sistema da agência de viagens seja dividido em módulos para gerenciar o sistema de faturamento e o sistema de devedores/credores. Queremos enviar uma listagem, do sistema de faturamento ao sistema de devedores/credores, com os recibos correspondentes a um determinado período de faturas (isto é, para um determinado período, devemos resumir, para cada cliente, o montante total faturado e então, gerar um recibo).

É informação hierárquica (estaremos enviando dados de recebimento específicos de cada documento de recebimento).

Os formatos mais comuns de troca de informações hierárquicas são geralmente Xml e Json, embora existam mais.

Sobre a linguagem

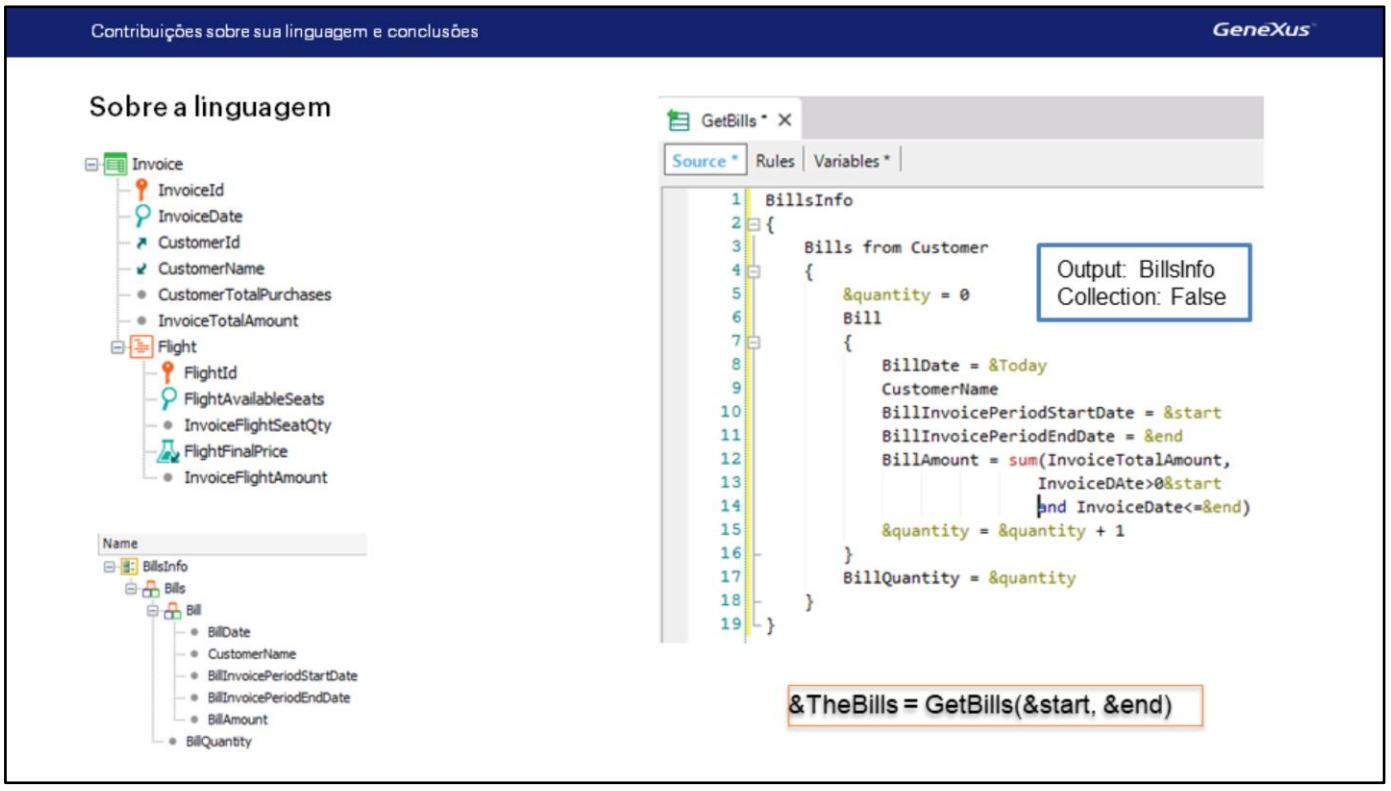


Devemos lembrar que, com um Data Provider, o foco está na linguagem de saída: a composição da Saída é indicada em uma estrutura hierárquica.

Então, para cada elemento na estrutura hierárquica, teremos que indicar –no Source do data provider- como ele é calculado.

Podemos representar a mesma informação estruturada usando os vários formatos existentes.

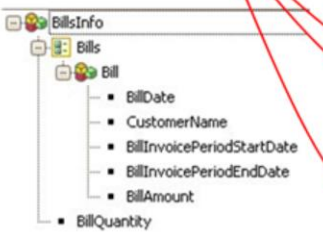
Essa é a idéia por trás do data provider. Se um novo formato para representar informação estruturada surgir no futuro, então o Data Provider permanecerá inalterado ... GeneXus implementará o método de transformação para esse formato, e o que faremos é usá-lo.



Neste caso, estamos usando uma estrutura mais complexa (um SDT com um elemento Quantity, e uma coleção de Bills).

Sobre a linguagem

- Componentes básicos:
  - Grupos
  - Elementos
  - Variáveis

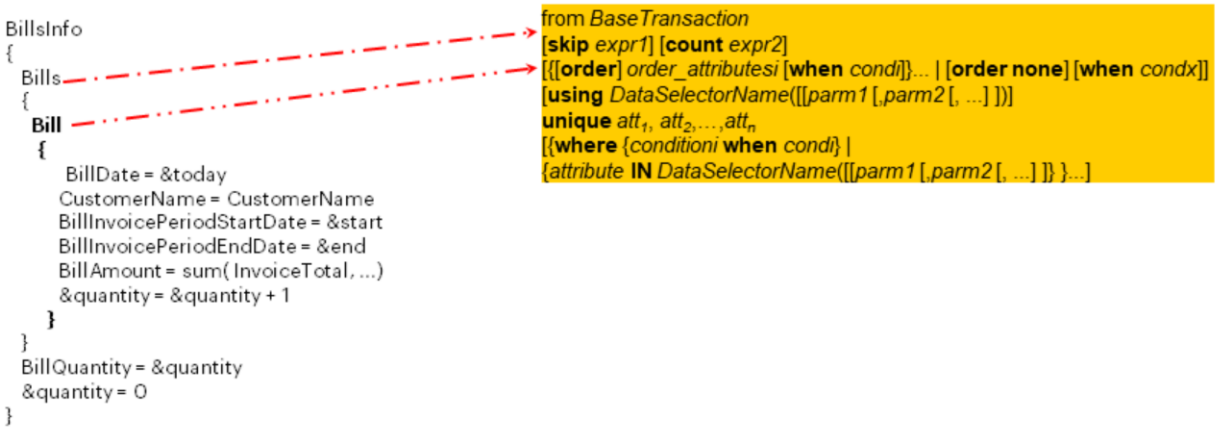



```
BillsInfo
{
   Bills
   {
      Bill
      {
         BillDate = &today
         CustomerName = CustomerName
         BillInvoicePeriodStartDate = &start
         BillInvoicePeriodEndDate = &end
         BillAmount = sum( InvoiceTotal, ... )
         &quantity =  &quantity + 1
      }
   }
   BillQuantity = &quantity
}
```

Aqui podemos identificar os componentes básicos na linguagem dos Data Providers.

Sobre a linguagem

- Um grupo repetitivo é análogo a um for each:
  - Determina tabela base (da mesma forma que em um for each)
  - Tem disponíveis as mesmas cláusulas que para um for each:



No exemplo, o grupo com o nome Bill será repetitivo. Por quê? Podemos responder a esta pergunta com outra pergunta: o que aconteceria se fosse um comando For Each, onde os elementos à esquerda das atribuições correspondem aos vários elementos de uma variável SDT? Nesse caso, a presença de CustomerName (à direita da segunda atribuição) nos permite afirmar que existe uma tabela base. Queremos iterar na tabela CUSTOMER, então escrevemos a cláusula "from Customer" de uma maneira análoga como faríamos no caso de um comando For Each.

Observe que, por outro lado, o grupo com o nome BillsInfo não será repetitivo, e não tem quaisquer cláusulas associadas, enquanto os elementos contidos nele são definidos com base em variáveis em vez de atributos:

```
BillQuantity = &quantity
&quantity = 0
```

E quanto ao grupo Bills? Note que, neste caso, o grupo contém apenas outro grupo. O grupo contido será repetitivo, então Bills será uma coleção de Bill. Portanto, o subgrupo Bill pode ser omitido (deixando apenas Bills) para que seja implícito. É assim que as cláusulas no grupo permitem que a definição de ordem e filtros possam estar associadas a este grupo.

Sobre a linguagem

- Os grupos podem ser repetidos no Source:

```
Clients
{
  Client
  {
       Name =  'Lou Reed'
       Country = 'United States'
       City = 'New York'

  }
  Client where CountryName = 'Mexico'
  {
       Name =  CustomerName
       Country = CountryName
       City = CityName
  }
}
```

O resultado retornado será uma coleção de N+1 itens: sendo N o número de clientes do México.

Se a condição fosse colocada no grupo Clients, então se aplicaria aos dois subgrupos de Client. E é por isso que as cláusulas são permitidas para operar ao nível dos grupos repetidos (itens) em vez de apenas ao nível do grupo que é coleção de itens.

Outros exemplos da linguagem

- Uso de parâmetros e cláusulas de paginação

```
parm(&PageNumber, &PageSize)

Customers
{
   Customer [Count = &PageSize] [Skip = (&PageNumber - 1) * &PageSize]
   {
     Code = CustomerId
     Name = CustomerName
   }
}
```

- Uso de variáveis, invocação a outro Data Provider, cláusula Input

```
CustomersFromAnotherDataProvider
{
   &CustomersSDT = GetCustomers() // a DataProvider that Outputs Customers collection
   Customer Input &Customer in &CustomersSDT
   {
      Id   = &Customer.Code
      Name = &Customer.Name
   }
}
```

O que tem sido considerado neste curso não é tudo o que pode ser dito sobre este assunto. Por exemplo, as variáveis utilizadas podem ser carregadas a partir de outro Data Provider, e podemos utilizar cláusulas específicas, como a Input, entre outros aspectos.

Você encontrará informações completas sobre a linguagem de Data Providers em: http://wiki.genexus.com /commwiki/servlet/wiki?5309,Toc%3AData+Provider+language

E documentos completos sobre este objeto em: http://wiki.genexus.com/commwiki/servlet/wiki?5270,Category%3AData+Provider+object,

GeneXus™

The power of doing.

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