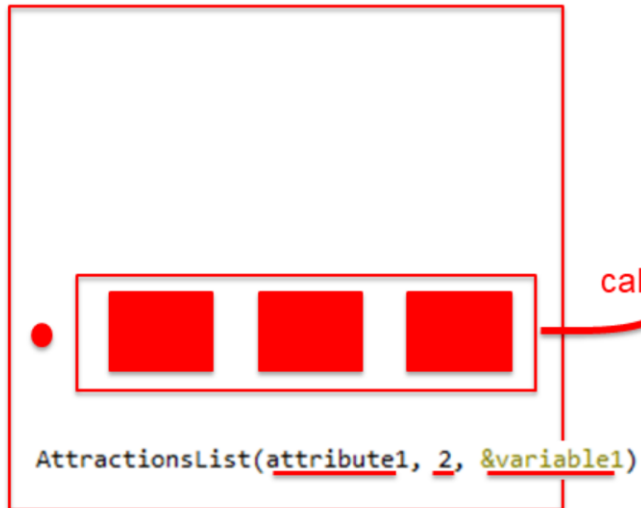


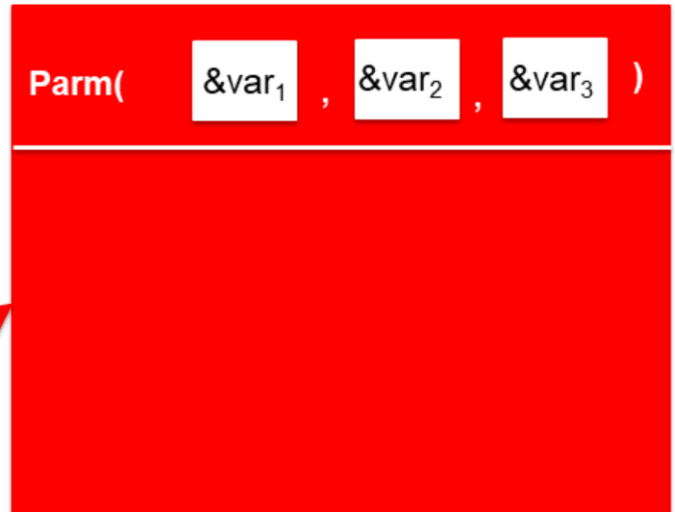
Quando o objeto invocado retorna um valor

GeneXus™ 16

Object A



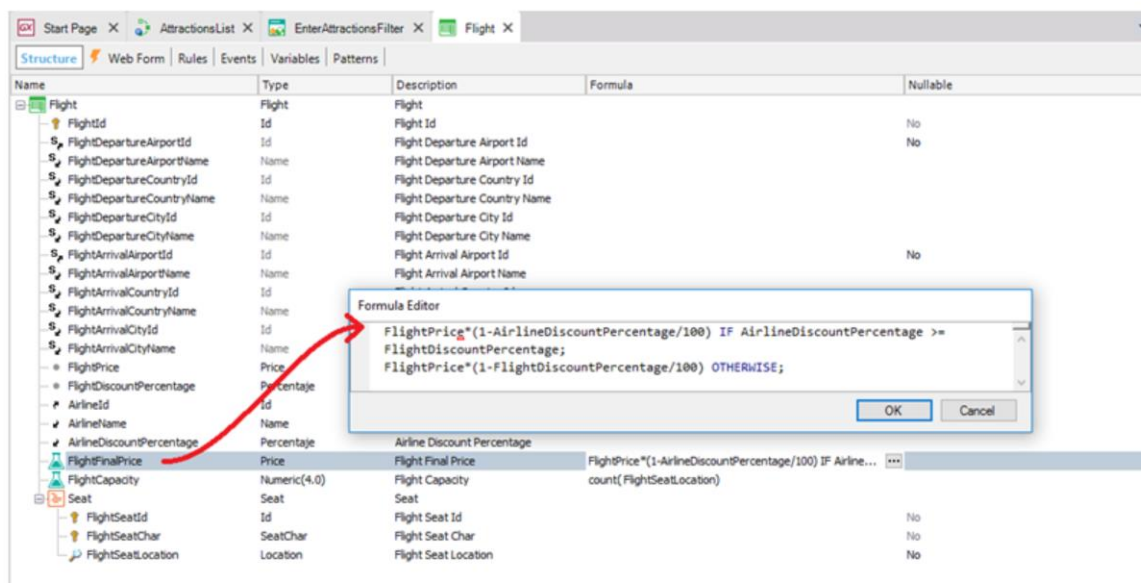
Object B



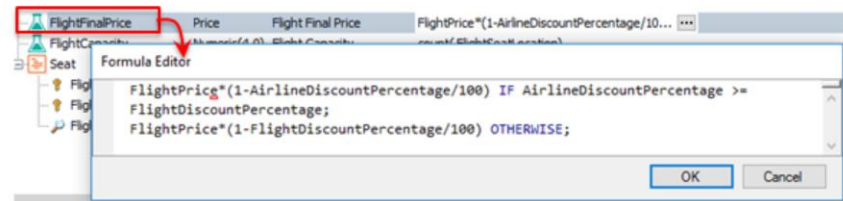
Já vimos como declarar parâmetros em um objeto para habilitá-lo a receber dados de outro objeto e executar as ações correspondentes de acordo com esses dados. Para este fim, usamos regras Parm e variáveis. Os exemplos que vimos envolveram parâmetros de entrada; isso quer dizer, parâmetros apenas recebidos pelo objeto.

Desta forma, se o objeto B tem uma regra Parm declarada com três variáveis, para invocar o objeto B, qualquer objeto terá que enviar três valores que, como vimos, podem ser salvos em atributos, ser uma expressão (como no caso de um valor fixo) ou ser salvos em variáveis.

Agora vamos ver o que acontece quando o objeto B deve **retornar** um valor para o objeto chamador, quando ele finaliza a execução.



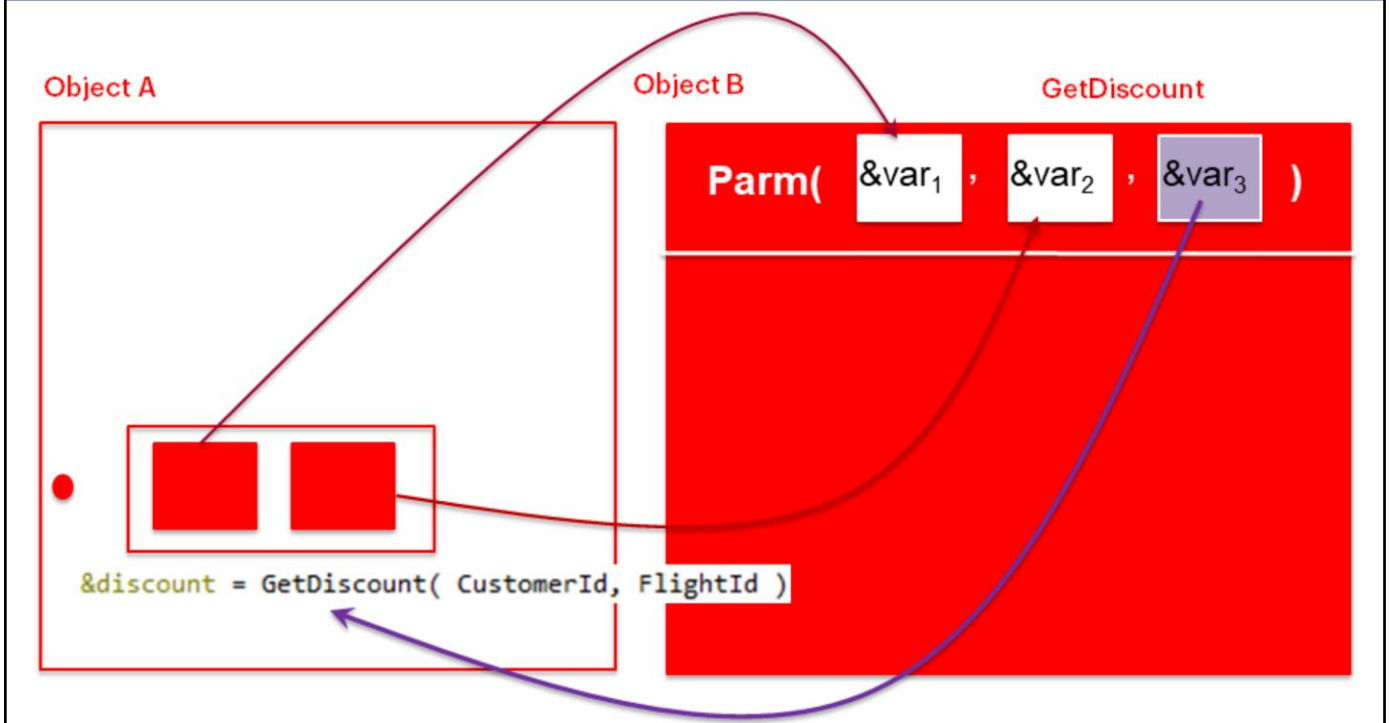
Na transação Flight tínhamos uma fórmula que calculava o preço de um voo de acordo com a porcentagem de desconto oferecido pela companhia aérea e a porcentagem indicada para o voo em si. É selecionado o maior desconto e aplicado.



Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4,0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
CustomerLastName	Character(20)	Customer Last Name		
InvoiceTotalAmount	Price	Invoice Total Amount		No
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightPrice	Price	Flight Price		
FlightArrivalCountryName	Name	Flight Arrival Country Name		
FlightArrivalCityName	Name	Flight Arrival City Name		
InvoiceFlightDiscount	Percentage	Invoice Flight Discount		No
InvoiceFlightAmount	Price	Invoice Flight Amount		No

Suponha que estamos criando uma transação para gravar as faturas emitidas para os clientes quando eles compram bilhetes de avião.

Além disso, suponha que o desconto é um cálculo mais complexo que implica não só o voo, mas também alguma condição relacionada com o cliente que está comprando um bilhete de avião. Por exemplo, o número de bilhetes que ele adquiriu no passado, se ele é um cliente recorrente, e se um destino é oferecido com um desconto. A porcentagem de desconto é determinada de acordo com estas condições mais complexas.



Para casos como este, nós podemos precisar implementar um procedimento que faz estes cálculos e retorna o valor resultante para o chamador. Por exemplo, podemos chamar este procedimento `GetDiscount`.

O procedimento terá de receber o cliente e voo como parâmetros de entrada. Ele irá retornar o desconto resultante.

A primeira questão é como esse resultado é recebido pelo objeto que precisa do resultado do procedimento. Tem que ser considerado como uma função que é chamada para executar uma ação com o resultado retornado.

Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4,0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
CustomerLastName	Character(20)	Customer Last Name		
InvoiceTotalAmount	Price	Invoice Total Amount		No
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightPrice	Price	Flight Price		
FlightArrivalCountryName	Name	Flight Arrival Country Name		
InvoiceFlightDiscount	Percentage	Invoice Flight Discount	GetDiscount(CustomerId, FlightId)	

```

1 InvoiceFlightDiscount = GetDiscount( CustomerId, FlightId );
  &discount = GetDiscount( CustomerId, FlightId );
  msg( "Free Flight" ) if GetDiscount( CustomerId, FlightId ) = 100;
  
```

Uma possibilidade é atribuir o resultado para um atributo. Por exemplo, podemos definir o atributo FlightDiscount na estrutura de transação Invoice como uma fórmula que é calculada pela invocação GetDiscount

Desta forma, a fórmula será avaliada em cada objeto onde o atributo InvoiceFlightDiscount é mencionado. O procedimento GetDiscount será invocado e executado, e quando finaliza a execução, será mostrado o resultado retornado como o valor do atributo fórmula.

Se não queremos definir esse atributo como uma fórmula, mas prefiro que seja um atributo armazenado na tabela correspondente e tê-lo armazenado com o resultado do procedimento apenas quando a transação é executada, digitamos nas regras a primeira atribuição que vimos acima.

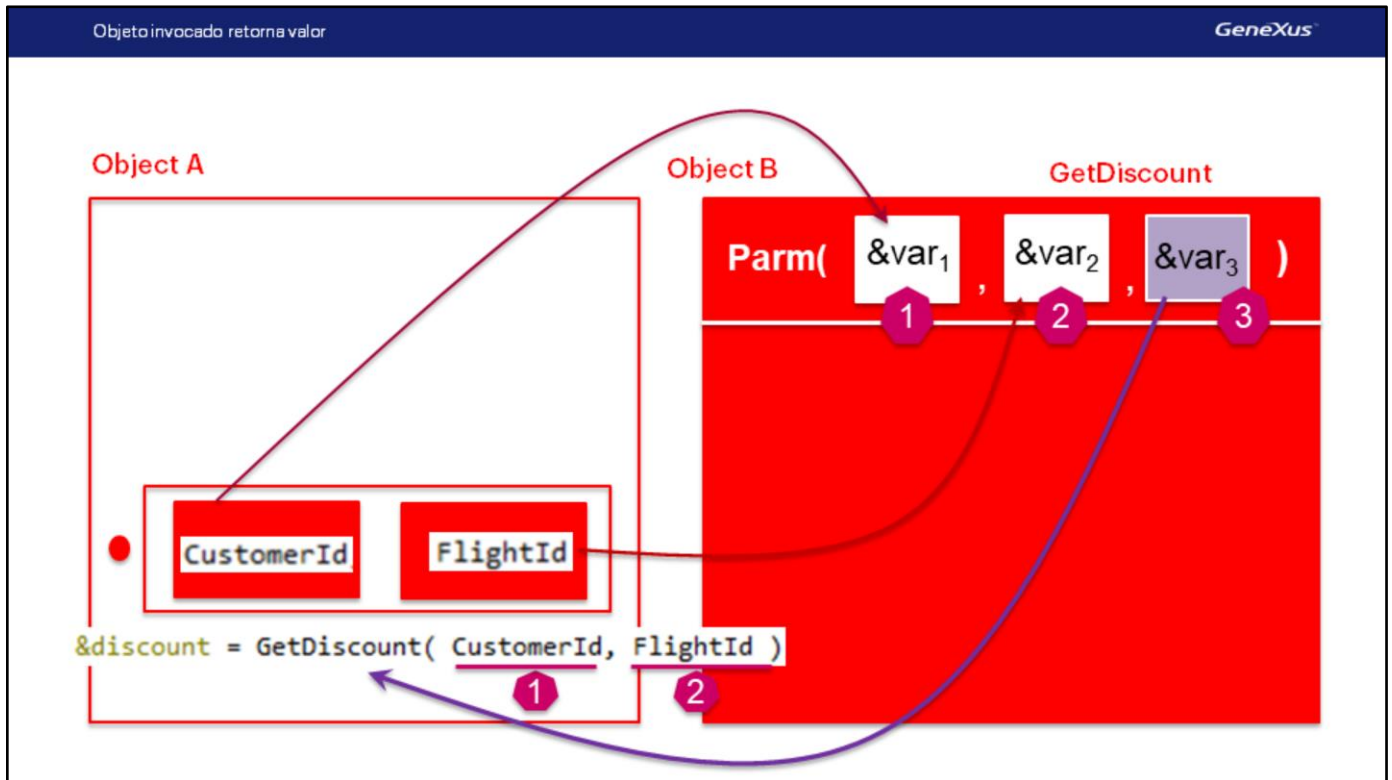
Além disso, o resultado da execução do procedimento poderia ser atribuído a uma variável. Também, pode não ser atribuído, mas usado em uma expressão em vez disso. Por exemplo, para condicionar o disparo de uma regra.

Ou das instruções em um procedimento ou um evento:

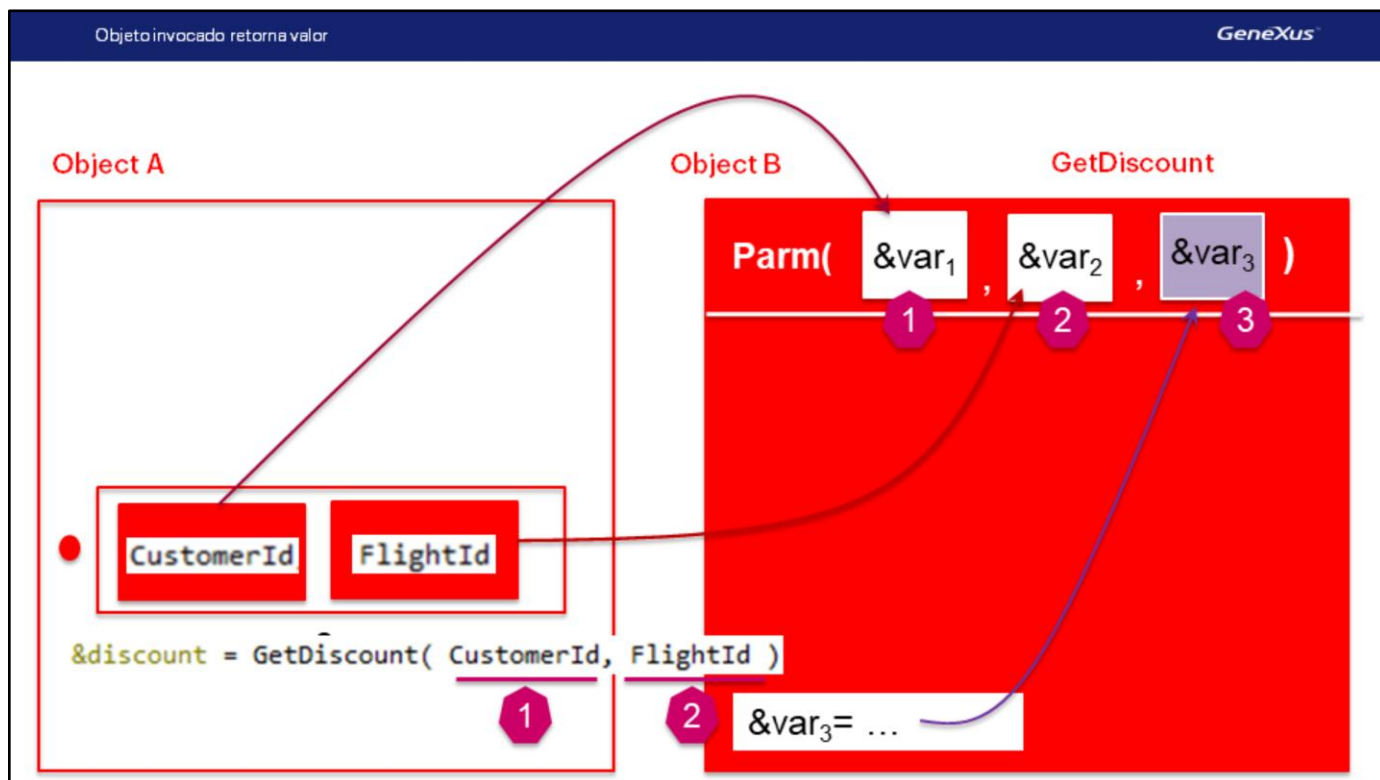
```
If GetDiscount( CustomerId, FlightId ) > 10
```

```
...
Endif
```

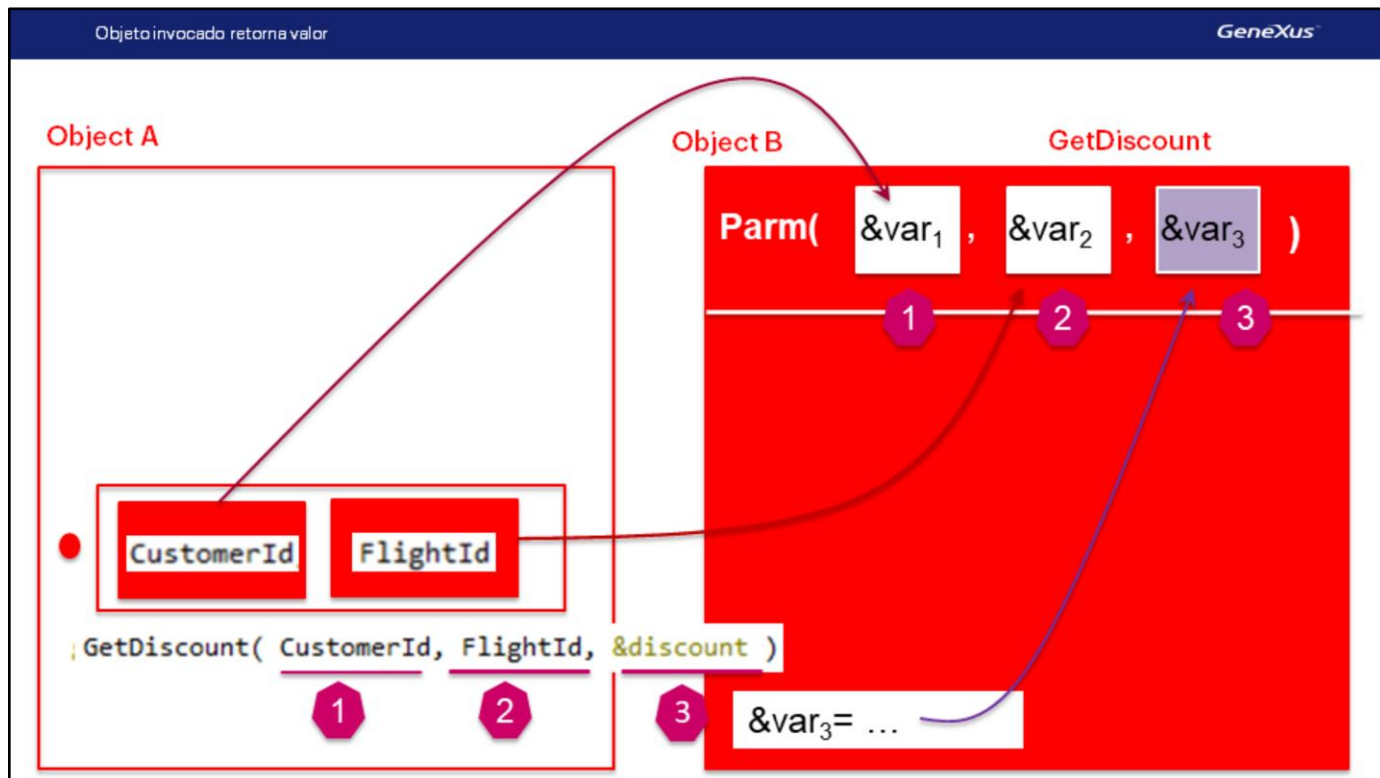
Não falaremos sobre como implementar o procedimento GetDiscount, porque não é relevante para o que estamos estudando agora. No entanto, temos que ver como a **regra parm** consta no objeto chamado quando a sintaxe de chamada pressupõe que o objeto retorna um valor, como nos exemplos que mencionamos.



Na seção de regras do procedimento `GetDiscount` temos que declarar a regra `Parm` com o número de parâmetros descritos na chamada... mais um no final...



.. terá de ser uma variável cujo valor é carregado no código do objeto (e no nosso caso, no Source do procedimento). O valor tomado pela variável quando o código finaliza a execução será o valor retornado para o objeto que o chamou.



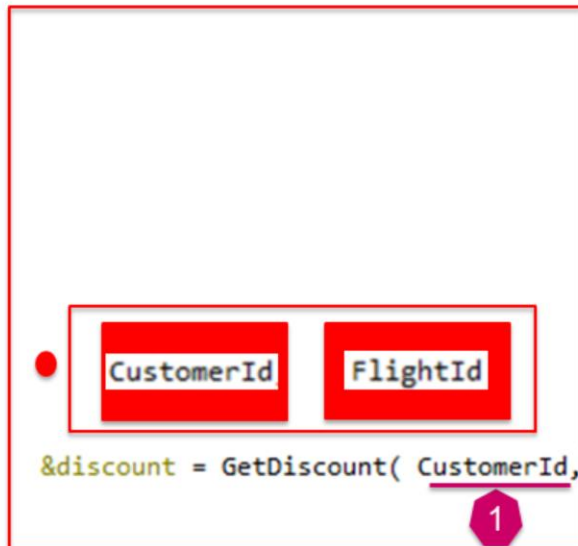
Para este mesmo Objeto B poderíamos ter feito a invocação como fizemos antes: onde os dois primeiros parâmetros são parâmetros de entrada e o último é um parâmetro de saída.

O único problema é que desta forma a invocação não indica claramente que `&discount` retornará carregado. Em outras palavras, não é claro que o objeto invocado retornará um valor. Nestes casos, usar a sintaxe de invocação que indica claramente é recomendado.

Receber em variável ou em atributo

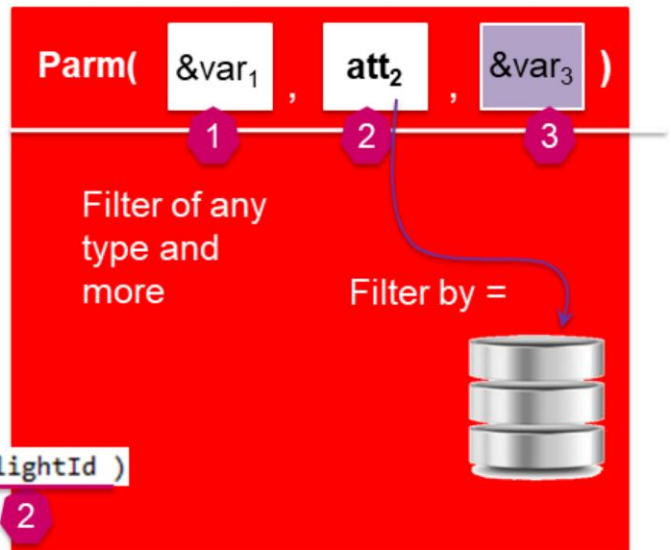
Por último, vamos analisar o caso em que um parâmetro da regra Parm é um atributo em vez de uma variável.

Object A



Object B

GetDiscount



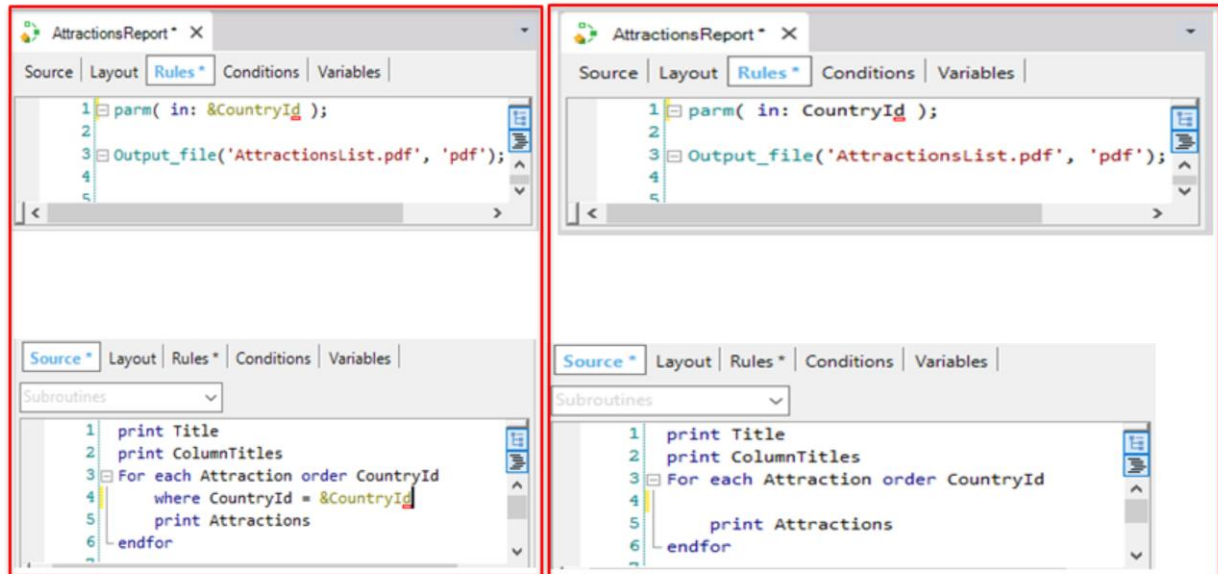
Qual é a diferença entre usar uma variável ou um atributo na regra **Parm** do objeto invocado?

Se o valor é recebido em uma **variável**, ele pode ser livremente usado na programação: pode ser usado como uma condição de filtragem para filtros como igualdade, superior a, superior a ou igual a, e assim por diante... também pode ser usado para uma operação aritmética, ou o que for necessário. É um espaço na memória com um nome que usamos dentro do objeto através de instruções explícitas, para fazer o que queremos.

Se, por outro lado, o valor é recebido em um atributo, é implícito, fixo e determinado. Recebemos valores em um atributo, quando acessamos o banco de dados de dentro do objeto. Em particular, uma tabela em cuja tabela estendida este atributo é armazenado. Então, quando um valor é recebido nesse atributo através de um parâmetro, será aplicado um filtro de igualdade. Serão considerados somente os registros que têm esse valor para o atributo.

Vejamos isto com um exemplo.

Exemplo: receber em variável vs receber em atributo



Fazemos uma cópia do procedimento AttractionsList com o nome de AttractionsReport.

Lembre-se que o procedimento no qual é baseado usa uma variável como um parâmetro: &CountryId. Ele é usado para filtrar as atrações na tabela Attraction pelo país.

Como podemos ver, está implementando um filtro de igualdade: ele irá listar apenas aqueles atrações cujo CountryId corresponde ao valor da variável &CountryId recebida em um parâmetro.

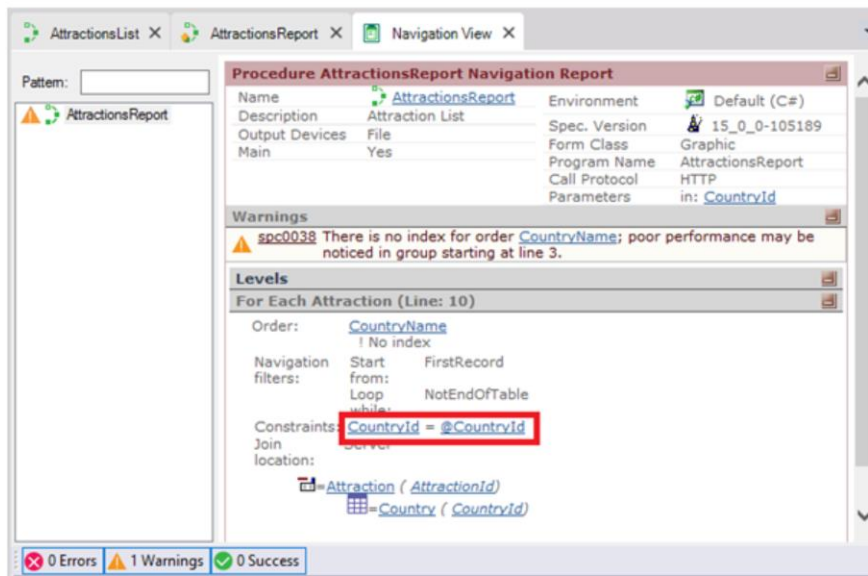
Poderíamos ter implementado exatamente o mesmo sem explicitamente indicar esse filtro.

Como? Ao receber o valor diretamente no atributo CountryId.

Quando recebermos o valor de um atributo na regra Parm, GeneXus usa um filtro de igualdade; isso quer dizer, somente os registros que têm o mesmo identificador de país são acessados.

Se olharmos para a lista de navegação deste objeto...

Lista de navegação



... vemos que o filtro é aplicado mesmo se a cláusula Where não está escrita.

É interessante notar que como o comando For Each está sendo executado ordenado por CountryName, a tabela inteira tem que ser percorrida através do filtro por valores de CountryId correspondente ao parâmetro.

Navigation List

The screenshot displays the GeneXus IDE interface with three panes: Source, Navigation View, and Procedure AttractionsReport Navigation Report.

Source Pane: Shows a code snippet for a report:

```
1 print Title
2 print ColumnTitles
3 For each Attraction order CountryId
4
5     print Attractions
6 endfor
7
8
9
10
```

Navigation View Pane: Shows a tree view with a red box highlighting the 'AttractionsReport' node.

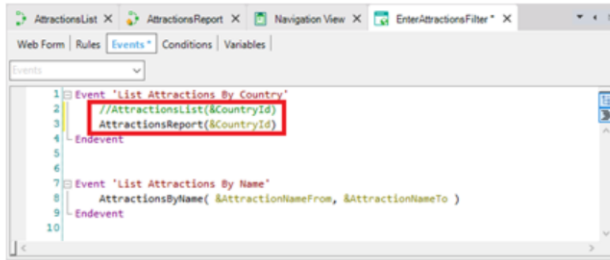
Procedure AttractionsReport Navigation Report Pane: Shows the configuration for the report. The 'Levels' section is highlighted with a red box:

Levels	
For Each Attraction (Line: 10)	
Order:	CountryId
Index:	AttractionId
Navigation	Start: CountryId = @CountryId
filters:	from: CountryId = @CountryId
Loop	while: CountryId = @CountryId
Join	Server
location:	Attraction (AttractionId)
	Country (CountryId)

The status bar at the bottom indicates 0 Errors, 0 Warnings, and 1 Success.

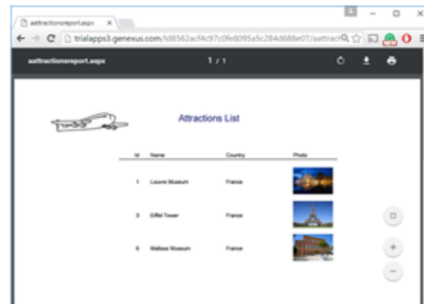
Por outro lado, se nós ordenamos pelo atributo usado para filtrar, vemos na lista de navegação que a tabela inteira não é mais percorrida.

DEMO



```
1 Event 'List Attractions By Country'
2 //AttractionsList(&CountryId)
3 AttractionsReport(&CountryId)
4 EndEvent
7 Event 'List Attractions By Name'
8 AttractionsByName( &AttractionNameFrom, &AttractionNameTo )
9 EndEvent
10
```

- F5 → executar Web panel, escolher France



Vamos substituir a invocação no web panel que chamou AttractionList com uma invocação a este outro procedimento: O que vem após uma barra dupla é considerado um comentário; isso quer dizer, ele não é interpretado por GeneXus como uma instrução. Desta forma, comentamos primeira invocação e digitamos uma nova:

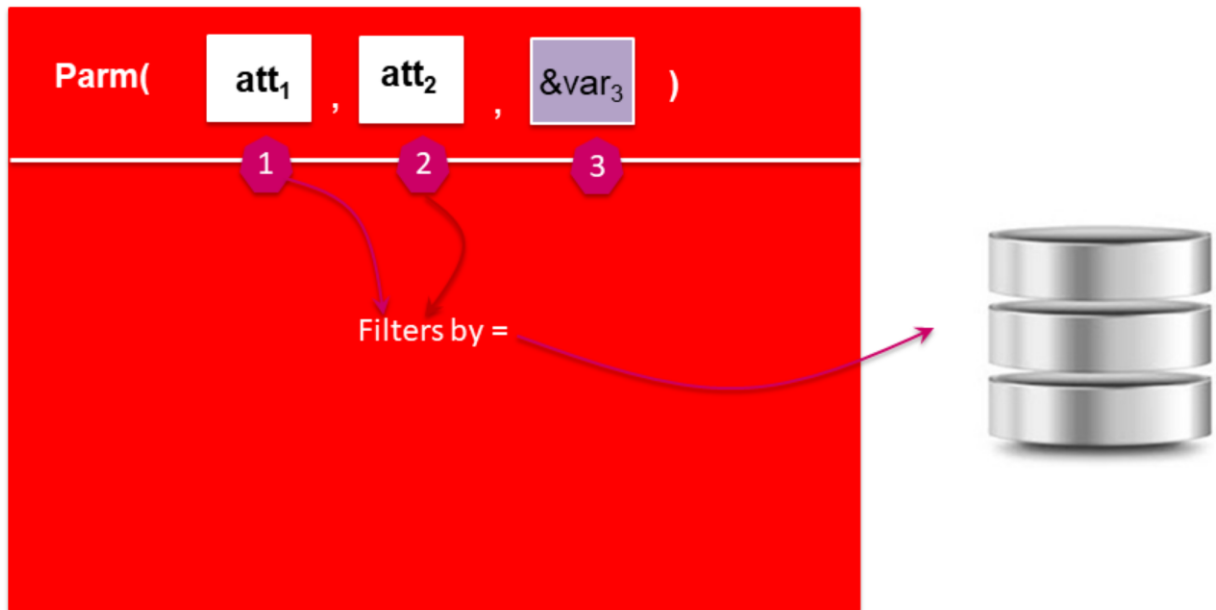
Executamos...

Selecionamos França e listamos...

Como podemos ver, a lista contém as atrações turísticas da França.

Por último, carregamos tudo para GeneXus Server.

Object B



Se recebemos mais de um valor usando atributos para recebê-los, apenas os registros que têm o mesmo valor de cada atributo recebido seriam acessados.

Além disso, não podemos mudar esses valores de atributo.


```
parm( inout: &var1, in: &var2, out: &var3 );
```

Object B

Parm(&var₁ , &var₂ , &var₃)

1

2

3

Filters by... > >= < <= Like, etc.

&var₁ = ...

Se nosso objetivo não é receber valores para aplicar um filtro de igualdade, a solução será a receber valores em variáveis em vez de usar atributos. Além disso, eles podem ser livremente usados na programação, por exemplo, para atribuir outros valores a eles, se necessário.

Neste sentido, devemos também dizer que não só a última variável da regra Parm pode ser usada como um parâmetro de saída; isso quer dizer, será retornado ao chamador. Todas as variáveis podem ser de entrada, saída ou entrada/saída. No exemplo, &variável 1, que como mencionado altera seu valor dentro do código do objeto, poderia ser uma variável de saída ou de entrada/saída.

Isso pode ser indicado digitando-se "in", "inout" ou "out" antes dos nomes das variáveis, como podemos ver aqui.

Comunicação entre objetos é essencial para qualquer aplicação GeneXus, pois permite que um objeto inicie a execução de outro objeto e envie ou receba informações de, e para ele.



Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications