

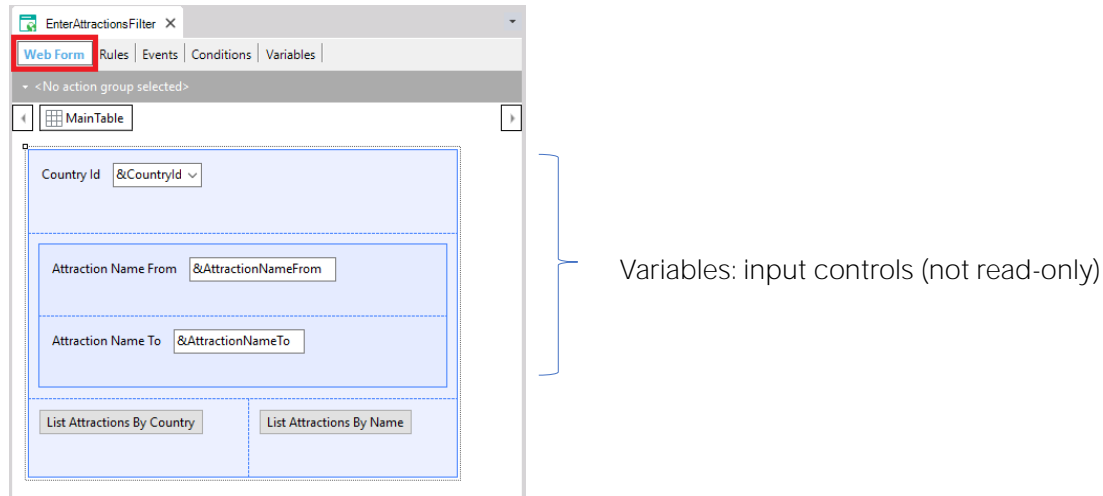
Web panel object. First steps

GeneXus™

Web Panel Object

Implements a highly flexible web screen

Example:



The screenshot shows a web form editor window titled "EnterAttractionsFilter". The "Web Form" tab is selected. The form contains several input controls: a dropdown menu for "Country Id" with variable "&CountryId", two text input fields for "Attraction Name From" and "Attraction Name To" with variables "&AttractionNameFrom" and "&AttractionNameTo", and two buttons labeled "List Attractions By Country" and "List Attractions By Name". A blue dashed border highlights the form area. A bracket on the right side of the form points to the variable names, with the text "Variables: input controls (not read-only)".

Web panels are the most versatile objects provided by GeneXus.

As we've seen in some examples, all web panels have a web form consisting of a web page that enables us to design and provide a variety of functionalities.

In the example we saw that the variables that we include in the web form are enabled for the user to assign values to them. This means that they are input controls; in other words, they are not read-only.

Example: variables in Web panel

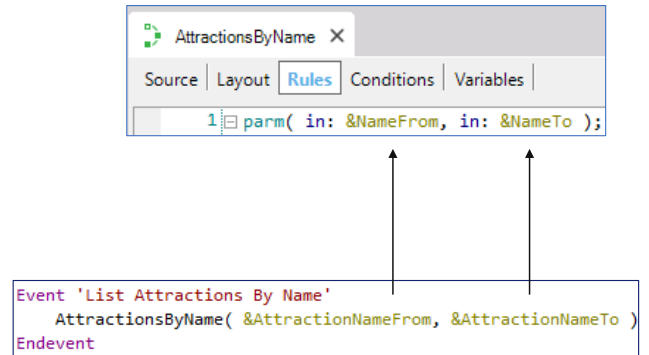
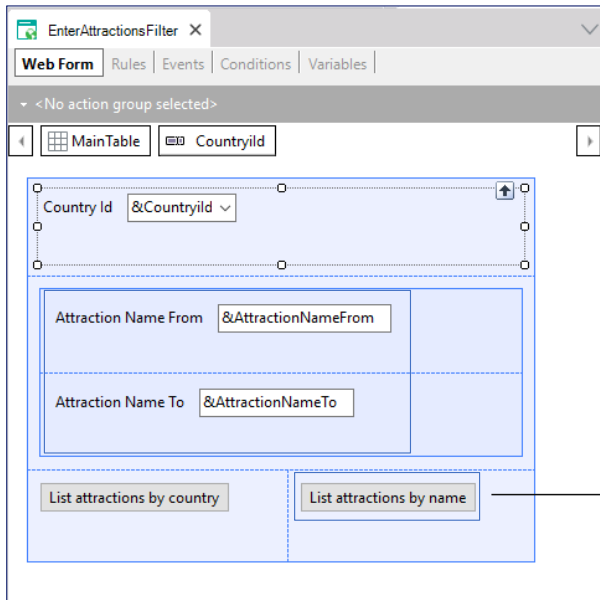
The screenshot displays the GeneXus IDE interface for a web form named 'EnterAttractionsFilter'. The design view shows a 'Country Id' dynamic combo box and two buttons: 'List attractions by country' and 'List attractions by name'. The event for the first button is defined as 'AttractionsList(&CountryId)'. The Properties window on the right shows the control type as 'Dynamic Combo Box' with 'CountryName' as the sort description. A database icon labeled 'Table Country' is shown to the right, with arrows indicating the data source for the control.

Control Name	&CountryId
Attribute	&CountryId
Label Caption	Country Id
ReadOnly	False
Return On Click	False
Appearance	
Label Position	Left
Class	Attribute
Invite Message	
Auto Resize	True
Width	4chr
Tooltip Text	
Control Info	
Control Type	Dynamic Combo Box
Data Source From	Attributes
Item Values	CountryId
Item Descriptions	CountryName
Sort Descriptions	True
Conditions	
Instantiated Attribut	

Specifically, this variable, of the dynamic combo type, expected the user to select one country from those loaded in the combo. After pressing the button “List Attractions By Country”, the associated event would be executed invoking the PDF file containing a list of attractions in that country.

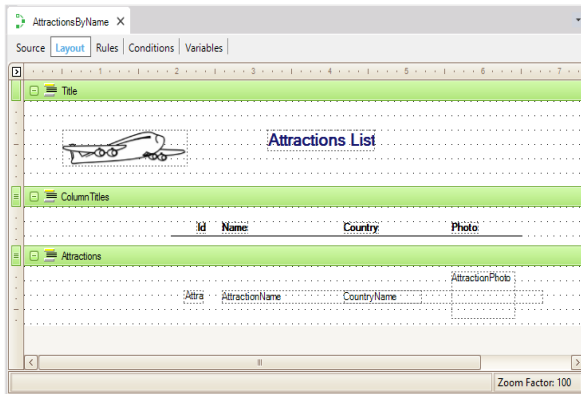
Here, the database is accessed only to load the **combo’s** values.

Example: variables in Web panel

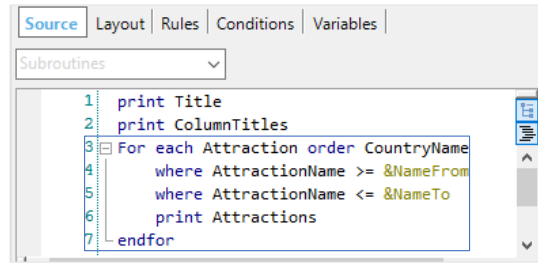
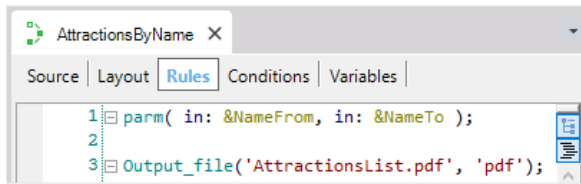


In these other variables, the user would enter a range of attraction names so that when this other button is pressed the PDF list showing the attractions within the range received by parameter would be invoked.

Example:



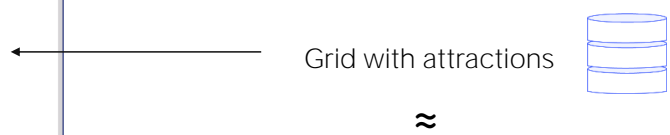
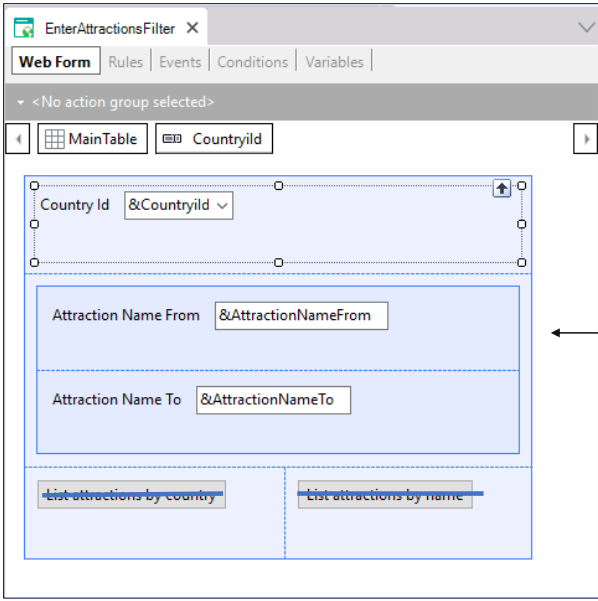
Can we implement this list in the previous web screen, which prompted the user for the filters?



Remember the layout. In the Source, we programmed the database query with the For Each, filtering by name.

But, why do we define these queries through PDF listings instead of doing it directly on the screen where the user is required to enter data for the filters?

Let's change the web panel so that it can query the DB

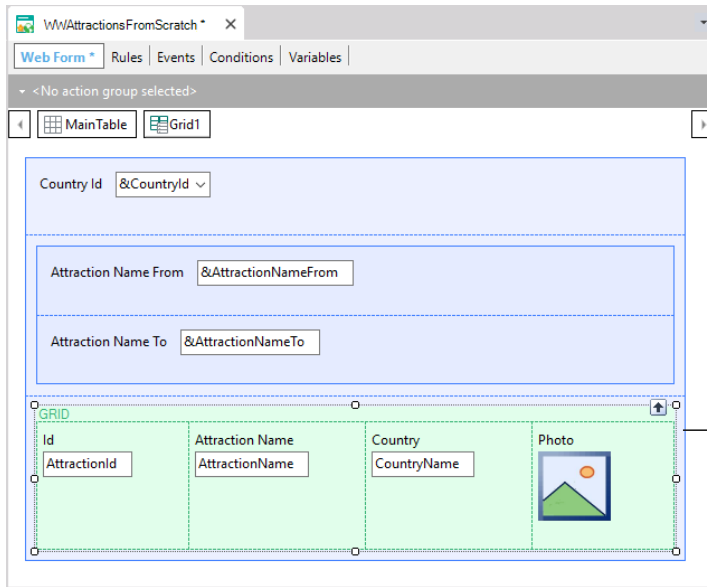


AttractionName	CountryName	AttractionPhoto

Why not add here a grid showing the desired attractions instead of the buttons?

The rows of the grid will be similar to the printblock that shows each attraction.

Web panels: interactive queries to the database



Attributes in web panel are output attributes: read-only



Apart from allowing the definition of variables to be used in actions programmed in buttons, web panels allow us to implement interactive queries to the database, which is in fact their main purpose.

“Interactive” implies that it is possible for the user to enter many different values – in variables – in the web page and then query the database for data matching the values entered, using them as filters, as we will see next.

Let's now save this web panel with a different name. We will be implementing something similar to a Work With element, so we will call it WWAttractionsFromScratch.

We remove the buttons and the associated events because they will no longer be necessary. Now, we insert a control of the grid type below the variables.

A screen opens up in order to select the attributes and/or variables that will be this grid's columns. Since we want to show the same we showed on the PDF listings, we select the AttractionId, AttractionName, AttractionPhoto and CountryName attributes and then press OK. We will then see that a grid has been created with these columns. We may change the column titles by editing the properties of each attribute comprised in the grid columns.

The attributes in a web panel form are, by default, output attributes. That is to say that they are readonly. This means that GeneXus understands that it must go to the database to search for their value and show it to the user.








Web panels: interactive queries to the database

Application Name

Country Id

Attraction Name From

Attraction Name To

Attraction Id	Attraction Name	Country Name	Attraction Photo
1	Louvre Museum	France	
2	Great Wall	China	
3	Eiffel Tower	France	
4	Christ the Redeemer	Brazil	
5	Smithsonian Institute	United States	
6	Matisse Museum	France	
7	Forbidden City	China	

Let's press F5 to run our new web panel, just as we have it so far.

We can see that all the attractions have been printed, with the data we indicated (ID, name, country and photo). We will also see that they have been ordered by AttractionId.

Grid with attributes is the equivalent to a For each command

The screenshot shows the GeneXus IDE interface. On the left, a web form is displayed with a grid component. The grid is labeled "Transacción base" and contains four columns: "Attraction Id", "Attraction Name", "Country Name", and "Attraction Photo". Above the grid, there are form fields for "Country Id", "Attraction Name From", and "Attraction Name To". Below the grid, a code editor shows a "For each" loop with the following code:

```

1 print Title
2 print ColumnTitles
3 For each Attraction where CountryName
4   print AttractionName
5   print Attractions
6 endfor

```

On the right, the Properties window for the "Grid: Grid1" control is shown. The "Base Trn" property is set to "Attraction". Other properties include "Class" (Grid), "Auto Resize" (True), "Rows" (0), and "Header" (empty). A database icon is visible to the right of the Properties window.

The mere grid with these attributes drove GeneXus to understand that it should go to the database to navigate the Attraction table, and then access Country to bring the country of the attraction, as we did with the For Each command (without the order and where clauses).

We can see that one of the grid properties is called Base Trn. This property is similar to the base transaction of the For Each command. In fact, to make sure that the Attraction base table is selected for the grid, as we want, we should indicate the base transaction, just like for the For Each command.

Grid with attributes is the equivalent to a For each command

The screenshot displays the GeneXus IDE interface. On the left, a web form titled 'Web Form' contains three input fields: 'Country Id' with a dropdown menu, 'Attraction Name From', and 'Attraction Name To'. Below these is a grid named 'Grid1' with four columns: 'Attraction Id', 'Attraction Name', 'Country Name', and 'Attraction Photo'. At the bottom, a code window titled 'AttractionsByName' shows the following code:

```

1: print Title
2: print ColumnTitles
3: For each Attraction order CountryName
4:   where AttractionName = &AttractionNameFrom
5:   where AttractionName <= &AttractionNameTo
6:   print Attractions
7: endfor

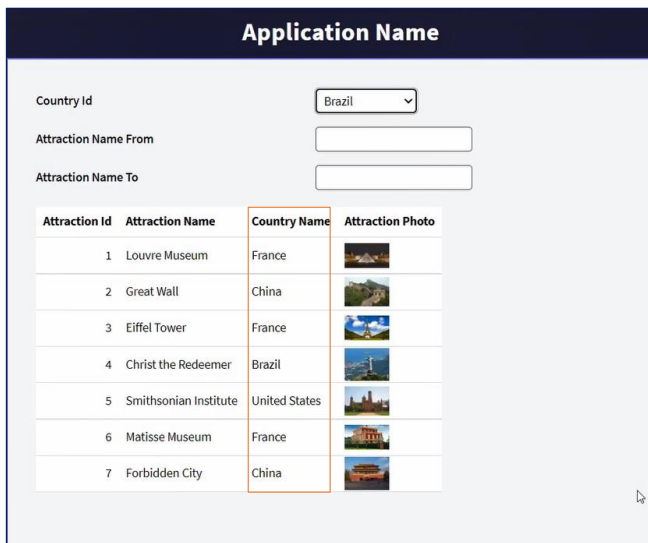
```

On the right, the 'Properties' window for 'Grid-Grid1' is open, showing the 'Order' property set to 'CountryName'. A small dialog box titled 'Grid1's Order' is also visible, showing the text 'CountryName'.

In addition, note that an Order property is available for the grid. This property corresponds to the Order clause of the For Each command.

So, if we want to order by country name, as in the listing, in the Order property we have to write the CountryName attribute.

Grid with attributes is the equivalent to a For each command



We press F5, and we will see that the grid is now ordered by country name.

Note that the navigation list of the web panel here indicates the navigation that has to be made to load the combo box of the &CountryId variable, which we have not used at all for the time being.

And, here, it indicates the navigation that will have to be made to load the grid. The list for this loading is identical to the list for a For Each command. As we can see, it has chosen the Attraction table, which will be run through according to the CountryName attribute of the Country table, the attribute that we specified in the Order property. That is, it will have to access that table to order how the Attraction table will be run through, and to show the CountryName of each record of the Attraction base table. It will run through the entire Attraction table.

Filter grid data

The screenshot displays the GeneXus IDE interface for a web form named 'Web Form'. The main workspace shows a form layout with several input fields: 'Country Id' (with a dropdown arrow), 'Attraction Name From', and 'Attraction Name To'. Below these is a grid component named 'Grid1'. The grid has four columns: 'Attraction Id', 'Attraction Name', 'Country Name', and 'Attraction Photo'. An arrow points from the 'Conditions' property in the Properties window to the grid.

The Properties window on the right shows the configuration for 'Grid: Grid1'. The 'Conditions' property is set to 'CountryId = &CountryId;'. Other visible properties include 'Control Name' (Grid1), 'Base Trn' (Attraction), 'Order' (CountryName), 'Unique', 'Save State' (False), 'Data Selector' (none), 'Class' (Grid), 'Auto Resize' (True), 'Rows' (0), 'Cell Padding' (1), and 'Cell Spacing' (2).

The 'AttractionList' subroutines window at the bottom shows the following code:

```

1 print Title
2 print ColumnTitles
3 for each Attraction order CountryName
4   where CountryId = &CountryId
5   print Attractions
6 endfor
7

```


So far, we haven't done anything with variables. But we wanted to use them to filter the data displayed in the grid, by country and as well as by attraction name.

In AttractionList we filtered by country. How do we indicate this filter for the grid? Through the Conditions property.




Filter grid data

The image displays two screenshots of an application interface, both titled "Application Name".

Left Screenshot: The "Country Id" dropdown menu is set to "Brazil". Below it are two empty text input fields for "Attraction Name From" and "Attraction Name To". The grid below shows one row of data:

Attraction Id	Attraction Name	Country Name	Attraction Photo
1	Christ the Redeemer	Brazil	

Right Screenshot: The "Country Id" dropdown menu is set to "France". Below it are two empty text input fields for "Attraction Name From" and "Attraction Name To". The grid below shows three rows of data:

Attraction Id	Attraction Name	Country Name	Attraction Photo
1	Louvre Museum	France	
3	Eiffel Tower	France	
6	Matisse Museum	France	

Note that, by default, the combo box takes the value Brazil, and that the grid only shows an attraction in Brazil.

If we select France, we see that the screen is refreshed and the grid is loaded again, this time with the attractions in France.

Conditional filtering of grid data

The screenshot displays the GeneXus IDE interface. The top window, titled 'Web Form', shows a form layout with a 'Country Id' dropdown menu, two text boxes for 'Attraction Name From' and 'Attraction Name To', and a grid of attraction data. The grid has columns for 'Attraction Id', 'Attraction Name', 'Country Name', and 'Attraction Photo'. The bottom window, titled 'AttractionsByName', shows the source code for the grid, which includes a 'for each' loop filtering attractions based on the 'Attraction Name From' and 'Attraction Name To' values.

The Properties window on the right shows the configuration for the 'Country Id' dropdown menu. The 'Control Type' is 'Dynamic Combo Box'. The 'Empty Item' property is set to 'True', and the 'Empty Item Text' property is set to 'GX_EmptyItemText'. An arrow points from the text 'Value: \"(None)\"' to the 'Empty Item Text' property.

We will probably want the combo to be first displayed without a selected value, with the attractions of all countries displayed.

To do this we must edit the combo box properties... and set the Empty Item property to True. This will add a **“(none)”** option to the combo. It will correspond to an empty value.

Conditional filtering of grid data

The screenshot shows the GeneXus IDE interface for a web form titled 'Web Form'. The form contains a dropdown menu for 'Country Id' and a grid below it. The grid has four columns: 'Attraction Id', 'Attraction Name', 'Country Name', and 'Attraction Photo'. The Properties window for the grid shows the condition 'CountryId = &CountryId'. A dialog box for 'Grid's Conditions' shows the condition 'CountryId = &CountryId when not &CountryId.IsEmpty();'.

The Source window shows the following code:

```

1 print Title
2 print ColumnTitles
3 For each Attraction order CountryName
4   where CountryId = &CountryId
5   print Attractions
6 endfor
7

```

So, we open the Conditions property and indicate that we want to have this condition applied only when the combo's value is not empty. When it is empty, the condition should not be applied.

Conditional filtering of grid data

Application Name

Country Id (None) ▾

Attraction Name From

Attraction Name To

Attraction Id	Attraction Name	Country Name	Attraction Photo
1	Louvre Museum	France	
2	Great Wall	China	
3	Eiffel Tower	France	
4	Christ the Redeemer	Brazil	
5	Smithsonian Institute	United States	
6	Matisse Museum	France	
7	Forbidden City	China	

Application Name

Country Id France ▾

Attraction Name From

Attraction Name To

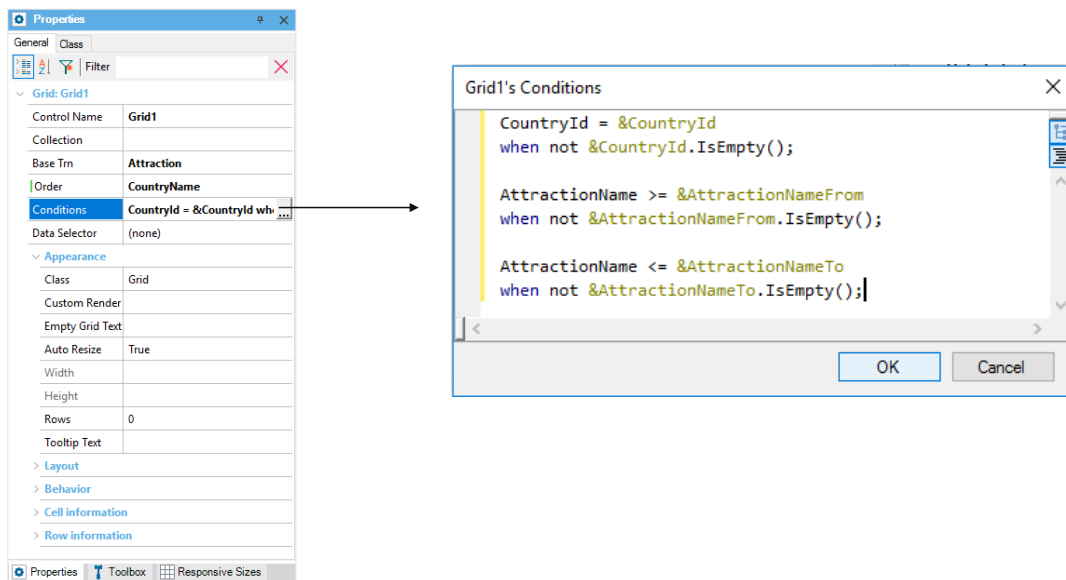
Attraction Id	Attraction Name	Country Name	Attraction Photo
1	Louvre Museum	France	
3	Eiffel Tower	France	
6	Matisse Museum	France	

We run it...

...and see how the (None) value is displayed in the combo. In addition, in this case there is no filtering for the attractions, and all of them are displayed.

If now we choose, France, for example, since the variable's value is not empty, the filter is applied and the attractions in France are displayed.

Filtering grid data by multiple conditions



We would also have to add the filters by attraction name that we want added to the other filter. So, if in the listing we filtered in the For Each command using these two Where clauses... we will add them to the grid as conditions.

AttractionName greater than or equal to the value of the &AttractionNameFrom variable of the form, which may be entered by the user. Once again, if the user doesn't enter a value in the variable, we will not want this filter to be applied. So, we use the When clause. This clause may also be used in the Where clause of the For Each command, in a completely analogous way.

We add the other filter.

Compatible orders and filters

The image displays three screenshots of a web application interface, illustrating different filter and order settings for a grid of attractions. The interface includes a 'Country Id' dropdown menu, 'Attraction Name From' and 'Attraction Name To' text input fields, and a grid of attraction records. A 'Grid's Order' dialog box is also shown, displaying the order expression: `CountryId, AttractionName when not &CountryId.IsEmpty()`.

Screenshot 1 (Left): Country Id: (None), Attraction Name From: a, Attraction Name To: f. Grid shows 2 records: Christ the Redeemer (Brazil) and Eiffel tower (France).

Screenshot 2 (Middle): Country Id: France. Grid shows 7 records: Louvre Museum (France), Great Wall (China), Eiffel Tower (France), Christ the Redeemer (Brazil), Smithsonian Institute (United States), Matisse Museum (France), and Forbidden City (China).

Screenshot 3 (Right): Country Id: France, Attraction Name From: a. Grid shows 3 records: Louvre Museum (France), Eiffel Tower (France), and Matisse Museum (France).

Grid's Order Dialog: CountryId, AttractionName when not &CountryId.IsEmpty()

We run again. And now we choose to see the attractions between A and F.

For the case when the user selects a country, we may instruct the web panel to then sort the information by CountryId, and inside CountryId by AttractionName, or otherwise, by AttractionName. This is meant to optimize the search of table records.

To do it, we edit the Order property of the grid and write the order first, conditioned to the selection, by the user, of a country in the combo. In this case, the data will be filtered by that country, and also the attractions will be listed in alphabetical order for that country. If the user left the combo empty, with the **"(none)"** value, then the following order -by AttractionName- will be selected.

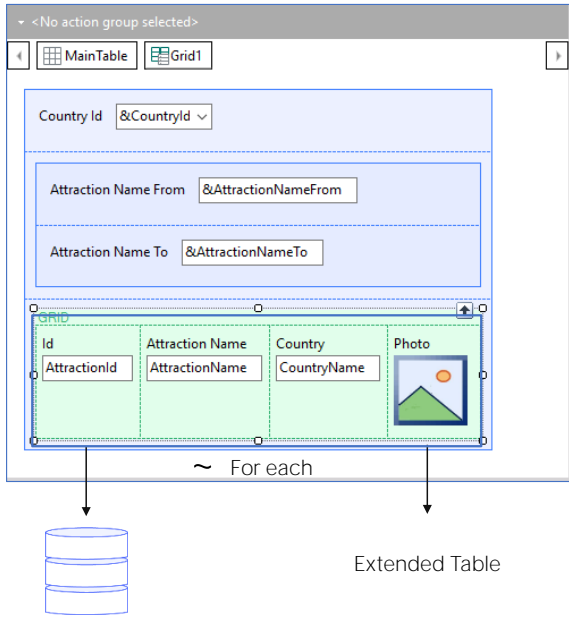
We will not go into further detail here. The purpose of mentioning this was just to show that we can also condition the way in which we want to have information ordered. It works exactly like in a For Each command.

We run it...Here, it ordered by AttractionName. And if we select France, it will order by France's ID, and inside it, by AttractionName.

In sum, attractions will always be shown in alphabetical order.

If, within France's attractions, we want to see those between the letters A and F, we will see that the grid will load filtering by the three conditions we had added.

In summary



The Properties window for 'Grid: Grid1' shows the following properties:

- Control Name: Grid1
- Collection: (empty)
- Base Trn: Attraction
- Order: CountryId, Attraction...
- Conditions: CountryId = &Country...
- Data Selector: (none)

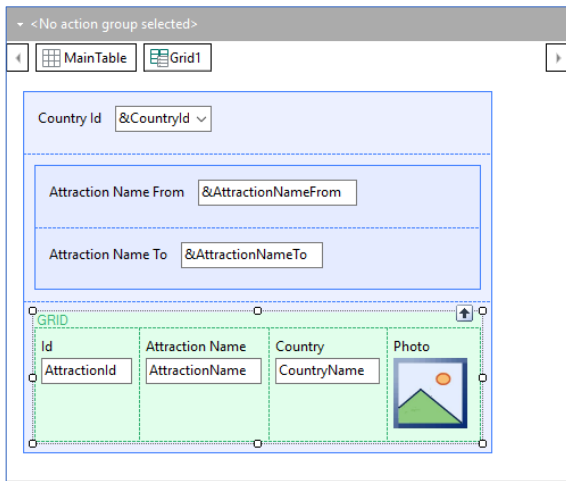
Base Table
~
For each
Order
Where

We implemented a web panel where, in addition to including some variables for which the user enters values, we also inserted a Grid control with attributes.

The attributes correspond to information in the database, so GeneXus understands that it has to search for it. A grid with attributes is like a For Each, so, we have the Base Trn property, as in the case of a For Each, to specify the level of the transaction whose associated table we want to run through. This table is called grid base table. When we don't specify it, as it may also be the case with a For Each command, then GeneXus will infer it based on the attributes used. However, we will not be considering such case at this point.

All the grid attributes will have to belong, as in a For Each command, to the extended table of that base table. Just like in a For Each we order data with the Order clause and filter the data to be returned by the query with one or several Where clauses, in order to do the same with the grid we have, respectively, the Order and Conditions properties.

Grid loading



```

For each
  Main_Code ← Print Attractions
  ...
endfor

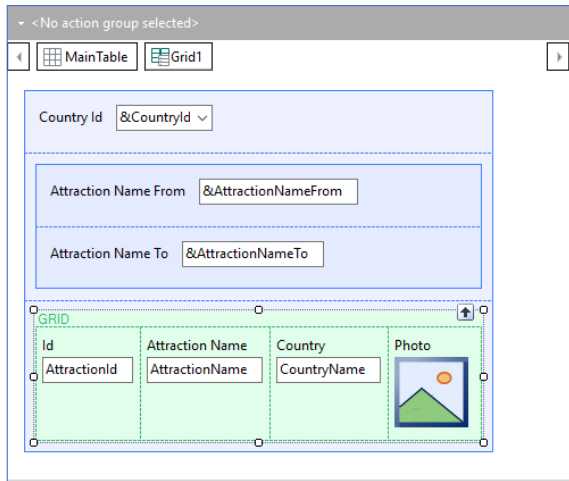
```

Attractions			
AttractionPhoto	CountryName	AttractionName	AttractionId
...

In a For Each command we program what we want to do with each record that meets the conditions inside its body.

For example, in the list of tourist attractions, the print Attraction command sends for print, in the output, what in this case would be the grid line, but in the case of the grid, we need not indicate it because it is done automatically.

Grid loading: Load event

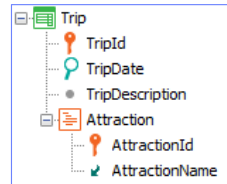


```

For each
  &trips = Count(TripDate)
  Print Attractions
endfor

```

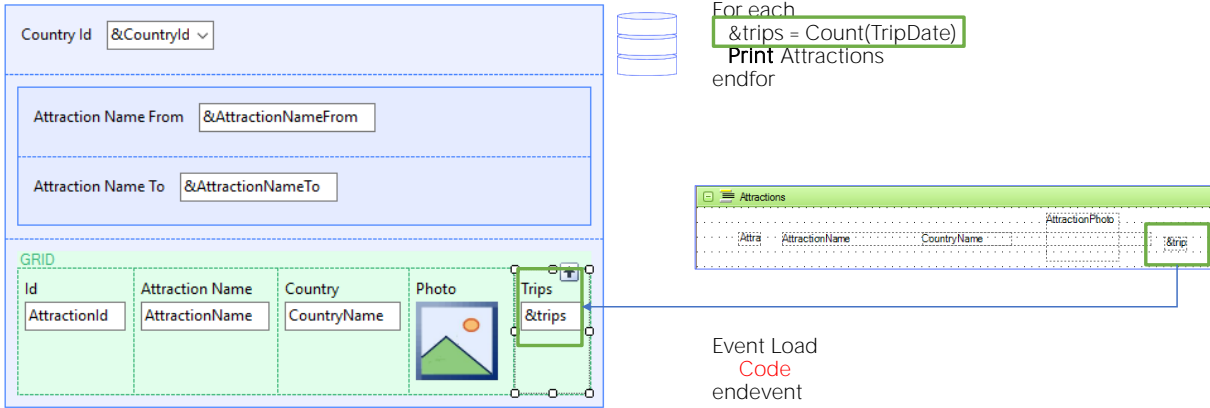
Attractions				
AttractionName	CountryName	AttractionPhoto	&trips	
AttractionName	CountryName	AttractionPhoto		



But let's suppose, for example, that we have a Trip transaction that records the trips offered by the travel agency. In a very simplified way, suppose that for every trip we only record the date on which it will take place, and its description; and then the tourist attractions that will be visited during that trip are recorded. Let's also suppose that in the list of tourist attractions we also want to see the number of trips associated with each attraction.

To do so, we only needed to define a numeric variable `&trips`, and inside the body of the For Each command (that is to say, when the For Each command is positioned on the record of its base table about to be processed) assign it the result of counting the trips of that attraction, to then include that variable in the print block.

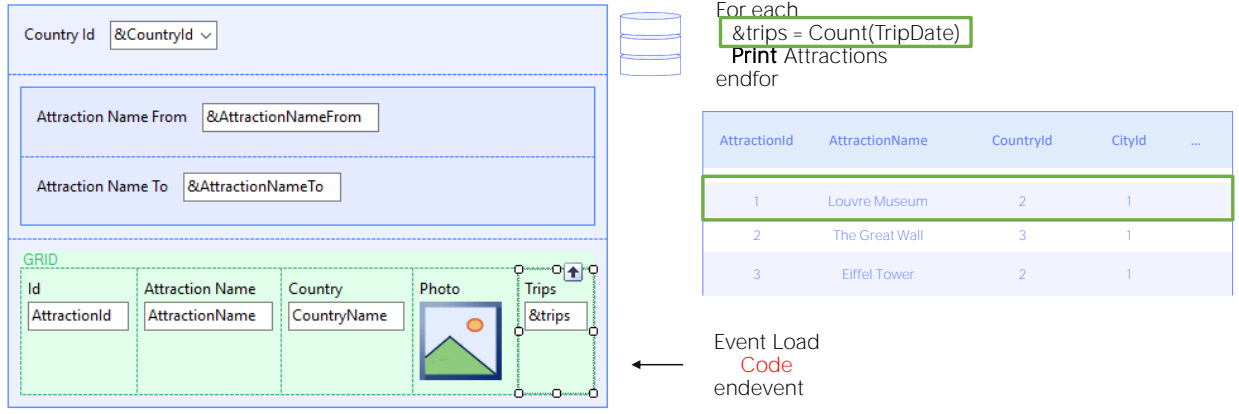
Grid loading: Load event



To do the same in the web panel, we right-click on the grid and select Insert Attribute/Variable. We create the &trips variables, with type Numeric(4,0), and we move it to the position where we want inside the grid. This corresponds to having inserted the variable in the print block. But, where do we indicate how the calculation is done? In the case of the For Each command, we did inside its body. And where do we do it in this case?

To do this we have the system's Load event.

Grid loading: Load event

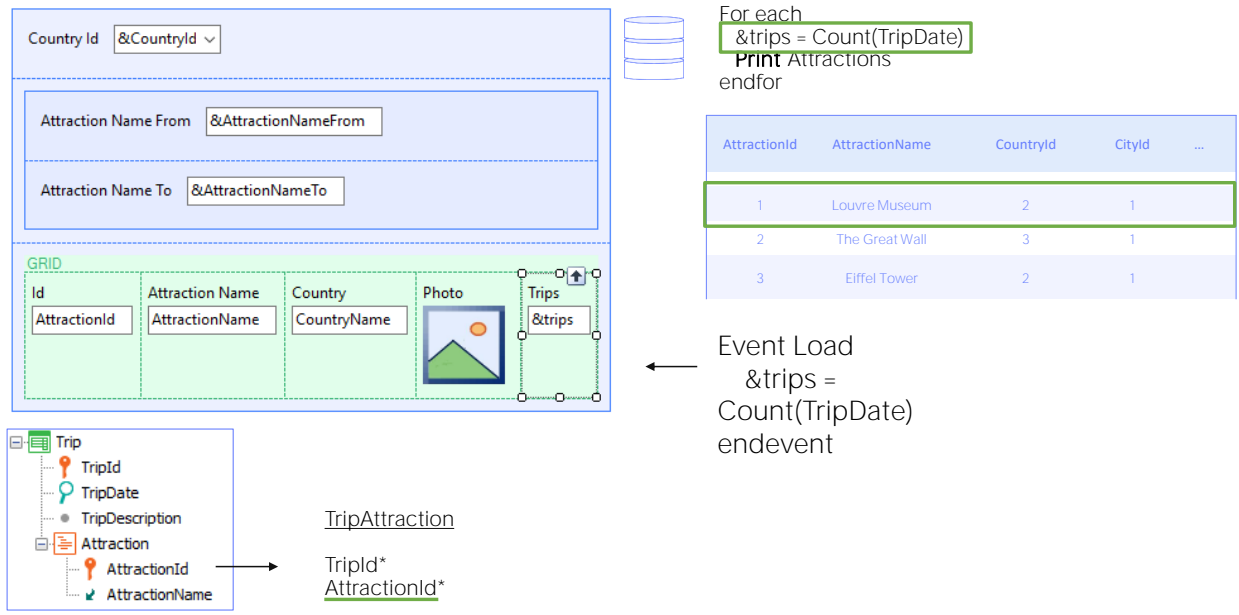


Inside it we program what we want executed when we're positioned on a record of the grid base table, right before the corresponding line is loaded in the grid.

In this case, there is where we would assign a value to the &Trips variable.

The Load event will be automatically executed for every record of the grid base table that meets the filtering conditions, immediately prior to the addition of the line to the grid.

Grid loading: Load event



That's why, when its code is executed, we know we're working with a record from the base table and its extended table; and this inline formula will not count all the trips. Instead, it will count only those from the TripAttraction table that correspond to this AttractionId, that of the Attraction record that we're about to load in the grid.

Note that even though the TripDate attribute belongs in the Trip table, GeneXus will not choose the Trip table as the **formula's** base table, but the TripAttraction table. We will not go into this here, but TripDate is in the extended table of TripAttraction, table which has a relationship to the Attraction table. GeneXus looked for a table which allowed relating the data.








Variable outside the grid to calculate the total

Application Name

Country Id (None) ▾

Attraction Name From

Attraction Name To

Attraction Id	Attraction Name	Country Name	Attraction Photo	Trips
4	Christ the Redeemer	Brazil		2
3	Eiffel Tower	France		2
7	Forbidden City	China		0
2	Great Wall	China		0
1	Louvre Museum	France		0
6	Matisse Museum	France		1
5	Smithsonian Institute	United States		1

Total Trips: 6 ← Variable outside the grid to total?

Let's implement it in GeneXus. We've already created the Trip transaction. Now we go to the events section of the Web Panel. In this combo box, we're offered predefined events; that is to say, system events that are generated at specific moments, and for which we may program code.

Among them is the Load event. Here we will program what we want executed every time we're positioned on a record from the Attraction table, before loading the line in the grid.

We run it...

Now we want to add the sum of trips in which the attractions displayed on the grid are included. That is to say, the sum of the values in this column:


Variable outside the grid to calculate the total

Country Id ▾

Attraction Name From

Attraction Name To

GRID

Id	Attraction Name	Country	Photo	Trips
<input type="text" value="AttractionId"/>	<input type="text" value="AttractionName"/>	<input type="text" value="CountryName"/>		<input type="text" value="Trips"/>

Total Trips

AttractionId	AttractionName	CountryId	CityId	...
1	Louvre Museum	2	1	
2	The Great Wall	3	1	
3	Eiffel Tower	2	1	

Event Load

```

&trips = Count(TripDate)
&totaltrips = &totaltrips +
&trips
endevent
&trips 
&totalTrips 

```

An efficient way of calculating the value that the variable will have to display is... every time a line is to be loaded in the grid, we should add the value of the &Trips variable of that line, to the value calculated so far in &totalTrips.

This means that, in the Load event, after calculating the value of &trips, we assign to &totalTrips the value that it contains so far plus the value of the &trips variable.


Variable outside the grid to calculate the total

Country Id ▾

Attraction Name From

Attraction Name To

GRID

Id	Attraction Name	Country	Photo	Trips
<input type="text" value="AttractionId"/>	<input type="text" value="AttractionName"/>	<input type="text" value="CountryName"/>		<input type="text" value="&trips"/>

Total Trips

AttractionId	AttractionName	CountryId	CityId	...
1	Louvre Museum	2	1	
2	The Great Wall	3	1	
3	Eiffel Tower	2	1	

Event Load

```
&trips = Count(TripDate)
&totaltrips = &totaltrips +
&trips
endevent
&trips 
&totalTrips 
```

For the second line to be loaded, the value of &trips is calculated and &totalTrips will contain the previous value, to which the value of the &trips variable will be added for this second line, and so on.

Once the last line has been loaded, the &totalTrips variable will have the desired value.

Refresh

Application Name

Country Id: (None) v

Attraction Name From:

Attraction Name To:

Attraction Id	Attraction Name	Country Name	Attraction Photo	Trips
4	Christ the Redeemer	Brazil		2
3	Eiffel Tower	France		2
7	Forbidden City	China		0
2	Great Wall	China		0
1	Louvre Museum	France		0
6	Matisse Museum	France		1
5	Smithsonian Institute	United States		1
Total Trips				6

Application Name

Country Id: France v

Attraction Name From:

Attraction Name To:

Attraction Id	Attraction Name	Country Name	Attraction Photo	Trips
3	Eiffel Tower	France		2
1	Louvre Museum	France		0
6	Matisse Museum	France		1
Total Trips				9

```

Event Load
  &trips = count( TripDate )
  &totalTrips = &totalTrips + &trips
- Endevent

Event Refresh
  &totalTrips = 0
- Endevent

```

Why does it show 9 instead of 3?

Let's run it.

We can see two things: first, the sum is being made correctly; second: because it is a variable, it is an input method where the user may change the value, and this doesn't make sense. So, the first thing to do is set it as Readonly.

To do so, we click on the variable in the form and, among its properties, we change its Readonly variable by setting it to True.

We may find it odd that &trips, which is also a variable, is shown as Readonly even when we did nothing in that sense. When no events have been programmed at the line level and when they are not run through by code, grid variables will always be Readonly. We will be seeing this in detail further ahead.

Let's also see what happens if, for example, we filter by France

Instead of showing a total of 3, it shows 9, which is the sum of the value displayed before -6- plus the 3 that it should be displaying now. Why did this happen?

After changing one of the filter variables, the web panel loaded the grid again, meaning that it queried the Attraction table in the database again, and it executed the Load event again for every record that met the filters criteria. The problem is that the &totalTrips variable should have been reset, returning it to

zero, before the start of the grid load.

Where do we do this? In the Refresh event.

Note that the order in which the events are written is not important. Here we will only indicate the code that will be run when each one of them is triggered.

Refresh Event

System event: It occurs every time the web panel is to be loaded, before going to the DB to search for information to load the grid (right before running the Load event for each line to be loaded).

Attraction Id	Attraction Name	Country Name	Attraction Photo	Trips
4	Christ the Redeemer	Brazil		2
3	Eiffel Tower	France		2
7	Forbidden City	China		0
2	Great Wall	China		0
1	Louvre Museum	France		0
6	Matisse Museum	France		1
5	Smithsonian Institute	United States		1

```
Event Load
    &trips = count( TripDate )
    &totalTrips = &totalTrips + &trips
Endevent

Event Refresh
    &totalTrips = 0
Endevent
```

Let's now press F5.

We will see that the total number of trips is now Readonly. To run this web panel for the first time, three events were triggered in sequence: the Start event, which is run only when the web panel is opened for the first time, the Refresh event, which set the variable to zero, and the Load event, as many times as lines were to be loaded in the grid. In this case, they were 7.

Refresh Event




Changing the value of the variable involved in the Conditions causes the Refresh and Load events to be triggered again (not Start).

Application Name

Country Id:

Attraction Name From:

Attraction Name To:

Attraction Id	Attraction Name	Country Name	Attraction Photo	Trips
3	Eiffel Tower	France		2
1	Louvre Museum	France		0
6	Matisse Museum	France		1

Total Trips: 3

Event Refresh

Event Load 3 times

```

Event Load
  &trips = count( TripDate )
  &totalTrips = &totalTrips + &trips
Endevent

Event Refresh
  &totalTrips = 0
Endevent

```

Now, if we filter by a country, such as France, we see that the number of trips is correctly calculated.

Changing the value of a variable that affects the conditions that must be met by the records to be loaded in the grid causes the Refresh event to be triggered again (and therefore, the &totalTrips variable is reset to zero), and the database is accessed to filter and load the records in the grid again. Therefore, the Load event is triggered again for every attraction in France that is to be loaded.

Work With Attractions of the Pattern

Actions over data:
Insert,
Update,
Delete
an attraction

Attraction

```
parm(in:&Mode, in:&AttractionId);
AttractionId = &AttractionId if not &AttractionId.IsEmpty();
```

Application Name

Attractions

INSERT

Q Name

Id	Name	Country Name	Category Name	Photo	City Name		
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro	UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris	UPDATE	DELETE
7	Forbidden City	China	Monument		Beijing	UPDATE	DELETE
2	Great Wall	China	Tourist site		Beijing	UPDATE	DELETE
1	Louvre Museum	France	Museum		Paris	UPDATE	DELETE
6	Matisse Museum	France	Museum		Nice	UPDATE	DELETE
5	Smithsonian Institute	United States	Museum		Washington	UPDATE	DELETE

Enum Values

Insert, Insert, INS; Update, Update, UPD; Delete, Delete, DLT; Display, Display, DSP

If we look at the web panel that we have implemented so far, we can see that it now looks like the object created by the WorkWith Pattern applied to the Attraction transaction.

Obviously, this object was a web panel.

What's most interesting is that, in addition to allowing us to filter the grid data, the WorkWith includes the option to run actions over the data. For example, updating an attraction's data or deleting the attraction or adding a new attraction.

To this end, the pattern inserted two controls at the grid line level, and another one outside. In any of these three cases, the action associated with each control consists in calling the Attraction transaction, sending it, as parameter, the mode in which the transaction must be opened if it is called to update, delete or insert data. In the first two cases, since Update or Delete will correspond to events of one line, the ID of the line attraction will be sent as a second parameter to the transaction, in order to update the data of that attraction, or in order to delete that attraction. For the Insert control outside the grid, 0 will be sent as a second parameter, because the attractions in Insert are autonumbered.

This is why the pattern modified the Attraction transaction by adding the Parm rule, among other things. As we can see, it receives two variables: the &Mode variable is a standard variable in transactions, 3 Character, which accepts one of four values indicated in the TrnMode enumerated domain: Insert (INS), Update

(UPD), Delete (DLT), and Display (DSP)

Upon receiving one of these four values, the transaction will know the mode in which it should be opened.

In addition, it will receive, as a second parameter, the attraction ID, in the `&AttractionId` variable, for updating, deleting or displaying.

Update action at the grid lines level

Country Id

Attraction Name From

Attraction Name To

GRID

Id	Attraction Name	Country	Photo	trips	Update
AttractionId	AttractionName	CountryName		&trips	

Total Trips

Event Start
 &Update.FromImage(updateIcon)
 Endevent

updateIcon X
 Images
 New Image
 Image
 updateIcon24.png
 (Default)
 Size:24x24 px

We add the image to the KB

&Update of Image type

Event &Update.Click
 Attraction(TrnMode.Update, AttractionId)
 Endevent

In our web panel, we will implement one of these actions over the transactions. For example, Update. The idea is to show an example of actions over the data.

We will have to insert a control in the grid. In the case of the pattern, a character variable called update is inserted. It is assigned the "UPDATE" text that we see in runtime. But we will choose to insert an image, which we must insert in the KB to start with. We will call it updateIcon.

Now we go to the web panel and drag the Attribute/Variable control from the toolbox to the last grid column, and define the new variable as &Update, of the Image type instead of character. We remove the title so that it isn't displayed as a column title and then we need to load that variable with the image we've just inserted in the KB. Where do we do this?

If the image changed according to the grid line, we would then do it in the Load event. But since the image will never vary and will remain the same for every line, then a good option is to do it in the Start event, which will be run only once, when the web panel is opened.

Now we need to associate an event with that image, so that when the user clicks on it, the event is triggered and its code is run, where we will invoke the Attraction transaction.

There are several options available to do this. One of them is to use the click event of the &Update variable control.

This will cause that, when the user clicks on the image for a line, the code that we type inside this event will be executed. In such case what we will want to do is to invoke the Attraction transaction. We will pass the Update mode, that is to say, the Update value of the TrnMode enumerated domain we saw before, and the AttractionId value corresponding to the grid line where the click was made.

Update action at the grid lines level

The image shows two screenshots of a GeneXus application. The left screenshot, titled "Application Name", displays a grid of attractions. The grid has columns for "Attraction Id", "Attraction Name", "Country Name", "Attraction Photo", and "Trips". The "Trips" column is highlighted with a green box, and a pencil icon next to the value "2" for the Eiffel Tower is also highlighted. Below the grid, a "Total Trips" row shows a value of 6. The right screenshot, also titled "Application Name", shows the "Attraction" update form. The "Name" field contains "Eiffel Tower" and is highlighted with a blue box. The "Country Id" field contains "3" and the "Category Id" field contains "2". A "Photo" field shows a thumbnail image of the Eiffel Tower. At the bottom right, there are "CONFIRM" and "CANCEL" buttons. A flow diagram at the bottom left shows a sequence of three boxes: "Start", "Refresh", and "Load", connected by arrows and enclosed in a green border. Arrows point from the pencil icon in the grid to the "Name" field in the form, and from the "CONFIRM" button to the "Start" box in the flow diagram.

Let's run it.

First, we see that the column of the &Trips variable is now editable. We had not defined it as Readonly explicitly because we had noticed, upon the execution, that it was already done. As we said, grid variables are first added as readonly, except when an event is defined at the line level, as in this case, or under other circumstances that we will not discuss now. Let's set it as Readonly for the next execution.

We now select a country, like France. And now we click on the update image for the Eiffel Tower. In update we see the name of the transaction. Now we modify something... for example, we can change the T in Tower from uppercase to lowercase.

We now confirm... and since the work with pattern, which we have not seen, has added a Return command to the transaction, to return to the caller, we will return to the web panel. This Return command is similar to invoking the web panel for the first time, so the Start event will be executed in it, followed by the Refresh and Load events as many times as the number of records that will be loaded.

When is it not possible to include a column in the grid?

The screenshot shows the GeneXus IDE interface. On the left, a web form is displayed with a grid containing columns for 'Attraction Id', 'Attraction Name', 'Country Name', 'Attraction Photo', and 'Trips'. The 'Attraction Id' column is highlighted in green. On the right, the Properties window for the 'AttractionId' attribute is open, showing various settings. The 'Visible' property is set to 'False', which is highlighted with a green box.

```
Event &Update.Click
  Attraction( TrnMode.Update, AttractionId )
Endevent
```

It must be
in the grid!
If we don't
want to
see it, we
can hide it.

That's why when returning all the attractions are loaded, without any filters, because the variables will naturally be empty when this panel is run again. Do we have a way to preserve the status that the variables had before invoking the transaction? Sure, but it **won't** be explained here.

What would happen if we hadn't added the AttractionId attribute in the grid? By clicking on the image to update, what AttractionId would have been sent as a parameter to the transaction? It would not have that value to send.

Since it will be used in an event at the line level, which will be triggered after the lines have been loaded, we cannot remove AttractionId from the grid, because here, we are no longer in the database. The grid has saved, upon the loading with the Load event, all its column values and nothing else. A later event will work only on the data loaded in the grid. So, if we don't want to see this column in the grid, we can hide it. It will continue to be present, though hidden. To this end, we use the Visible property with its value set to False.















When is it not possible to include a column in the grid?

Application Name

Country Id

Attraction Name From

Attraction Name To

Attraction Name	Country Name	Attraction Photo	Trips
Christ the Redeemer	Brazil		2 
Eiffel tower	France		2 
Forbidden City	China		0 
Great Wall	China		0 
Louvre Museum	France		0 
Matisse Museum	France		1 
Smithsonian Institute	United States		1 

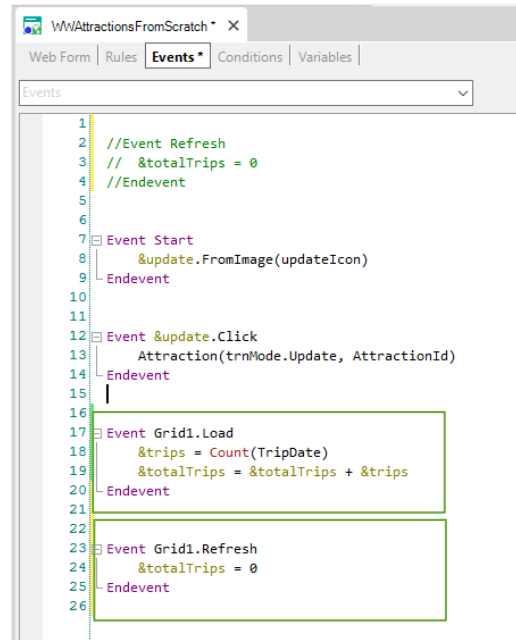
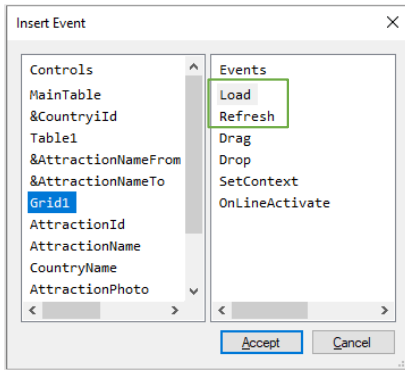
Total Trips 6

All Categories

Before we move on, let's say that, obviously, a web panel can have many grids on its screen, and not just one. For example, we might want to show all the categories next to it, in another grid.

Each grid will possibly need to run code as each line is loaded. But how would we specify this if we have a single Load event?

When is it not possible to include a column in the grid?



If we choose Insert / Event we will see that for the grid control we also have the Load event. In this case, since we had only one grid, we didn't worry about it. However, to plan for the future, in case we want to add another one, maybe it would have been a good idea to use that of the specific grid, which is valid even when grids are added, instead of using the Load event that is valid only when the web panel has up to one grid.

Let's try it... We can see the &trips variable was loaded correctly, just like before.

There is also a Refresh at the grid level...

That will always be triggered before loading (but unlike Load, the generic Refresh is still valid for cases of more than one grid. It will be triggered before the grids are loaded and then each grid's Refresh and Load events will be triggered):

Let's try it...

Next, we will see some other features and, above all, we will review all the events, their logic, and where and how to program them, to clarify everything.

*GeneXus*TM

training.genexus.com
wiki.genexus.com