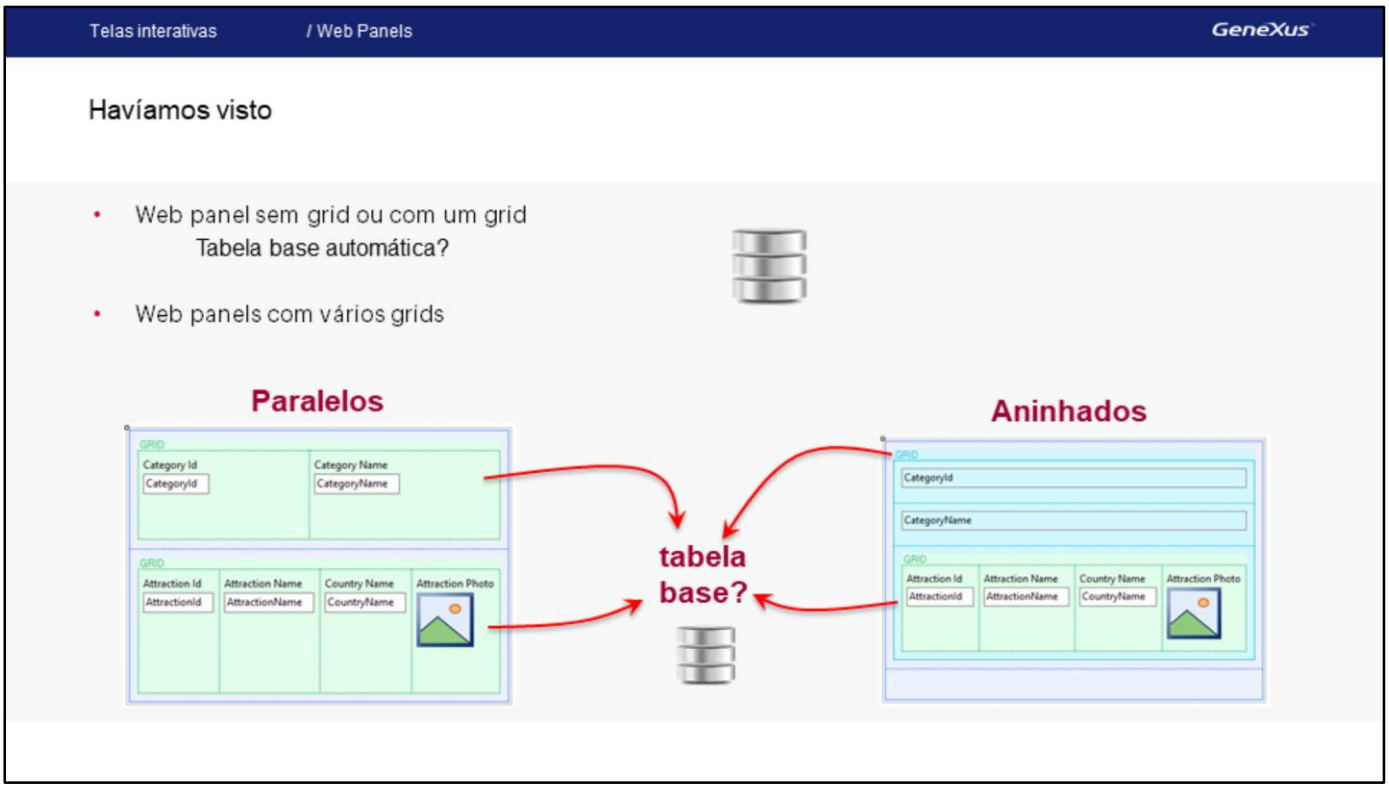
Telas interativas

Mais sobre webpanels

GeneXus 16

Tabelas base

Anteriormente vimos que simplesmente colocando atributos em um grid, GeneXus entende que deve ir ao banco de dados para navegar na tabela correspondente para recuperar os dados, de forma análoga a um comando For Each.
Vamos ver os diferentes casos em que o GeneXus pode determinar uma tabela base a ser percorrida.



Tínhamos visto o caso do Web Panel com atributos "soltos" no form, sem grid. Também o de um Web Panel com um grid com atributos e também sem atributos.
E havíamos deixado colocada a pergunta: quando o web panel não tem nenhum grid ou tem um, onde exatamente busca GeneXus a aparição de atributos para determinar se associa uma tabela base implícita ou não ao Web Panel? E, para aparecer atributos nesses lugares, quais critérios devem atender?

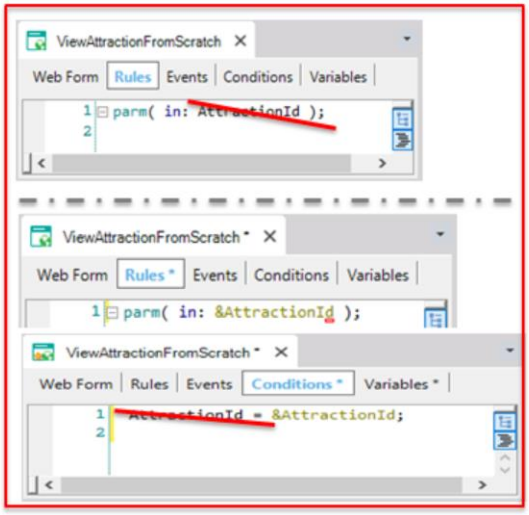
Por outro lado, vimos que se podem incluir vários grids em um Web Panel, tanto de forma paralela como aninhada. E, novamente, esses grids poderão cada um ter ou não, tabela base associada.

Web Panel sem grid

- Não há grid → não há propriedade Base Transaction

- Atributos no **form** (visíveis ou ocultos)
- Atributos em **eventos** (fora de todos os For each)

Tabela base: mínima tabela estendida

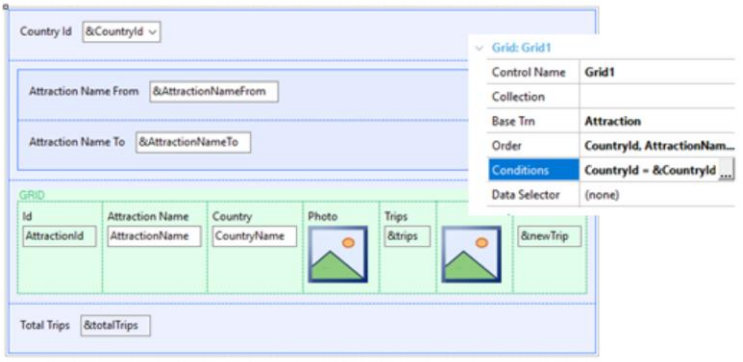


Quando o web panel não tem nenhum grid, mas possui atributos quer seja no form (tanto visíveis como ocultos) ou em eventos (sempre e quando estejam fora de um comando for each), o web panel terá uma tabela base.
Como é determinado? Da mesma maneira que um for each ao qual não especificamos a transação base: escolhendo a mínima tabela estendida que contenha todos os atributos mencionados onde indicamos.

Os atributos que são especificados na regra parm ou nas Condições gerais (ou seja, a guia Conditions) não serão levados em consideração ao determinar a tabela base. Eles serão usados como filtros APÓS a tabela base ter sido determinada.

Web Panel com um grid

- Existe um grid → tabela base do web panel = tabela base do grid



- Propriedade **Base Transaction**
- Atributos no **form (grid e fora do grid**, visíveis ou ocultos)
- **Order** do grid
- **Conditions** do grid
- **Data Selector** do grid
- Atributos em **eventos** (fora de todos os For each)

Tabela base: a associada à **Transação Base** se existe definida
mínima tabela estendida caso contrário

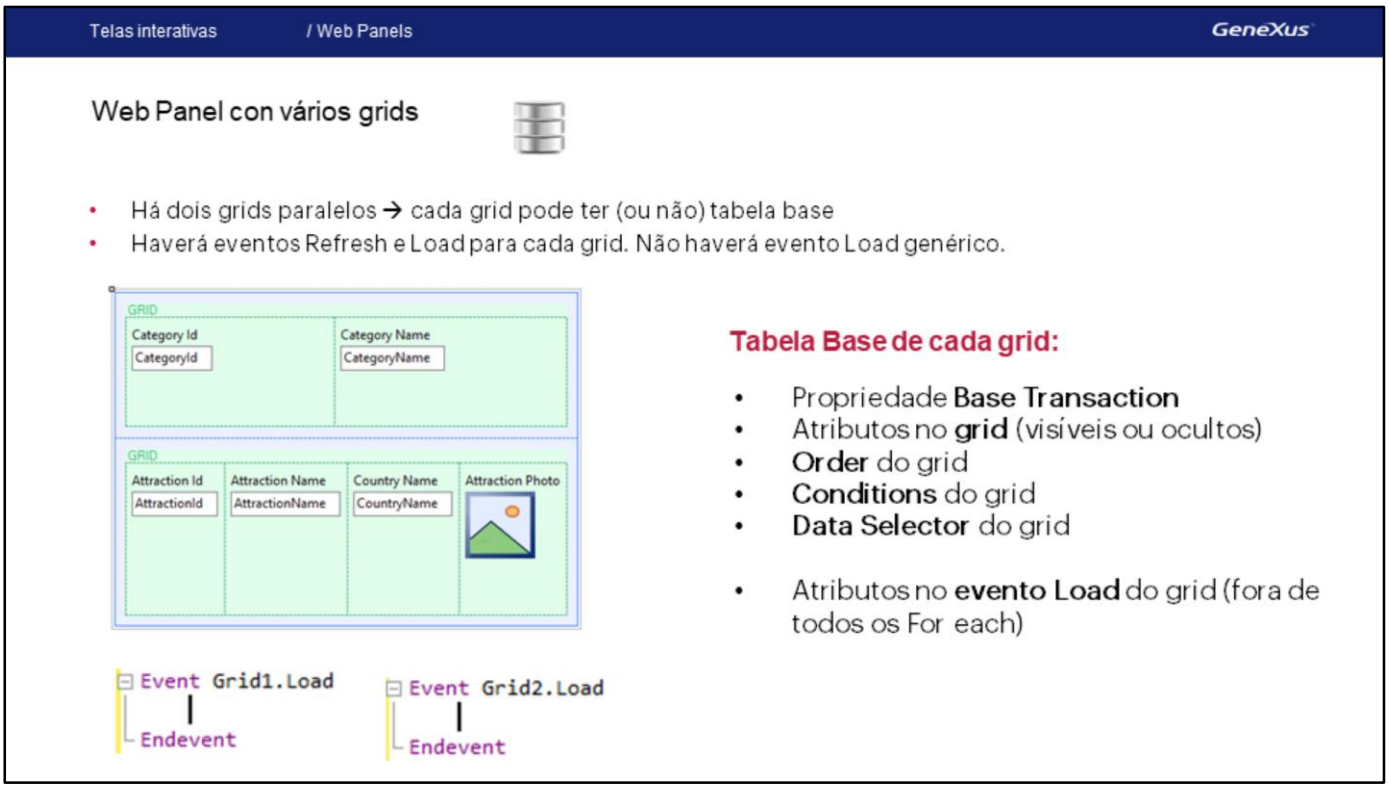
Quando o web panel possui um grid, então para determinar a tabela base, GeneXus observa o que está indicado acima.

Se o grid tem uma Transação Base configurada, sua tabela base será a tabela base do grid/web panel. Os atributos mencionados nos lugares indicados deverão pertencer à sua tabela estendida (exatamente como no caso de um for each).

Se o grid não tiver uma Transação Base configurada, então a tabela base é determinada como a correspondente à mínima tabela estendida que contém todos os atributos dos lugares mencionados.

NÃO participam os atributos das Condições gerais (os da guia Conditions) e nem os da regra Parm. Nem os atributos que estão dentro de um for each.

Observe-se que ao grid pode ser aplicado um DataSelector para filtrar e ordenar suas informações, assim como fizemos no For each (e também em um Grupo de Data Provider).



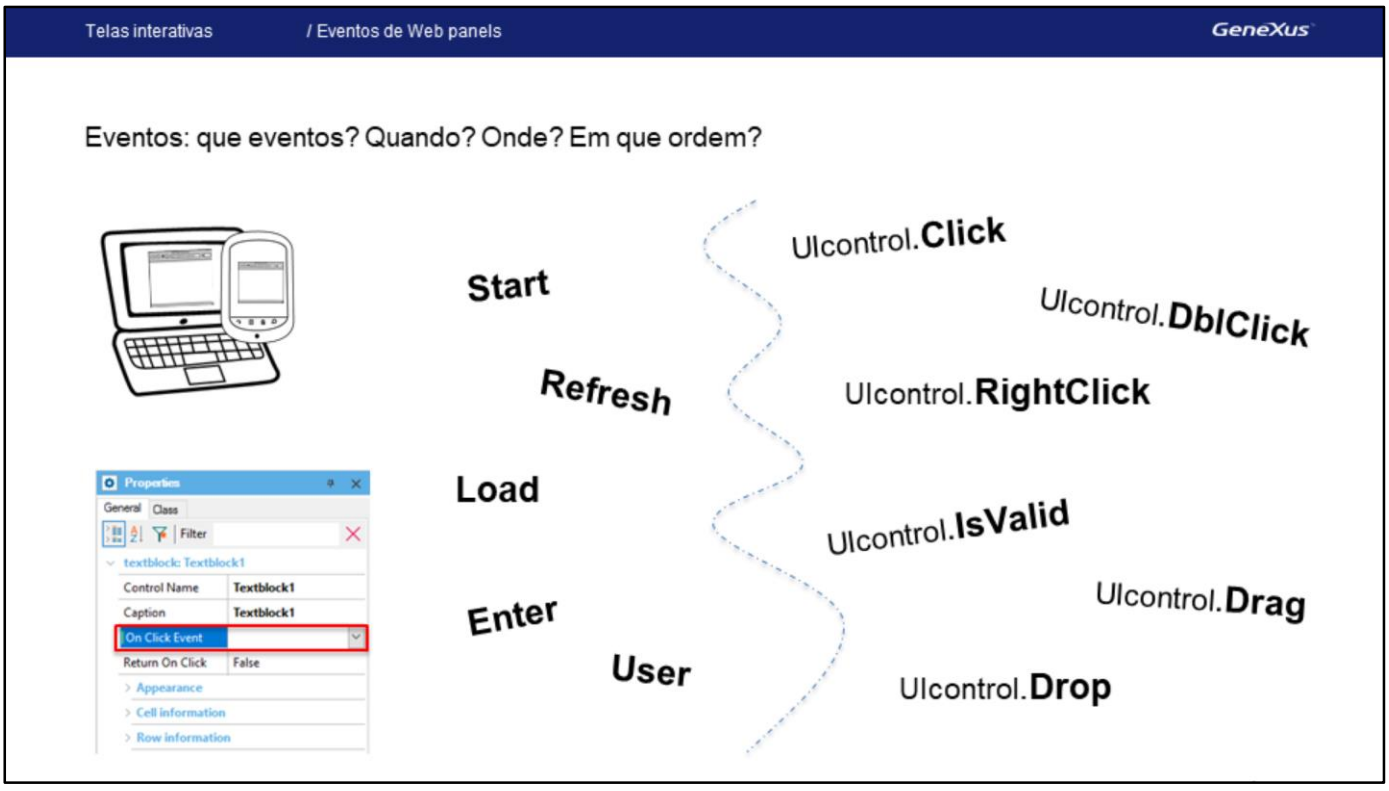
Quando há mais de um grid em um web panel, **a tabela base de cada grid** é determinada **considerando exclusivamente os atributos mencionados acima.**

Cada grid terá seu evento Load e a sintaxe se mostra acima. Não pode ser usado o evento Load genérico, pois não se saberia de qual grid se trata.

Ao contrário do caso de um único grid, os atributos que estão fora de for eachs em qualquer evento que não sejam "seu" Load, não participarão da determinação das tabelas base. Mas deverão obedecer que pertençam à tabela estendida de algum dos grids. Caso contrário, GeneXus irá advertir na lista de navegação.

Se houver atributos na parte fixa, a tabela base do primeiro grid no form é determinada levando em consideração os atributos soltos, e os demais grids sem levá-los em conta.
Se quiser ver mais sobre este caso especial, acesse nossa wiki: http://wiki.genexus.com/commwiki/servlet/wiki?6105,Determining+the+Base+Table+for+Each+Grid+in+a+Web+Panel

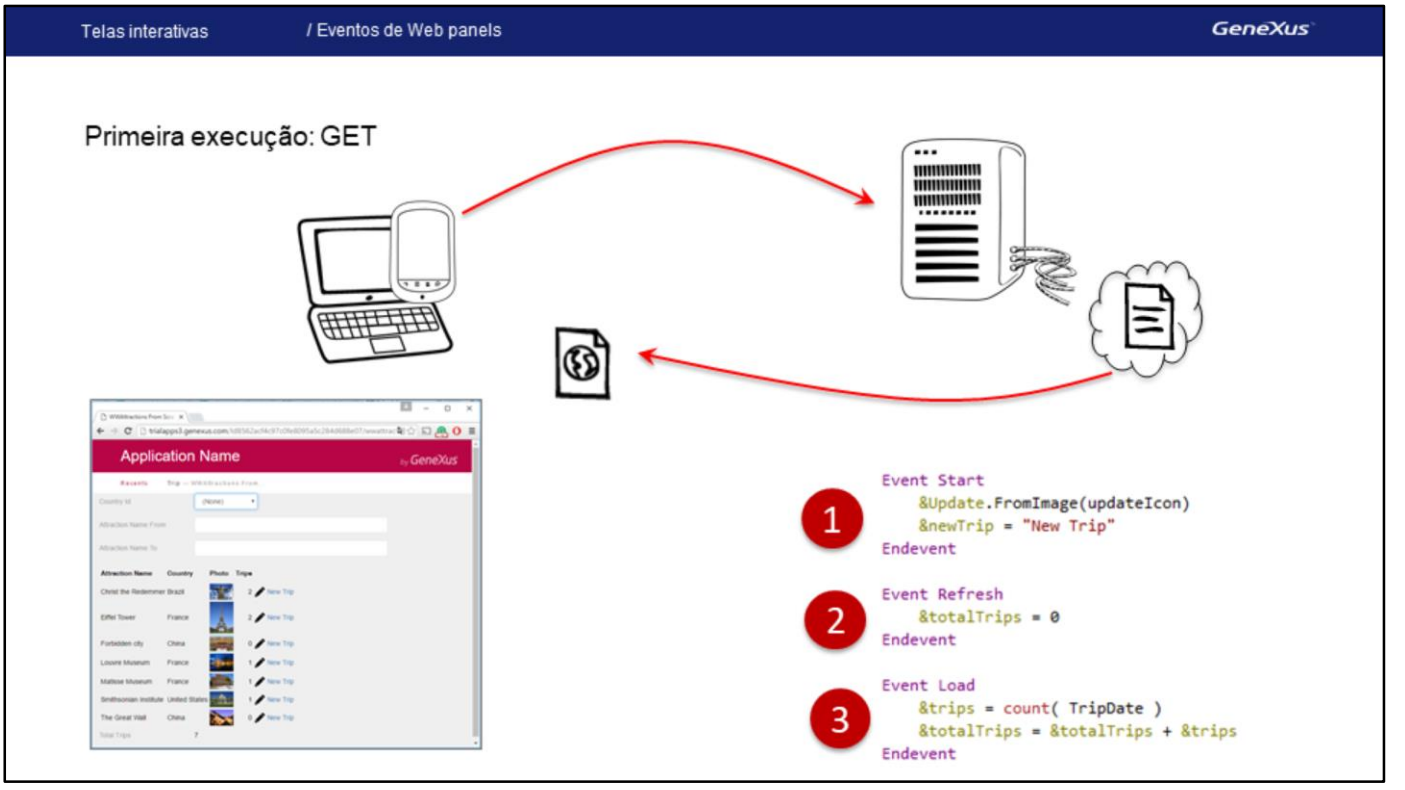
Eventos



Já vimos e programamos diferentes eventos nos Web panels (por exemplo, o clique, associado a controles incluídos no form, mas, dependendo do controle, também podem ser programados o duplo clique, botão direito, etc.).

Também apresentamos e usamos os eventos Start, Refresh e Load associados ao objeto web panel. O evento Enter é um evento do sistema que pode ser associado a qualquer controle inserido no form e que também pode ser executado quando o usuário pressiona a tecla Enter. Também definimos para botões, eventos de usuário, explicitamente. Ou seja, damos um nome qualquer para um evento e o associamos a um botão ou a qualquer outro controle (observe-se a propriedade **On Click Event** que possuem os controles simples, no form).

Veremos agora mais detalhadamente os eventos dos web panels, onde se executa cada um (se no servidor onde a aplicação está instalada ou no cliente –Browser–) e em que ordem.



Em qualquer aplicação web, teremos uma máquina –PC, notebook ou dispositivo inteligente– com conexão à Internet com a qual o usuário acessará a aplicação através de um navegador; e, por outro lado, o servidor, que será onde se encontram todos os programas da aplicação gerados por GeneXus. Entre eles, os correspondentes aos web panels (por exemplo, o programa gerado para o web panel que implementamos a partir do zero).

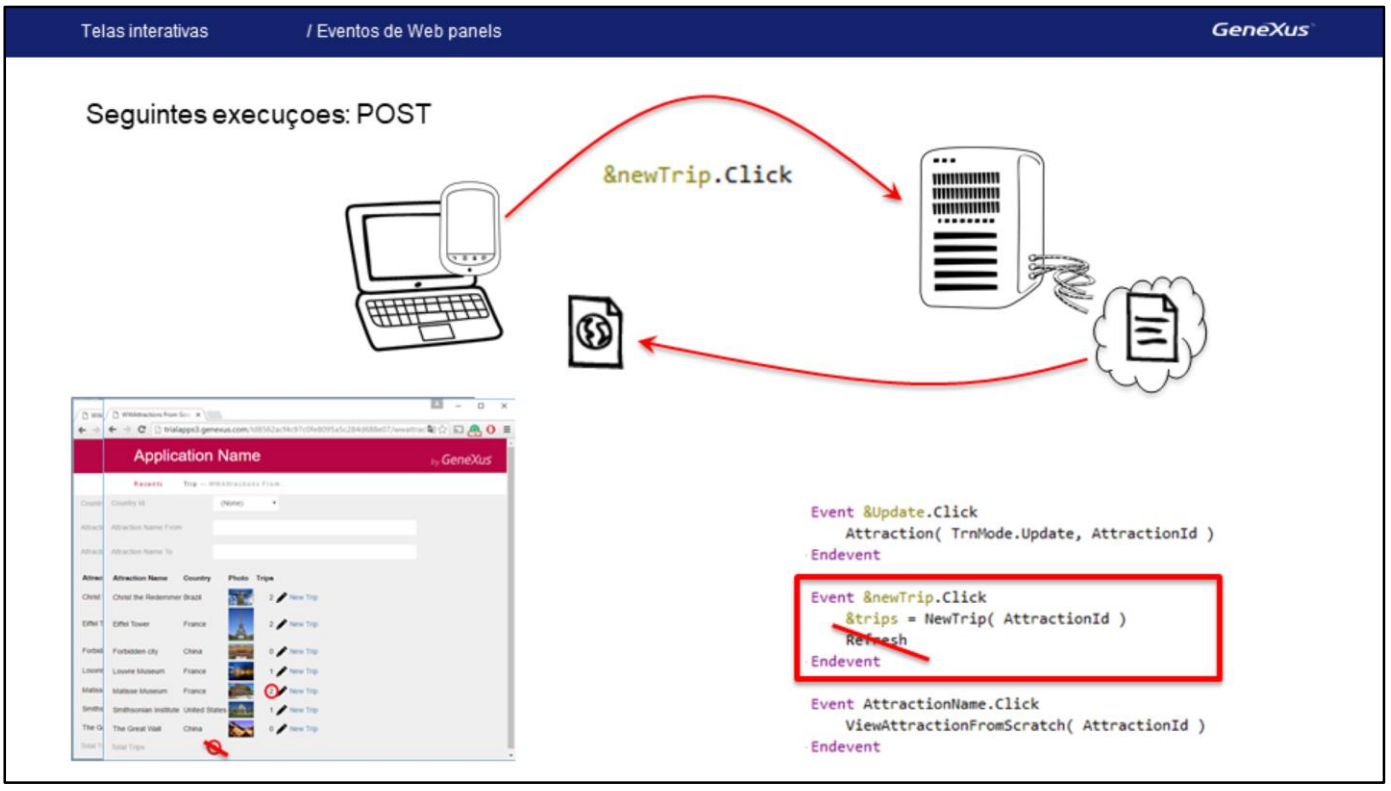
O que acontece quando invocamos o web panel a partir do navegador **pela primeira vez**? O que é conhecido como fazer um **GET**.

O cliente solicita ao servidor que execute o programa associado ao web panel e retorna como resultado um arquivo html que indique ao navegador como desenhar a tela (com quais dados, com qual formato, etc.).

Mas o que executa o programa no servidor para montar esse arquivo html?

Os eventos Start, Refresh e Load, nessa ordem. Se trata-se de um web panel com tabela base, como é o caso, o evento Load será executado N vezes: uma para cada registro que atenda às condições.

É importante ter claro que, nesta primeira execução do web panel (GET), o usuário não tem tempo para escolher qualquer valor no Dynamic Combo Box, nem nas variáveis para filtrar pelo nome da atração. É a primeira chamada ao objeto. Portanto, a variável &CountryId estará vazia... e a condição definida contempla que se filtre apenas se a variável &CountryId não estiver vazia (when not &CountryId.IsEmpty()). O mesmo acontecerá com as variáveis &AttractoinNameFrom e &AttractionNameTo. Portanto, não se realizará nenhum filtro e a página será exibida com o grid carregado com todas as atrações de todos os países.



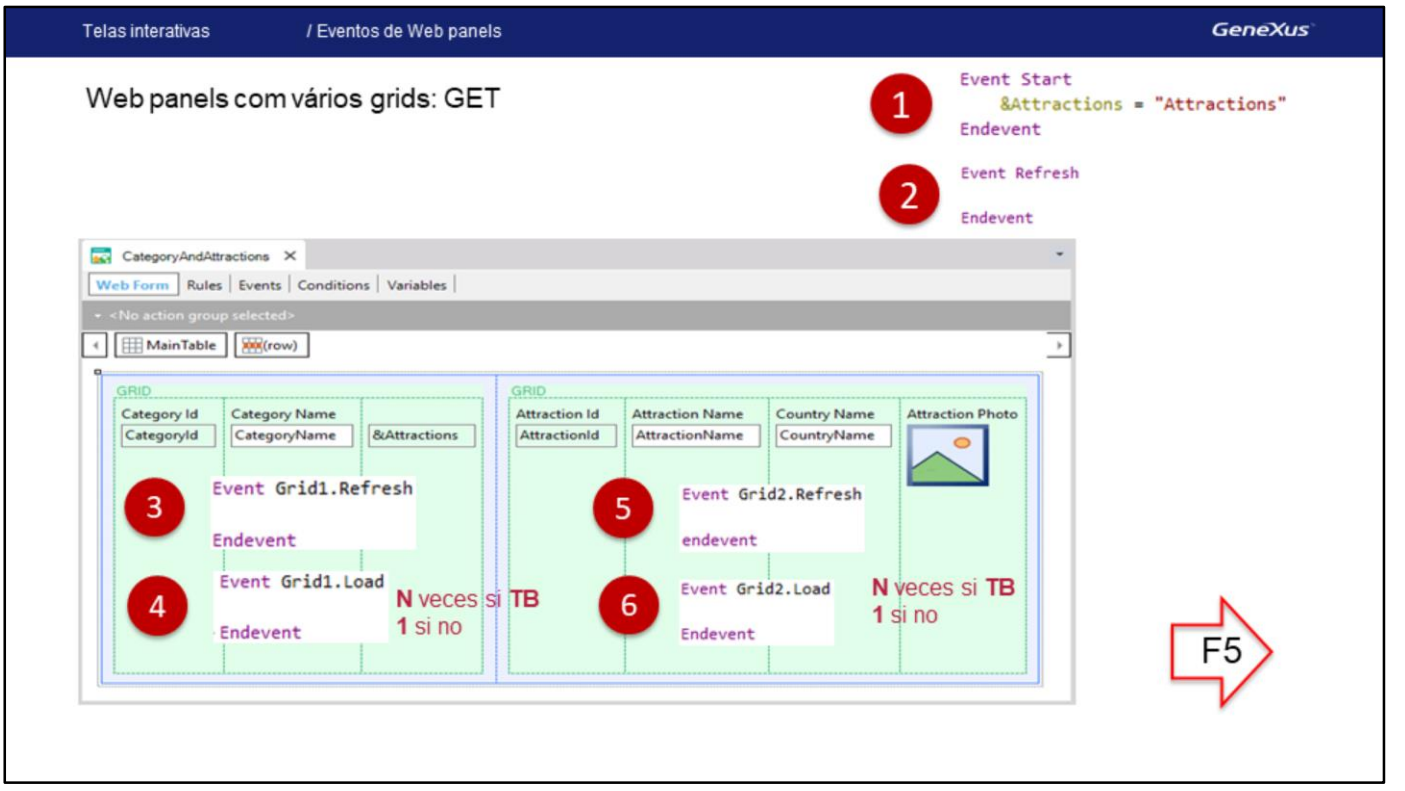
Em forma geral, um **POST** ocorre cada vez que uma ação é efetuada no cliente, que requer o retorno ao servidor para executar.

Ações deste tipo podem ser pressionar a tecla Enter ou algum botão ou controle associado a um evento.

Quando um POST é executado, acontece o seguinte:
- A informação é lida na tela
- O evento de usuário que causou o Post é executado.

Por exemplo, se o usuário clicar na opção do grid New Trip, se envia ao servidor o AttractionId da linha sobre a qual foi clicada. No servidor, se invoca o procedimento NewTrip passando-lhe como parâmetro esse valor de AttractionId. Como o valor retornado pelo procedimento se carrega na variável do grid, &trips, isto faz com que automaticamente volte a ser carregada a linha no html resultante.

Se precisarmos recarregar tudo (por exemplo, para que a variável &totalTrips seja mostrada com o valor correspondente), então, colocando o comando Refresh no evento do usuário voltará a disparar Refresh e Load, como já vimos aulas atrás. Isso é executado no Servidor, antes de montar o html que é retornado ao cliente.

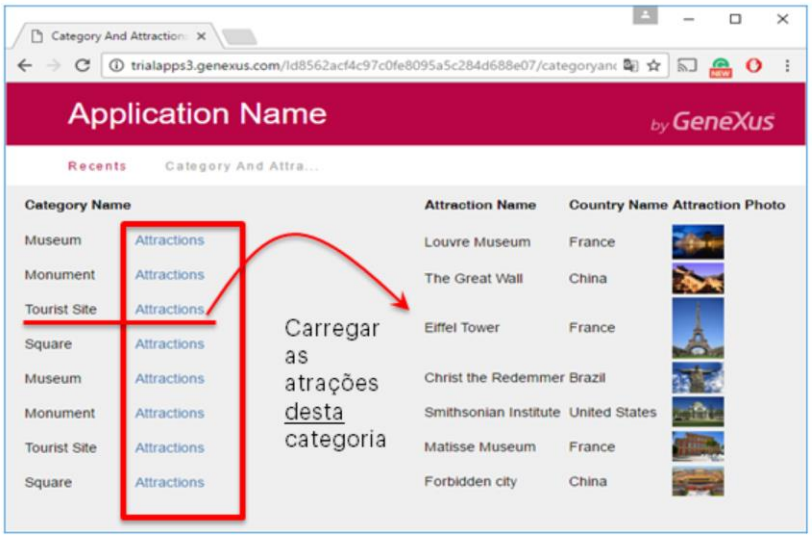


Para Web panels com vários grids, o evento Refresh que agora será genérico (do web panel como um todo), iniciará o evento Refresh e o Load de cada grid.

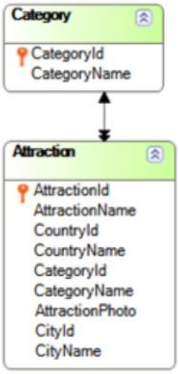
Dependendo se o grid possui uma tabela base ou não, o evento Load, como no caso de um único grid, será executado N vezes, uma vez para cada registro da tabela base a ser carregado como linha do grid; ou apenas uma, se não tiver tabela base.

A ordem de disparo dos eventos é a apresentada acima. Dependerá da ordem em que os grids estão na tela (da esquerda para a direita, de cima para baixo).

Web panels com vários grids: GET

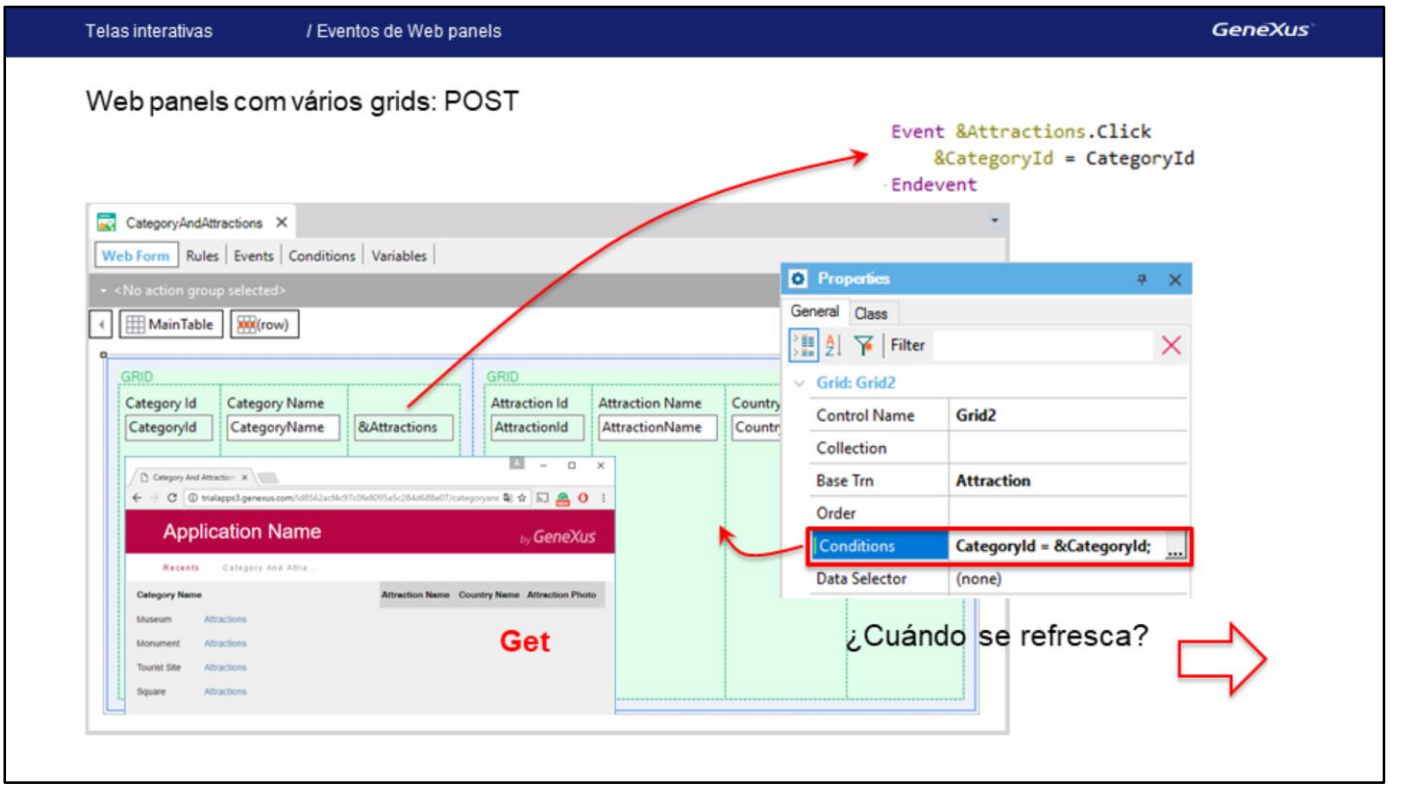


Embora as tabelas base de cada grid estejam relacionadas por uma relação 1 a N, as cargas não são automaticamente relacionadas



Para que no grid de atrações sejam mostrados apenas as da categoria escolhida no grid à esquerda, teremos que especificar o filtro explicitamente.

Para fazer isso, adicionamos uma coluna com a variável &Attractions, do tipo character(15), com o texto Attractions (especificamos isso no evento Start, porque alcançamos defini-lo quando o web panel é aberto. Não irá mudar mais tarde) . Assim, quando o usuário clica nessa opção, devemos pedir-lhe para carregar (novamente) o grid de atrações.



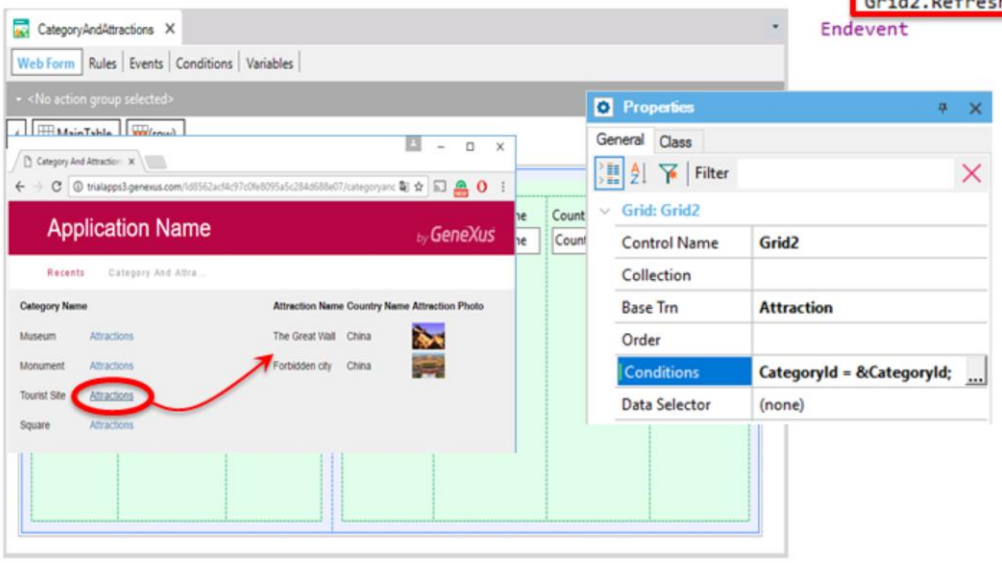
Para programar a carga do grid de atrações a partir da categoria escolhida pelo usuário no outro grid, criamos uma variável &CategoryId à qual atribuiremos valor cada vez que o usuário clicar em "Attractions" para a linha desejada. Observemos que especificamos como condição para que uma atração seja carregada no grid (Grid2) que seu CategoryId corresponda ao valor da variável.

Desta forma, na primeira execução, nenhuma atração irá cumprir a condição, pois, nesse caso, a variável &CategoryId estará vazia.

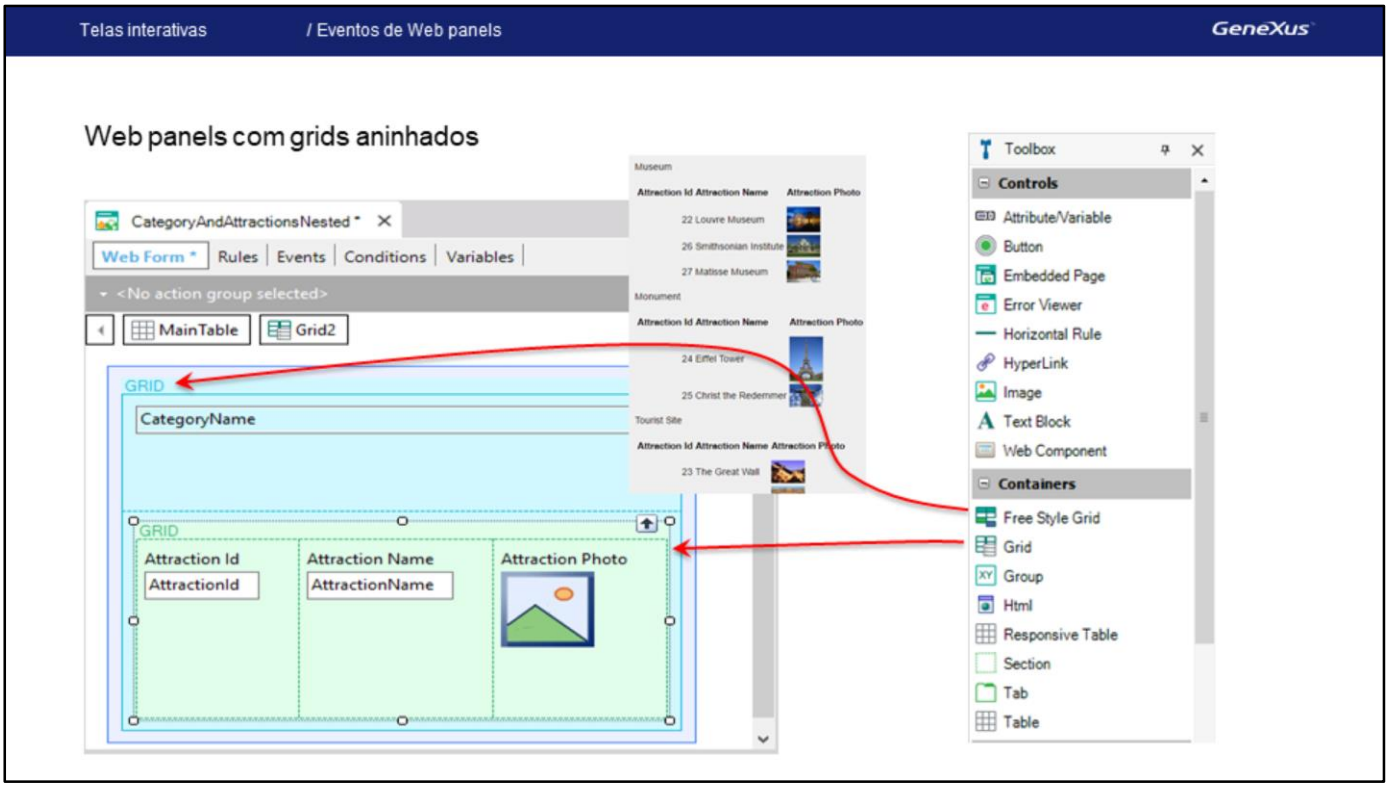
Então, quando o usuário clicar em &Attractions, é dado valor à variável a partir do CategoryId dessa linha. Mas se não fizermos nada mais, nunca será atualizado o Grid2. Temos que pedir ao grid2 que se atualize

Web panels com vários grids: POST

```
Event &Attractions.Click
    &CategoryId = CategoryId
    Grid2.Refresh()
Endevent
```



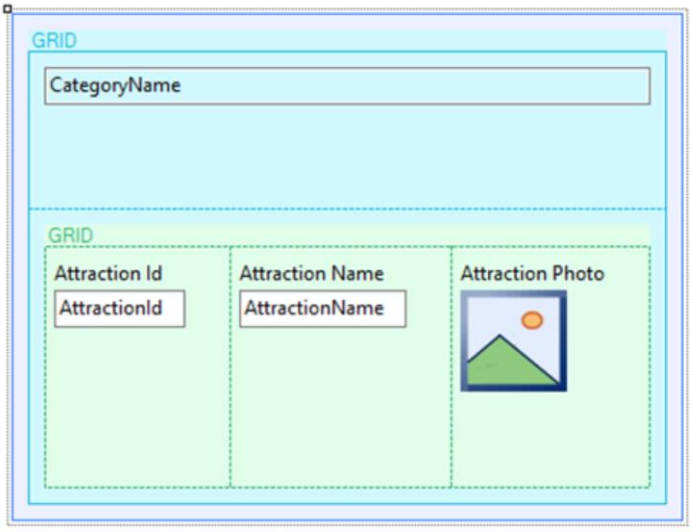
E fazemos isso com o método Refresh desse grid.



Para aninhar grids, são utilizados os grids Freestyle. Por exemplo, ao invés de ver todas as categorias e clicando em uma, ver suas atrações, podemos ver tudo aninhado, isto é, para cada categoria, suas atrações. Como se fosse um caso de for eachs aninhados. É análogo.

Aqui as navegações sim se relacionam.

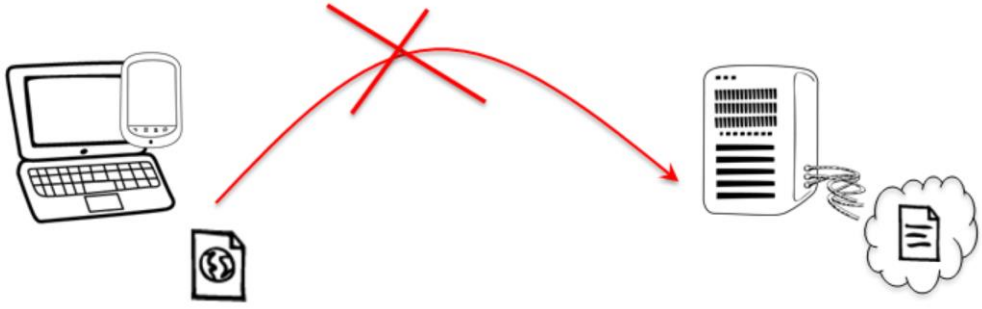
Web panels com grids aninhados (GET)



1. Start
2. Refresh
3. Grid1.Refresh
4. Grid1.Load
    5. Grid2.Refresh
    6. Grid2.Load

A ordem de disparo dos eventos é conforme o esperado. Primeiro Start do web panel, então Refresh e, em seguida, Refresh do grid e Load. Se tem tabela base, o Load disparará N vezes, uma para cada linha. Caso contrário, se disparará apenas uma.
Em qualquer caso, após o Load do grid freestyle, se diaparará Refresh e Load do Grid aninhado. Mais uma vez, este segundo Load poderá ser executado apenas uma vez, ou N, dependendo se possui tabela base ou não.

Alguns eventos podem ser resolvidos no cliente, sem POST para o Server



```
Event Enter
    Textblock1.Caption = "Please, click Attractions in order to see the Attractions of the Category"
endevent
```

Nem toda ação realizada pelo usuário e associada a um evento produzirá um POST para o servidor. Alguns podem ser resolvidos no cliente.

Por exemplo, se em um evento associado a entrada, usuário ou controle, um controle se torna invisível, ou a legenda, a cor, etc. são alterados, isso é resolvido no próprio cliente.

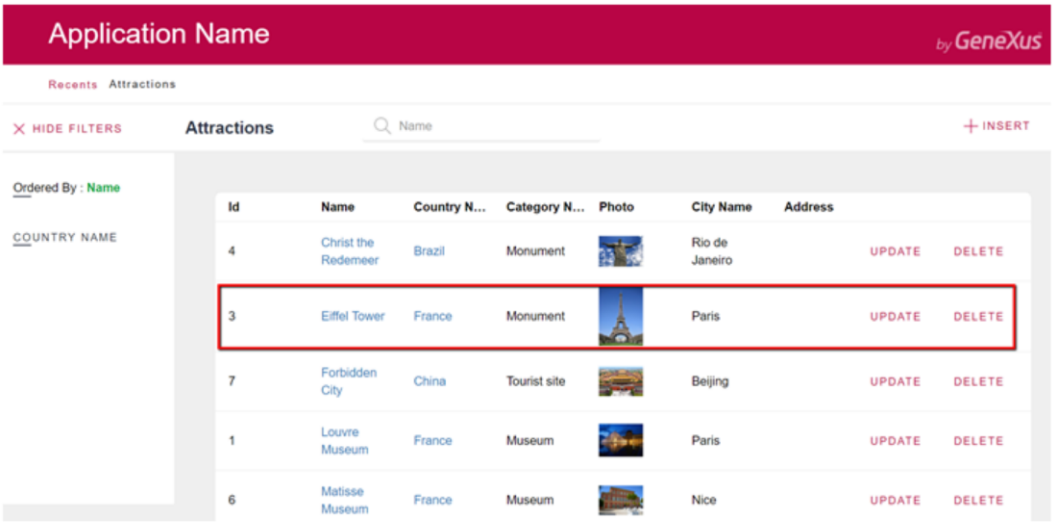
Seleção múltipla em um grid

Em muitas ocasiões, devemos trabalhar com um conjunto selecionado de elementos, para fazer algo com eles mais tarde.

Dado que um grid nos permite visualizar muitos elementos, é natural que queiramos selecionar vários elementos de um grid.

Em seguida, veremos como podemos fazer uma seleção múltipla em um grid e como podemos percorrer os elementos para processá-los.

Seleção de uma única linha de um grid



Suponhamos, por exemplo, que temos um grid com dados de atrações.

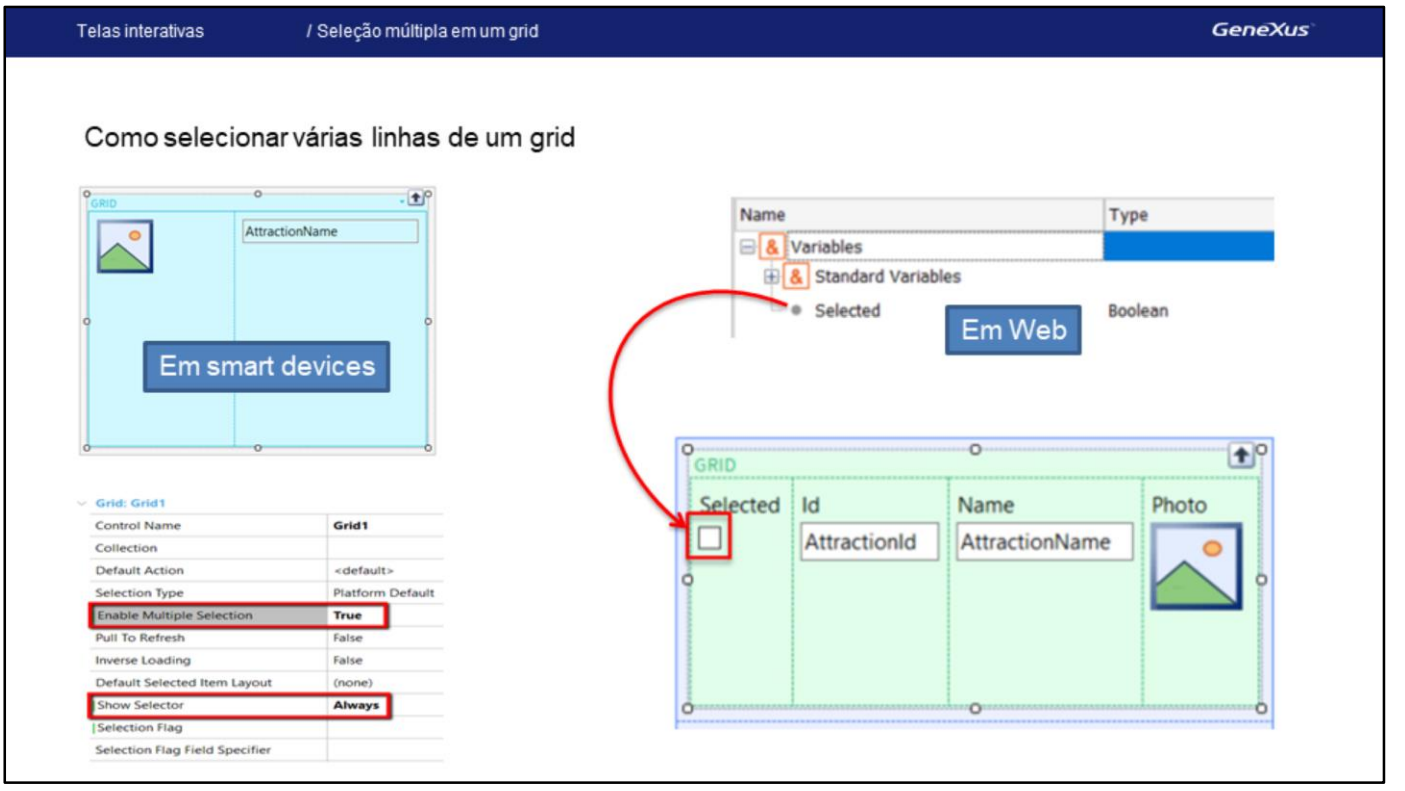
Se queremos trabalhar com uma atração específica, é necessário marcar uma linha do grid como selecionada, de forma a poder acessar os valores de cada coluna dessa linha.

Para selecionar uma única linha de um grid: AllowSelection=True

Em uma aplicação Web, se queremos marcar uma linha do grid como selecionada, vamos às propriedades do grid e, na propriedade AllowSelection, selecionamos o valor True.

Com este valor, também se disponibiliza a propriedade Allow Hovering, cujo valor padrão é True. Isso significa que ao passar o mouse sobre as linhas, elas são pintadas com uma cor.

Com AllowSelection em True, ao clicar em uma atração, a linha fica marcada com outra cor. Essas cores podem ser configuradas no objeto Theme, nas classes atribuídas às propriedades Selected Row Class e Hover Row Class da classe Grid.



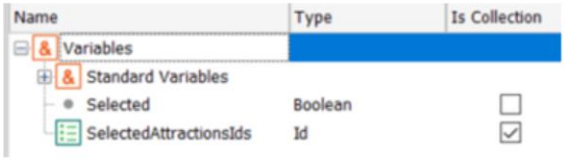
Suponhamos que a agência de viagens exija que, a partir de um grid de atrações, selecionemos aquelas que são de nosso interesse e imprimamos uma lista com as atrações selecionadas.

Em uma aplicação Web, podemos usar uma propriedade do grid para selecionar uma única linha, como já vimos, mas não temos propriedades que nos permitam indicar que queremos selecionar várias linhas de cada vez.

Diferentemente, em uma aplicação para Smart Devices, sim é possível, utilizando a propriedade **Enable Multiple Selection** do grid. Para poder fazer com que apareça o checkbox que nos permite selecionar a linha, utilizamos a propriedade **Show Selector**.

Para o que a agência de viagens nos pede, como precisamos de uma tela web, devemos resolver isso de outra maneira. Uma solução seria adicionar uma variável do tipo boolean ao grid, usá-la para marcar a linha selecionada e depois percorrer as linhas do grid para processar as que foram selecionadas. Vamos ver como fazer isto.

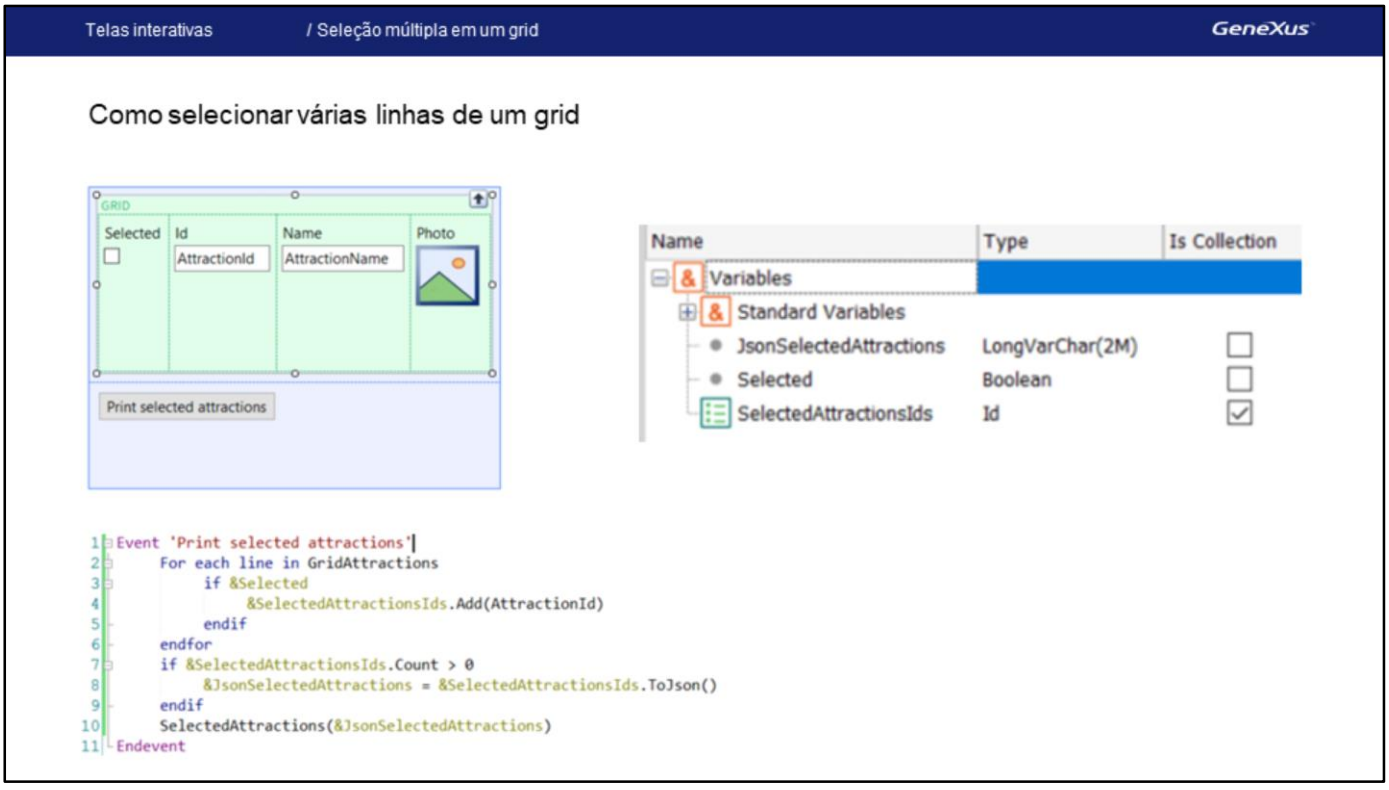
Como selecionar várias linhas de um grid



Primeiro vamos criar um webpanel de nome SelectedAttractions, no qual inserimos um grid, com os atributos AttractionId, AttractionName e AttractionPhoto.
Em seguida criamos uma variável de nome Selected, do tipo boolean e a incluímos no grid, na coluna mais à esquerda.

Depois incluímos um botão e ao evento damos o nome "Print selected attractions". Neste evento, vamos invocar o objeto procedimento que imprime as atrações selecionadas e para poder enviá-las para ele, devemos guardá-las primeiro em uma coleção.

Para isto, adicionamos uma variável chamada SelectedAttractionsIds, já que nos é suficiente guardar na coleção apenas os identificadores das atrações escolhidas. Lhe atribuímos o tipo de dados Id e a marcamos como coleção.



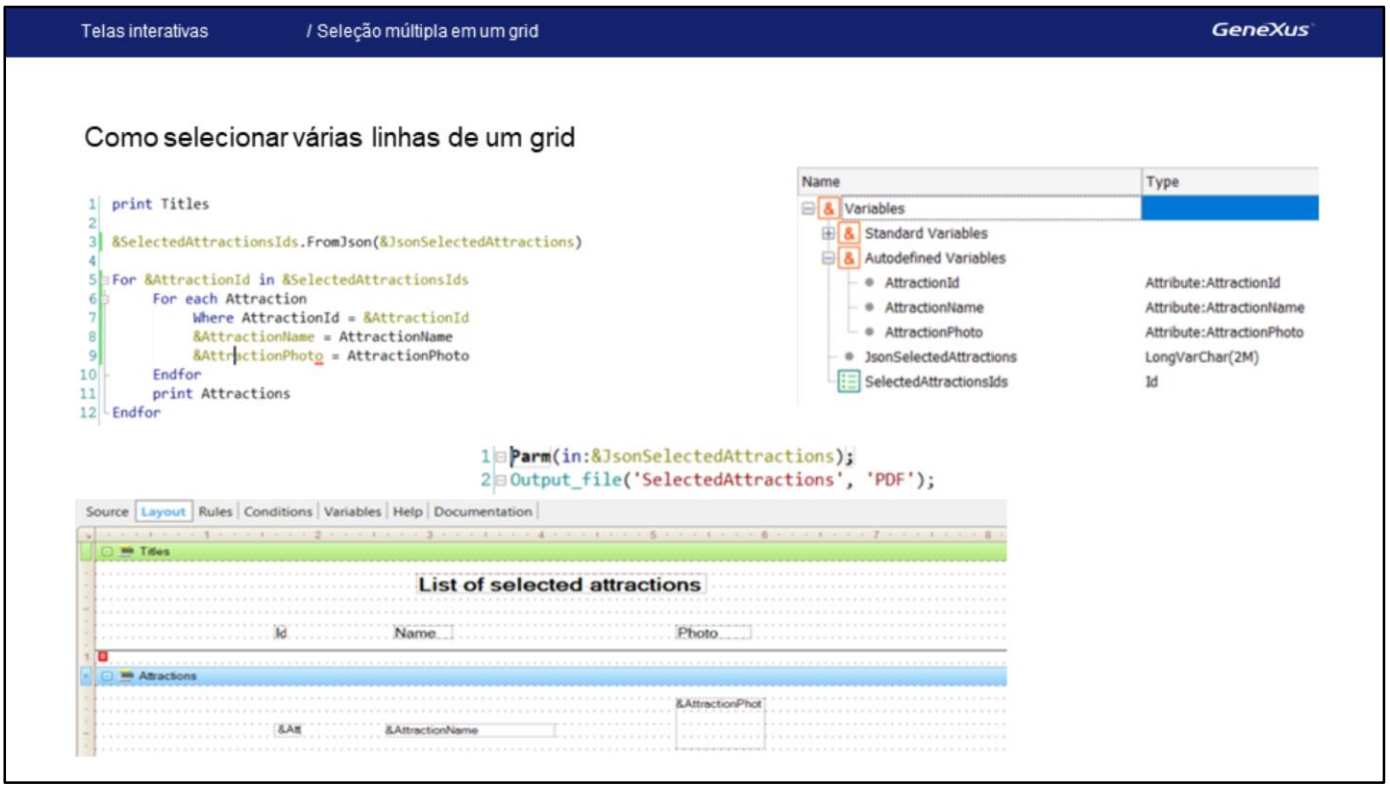
Para percorrer as linhas de um grid, usamos a instrução **For each line** ... in... o nome do grid, que neste caso é GridAttractions. Esta instrução se posicionará em cada linha do grid e nos permitirá recuperar os valores que estão tomando suas colunas em cada linha. Algo importante a levar em conta é que o For each line só percorrerá os registros carregados no grid, se houverem mais registros nas páginas seguintes, eles não serão incluídos. Apenas os registros que podem ser vistos nas linhas do grid podem ser percorridos.

Para verificar se a atração foi selecionada ou não, escrevemos If &Selected e, caso isso se cumpra, adicionamos o AttractionId à coleção, então escrevemos &SelectedAttractionsId.Add(AttractionId) e fechamos o if. Agora fechamos o for each line com um endfor. Se estivéssemos em uma aplicação para Smart Devices, em vez da variável booleana, usamos a propriedade **Show Selector** do grid e, para percorrer as linhas, temos o comando **For each selected line**.

Retornando ao nosso webpanel, quando todas as linhas do grid forem percorridas, passaremos a coleção das atrações selecionadas para a lista. Mas, para poder fazer isso, não podemos passar a variável coleção como parâmetro, mas devemos serializar seu conteúdo, ou seja, gerar um arquivo de texto com um formato estruturado, por exemplo, um JSON ou um XML.

Criaremos uma variável para guardar o JSON, que chamamos de JSonSelectedAttractions do tipo LongVarChar.... e a carregamos a partir da coleção, utilizando o método ToJSon.

Agora sim, chamamos o procedimento SelectedAttractions e lhe passamos por parâmetro o JSON que obtivemos anteriormente.



Criamos um objeto do tipo procedimento, que chamamos de SelectedAttractions e definimos as variáveis AttractionId, AttractionName e AttractionPhoto. Em seguida, criamos uma variável chamada JSonSelectedAttractions, do tipo LongVarChar e uma variável SelectedAttractionsIds do tipo Id, a qual marcamos como coleção,
Agora vamos para as regras e escrevemos uma regra Parm, que recebe como entrada a variável JSonSelectedAttractions. Aproveitamos e já incluímos a regra Output_file.... e marcamos o procedimento como Main e o Call Protocol em HTTP.

Em Layout, renomeamos o printblock com o nome Titles e arrastamos um Text Block no qual colocamos o texto “List of selected attractions”. Também adicionamos títulos para as colunas da lista, um com o texto “Id”, outro “Name” e outro “Photo”. E finalmente colocamos uma linha abaixo desses títulos das colunas.

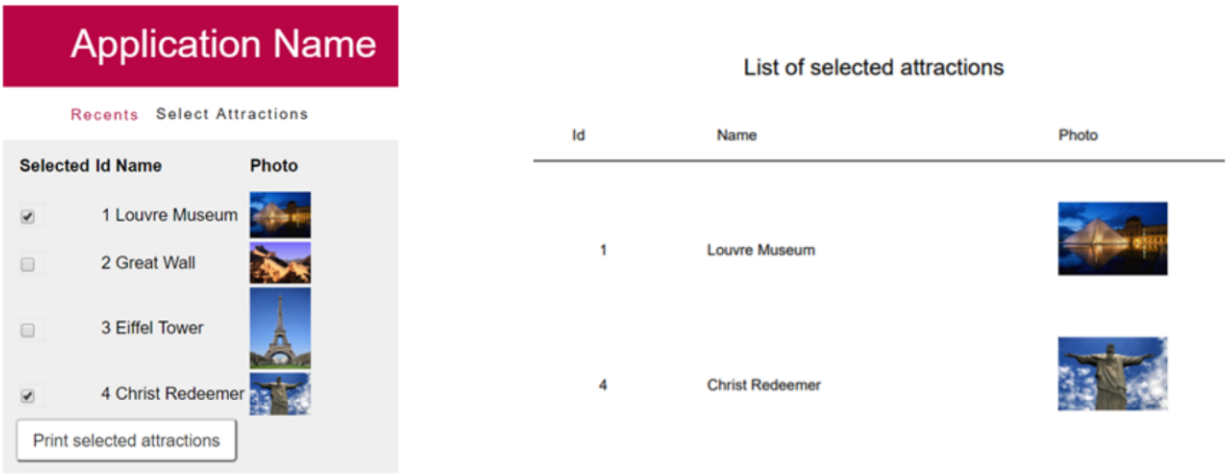
Agora inserimos outro printblock, o chamamos de Attractions e arrastamos as variáveis AttractionId, AttractionName e AttractionPhoto.

No source escrevemos print Titles.... E agora carregamos a variável coleção SelectedAttractionsId com o conteúdo da variável JSonSelectedAttractions, usando o método FromJson.

Com isto, os Ids das atrações selecionadas foram salvos na coleção, então devemos percorrer a mesma para acessá-los, então escrevemos: For &AttractionId in &SelectedAttractionsIds e, em seguida, com cada AttractionId acessado, recuperaremos o nome da atração e sua foto, para isso, escrevemos For each Attraction, Where AttractionId = &AttractionId e carregamos as variáveis &AttractionName e &AttractionPhoto a partir dos atributos correspondentes.
Fechamos o for each, imprimimos o printblock com os dados das atrações e finalmente fechamos o For in.

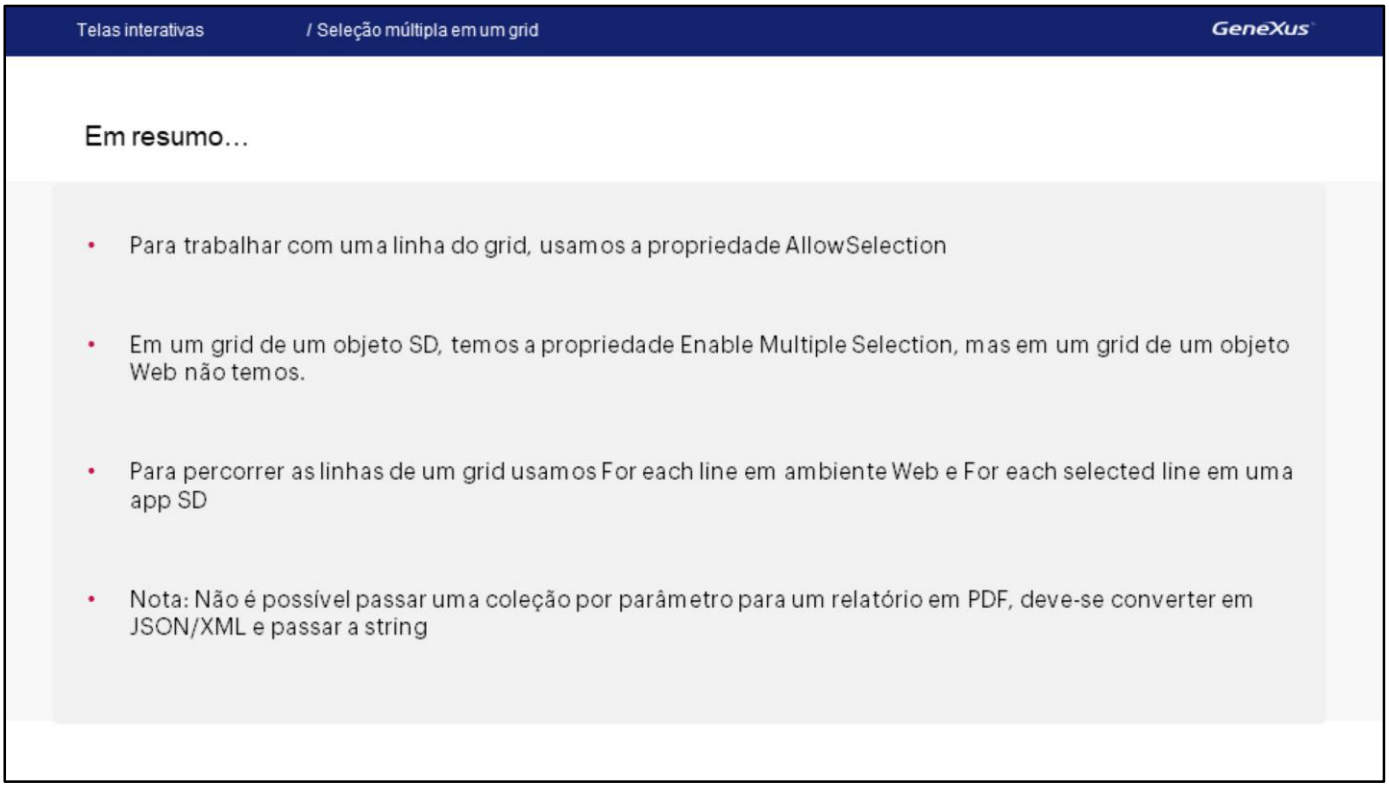
Agora sim estamos em condições de testar o que havíamos programado, então damos F5.

Como selecionar várias linhas de um grid



No Developer Menu, clicamos em SelectAttractions e vemos que se abre o webpanel, mostrando a lista de atrações. Vemos que o controle associado à variável booleana Selected, é um check box, que nos permite selecionar as atrações.

Vamos escolher o Museu do Louvre e a Torre Eiffel... e pressionaremos Print selected attractions. Vemos que se abre a lista, mostrando as atrações que havíamos selecionado.



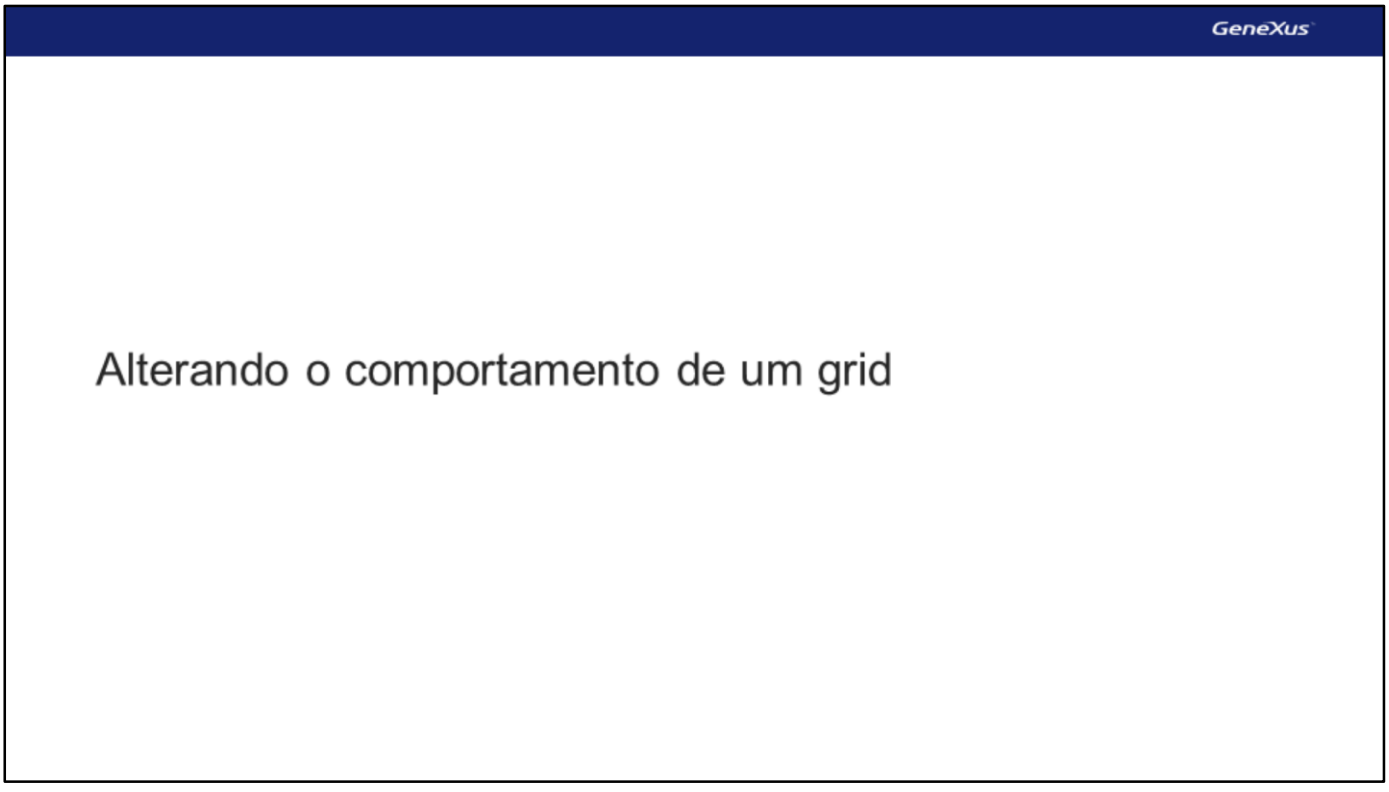
Repassemos os conceitos vistos até agora.

Se queremos selecionar uma única linha de um grid, definimos a propriedade AllowSelection como True, com a qual podemos acessar os valores de cada coluna do grid, para a linha selecionada.

Se queremos selecionar mais de uma linha ao mesmo tempo, se estivermos em uma aplicação para Smart Devices, o grid conta com a propriedade Enable Multiple Selection. Ao atribuir-lhe o valor True, é possível selecionar várias linhas para serem processadas posteriormente.
Em um grid de uma aplicação web, não temos essa propriedade, então devemos usar uma variável booleana no grid, que nos permitirá salvar a seleção da linha.

Para percorrer as linhas de um grid em uma aplicação Web, temos o comando For each line. Este comando irá iterar apenas para as linhas que estejam carregadas no grid em um determinado momento e em cada iteração teremos os valores das colunas do grid para a linha percorrida. Para saber se uma linha foi selecionada, usamos o valor de uma variável booleana que adicionamos para essa finalidade.
No caso de uma aplicação SD, usamos a propriedade Show Selector e o comando For each selected line.

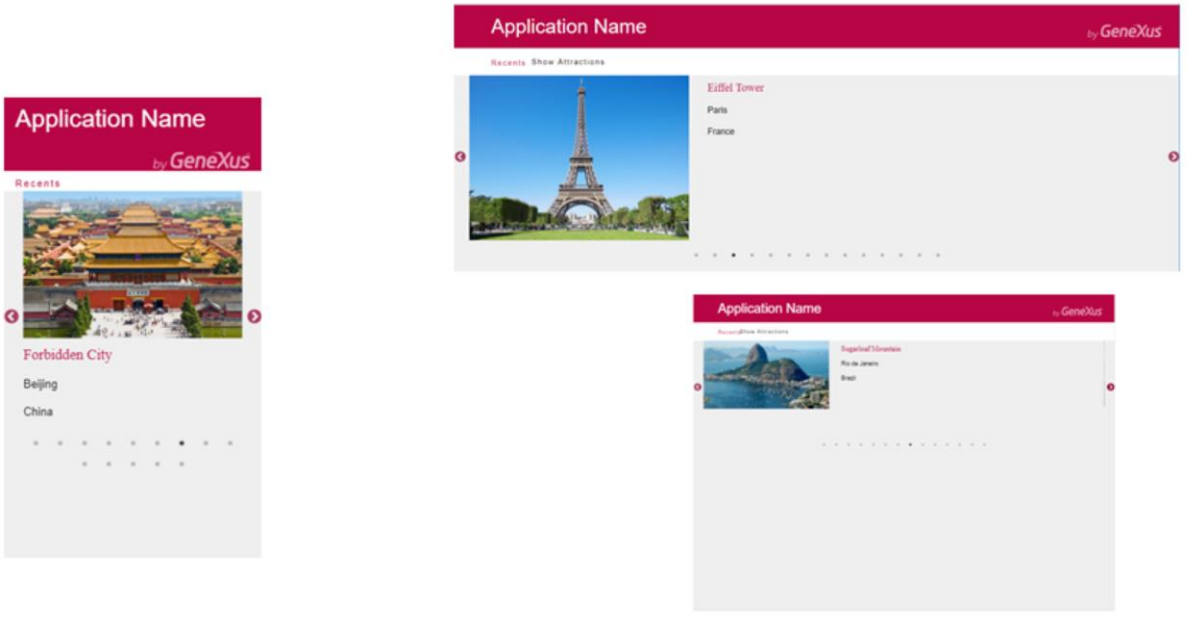
Não é possível passar uma variável coleção por parâmetro para um objeto web, como um webpanel ou um procedimento com protocolo HTTP. Para enviar as informações das atrações selecionadas, usamos uma coleção, mas depois serializamos estas informações gerando um arquivo de texto estruturado, por exemplo, usando o formato JSON ou XML.



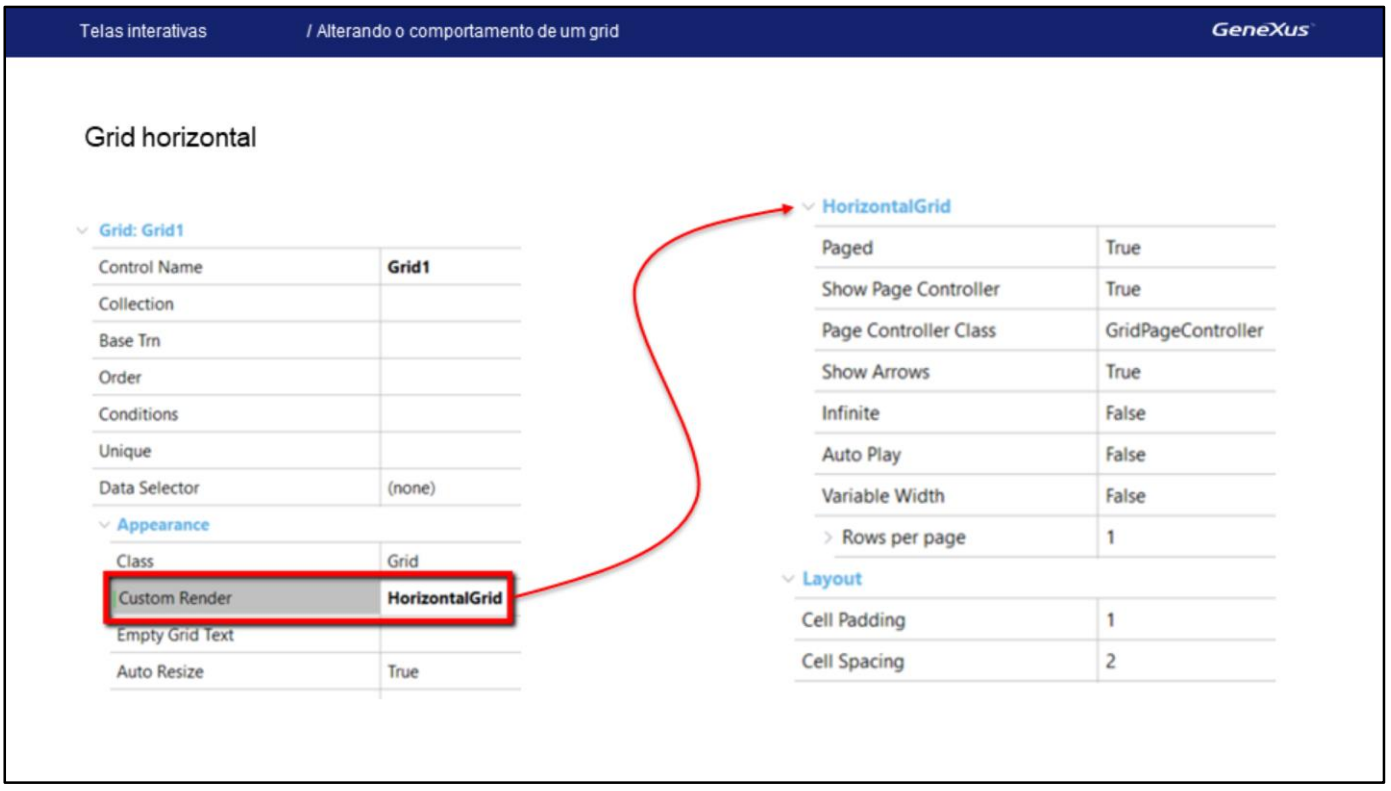
Alterando o valor de algumas propriedades do grid, podemos fazer com que o mesmo altere a maneira de exibir informações, por exemplo, mostrar os dados horizontalmente, com scroll infinito, ou de forma flexível, customizando a exibição de acordo com seu conteúdo.

Vejamos a seguir alguns casos.

Grid horizontal



Um grid horizontal é um grid que nos permite mostrar informações como se fosse um carrossel, ou seja, podemos paginar a partir da direita e os elementos se moverão horizontalmente, para a esquerda.



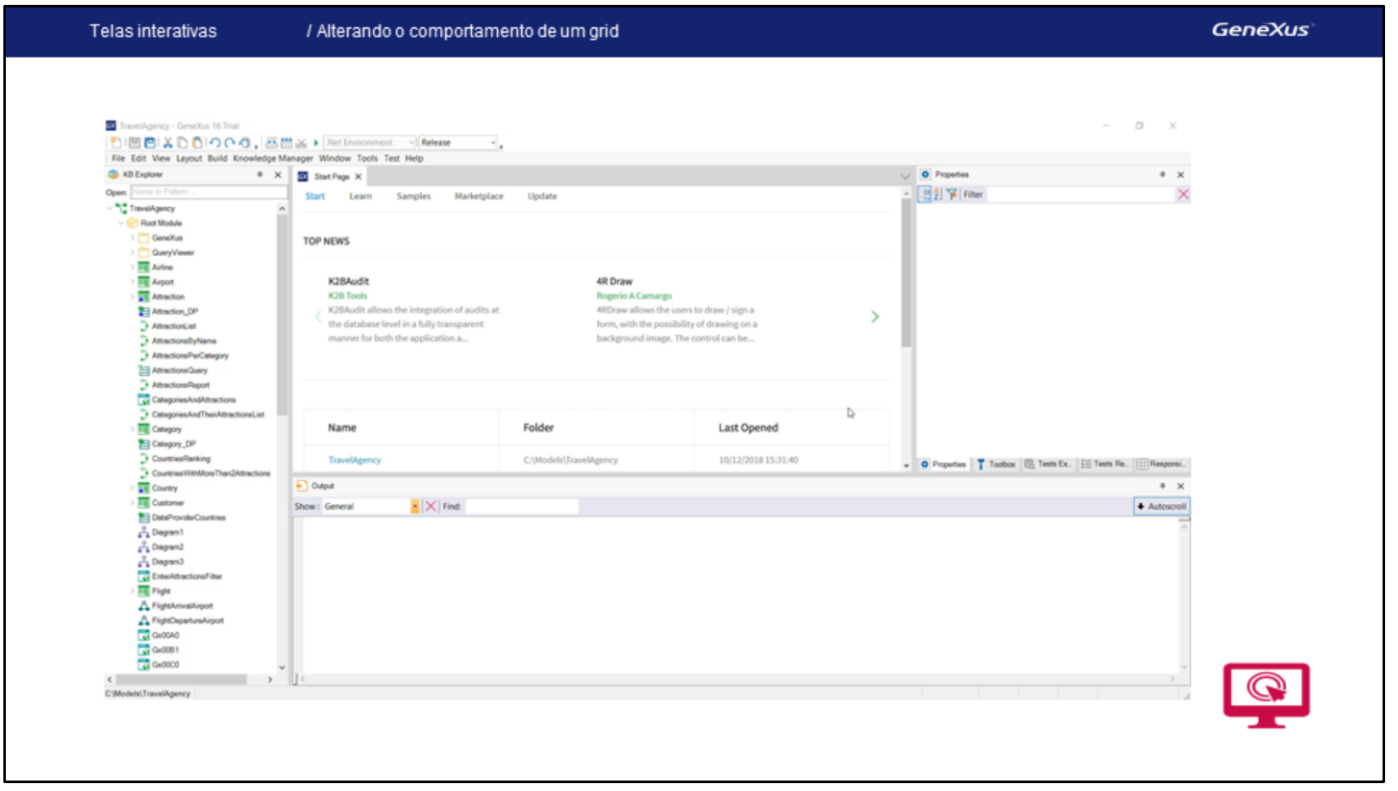
Para fazer um grid exibir seu conteúdo horizontalmente, ou seja, em formato carrossel, devemos atribuir à propriedade Custom Render o valor HorizontalGrid.

Ao atribuir este valor, serão habilitadas uma série de propriedades, através das quais podemos configurar a aparência do grid. Por exemplo, a propriedade Show Page Controller determina se será visto ou não o paginador e a propriedade Page Controller Class nos permitirá definir a **classe** na qual este paginador se baseia.

Esta classe é como um modelo para o qual podemos alterar propriedades e todos os controles baseados na classe herdarão esses valores.
Há um certo paralelismo com o conceito de domínio que vimos antes. Alterando os valores das propriedades de uma classe, podemos personalizar todos os controles baseados nela.
Estas classes, que definem a aparência e o comportamento de cada controle na tela, são agrupadas sob o **objeto Theme**, que estudaremos mais tarde.

Voltando às propriedades que nos permitem personalizar nosso grid horizontal, vemos que podemos decidir se queremos ver ou não as setas do grid que usamos para mudar de registro, definir se a largura do grid é variável, como será paginado, etc..

Vejamos um exemplo de um grid deste tipo.



[Demo: https://youtu.be/b76ee3Iu3hQ]

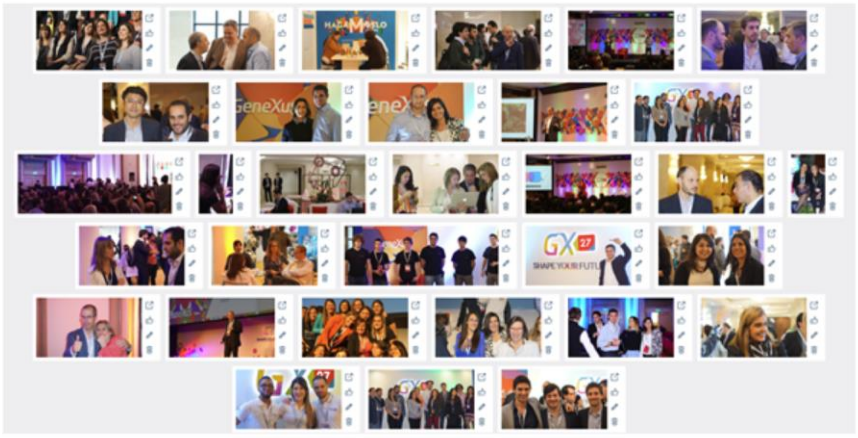
Vejamos uma demonstração de como configurar um grid horizontal usando a propriedade **Custom Render**.
Para testar o anterior, temos um webpanel ShowAttractions com um grid. Em particular, é um FreeStyleGrid, mas o exemplo também é válido para um grid padrão.
Para o grid, adicionamos os atributos AttractionName, AttractionPhoto, CityName e CountryName e uma tabela responsiva para melhorar o alinhamento.

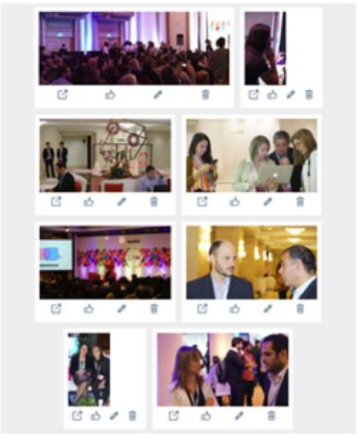
Se executamos este webpanel... vemos o grid com seu comportamento habitual. Agora, atribuímos o valor Horizontal grid à propriedade **Custom Render** e executamos.

Vemos que, somente fazendo isto, as informações são exibidas horizontalmente, como em um carrossel, mostrando uma atração de cada vez.
Também vemos o controle da paginação do carrossel e as setas à direita e à esquerda da página. Como dissemos, tudo isto é configurável a partir do objeto Theme.

Como estamos usando o Chrome, se pressionarmos F12, podemos escolher o tamanho da tela. Se escolhermos outros dispositivos, veremos que as informações são ajustadas de maneira responsiva, mesmo se mudarmos a orientação do mesmo.

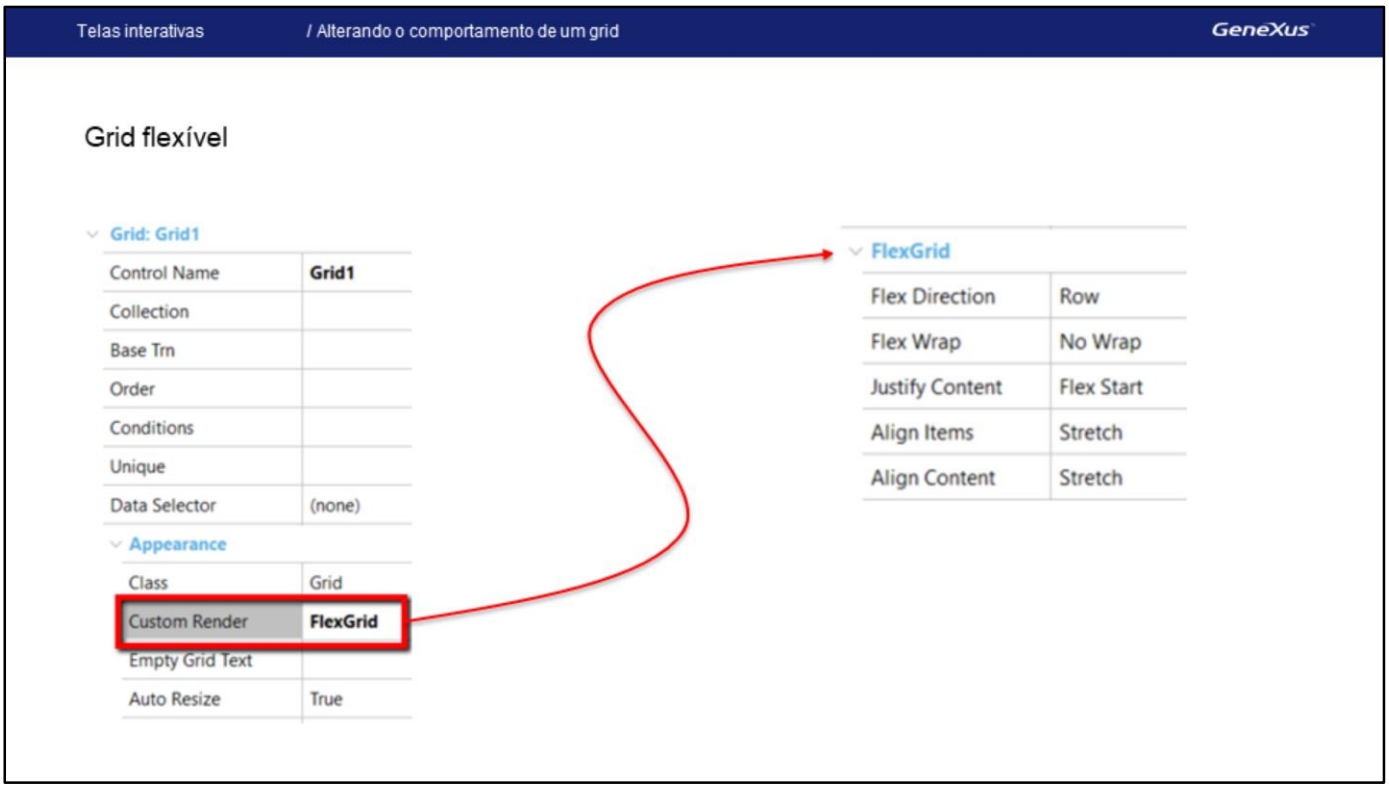
Grid flexível





Um grid flexível nos permite visualizar as informações de uma forma adaptável às nossas necessidades e nos permite uma personalização maior do conteúdo do que a permitida pelo Responsive Web.

Quando adicionamos fotos, por exemplo, nunca sabemos o comprimento ou largura de cada elemento que devemos mostrar. Estes elementos devem ser reordenados automaticamente da melhor maneira, dependendo do tamanho da tela. Por exemplo, na figura à esquerda, vemos que as fotos têm as ações à direita e na figura à esquerda, com uma tela menor, os elementos se ajustaram à largura e os controles das fotos se moveram automaticamente sob as fotos.



Para fazer um grid exibir seu conteúdo de forma flexível, devemos atribuir à propriedade **Custom Render** o valor FlexGrid.

Ao atribuir este valor, uma série de propriedades será habilitada, através das quais podemos configurar a aparência do grid.

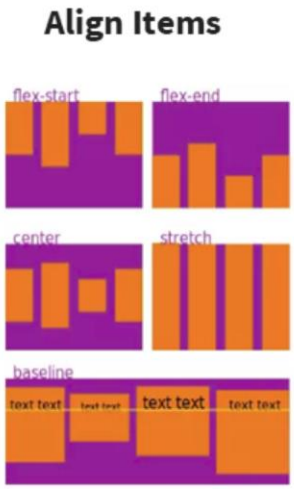
A propriedade **Flex Direction** nos permite alterar a direção em que os elementos são exibidos na tela, se é por linha ou linha invertida, por coluna ou coluna invertida.

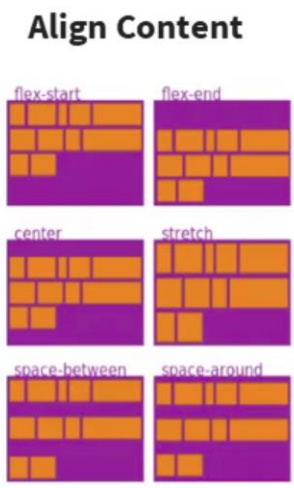
Flex Wrap estabelece como os elementos serão acomodados quando ocupam mais do que o comprimento de uma linha e nem todos podem ser exibidos. O valor Wrap acomoda-os na linha seguinte.

Justify Content nos permite estabelecer como o conteúdo será alinhado, se da esquerda para a direita, da direita para a esquerda, centralizado, que a separação seja ajustada para ocupar toda a largura disponível ou se espaços são adicionados entre os elementos.

Grid flexível

FlexGrid	
Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Align Content	Stretch

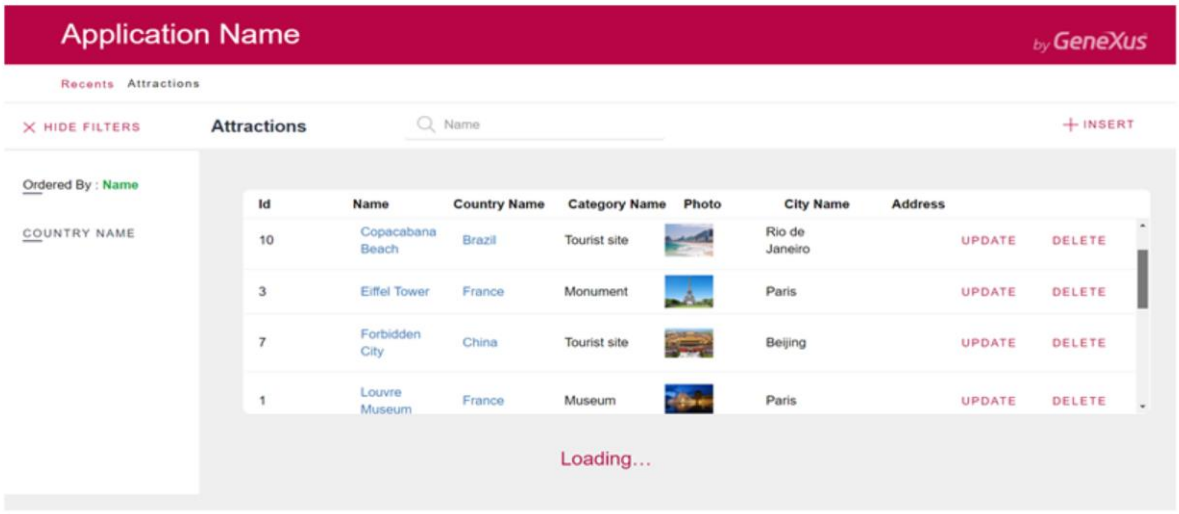




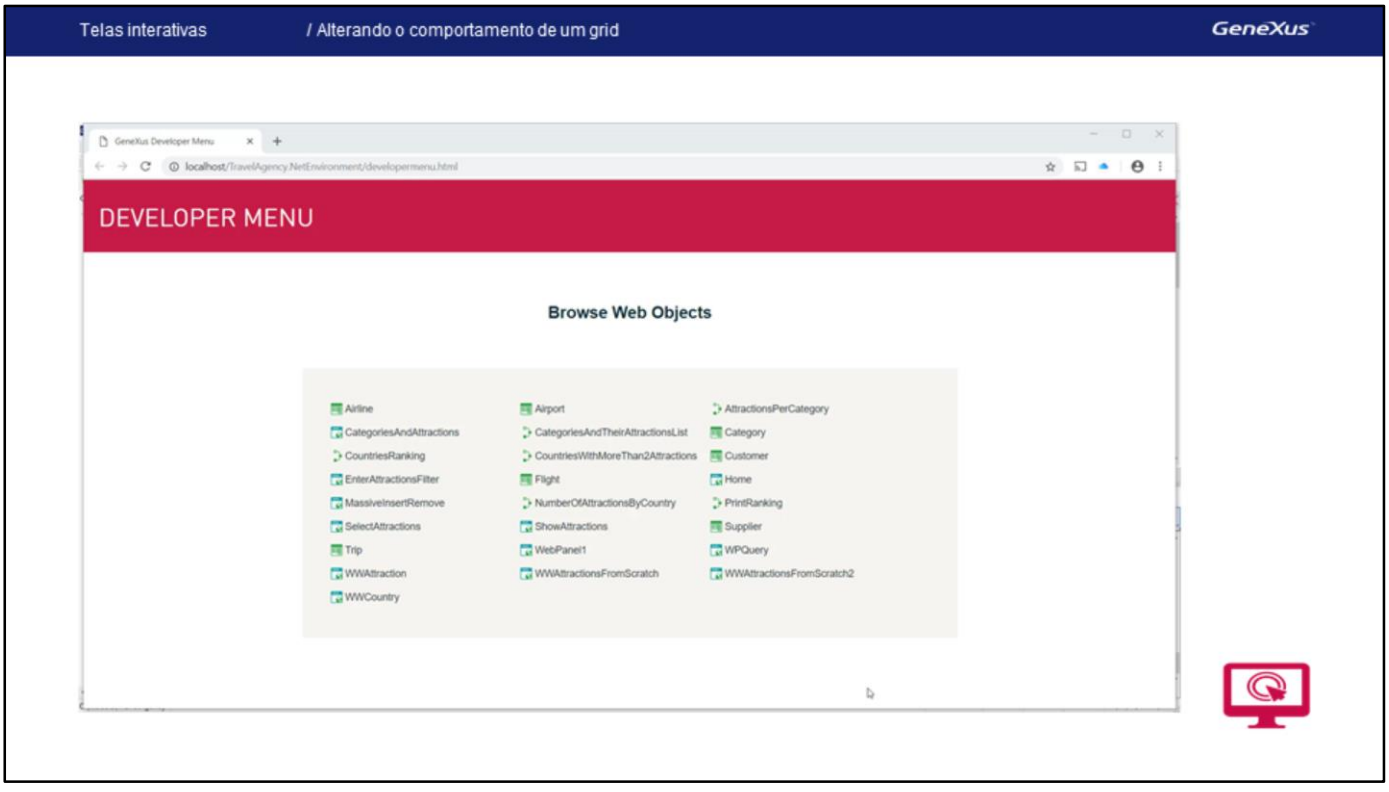
A propriedade **Align Items** nos permite selecionar como são alinhados os itens nas linhas.

Align Content nos permite alinhar a linha quando houver espaço extra no contêiner.

Grid com scroll infinito



Uma característica que podemos necessitar é que, em vez do grid nos mostrar as informações paginadas, as linhas sejam carregadas à medida que rolamos, na mesma página.



[Demo: https://youtu.be/jWNAkRXwGw0]

Para os fins desta demonstração, adicionamos algumas atrações. Se formos "trabalhar com Atrações", vemos que há 10 atrações exibidas na página e que ao final da mesma, temos o controle de paginação. Se pressionarmos o botão seguinte, veremos mais 4 atrações.

Se quisermos alterar o valor padrão de 10 para 5 registros por página, abrimos a instância do pattern WorkWithAttraction, clicamos no nó Selection, na propriedade **Rows per page**, escolhemos o valor **<custom>** e, na propriedade **Custom Rows**, atribuímos o valor 5.
Pressionamos F5... E vemos que as atrações estão separadas em 3 páginas.

Se precisarmos processar as informações das atrações e, em vez de 14 atrações, tivéssemos 1000, mesmo deixando o valor padrão de 10 registros por página, há muitas páginas a serem processadas. Seria muito mais simples se pudéssemos fazer com que o grid carregasse os registros automaticamente, conforme rolamos para baixo, sem ter que paginar à mão. Isto é o que é conhecido como ***scroll infinito***.

Para obter esse comportamento no grid do padrão WorkWith, vamos para o nó Selection e na propriedade **Paging mode** selecionamos o valor **<infinite scrolling>**. Pressione F5 e vemos que a barra de rolagem aparece no grid e o controle de paginação desaparece. Ao mover o scroll, a mensagem "Loading..." aparece, indicando que os registros estão sendo carregados da Base de Dados e podemos rolar indefinidamente até vermos todas as atrações que carregamos na tabela.

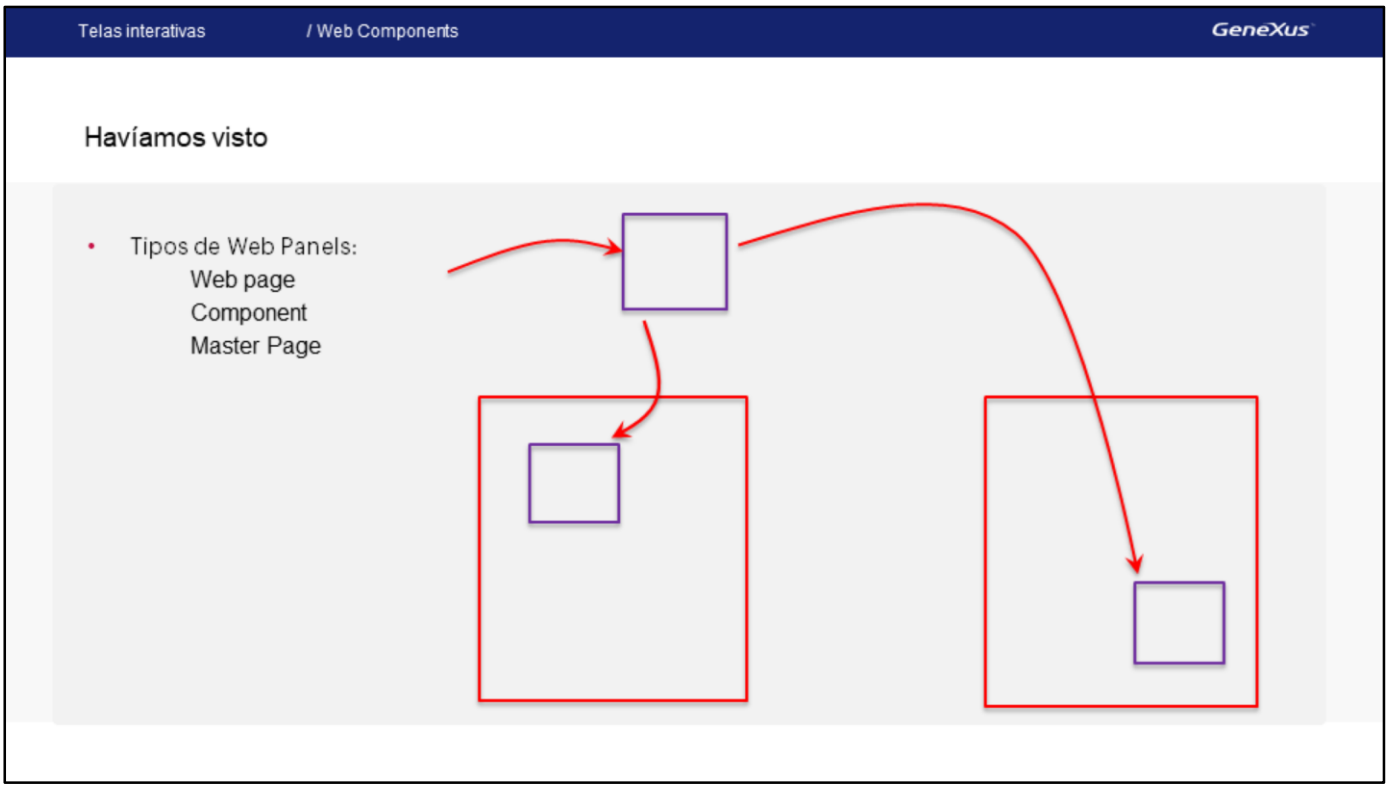
Se quisermos alterar o número de registros por página em um grid padrão que não esteja em um padrão Work With, alteraríamos a propriedade **Rows** do valor 0, que é o valor padrão, para o valor desejado. O valor 0 é igual a "unlimited" e torna todos os registros no grid visíveis de uma só vez.
Pressionamos F5... E vemos que o controle de paginação desaparece e a barra de rolagem não aparece no grid. Sim, o scroll aparece na página, porque o grid é muito longo para vê-lo completamente, e a página nos faz rolar. (Nota: Para ver a imagem em tamanho maior que o padrão, edite a coluna AttractionPhoto, defina sua propriedade Auto Resize em False e altere os valores das propriedades Height e Width).

Para obter a rolagem infinita em um grid padrão que não esteja em um padrão Work With, atribuímos a

propriedade **Rows** com um valor diferente de 0 e alteramos a propriedade **Paging** para o valor **Infinite Scrolling**.

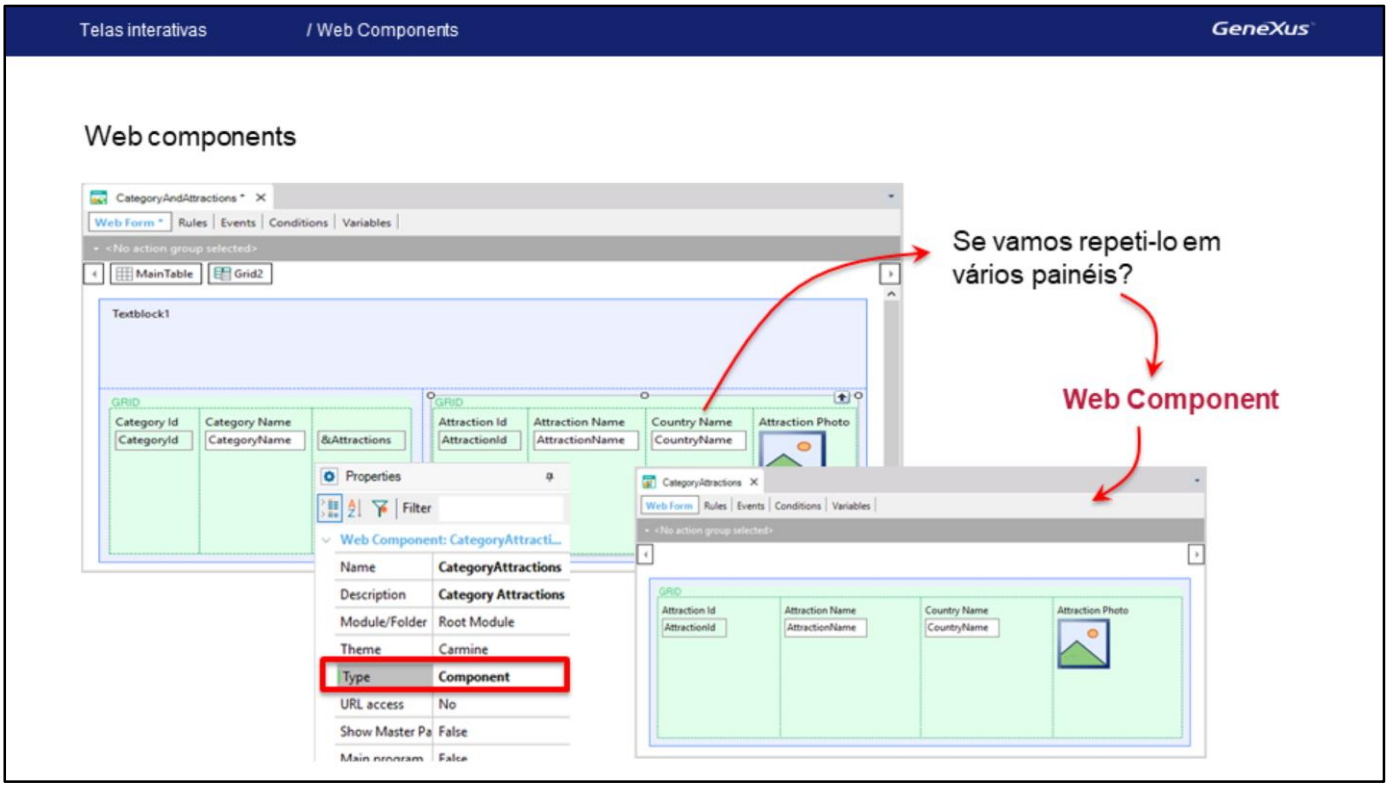
Em um grid Free Style, o valor padrão da propriedade Rows é <unlimited> e este valor não pode ser alterado.

Web components



Em outra aula, vimos que existem três tipos de web panels.

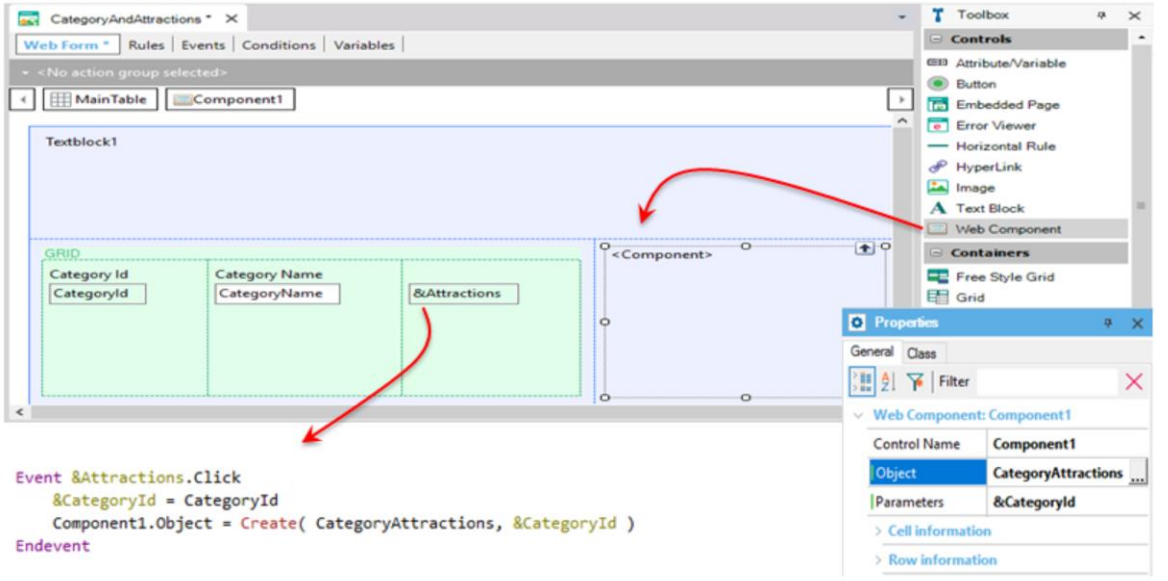
Nestes vídeos, nós só vimos o tipo Web Page, mas há Web panels que podem ser definidos como componentes, para o caso em que uma parte de uma página web precise ser repetida em vários web panels. Ao defini-lo como componente, pode ser inserido em outros objetos com uma tela web.



Por exemplo, suponhamos que o grid que mostra as atrações da categoria escolhida se repita em diversos web panels.
Então, será melhor ter um web panel do tipo Component com ele e toda a sua programação (por exemplo, substituímos a condição que tínhamos no nível do grid, pela regra parm recebendo no atributo CategoryId, que, como sabemos, realiza um filtro automático).

Então...

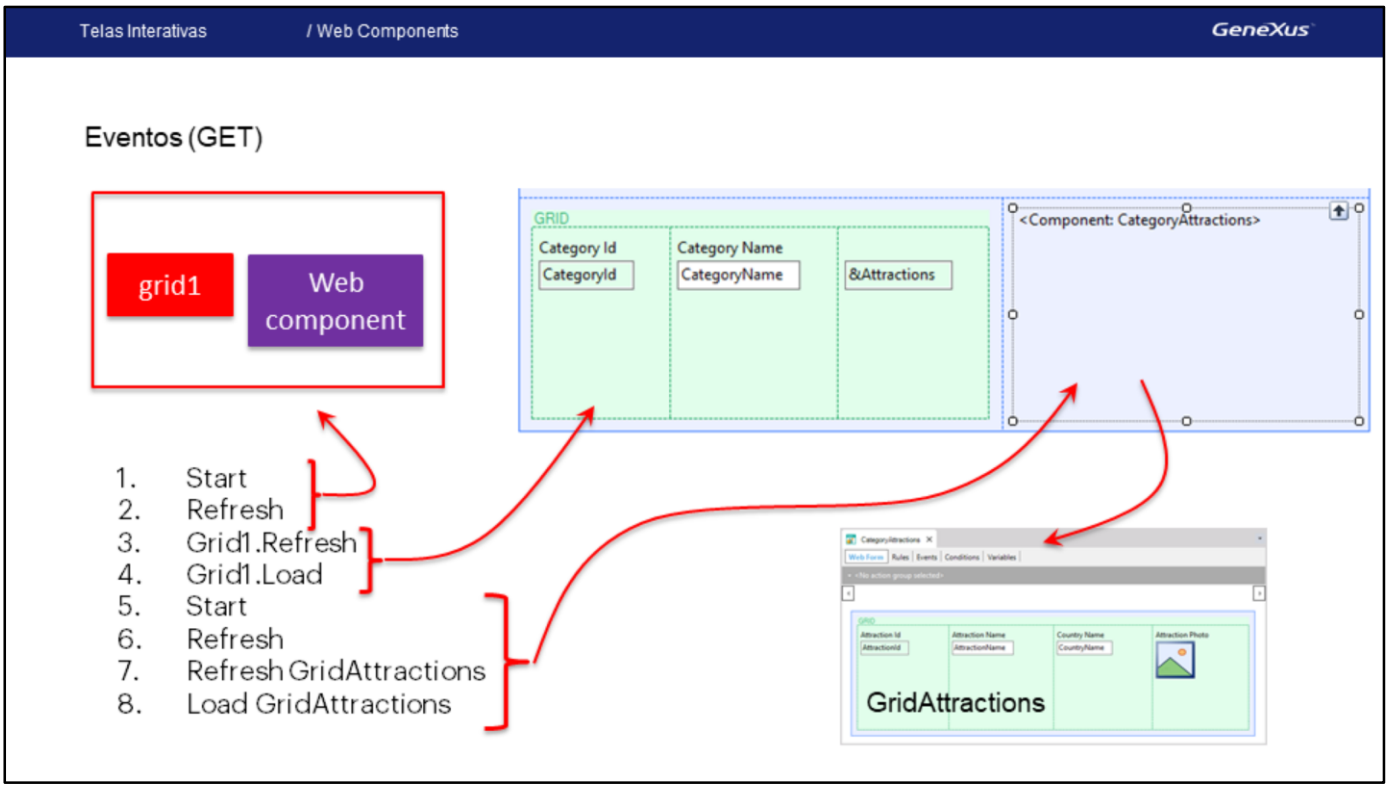
Web components



No web panel onde inicialmente tínhamos o grid de atração, colocamos em seu lugar controle Web Component.
Tudo o que temos a fazer é dizer-lhe qual objeto do tipo Web Component será carregado ali dentro.
Podemos fazê-lo tanto pela programação, quanto pelas propriedades do controle Component, como vemos acima.
Lá, indicamos o objeto que será carregado e o parâmetro que devemos enviar. Nesse caso, o valor da variável &CategoryId.

Mas agora devemos também modificar a programação do clique do grid de categorias. Uma vez que para a variável &CategoryId é dado o valor da categoria da linha, temos que obter que o web component volte a ser criado dentro do controle, para que possa receber o valor da variável por parâmetro. Para isso, usaremos a propriedade Object no nível de programação, como mostrado.

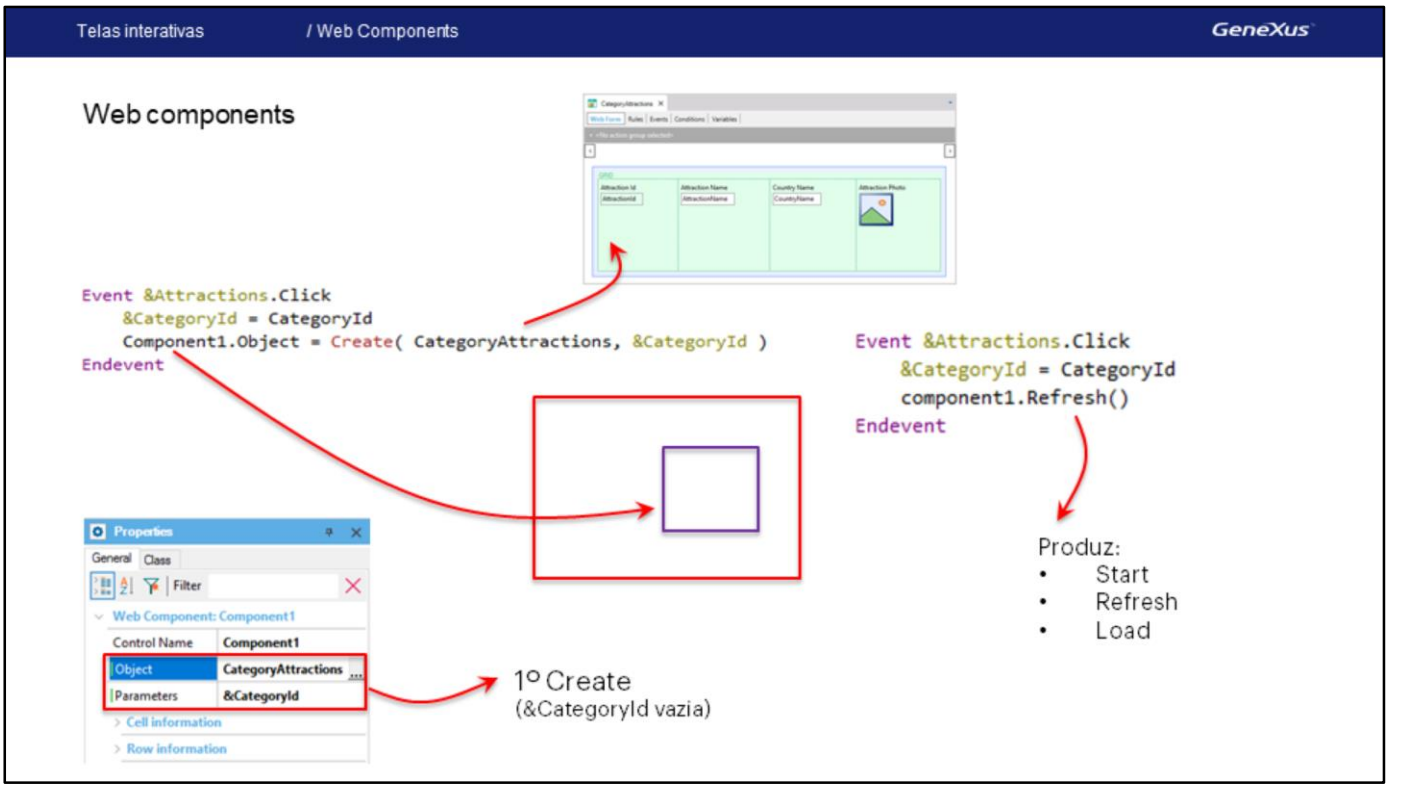
Existe o método Refresh() no nível do componente. Veremos depois de ver os eventos executados no GET.



Se há um web component dentro de um web panel, os eventos que se disparam e a ordem são os esperados. Se dispara:

1. Evento Start do web panel
2. Evento Refresh geral do web panel
3. Evento Refresh da grid1
4. Evento Load da grid1
5. Eventos do componente (Start, Refresh, Refresh e Load de cada grid que é encontrado)

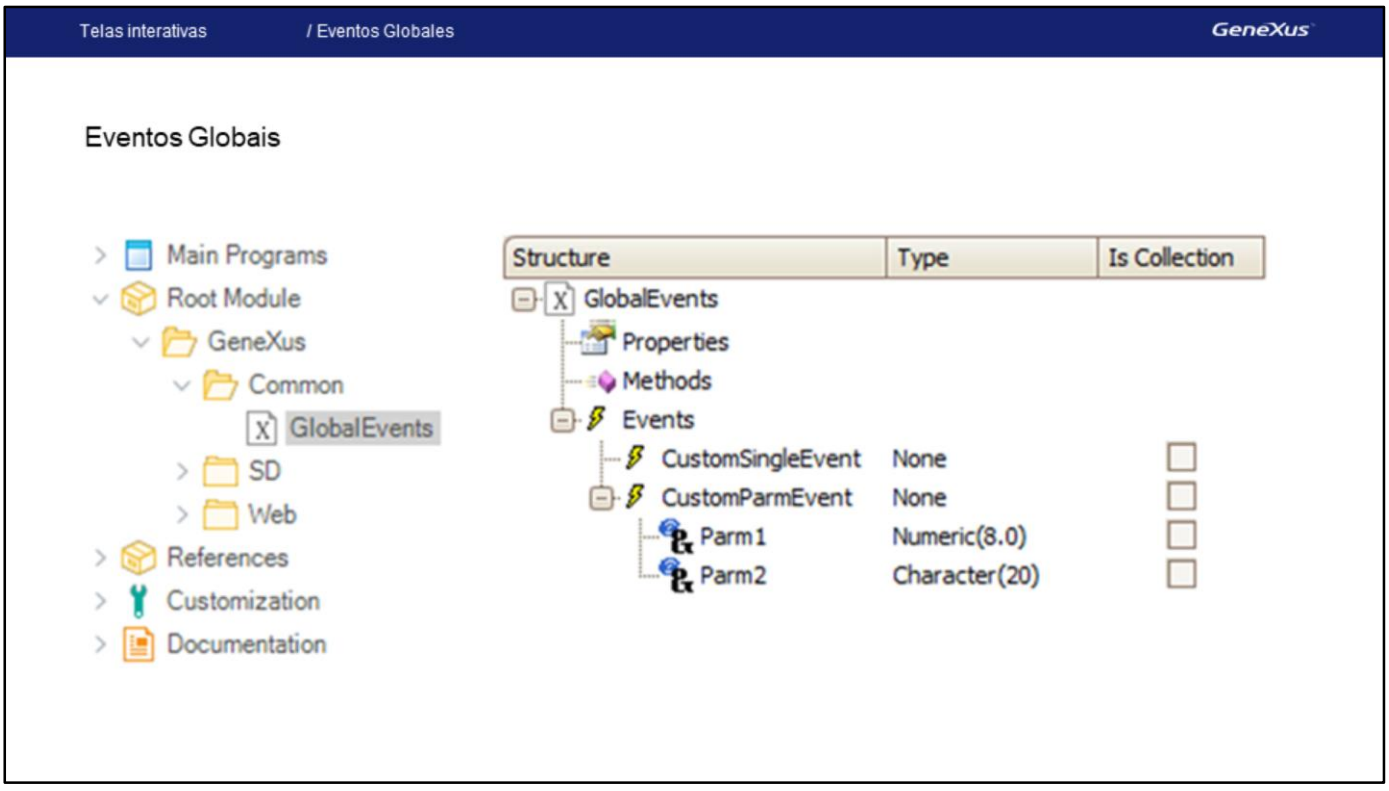
Então, se houver um evento de usuário ou de um controle no nível do componente, esse evento será executado e somente a parte do componente correspondente será atualizada. Aqui você pode ver mais informações: http://wiki.genexus.com/commwiki/servlet/wiki?22472,Event+Execution+Scheme,



O método Refresh no nível do Web component disparará os eventos Start, Refresh e Load do web componente e recarregará toda a sua área de tela. Por que não o usamos? Pelo parâmetro que queremos enviar ao objeto que criamos lá dentro. É que o método Refresh enviará por parâmetro o valor que tinha &CategoryId no momento do Create (que neste caso terá sido quando o GET foi feito, ou seja, quando a tela foi desenhada pela primeira vez). Naquele momento, a variável estava vazia.

Não vamos entrar em detalhes aqui sobre isso.

Eventos Globais

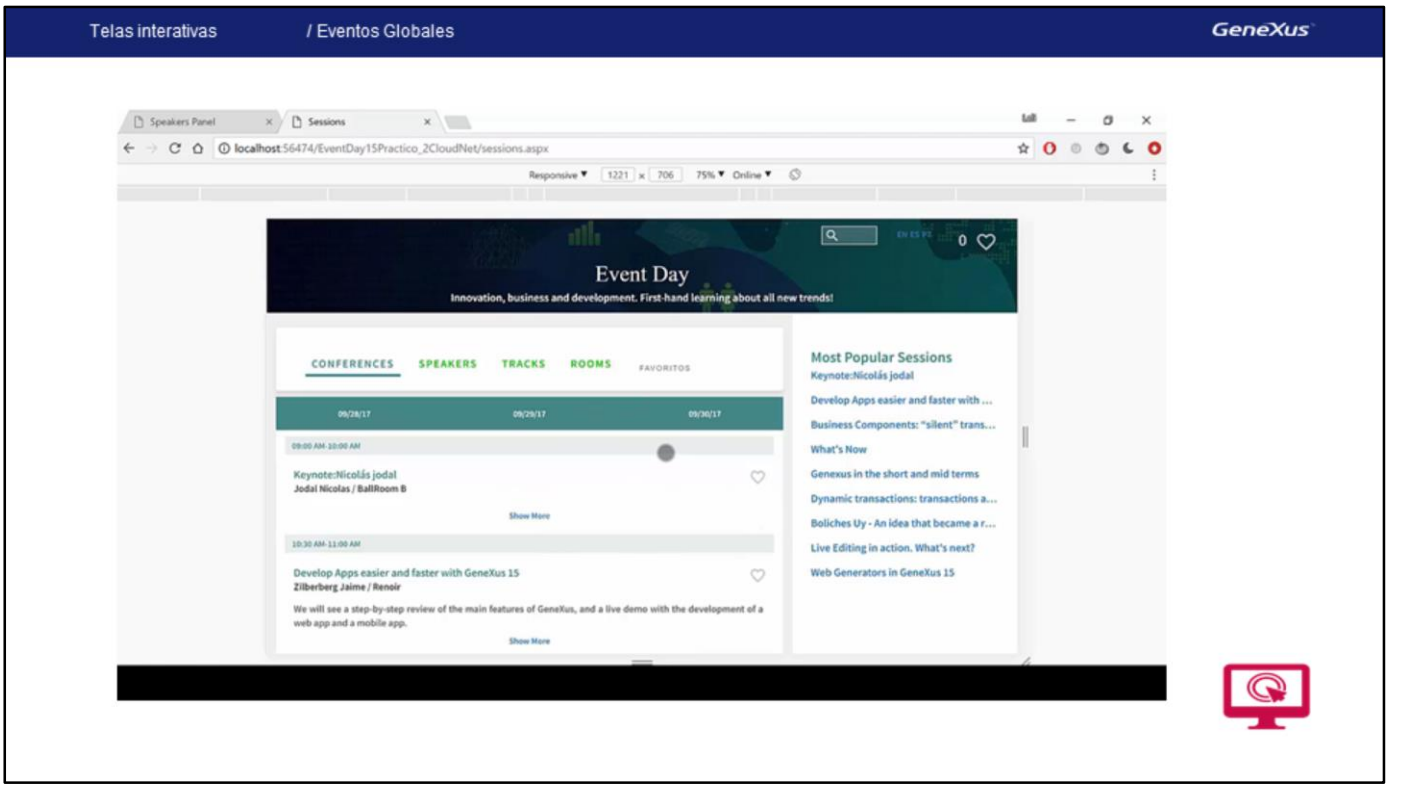


Muitas vezes precisamos implementar uma solução em que uma ação em um componente na tela causa uma reação em outro componente da tela.

Por exemplo, disparar uma ação em um webcomponent quando o usuário seleciona um item de um menu, que está em outro componente.

Como a comunicação entre webcomponents é limitada – a única comunicação viável é de pais para filhos – os eventos globais nos permitem estabelecer comunicação entre webcomponents que não estão relacionados.

Mais Informações: https://wiki.genexus.com/commwiki/servlet/wiki?31167



[DEMO: https://youtu.be/D8akim-gRSQ]

Suponhamos que temos uma aplicação Web para um evento. Se formos ao componente que exibe as conferências, ao clicar no botão Favorito, está sendo atualizado um componente que mostra o total de favoritos, encontrado na master page e que não tem uma relação direta com o componente que o está disparando.
Vamos ver como conseguimos isto no GeneXus

Para isso, temos um novo objeto chamado GlobalEvents. Podemos definir quantos objetos Global Events quisermos, basicamente este é um objeto externo, e poderíamos definir GlobalEventsBackend, GlobalEventfrontend, etc. dependendo de como queremos agrupá-los, é importante lembrar que são globais e, portanto, é importante usá-los com certo critério e de forma ordenada para que não gerem muito ruído.

Temos aqui nosso GlobalEvents que basicamente o que diz é que se atualize a quantidade de favoritos.

Em nosso caso, nosso Evento não recebe parâmetros, mas é possível implementar eventos que sim, se fosse necessário passar parâmetros, seguiria definindo-os, associados ao evento, com seus tipos de dados, etc. e no momento em que é invocado, é responsabilidade do objeto que dispara o evento definir quais são os parâmetros e, em seguida, todos os componentes que implementam o evento ou que escutam esse evento receberão os parâmetros com este formato.

Como disparamos este evento global? No nosso caso, estamos fazendo isso a partir do componente de sessões que, basicamente, quando se clica na imagem de favoritos, é feita a alternancia que mostra a imagem em cinza ou em amarelo e também se dispara o GlobalEvents.

Quem recebe/executa este GlobalEvents? Essa uma decisão nossa, e qualquer componente que faça sentido pode fazê-lo.
No nosso caso, o componente que mostra a quantidade de eventos favoritos está fazendo isso, basicamente este componente implementa GlobalEvents.UpdateFavoritesQuantity e basicamente aqui fará a chamada que necessite para atualizar a informação, que poderia ser um refresh etc.

Em suma, se tivéssemos mais de um componente que precisa se inscrever neste evento, isso é possível. Por exemplo, poderíamos querer exibir informações sobre favoritos em diferentes partes das telas e, quando alguém adiciona um evento ou um conferência a seus favoritos, poderíamos estar atualizando componentes diferentes, cada um deles implementando o Events.UpdateFavoritesQuantity.

Mais Informações: https://wiki.genexus.com/commwiki/servlet/wiki?31167

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentación	wiki.genexus.com
Certificaciones	training.genexus.com/certificaciones