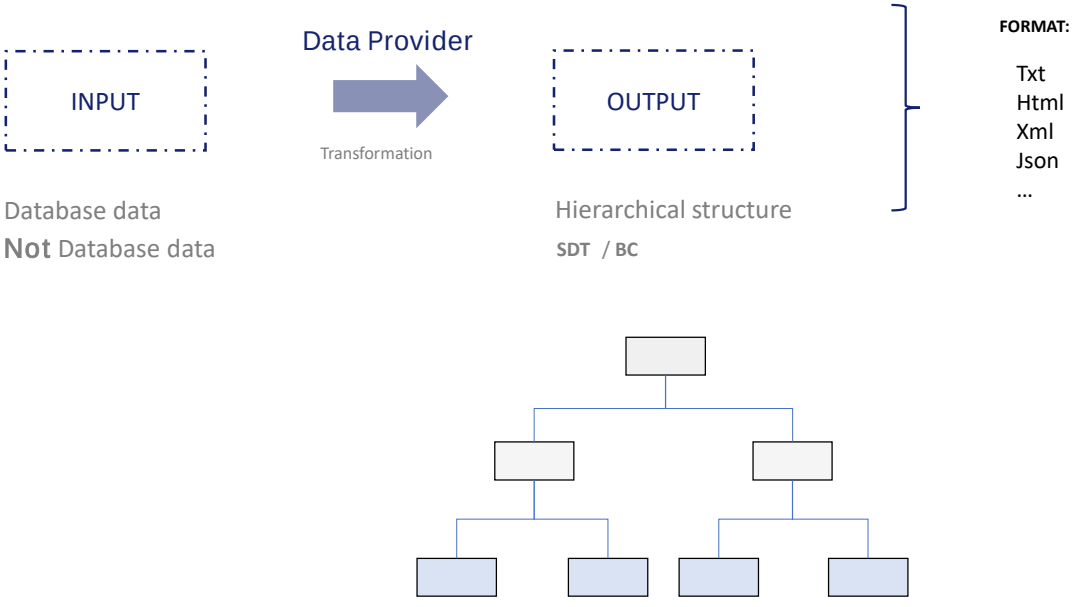
Data Providers

Linguagem e alguns exemplos

GeneXus

Data Provider



O objetivo dos Data Providers é obter informações hierárquicas para que quem precisa possa, depois, fazer algo com elas.

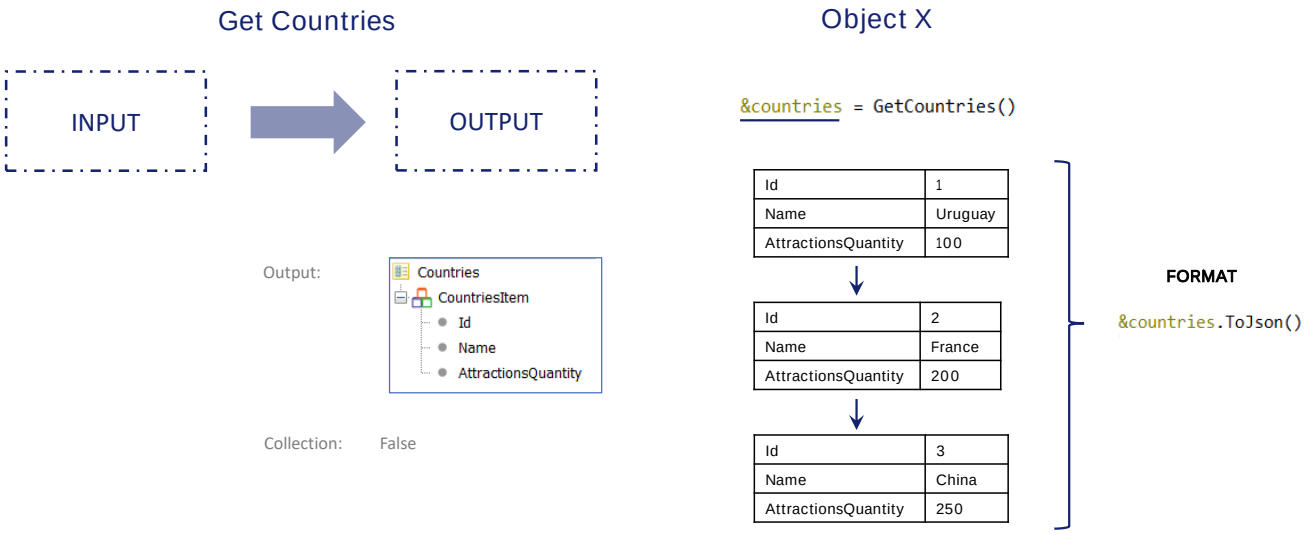
Lembremos que nos Data Providers o foco está localizado na linguagem de saída: se indica em uma estrutura hierárquica como é composto esse output. Por isso se fala de um processo de transformação dos dados de entrada nessa saída estruturada. Dados que podem ser da base de dados ou não.

A forma de representar estruturas hierárquicas no GeneXus é com o objeto SDT, junto com a possibilidade de definir coleções. Obviamente, um Business Component pode ser pensado estruturalmente como um SDT. É por isso que na propriedade Output do Data Provider podemos especificar tanto um SDT como um Business Component. Também temos a propriedade Collection para poder indicar que a saída será uma coleção desse tipo de dados indicado, ou não, ou será um único item.

Portanto, um Data Provider sempre devolverá ao chamador uma hierarquia, seja um SDT, uma coleção de SDTs, um Business Component ou uma coleção de Business Components.

Quem o invoca, portanto, é o responsável por fazer o que necessite com essa informação hierárquica. Por exemplo, convertê-la em outro formato de representação de informação hierárquica, como XML ou JSON, que são formatos úteis para interagir com terceiros.

Data Provider



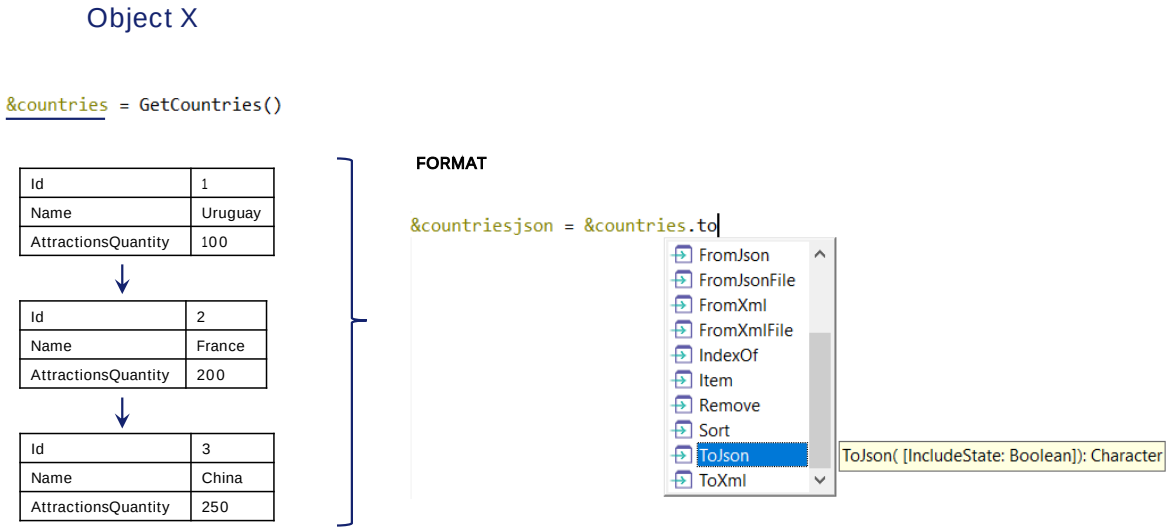
Neste exemplo, temos um Data Provider denominado GetCountries, que devolverá o tipo de dados que chamamos de Countries. Como se vê, será uma coleção de SDTs simples, cada um dos quais assumirá o nome CountriesItem.

Nas propriedades do Data Provider teremos:

A propriedade Output, com o objeto SDT denominado Countries.
E a propriedade Collection em False, já que não queremos uma coleção de Countries, se estivesse em True seria uma coleção de coleções.

Neste exemplo, quem invoca o Data Provider está atribuindo seu resultado à variável countries do mesmo tipo de dados que a saída. Este resultado será uma coleção específica em memória, com valores específicos, aqueles calculados dentro desse Data Provider.

Então, o programa que está trabalhando com essa variável pode fazer o que quiser com ela, por exemplo, converter seu conteúdo para o formato Json.

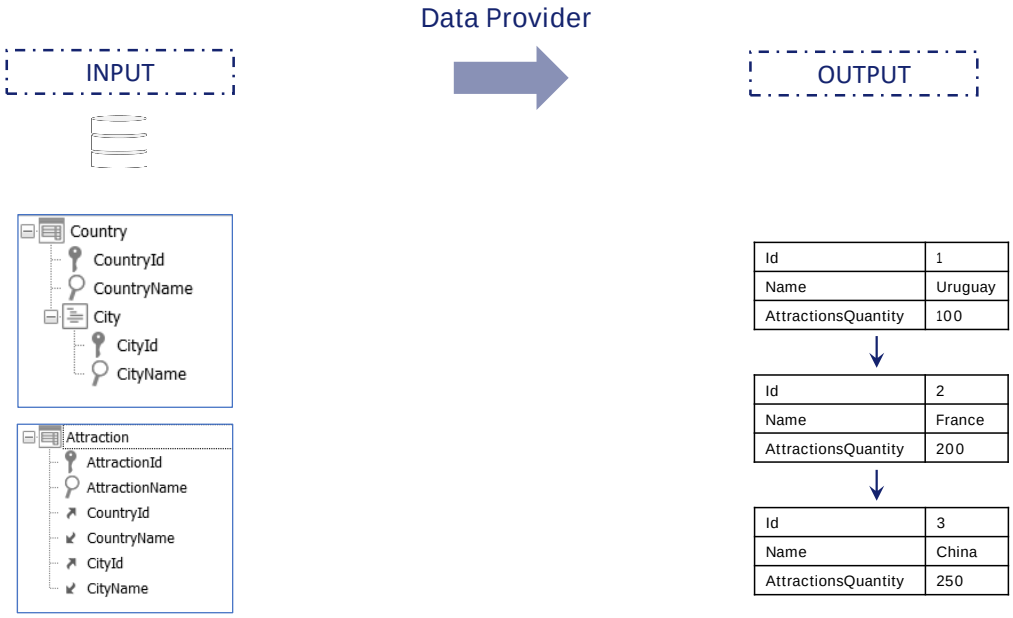


Aqui vemos como GeneXus oferece diferentes métodos de conversão entre SDTs e alguns desses outros formatos.

Se no futuro aparecer um novo formato de representação de informação estruturada, o Data Provider permanecerá inalterado. GeneXus implementará o método de transformação para esse formato, e você só terá que utilizá-lo.

Podemos tanto converter de SDT para outro formato quanto vice-versa: desse outro formato para SDT.

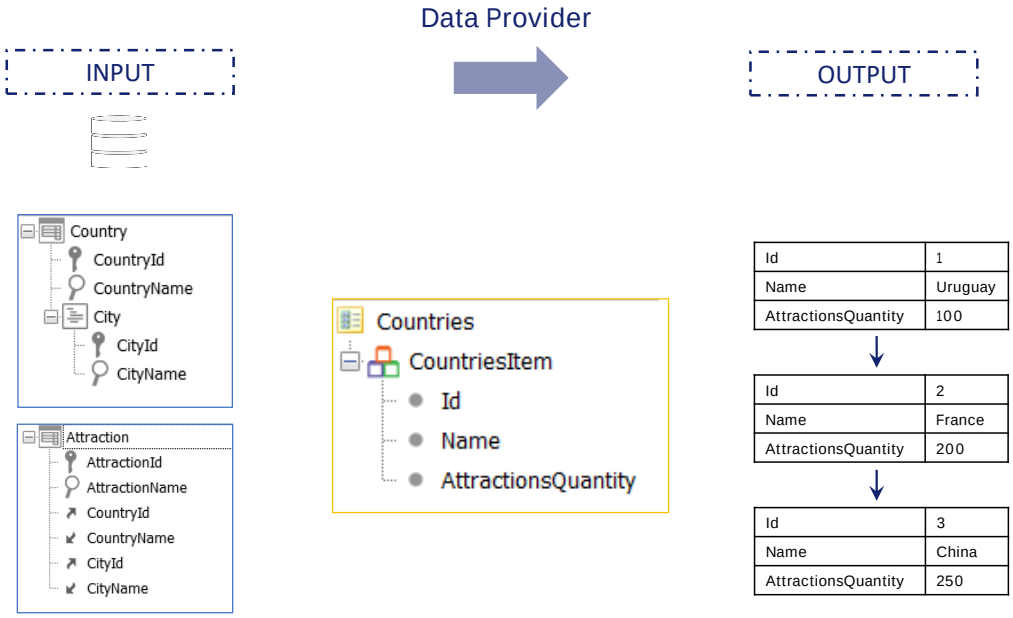
Isto já não tem a ver com o Data Provider em si, mas com os tipos de dados estruturados.
A obtenção da coleção de países poderia ter sido conseguida com um procedimento em vez de um Data Provider, e a parte da conversão seria idêntica.



Vejamos este exemplo. Suponhamos que, no contexto de uma aplicação para uma agência de viagens, necessitamos exibir em tela um ranking de países, ordenados do maior para o menor por quantidade de atrações turísticas para visitar que cada um possui.

Em nossa realidade temos as transações Country e Attraction com os seguintes atributos.
Uma maneira simples de conseguir isso é declarar um Data Provider que devolva uma coleção de países onde para cada um se adicione, além de seu nome e identificador, a quantidade de atrações que possui. E, em seguida, processar essa coleção na ordem inversa por essa quantidade.

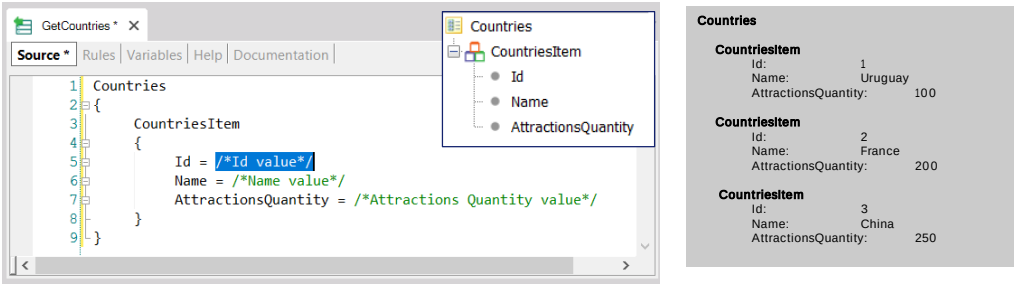
Como dissemos, a linguagem do Data Provider coloca o foco na saída, são calculados os elementos do ponto de vista da hierarquia que resultará.



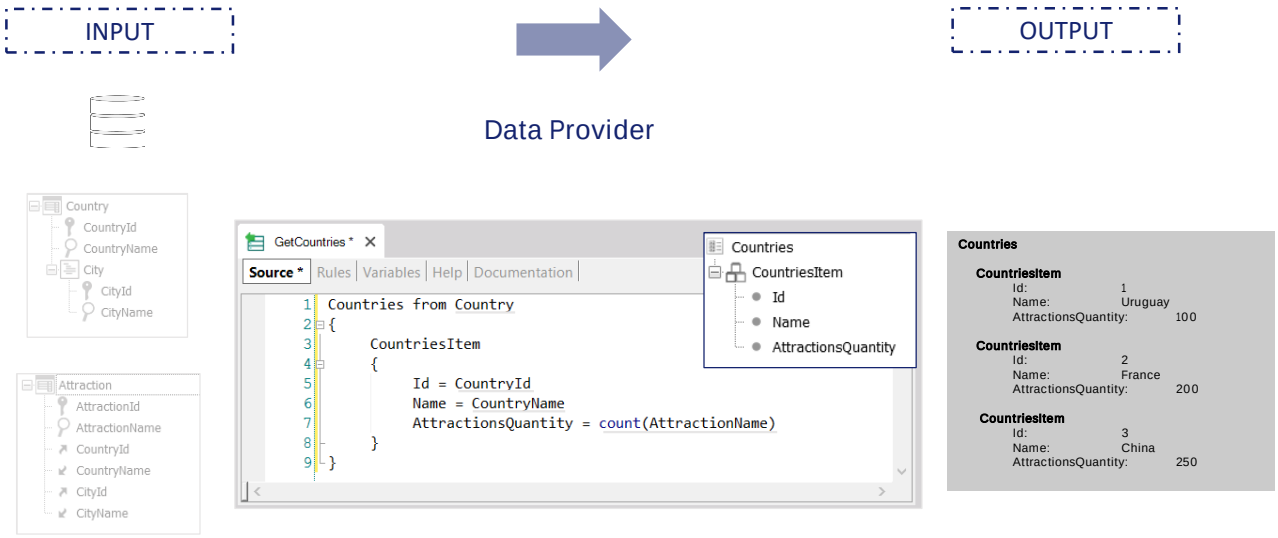
Para representar este exemplo, criamos a seguinte estrutura de dados que será aquela que, posteriormente devolva o Data Provider. E então devemos carregar este objeto SDT dentro do Source do Data Provider.

INPUT → OUTPUT

Data Provider



Ao arrastar dentro dele o SDT que será o output do Data Provider, já nos apresenta a estrutura que temos que carregar. Vemos claramente como sua linguagem está orientada para a declaração de saída.



Aqui vemos qual seria o Input de nosso Data Provider, ou seja, de onde estão sendo retirados os dados. Temos uma transação base especificada, atributos e uma fórmula inline. Claramente, as informações estão sendo retiradas da base de dados, para transformá-las na informação hierárquica requerida.

INPUT

OUTPUT

Data Provider

GetCountries

Source | Rules | Variables | Help | Documentation

```
Countries
{
     CountryiesItem
     {
          Id = 1
          Name = "Uruguay"
          AttractionQuantity = 100
     }
     CountryiesItem
     {
          Id = 2
          Name = "France"
          AttractionQuantity = 200
     }
     CountryiesItem
     {
          Id = 3
          Name = "China"
          AttractionQuantity = 250
     }
}
```

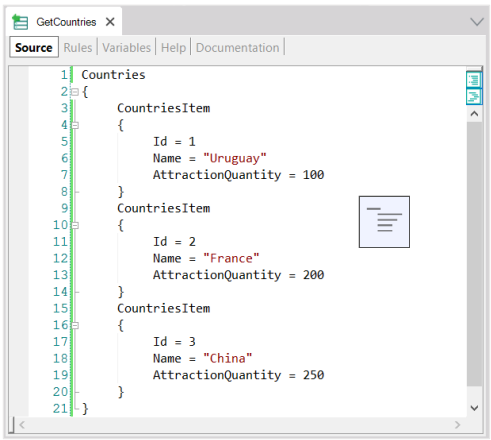
Countries

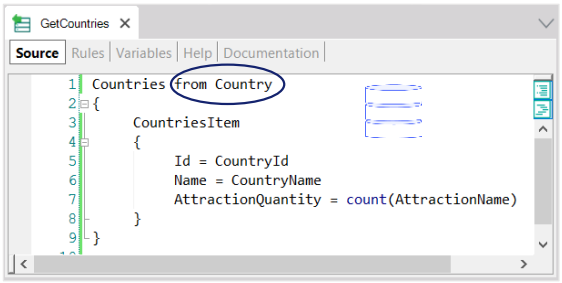
CountriesItem
Id: 1
Name: Uruguay
AttractionsQuantity: 100

CountriesItem
Id: 2
Name: France
AttractionsQuantity: 200

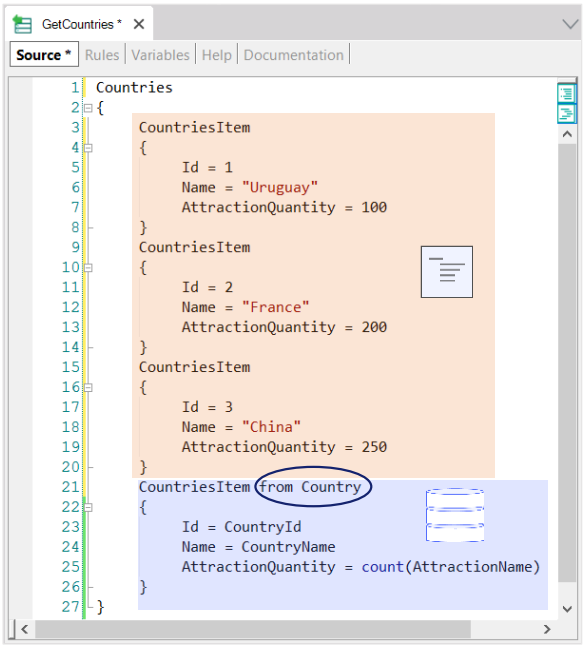
CountriesItem
Id: 3
Name: China
AttractionsQuantity: 250

O mesmo resultado teríamos obtido se carregássemos de maneira estática os dados no Source, ou seja, se o input não fosse retirado da base de dados, mas fosse codificado manualmente. Como vemos, por não haver transação base ou atributos, GeneXus não trará informações da base de dados.





INPUT MIXTO

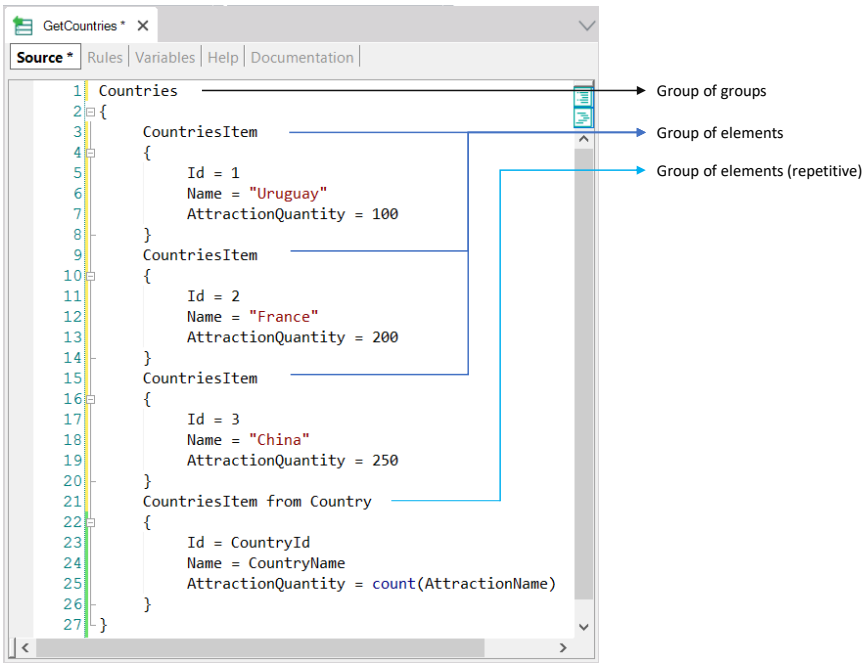


También podemos ter um input misto: uma parte codificada de forma estática e outra retirada da base de dados.

Neste exemplo, vemos no Source do Data Provider que apresentará na saída três itens da coleção carregados de maneira estática e, em seguida, mais N itens carregados da base de dados a partir dos registros da tabela Country.

Observe que tivemos que mover a cláusula from para que se aplique ao subgrupo CountriesItem final e não a todos. Pois, como vimos, esta parte estática, codificada manualmente por nós, não retira registros da base de dados.

- Groups
- Elements
- Variables




```
Countries
{
    CountriesItem
    {
        Id = 1
        Name = "Uruguay"
        AttractionQuantity = 100
    }
    CountriesItem
    {
        Id = 2
        Name = "France"
        AttractionQuantity = 200
    }
    CountriesItem
    {
        Id = 3
        Name = "China"
        AttractionQuantity = 250
    }
    CountriesItem from Country
    {
        Id = CountryId
        Name = CountryName
        AttractionQuantity = count(AttractionName)
    }
}
```

Veremos agora os principais componentes da linguagem do Source de um Data Provider.

Teremos grupos, elementos e também podemos utilizar variáveis.

Os elementos são análogos aos membros de um SDT. Se pensarmos na hierarquia como uma árvore, os grupos são os galhos e os elementos as folhas. Ou seja, os grupos são elementos compostos; podem ser compostos de outros grupos e/ou de elementos.

Os grupos poderão ser estáticos ou carregados de modo dinâmico, a estes chamamos de grupos repetitivos. Em nosso exemplo, os três primeiros grupos são estáticos, são carregados com dados fixos, enquanto o último é um grupo que terá tabela base associada e, portanto, produzirá N itens na saída, um para cada registro da tabela base considerado.

Um grupo com tabela base será equivalente a um comando for each.

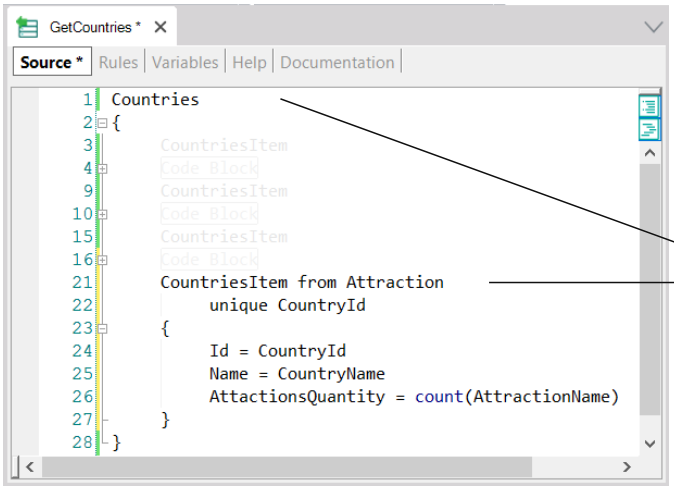
from *BaseTransaction*
[**skip** *expr1*] [**count** *expr2*]
[{[**order**] *order_attributesi* [**when** *condi*]}... | [**order none**] [**when** *condx*]]
[**using** *DataSelectorName*([[*parm1* [,*parm2* [, ...]])]
unique att_1, att_2...,att_n
[{**where** {*conditioni***when** *condi*} |
{*attribute* **IN** *DataSelectorName*([[*parm1* [,*parm2* [, ...]]} }...]

Os grupos permitem especificar transação base, embora, diferente do for each, aqui se especifica precedendo o nome da transação base ou nível com a palavra "from".

Observemos que neste exemplo o que fizemos foi, em vez de percorrer a tabela base COUNTRY, percorrer ATTRACTION, utilizando a cláusula unique de forma que, se houver muitas atrações de um país, seja levada em consideração apenas uma, e para essa se contem todas as demais atrações que tenham o mesmo país. Dessa forma, na saída serão listados apenas os países com atrações.

Vale exatamente o mesmo que todo o visto para o comando for each.

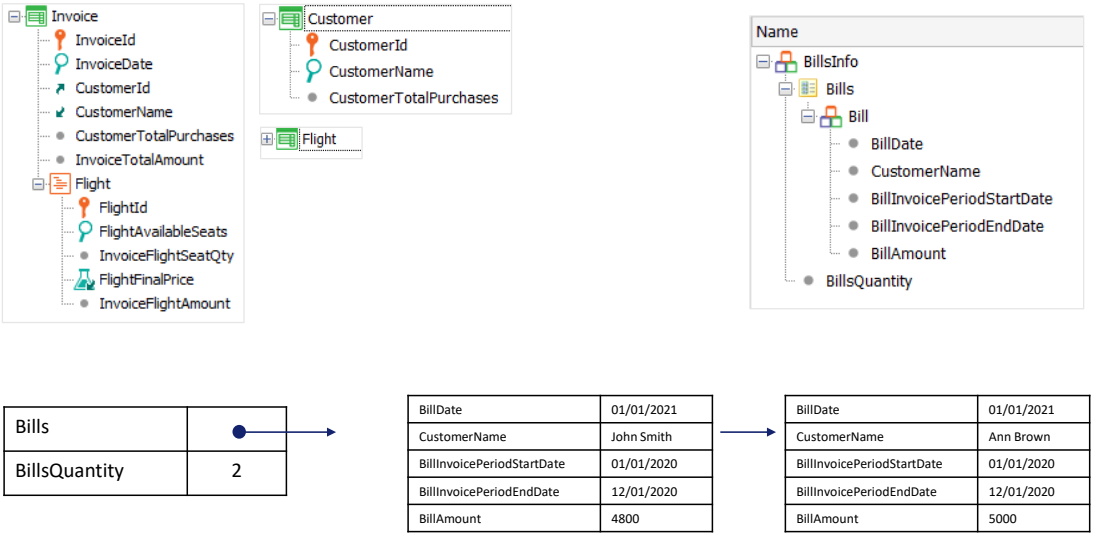
Data Provider language



from *BaseTransaction*
[**skip** *expr1*] [**count** *expr2*]
[{[**order**] *order_attributesi* [**when** *condi*]}... | [**order none**] [**when** *condx*]]
[**using** *DataSelectorName*([[*parm1* [,*parm2* [, ...]])]
unique att_1, att_2...,att_n
[{**where** {*conditioni***when** *condi*} |
{*attribute* **IN** *DataSelectorName*([[*parm1* [,*parm2* [, ...]]} }...]

Se os grupos estáticos não fossem requeridos, então seria o mesmo especificar as cláusulas no nível do subgrupo CountriesItem que do grupo pai, Countries, coleção de CountriesItem.

Neste caso, então, declarar as cláusulas no nível do grupo pai é equivalente a fazê-lo no nível do grupo filho.



Bills	
BillsQuantity	2

BillDate	01/01/2021
CustomerName	John Smith
BillInvoicePeriodStartDate	01/01/2020
BillInvoicePeriodEndDate	12/01/2020
BillAmount	4800

BillDate	01/01/2021
CustomerName	Ann Brown
BillInvoicePeriodStartDate	01/01/2020
BillInvoicePeriodEndDate	12/01/2020
BillAmount	5000

Vejamos agora este outro exemplo. Temos um Data Provider que devolverá como estrutura um SDT, que possui uma coleção de Bills e um elemento Quantity.

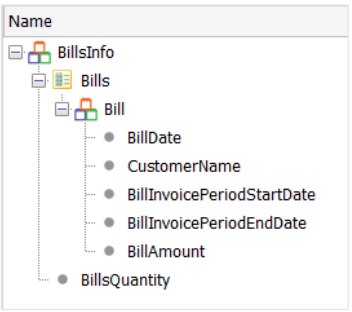
Em nossa aplicação temos a transação Invoice, que consiste em dois níveis e com os seguintes atributos. A transação Flight independente, e a transação Customer.

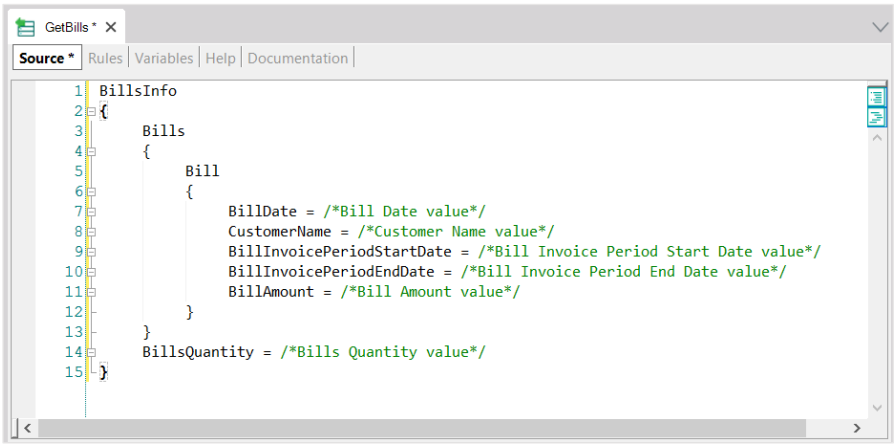
O que queremos é que a partir das faturas que foram geradas para cada cliente entre um par de datas dadas, seja gerado um recibo de pagamento, para o total de todas essas faturas. Pode ser que entre essas duas datas de faturamento nem todos os clientes tenham recibos a serem gerados, uma vez que não tenham faturas feitas nesse intervalo de datas. Na estrutura a ser devolvida é necessário saber quantos recibos são obtidos do cálculo, e por isso temos o membro BillsQuantity.

Se no intervalo de datas 01/01/2020 a 12/01/2020 somente houver faturas dos clientes John Smith e Ann Brown, a saída será conforme representa a imagem: uma variável SDT com dois membros: um do tipo coleção e o outro do tipo Numeric. A coleção terá dois itens, conforme mostrado.

Em outras palavras, o DataProvider devolverá uma estrutura com dois elementos: a coleção de recibos, por um lado, e a quantidade de elementos dessa coleção, por outro.

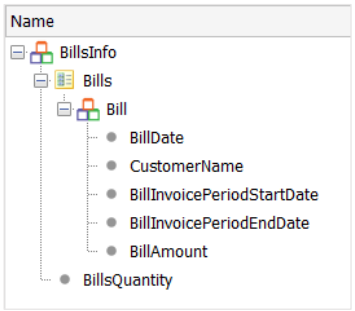
Output: BillsInfo
Collection: False



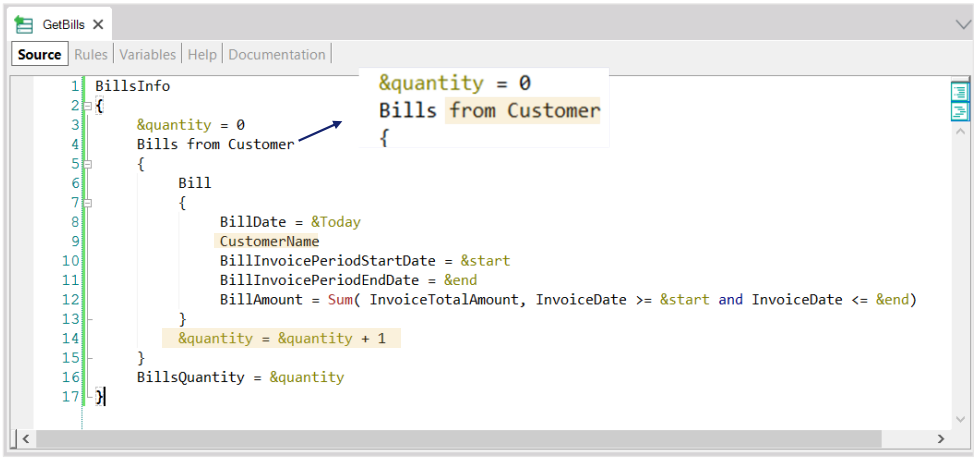


Se arrastamos para o Source do Data Provider o SDT, vemos como é inicializado, onde a propriedade Collection ficará com seu valor default "False". Isto é o que queremos, porque não vamos devolver uma coleção de BillsInfo, mas um só elemento desse tipo, que em particular conterá, entre outras coisas, uma coleção de Bills.

Output: BillsInfo
Collection: False



parm(in: &start, in: &end);



Programemos a regra parm para receber por parâmetro o intervalo de datas de faturamento.

E então para o grupo Bills, que representa a coleção, especificamos a transação base.

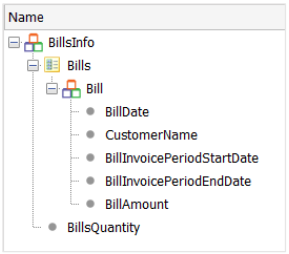
Por que colocamos CustomerName sem atribuir um valor a ele? Porque ele se chama igual ao atributo CustomerName e estamos navegando na tabela Customer. Portanto, podemos utilizar essa notação abreviada. Equivale a ter escrito: CustomerName igual a CustomerName, onde o da esquerda é o membro do SDT e o da direita é o atributo da tabela Customer.

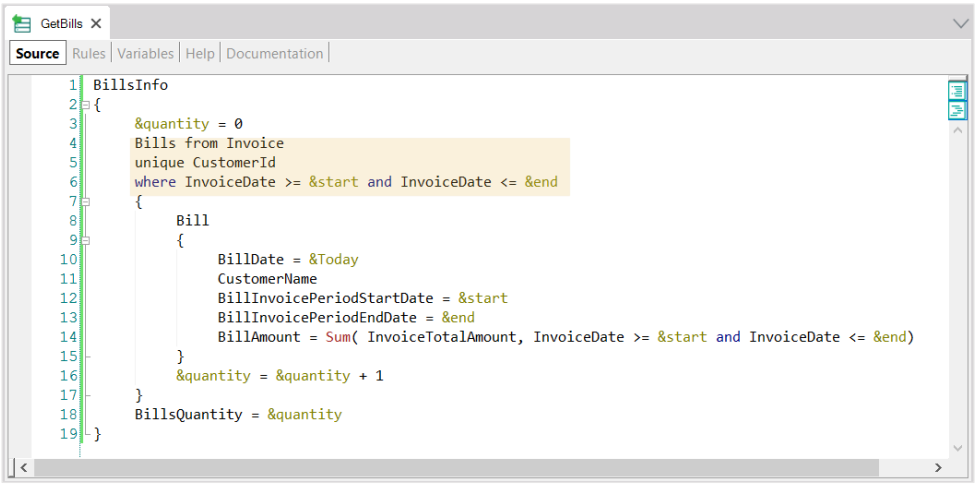
Observemos que o uso das variáveis é igual ao que fazemos em um for each.

Temos um problema: neste caso vamos devolver um item Bill na saída mesmo para clientes que não tenham faturas no intervalo recebido por parâmetro. Como faríamos para que isto não acontecesse?

Output: BillsInfo
Collection: False

parm(in: &start, in: &end);



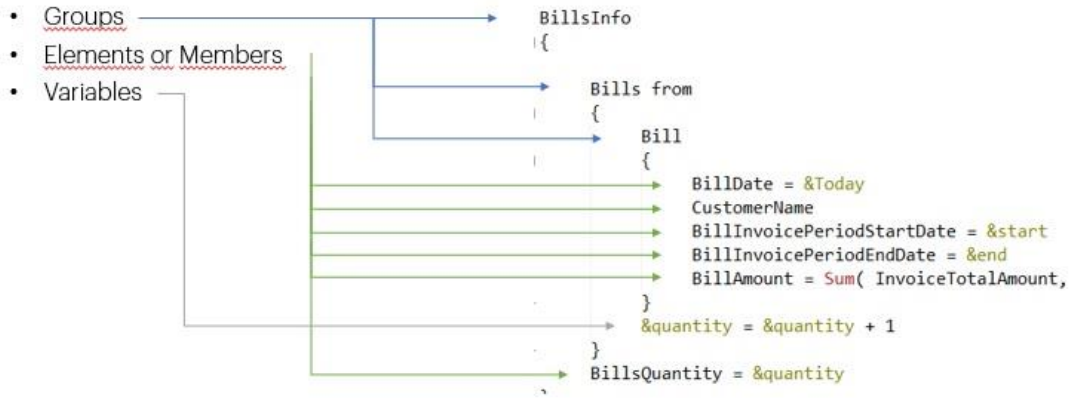


Uma maneira é alterando a transação base para Invoice e ficando com os registros de Invoice sem repetir o CustomerId e com datas no intervalo desejado.

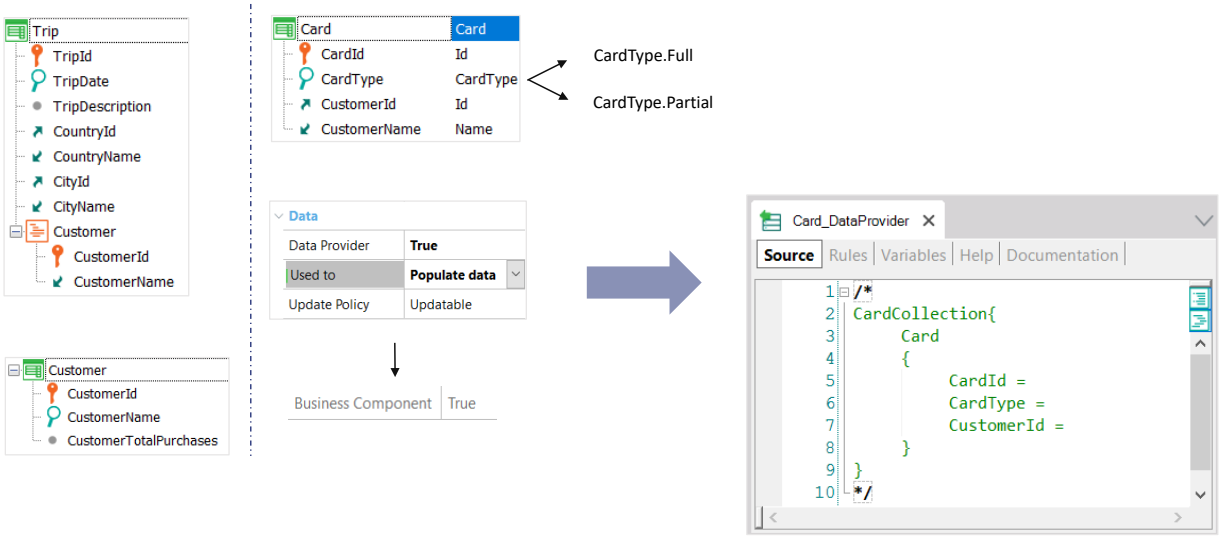
Depois no Sum interno contaremos os totais de todas as faturas do cliente que estão nesse mesmo intervalo de datas. Como dissemos, é idêntica à lógica do for each.

Basic components

- Groups
- Elements or Members
- Variables



Aqui, vemos mais uma vez os componentes básicos da linguagem de um Data Provider.



Agora, vejamos este outro exemplo. Queremos preencher com dados a tabela associada a uma nova transação chamada Card, utilizando seu Data Provider associado e a partir de dados de outras tabelas da base de dados.

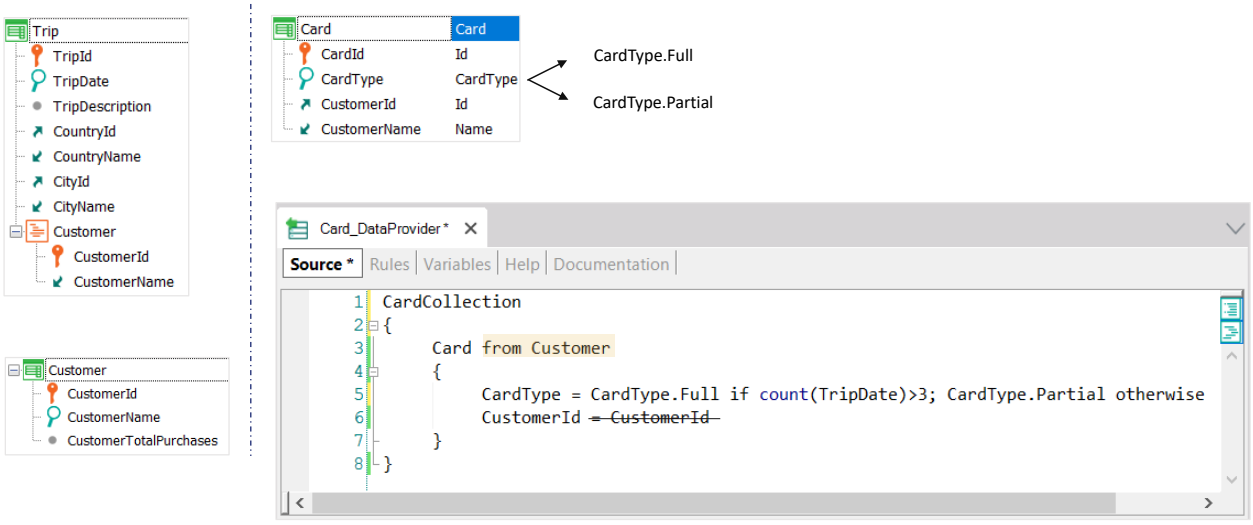
Temos a transação Customer para registrar os clientes da agência de viagens e Trip para registrar cada viagem ou excursão oferecida pela agência a uma cidade determinada. Temos um subnível com clientes registrados para a viagem.

Suponhamos que a agência de viagens decide que todos os clientes que tenham contratado mais de 3 viagens receberão um cartão especial de tipo "Full Services", que lhes permitirá usufruir de todos os serviços gratuitamente. E caso tenham contratado menos de 3 excursões, receberão o cartão de tipo "Partial Services".

Cada cartão tem um identificador que se autonumera, um cliente e um tipo de cartão, para o qual definimos um domínio enumerado que admite somente os valores "Full" ou "Partial".

Agora, queremos que a tabela associada à transação Card seja inicializada com a informação correta, então ativamos a propriedade Data Provider e deixamos o valor de Populate data para "Used to", de modo que criará automaticamente o DataProvider que vemos, e ativará a propriedade Business Component, para inserir os cartões que sejam devolvidos por esse Data Provider ao ser executado automaticamente na primeira execução.

Como declaramos o Source do Data Provider?



Criaremos um Business Component Card na coleção para cada cliente. Por isso para o grupo Card especificamos transação base Customer. Sabemos, portanto, que é um grupo com tabela base.

Removemos o elemento CardId do grupo Card, pois o domínio Id do atributo CardId da transação é autonumerado.

A seguir, observemos como carregamos o valor do elemento CardType utilizando uma fórmula inline condicional. Assumirá o valor do enumerado CardType.Full, desde que o resultado da execução da fórmula inline count(TripDate) seja maior que 3; caso contrário, será atribuído o valor CardType.Partial.

Essa fórmula contará os registros da tabela Trip, filtrando por CustomerId.

E então ao elemento CustomerId, que corresponderá ao Business Component, é atribuído o valor do atributo CustomerId da tabela base do grupo Customer. Podemos utilizar a notação abreviada e remover a atribuição.
Aqui, portanto, vemos um exemplo em que o Data Provider devolve uma coleção de Business components cujos dados são obtidos de outra tabela.

É idêntico, mais uma vez, ao caso de um For each.

Para continuar pesquisando sobre este tema, convidamos a visitar nossa wiki.

GeneXus™

training.genexus.com
wiki.genexus.com