

# DATA PROVIDERS

Contribuições sobre sua linguagem e conclusões

*GeneXus 16*

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

## Características de um Data Provider

### Output:

- SDT simples
- SDT coleção
- BC simples
- BC coleção

### Parâmetros:

- Variável
- Atributo

### Origem dos dados:

- Dados fixos
- Dados provenientes do BD:
  - Para carregar um SDT: de uma ou várias tabelas
  - Para carregar um BC:
    - trazidos da mesma tabela
    - de outra tabela

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

```
CategoryCollection
{
  Category
  {
    CategoryName = "Museum"
  }
  Category
  {
    CategoryName = "Monument"
  }
  Category
  {
    CategoryName = "Tourist site"
  }
}
```

Este grupo é adicionado para maior clareza, não é necessário se usarmos a propriedade Collection=True

O data provider **não tem tabela base** porque são dados fixos

Estes não são nomes de atributos, mas dos elementos do business component baseado na transação 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.

## Dados provenientes da base de dados

### Carga de um SDT de uma ou várias tabelas: Ranking de países

The screenshot displays the GeneXus IDE interface. The top window, titled 'SDTCountries', shows the 'Structure' tab with a tree view of the SDT. The 'SDTCountries' collection contains an 'SDTCountriesItem' with three fields: 'Id' (Type: Id), 'Name' (Type: Name), and 'CountryAttractionsQuantity' (Type: Numeric(4,0)).

The bottom window, titled 'DPRankingCountriesWithAttractionsQty', shows the 'Source' tab with the following code:

```
1 SDTCountries from Country
2 {
3   SDTCountriesItem
4   {
5     Id = CountryId
6     Name = CountryName
7     CountryAttractionsQuantity = count(AttractionName)
8   }
9 }
```

Two callout boxes provide additional context:

- A blue callout box points to the 'Country' table in the code, stating: "A tabela base do data provider é: COUNTRY".
- A blue callout box points to the 'count(AttractionName)' formula, stating: "A tabela navegada pela fórmula é: ATTRACTION".

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

```
Country from Country
{
  CountryId = CountryId
  CountryName = CountryName
  CountryFlag = CountryFlag
}
```

A tabela base do data provider é: COUNTRY

```
Country from Country
{
  CountryId
  CountryName
  CountryFlag
}
```

Como os nomes dos os elementos dos business components são os mesmos que os nomes dos atributos, é possível usar notação abreviada

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

## Dados provenientes da base de dados

- Carga de um business component com dados de uma tabela diferente

```
ServiceCard from Customer
{
  ServiceCardCardType = Type.Full if count(TripId)>3; Type.Partial otherwise
  CustomerId = CustomerId
}
```

A tabela base do data provider é: CUSTOMER

Os elementos são de um business component da transação SERVICECARD

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

Name	Type
Customer	Customer
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
CustomerAddress	Address, GeneXus
CustomerPhone	Phone, GeneXus
CustomerEMail	Email, GeneXus
CustomerAddedDate	Date
CustomerMiles	Numeric(4.0)
CustomerFreeTrips	Numeric(4.0)
Trip	Trip
TripId	Id
TripDate	Date
TripDescription	Description
CountryId	Id
CityId	Id
CityName	Name
TripIsFree	Numeric(4.0)
CustomerTripMiles	Numeric(4.0)

**Business Component = True**

Name	Type
ServiceCard	ServiceCard
ServiceCardId	Id
ServiceCardType	Type
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)

Domain: Type

Enum Values: { Full  
Partial

**Nos pedem: Criar cartões massivamente (ServiceCard), somente para clientes que não possuem:**

- "Full" → para clientes com mais de 3 excursões contratadas
- "Partial" → em outro caso

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").

## Solução...

1



2

Propomos um **Data Provider** que carregue e retorne o conjunto de cartões a serem gerados:

Business Component = True

Name	Type
ServiceCard	ServiceCard
ServiceCardId	Id
ServiceCardType	Type
CustomerId	Numeric(4,0)
CustomerName	Character(20)
CustomerLastName	Character(20)

Arrastamos a transação  
para o source do Data  
Provider



```
ServiceCard from Customer
{
  ServiceCardId =
  ServiceCardCardType =
  CustomerId =
}
```

Não são atributos, mas elementos da estrutura que será carregada em memória.

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.



## Solução...

3

```
ServiceCard from Customer
{
  ServiceCardId =
  ServiceCardCardType =
  CustomerId =
}
```

→ **ServiceCardId** baseado em domínio  
autonumerado

... a estrutura do Data Provider ficará...

Também pode ser utilizado um procedimento que retorne um valor

```
ServiceCard from Customer
{
  ServiceCardCardType = Type.Full if count(TripId)>3; Type.Partial otherwise
  CustomerId = CustomerId
}
```

Tabela base do DP:  
**CUSTOMER**

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 Autounumber 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.

## Solução...

... mas só queremos percorrer os clientes que ainda não têm cartão...

```
ServiceCard from Customer
Where count(ServiceCardType) = 0
{
  ServiceCardCardType = Type.Full if count(TripId)>3; Type.Partial otherwise
  CustomerId = CustomerId
}
```

... vamos simplificar...

```
ServiceCard from Customer
Where count(ServiceCardType) = 0
{
  ServiceCardCardType = Type.Full if count(TripId)>3; Type.Partial otherwise
  CustomerId
}
```

... vamos analisar a saída ...

Output	
Output	ServiceCard
Collection	True
Collection Name	Cards

← Se configurou automaticamente ao arrastar a trn

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.

## Solução ...

Invocando o Data Provider...

The screenshot shows the GeneXus IDE interface. At the top, there are tabs for 'Web Form \*', 'Rules', 'Events', 'Conditions', and 'Variables'. Below the tabs, there is a section for 'Service Cards' with a 'Generate cards' button. A red arrow points from this button to the 'Event Enter' configuration area. In the 'Event Enter' area, the code `&ServiceCardCollection = DPCards()` is shown, underlined with a blue dashed line. Below this, the 'EndEvent' label is visible. At the bottom, the 'Variables' tab is active, showing a table with columns 'Name', 'Type', 'Is Collec...', and 'Description'. The table contains one variable named 'ServiceCard' of type 'ServiceCard', which is highlighted with a red dashed line and a red circle containing a checkmark.

Name	Type	Is Collec...	Description
ServiceCard	ServiceCard		Service Card

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.

## Solução ...

Gravando os cartões...

The screenshot shows the GeneXus IDE interface. At the top, there's a menu bar with 'Web Form \*', 'Rules', 'Events', 'Conditions', and 'Variables \*'. Below the menu, there's a data table with columns: Service Card Id, Service Card Type, Customer Id, Customer Name, and Customer Last Name. The table contains one row with values: &ServiceCard.item().ServiceCardId, &ServiceCard.item().ServiceCardType, &ServiceCard.item().CustomerId, &ServiceCard.item().CustomerName, and &ServiceCard.item().CustomerLastName. Below the table, there's a 'Generate cards' button. A red arrow points from this button to a code block in the 'Event Enter' section. The code block contains the following code:

```

Event Enter
  &ServiceCardCollection = DPCards()

  For &oneCard in &ServiceCardCollection
    &oneCard.Save()
  EndFor

  Commit

EndEvent

```

At the bottom, there's a 'Variables \*' window showing a table of variables:

Name	Type	Is Collec...	Description
& Variables			
& Standard Variables			
ServiceCard	ServiceCard	<input checked="" type="checkbox"/>	Service Card
OneCard	ServiceCard	<input type="checkbox"/>	One Card

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.

## Solução melhor

Gravando os cartões...

**Event Enter**

```
&ServiceCardCollection = DPCards()
If &ServiceCards.Insert()
    Commit
endif
EndEvent
```

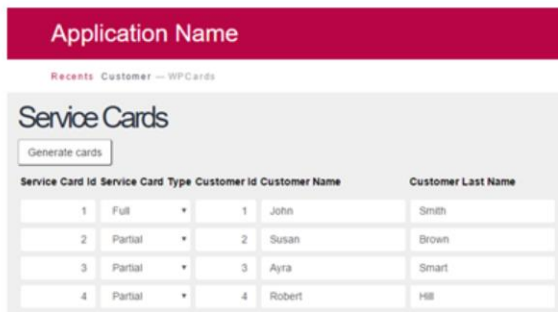
Name	Type	Is Collec...	Description
& Variables			
Standard Variables			
ServiceCard	ServiceCard	<input checked="" type="checkbox"/>	Service Card
OneCard	ServiceCard	<input type="checkbox"/>	One Card

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?

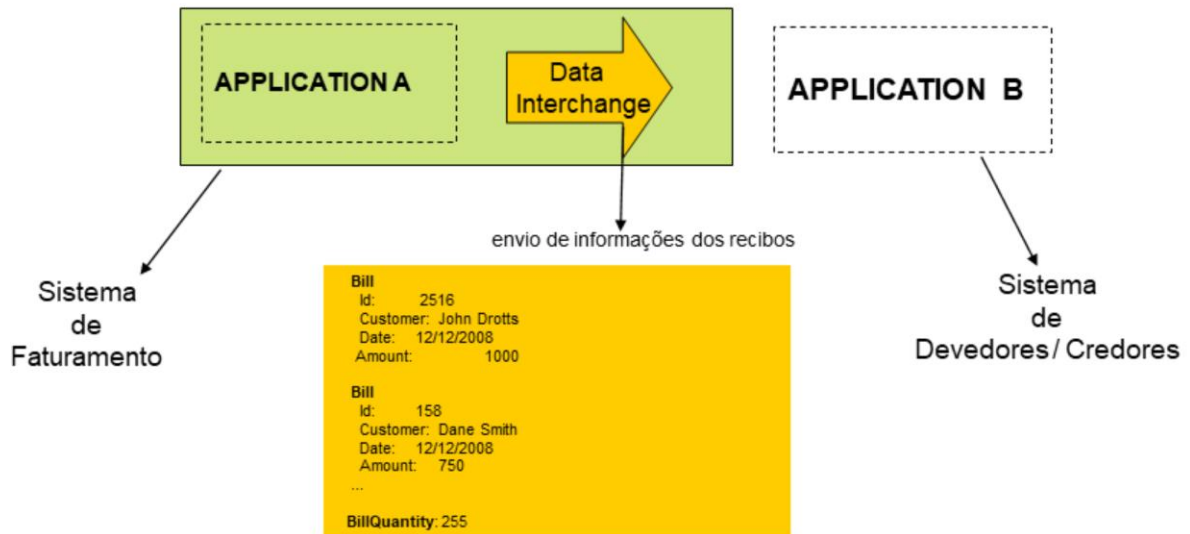
Eles não serão, porque, no data provider, filtramos que só queríamos navegar clientes sem cartões.

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.

## Sobre a linguagem

- Cenário: troca de informações hierárquicas entre módulos de uma mesma aplicação.

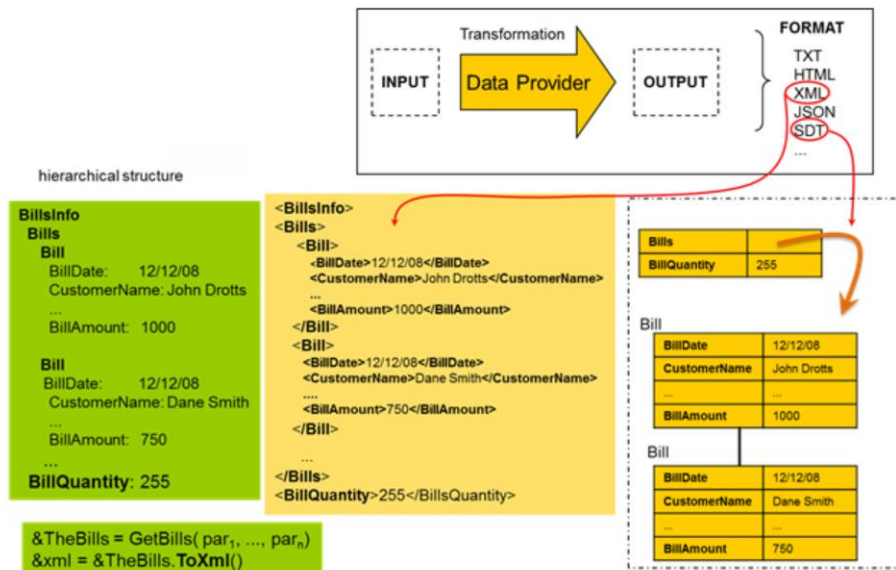


Suponha que o sistema da agência de viagens seja dividido em módulos para gerenciar o sistema de faturamento e o sistema de devedores/credores. Queremos enviar uma listagem, do sistema de faturamento ao sistema de devedores/credores, com os recibos correspondentes a um determinado período de faturas (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.

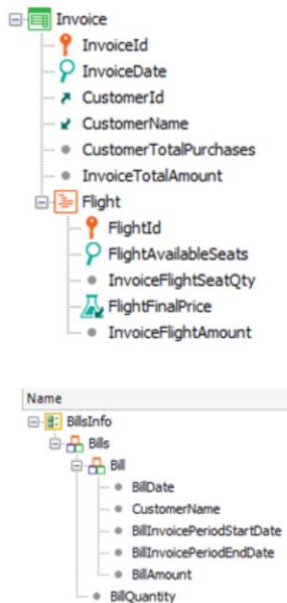
Então, para cada elemento na estrutura hierárquica, teremos que indicar –no Source do data provider– como ele é calculado.

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.



## Sobre a linguagem



```

GetBills * X
Source * Rules Variables *
1 BillsInfo
2 {
3   Bills from Customer
4   {
5     &quantity = 0
6     Bill
7     {
8       BillDate = &Today
9       CustomerName
10      BillInvoicePeriodStartDate = &start
11      BillInvoicePeriodEndDate = &end
12      BillAmount = sum(InvoiceTotalAmount,
13                    InvoiceDate>0&start
14                    and InvoiceDate<=&end)
15      &quantity = &quantity + 1
16    }
17    BillQuantity = &quantity
18  }
19 }

```

Output: BillsInfo  
Collection: False

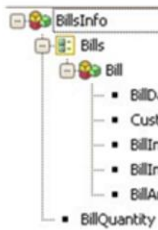
&TheBills = GetBills(&start, &end)

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:

```

BillsInfo
{
  Bills
  {
    Bill
    {
      BillDate = &today
      CustomerName = CustomerName
      BillInvoicePeriodStartDate = &start
      BillInvoicePeriodEndDate = &end
      BillAmount = sum( InvoiceTotal, ...)
      &quantity = &quantity + 1
    }
  }
  BillQuantity = &quantity
  &quantity = 0
}

```

```

from BaseTransaction
[skip expr1] [count expr2]
[{{order} order_attributesi [when condi]}... | [order none] [when condx]]
[using DataSelectorName([[parm1 [,parm2 [, ...] ]])]
unique att1, att2, ..., attn
[{{where} {conditioni when condi} }
{attribute IN DataSelectorName([[parm1 [,parm2 [, ...] ]]}] }...

```

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

```
Customers parm(&PageNumber, &PageSize)
{
  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](http://training.genexus.com)

Documentation

[wiki.genexus.com](http://wiki.genexus.com)

Certifications

[training.genexus.com/certifications](http://training.genexus.com/certifications)