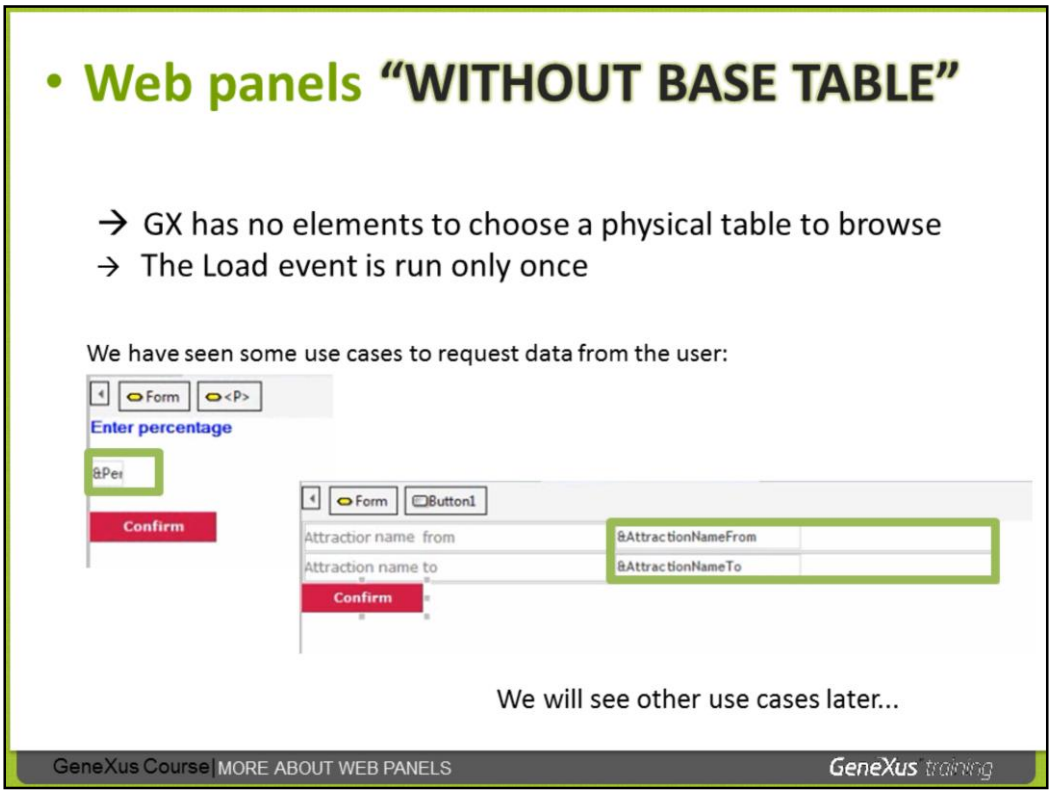
INTERACTIVE SCREENS

More about Web Panels

WITH BASE TABLE?
WITHOUT BASE TABLE?

(REVIEW AND MORE KNOWLEDGE...)



Although the examples of web panels "WITHOUT BASE TABLE" we have seen so far are cases in which we need to request data from the user, there are also use cases of web panels "without base table" with variables in the grid (there can also be variables in the flat part of the form) which may not necessarily be the target of the web panel, to offer data entry.

As the Load event in the web panels "without base table" is run only once, it is possible to include within said event an explicit programming with For each, assigning values to the variables on the grid (for example, with the values of the navigated attributes, formulas, etc.), and once all the necessary variables to add a line have been initialized, we have the Load command, whose use only applies within the Load event, to load a line on the grid.

We will provide a thorough insight on this other possible use of the web panels "without base table" later.

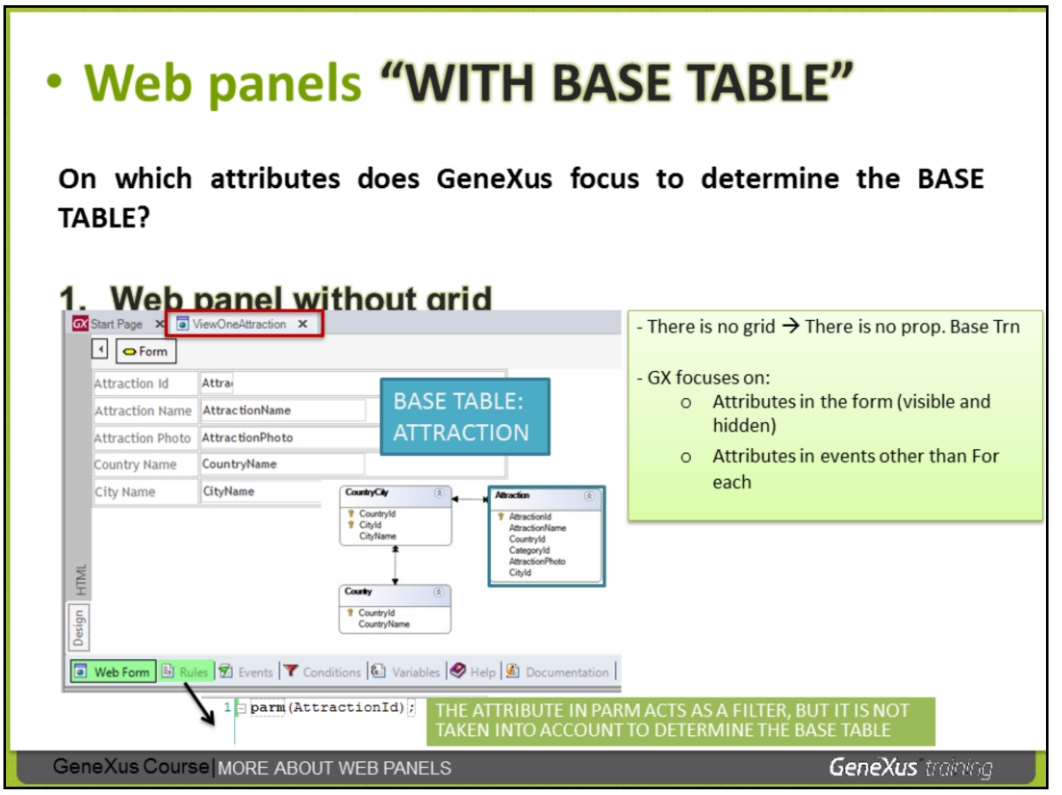
As we have mentioned, the web panel is a very flexible object that allows multiple uses, so it is useful for many more things.

• Web panels "WITH BASE TABLE"

On which attributes does GeneXus focus to determine the BASE TABLE?

Let's divide in cases:

1. Web panel without grid
2. Web panel with 1 grid
3. Web panel with more than 1 grid



In this web panel 1 grid has not been included, but attributes containing 3 physical tables were inserted in the form (they were automatically inserted in a table to appear aligned in the form).

The web panel has only the attributes shown in the form and the parm rule shown as defined.

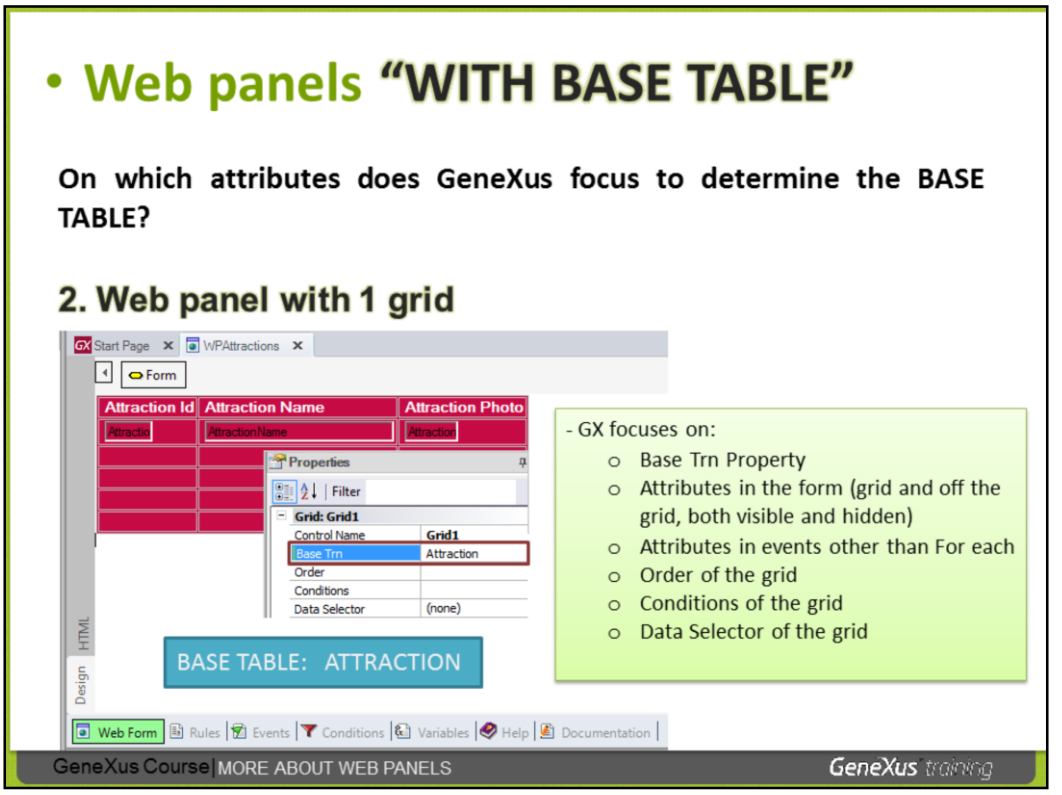
It's not possible to indicate the transaction base, because there is no grid. But the GeneXus analyst has inserted the attributes, knowing that GeneXus would analyze in which tables they are and, taking into account the relations, **it would determine a base table and access its extended table** to get the values of all the required attributes. So, in this example, it is clear that GeneXus determines the browsing of the physical table ATTRACTION, because since ATTRACTION is the base table, the rest of the tables are in the extended table and all requested data can be reached.

But, of all the ATTRACTION registries, which one will be displayed?

If the Parm rule was not defined, all the physical table would be browsed and finally the data of the last registered attraction will be viewed.

But the parm rule receives the identifying **attribute** AttractionId; therefore, we understand that this web panel will be invoked from some other object, sending by parameter a certain attraction identifier and said value will be used as a filter for equality, so 1 specific registry will be queried in the ATTRACTION physical table and its data and the related data of its extended table will be displayed.

So the web panel has an **implicit For each or, in other words, an automatic browsing determined by GeneXus**. Therefore, we can say that this web panel **has a BASE TABLE.**



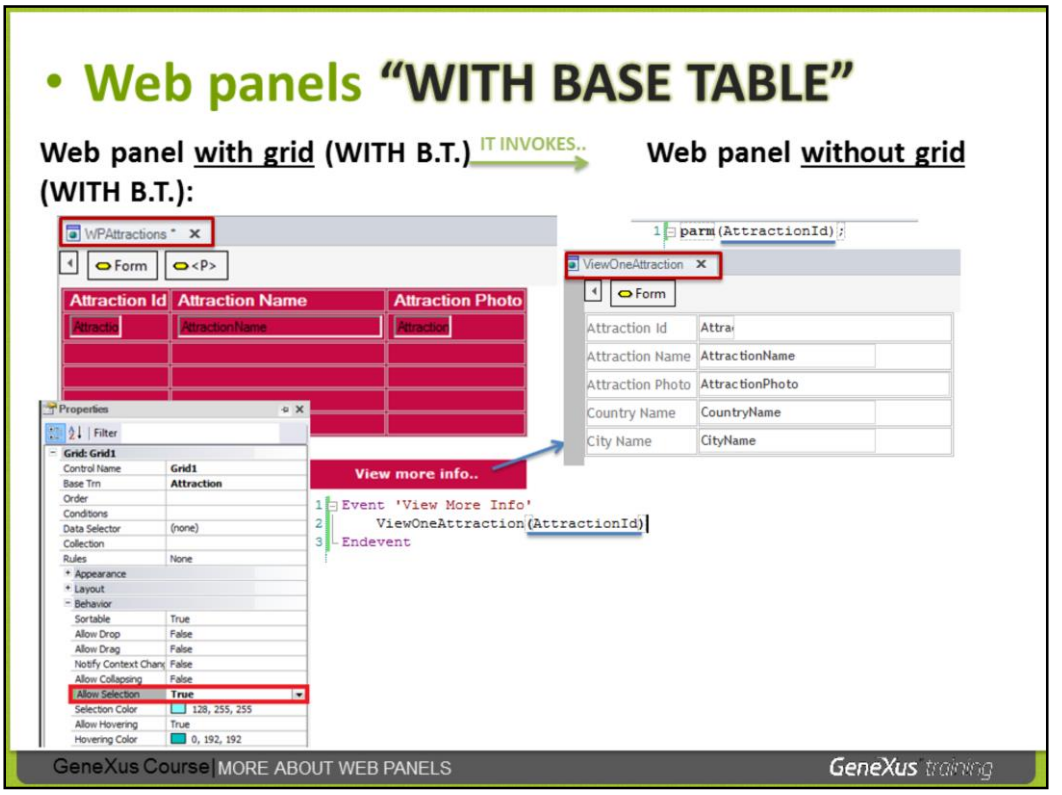
This web panel only contains in its form the grid displayed, with attributes of the physical table ATTRACTION and the Base Trn del grid = Attraction property (only defined in this object).

GeneXus has clear information to know which base table to browse: ATTRACTION.

So, at runtime, all attractions will be browsed and displayed in the grid of the web panel, and all attractions (as there are no filters in the conditions) will appear on their grid arranged by AttractionId (as no order was specified either).

To say that the **base table of the web panel is ATTRACTION** is the same as saying that **the base table of the grid is ATTRACTION**, as there is only 1 grid; therefore, ATTRACTION is the base table of both the grid and the web panel. This means that there is an implicit for each we have <u>not</u> defined, but it was determined by GeneXus automatically for the web panel.

Data Selector is a grid property that can be either configured or not. We will see what this is all about in due course.

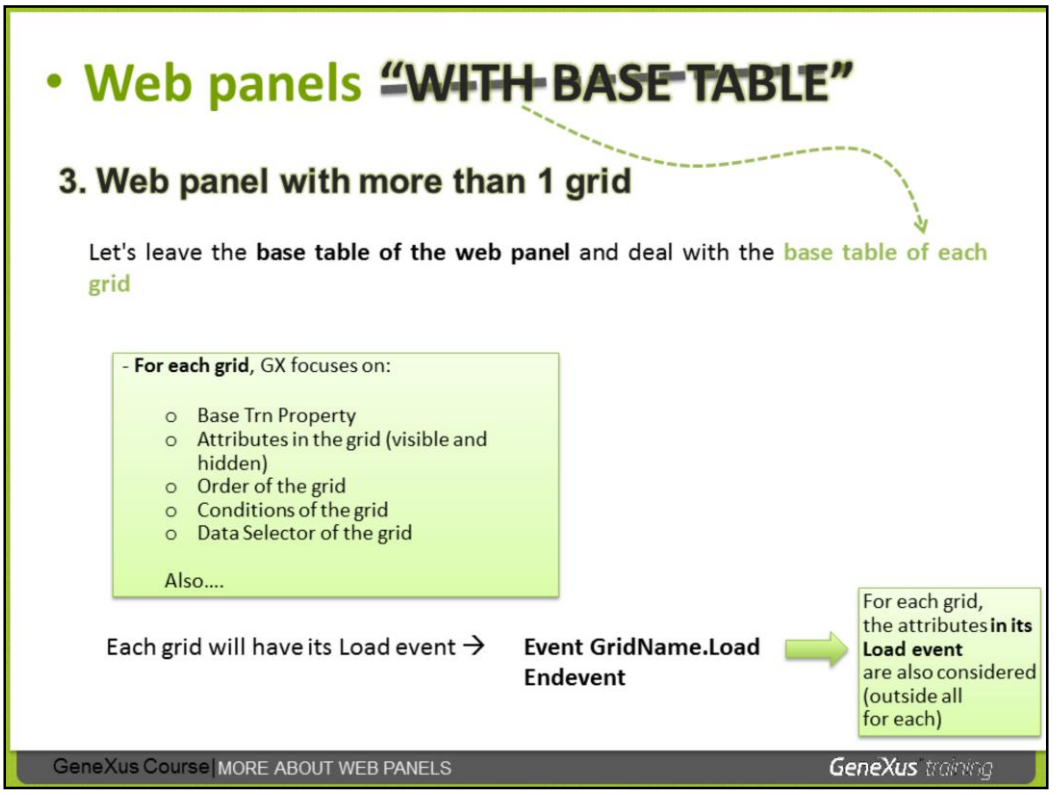


We are proposing here that one of the web panels we have defined invokes the other, to make use of both, integrate what we have defined and learn more.

We know that the web panel "WPAttractions" (the one that contains the grid) has a base table: ATTRACTIONS and it loads all the registered attractions in the grid.

We have now made 2 changes to this web panel. First, we have set the grid property Allow Selection = True, for the user, by moving the cursor over the grid during runtime, to also see a scrolling color line and to click on one of the lines; when doing so, it will be selected and painted.

The second thing we have added to the web panel is a button, for the user to press it, precisely when selecting a line, and the web panel "ViewOneAttraction" is invoked, providing as a parameter the attraction identifier of the selected line, and said value is received on the invoked object, acting as filter for equality to show the data of that attraction.



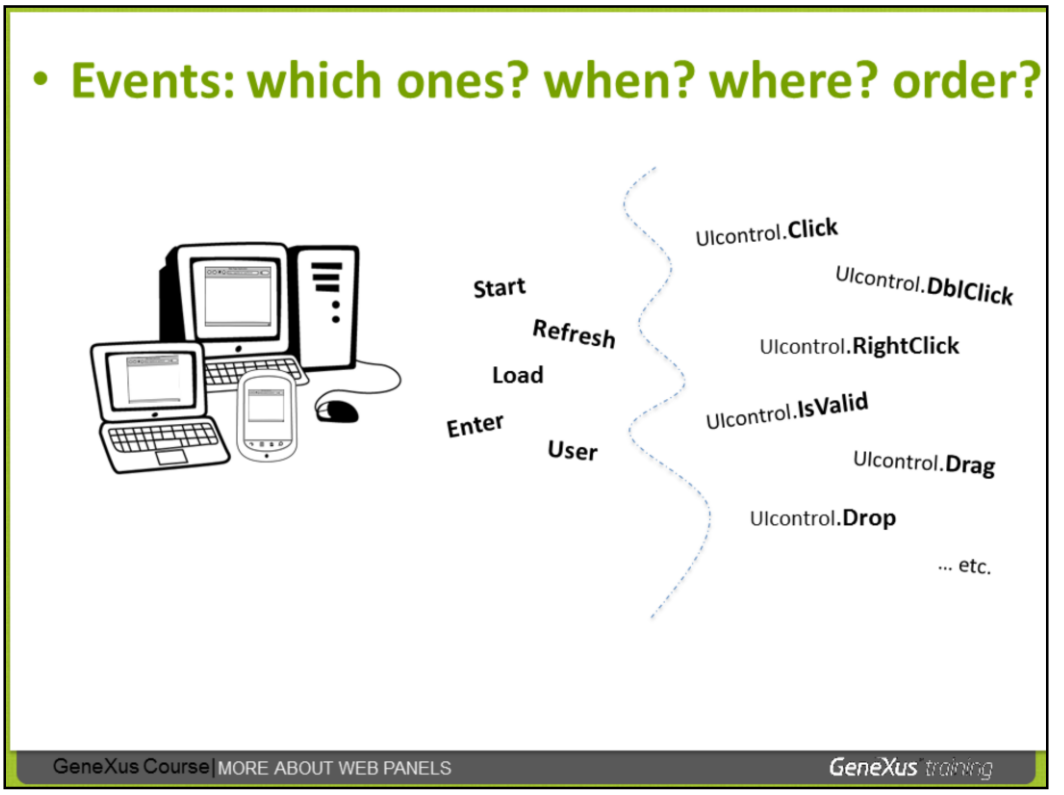
When there is more than 1 grid in a web panel, **<u>the base table of each grid</u>** is determined **considering exclusively the attributes mentioned in the slide**.

Each grid will have its Load event and the syntax is shown in the slide.
The generic Load event cannot be used, as it would be impossible to know which grid we would be dealing with.

Unlike the case of only one grid, the attributes outside the for eachs in any event not being "its" Load, will not be part of the determination of the base tables. But they must belong to the extended table of any of the grids. Otherwise, GeneXus will note it in the browsing list.

If there are attributes in the fixed part, the base table of the first grid is determined taking into account the loose attributes, the other grids not taking them into account. If you wish to see more about this special case, access our wiki: http://wiki.gxtechnical.com/commwiki/servlet/hwikibypageid?6105

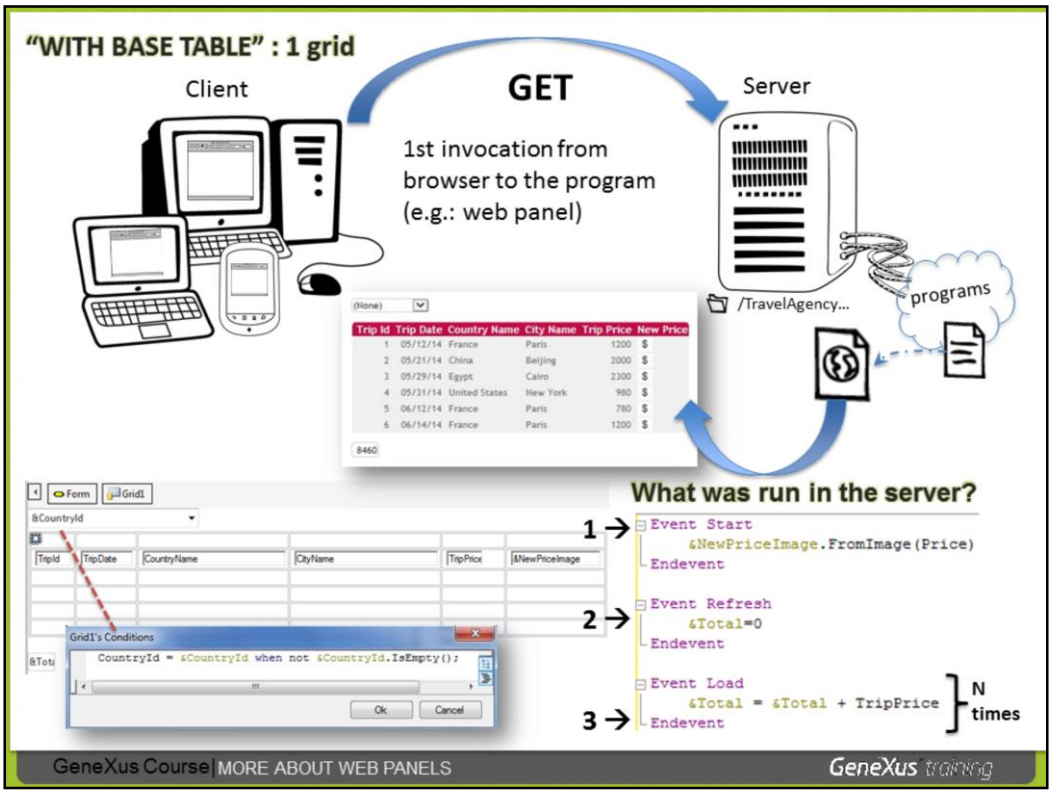
EVENTS
which ones?
when?
where?
order?



We have seen and programmed different events in Web panels (for example, the click, associated with controls included in the form but, depending on the control, the double click, right button, etc., can also be programmed).

We have also presented and used the Start and Load events associated with the web panel object. We saw that the Enter event associated by default with every button is inserted in the form, and we have also explicitly defined user events for buttons. The web panels also have the Refresh event.

We will see now in more detail the web panel events, where each one of them is run (if in the server where the application is installed or client) and in which order.



In every web application, we will have a machine –PC, notebook or smart device– connected to the Internet, with which the user will access the application through a browser; and the server, on the other hand, where all the application programs generated by GeneXus are. Among them, those of the web panels (for example, the program generated for the web panel shown in the slide).

This web panel allows the selection of a country (of all the countries stored) and the chosen country identifier shall remain in the &CountryId variable presented with the Dynamic Combo Box control (having set the properties associated with said control).

There is 1 defined condition, as shown in the slide, for all the trips registered <u>for the country chosen in the variable</u> &CountryId to be loaded in the grid.

Within the grid in the last column there is an image-like variable, to which we will assign an image with the peso sign ($), in the start event.

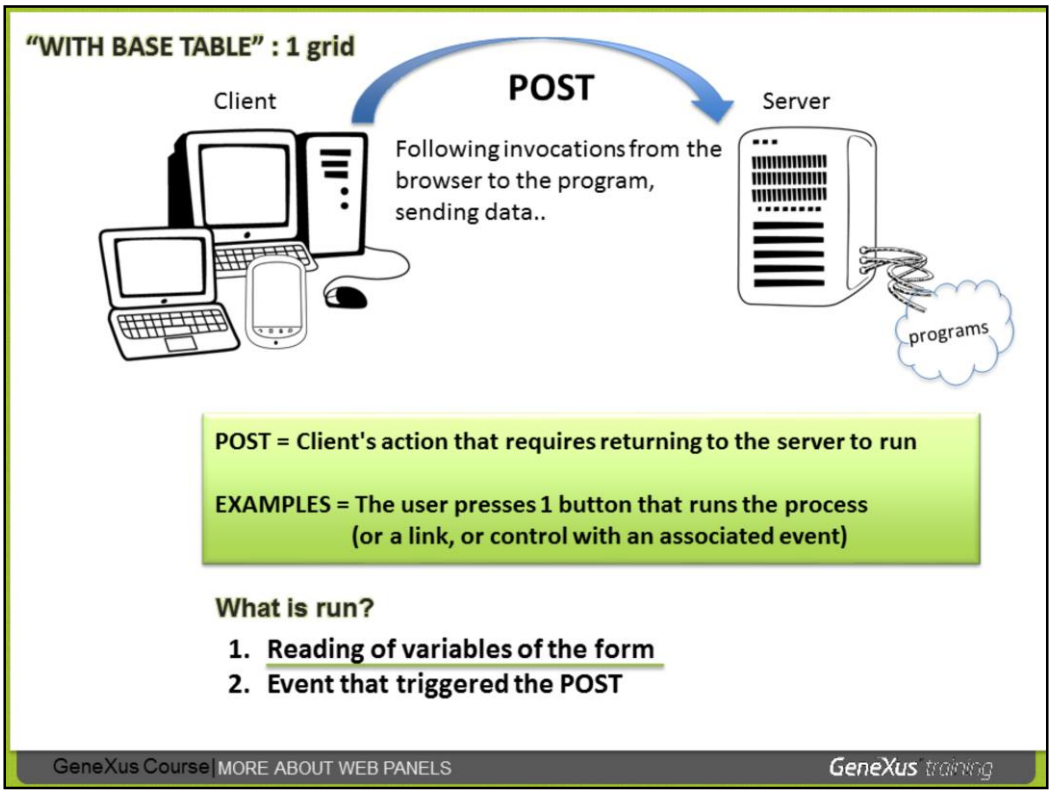
Under the grid, we would like to see the prices of all the trips shown (in the &total variable).

Ok. What happens when we invoke the web panel from the browser **for the first time**? What is known as making a **GET**.

The <u>client</u> asks the <u>server</u> to run the program associated with the web panel and send back as a result a html file indicating the browser how to draw the screen (with which data, format, etc.).

But, what does the program run in the server to create this html file?

First, the Start event. And because of what we have defined in it, the money symbol image is loaded in the &NewPriceImage variable.

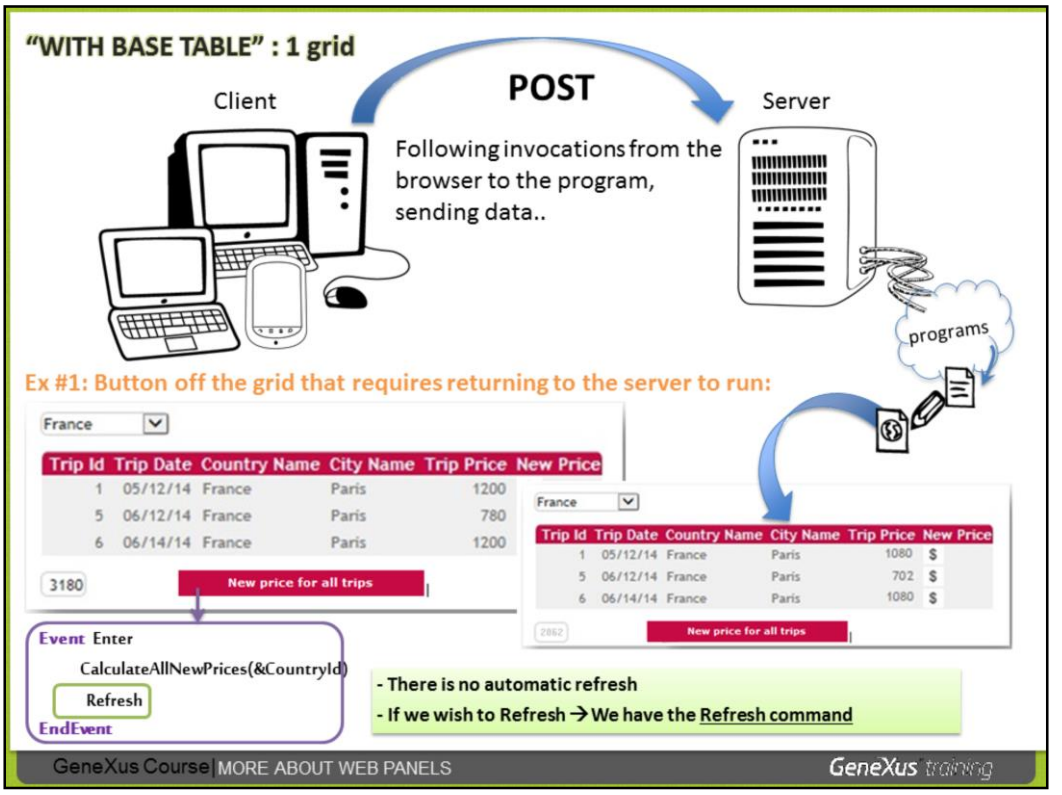


Generally speaking, a **POST** is produced every time an action is performed in the client, which requires going back to the server to run.

Examples of these actions can be pressing the Enter key or any button or control associated with one event.

When a POST is run, this is what happens:
- The variables in the screen are read
- The user event that brought about the Post is run.

Both the form variables and the web panel parameters are accessible to the event.



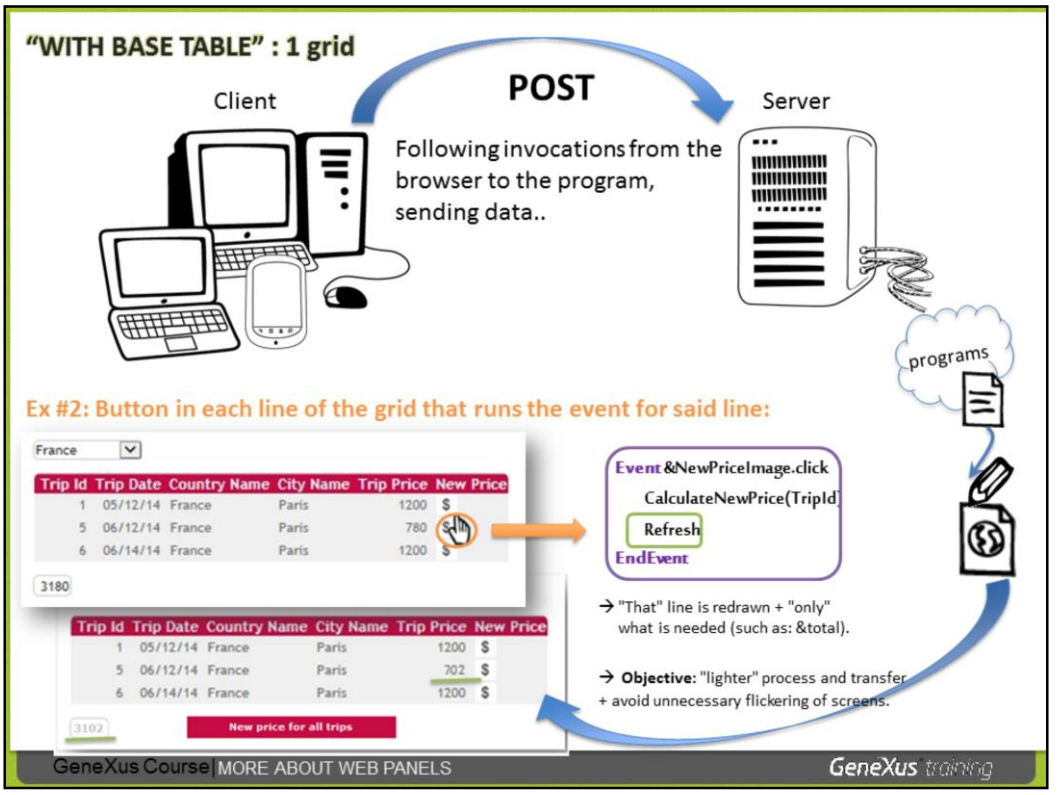
In this example, there is a button under the grid, which allows to increase the prices of all the trips shown in the grid (that is to say, those of the country chosen in the combo). In the event associated with the button, a procedure is invoked and the value of the &CountryId variable is sent for parameter. It is not important to know the criteria to obtain the new prices (it may be based on the date, verifying in a table if there is any special offer today... or other verifications). The fact is that when the user presses the button, the procedure will be invoked, and it will browse with a For each all the trips of the country received as parameter and it will update the price of each trip in the TRIP table, based on the criteria requested by the travel agency.

It is important to know and consider that events (both user events and events associated with controls) **do not Refresh automatically** (as the programmer must determine if that is necessary or not; he/she may want to run a process in the server and to not refresh after the form).

If the form needs to be refreshed, GeneXus provides the command: Refresh. In this example, the procedure will update the database (the trip prices) and the developer needs to include the Refresh command immediately after invoking the procedure.

The Refresh command will run the Refresh event, so the code defined in said event will be run and afterwards the Load event will be run as many times as registries complying with the conditions… and they will be loaded in the file that will be returned to the client for the browser to draw the "refreshed" page.

To sum up: when the user presses the button, the form variables are read (in this case &CountryId) and then the event associated with the button pressed is run. The event has 2 lines in its code: 1) running the procedure and 2) running the refresh command.



Now we are presenting another example of POST.

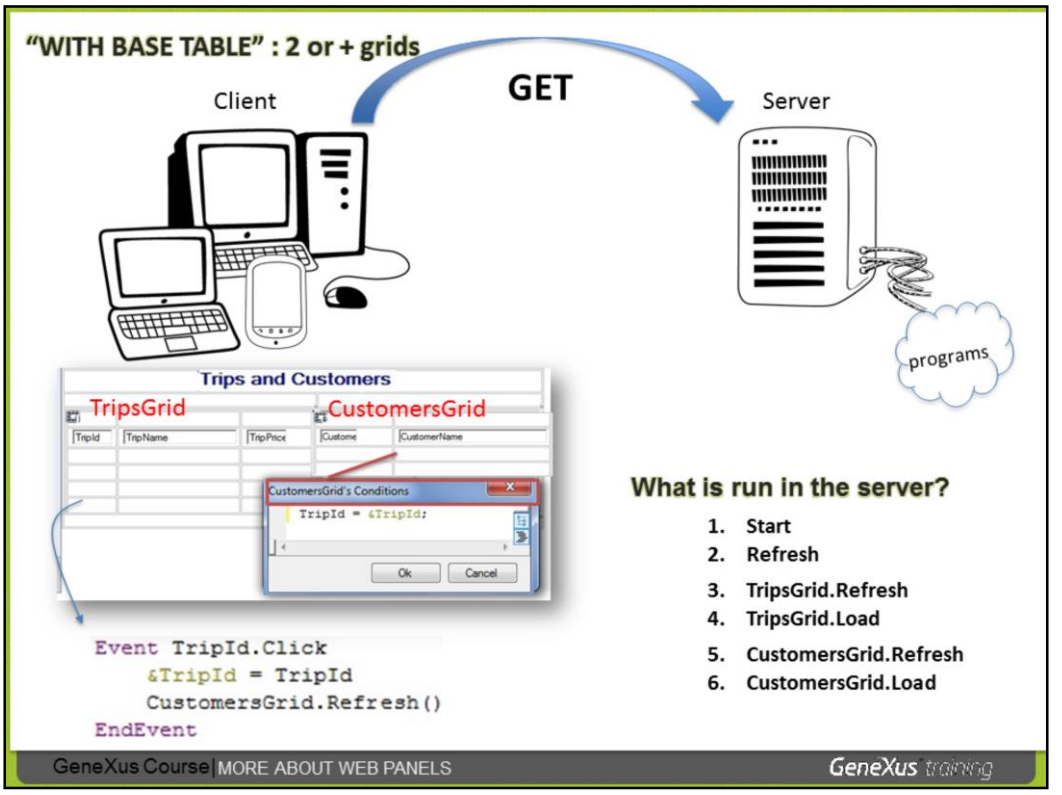
Here, there is 1 variable inside the grid, in the last column (where there is one image of the $ sign loaded) and has the click event programmed.

When the user selects the image for one line, the associated event will be run, and like we said, this is what will happen:

•The form variables will be read (&CountryId)
•The event that brought about the POST will be run

The event invokes a procedure and sends the TripId of the line to it, for it to process the calculation of that trip and save its new price. Then, a Refresh was requested.

The behavior in this case will be that the line implied of the grid will be updated, **while the rest of the lines will remain the same and will not be reloaded**.



We will see now what event triggering is like when there is more than 1 grid in the form.

The top web panel has 2 grids. As we have already mentioned, each grid will have its base table. The grid called TripsGrid has the base table: TRIPS and the grid called CustomersGrid has the base table CUSTOMERS (for the attributes each one includes + the Trn Base property of each one of them, etc.).

The purpose is that when the user clicks on a trip identifier they see the clients registered to take the trip in the other the grid.

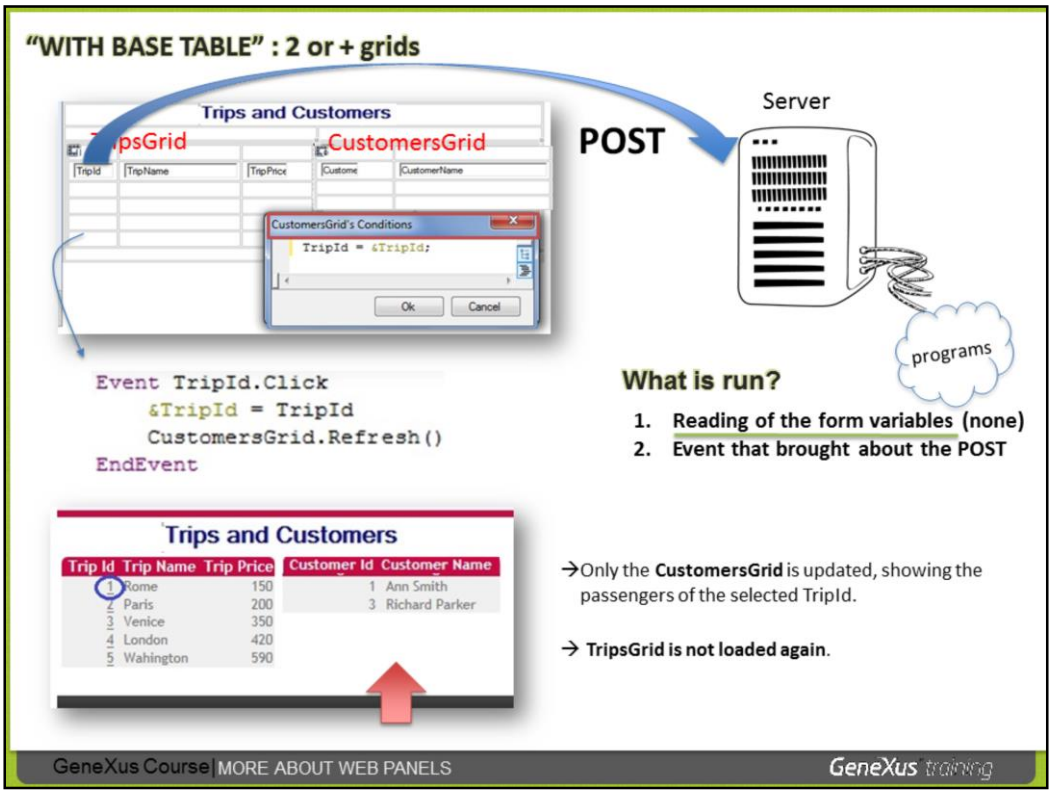
The GET is the first thing run by 1 web panel. What happens in the GET in this case?

When running this web panel from the client, the following will be run in this order:

1. The Start event
2. The general Refresh event that executes its code (if programmed) and calls refresh and load of each grid. Therefore, it calls:
3. Refresh event of the first form grid, and this calls
4. Load form said grid, once for each registry. And then…
5. Refresh event of the second form grid, and this calls
6. Load event of that grid, once for each registry that meets the conditions.

Since no one has yet clicked and the &TripId variable is empty when the GET is run, no line will be loaded for the second grid.

We will now see what happens when the user clicks on the identifier of 1 Trip and therefore a POST occurs.



When the user clicks on 1 TripId of the 1st grid, the reading of variables takes place in the screen (in this example there is none) and then the click event that causes the POST is run.

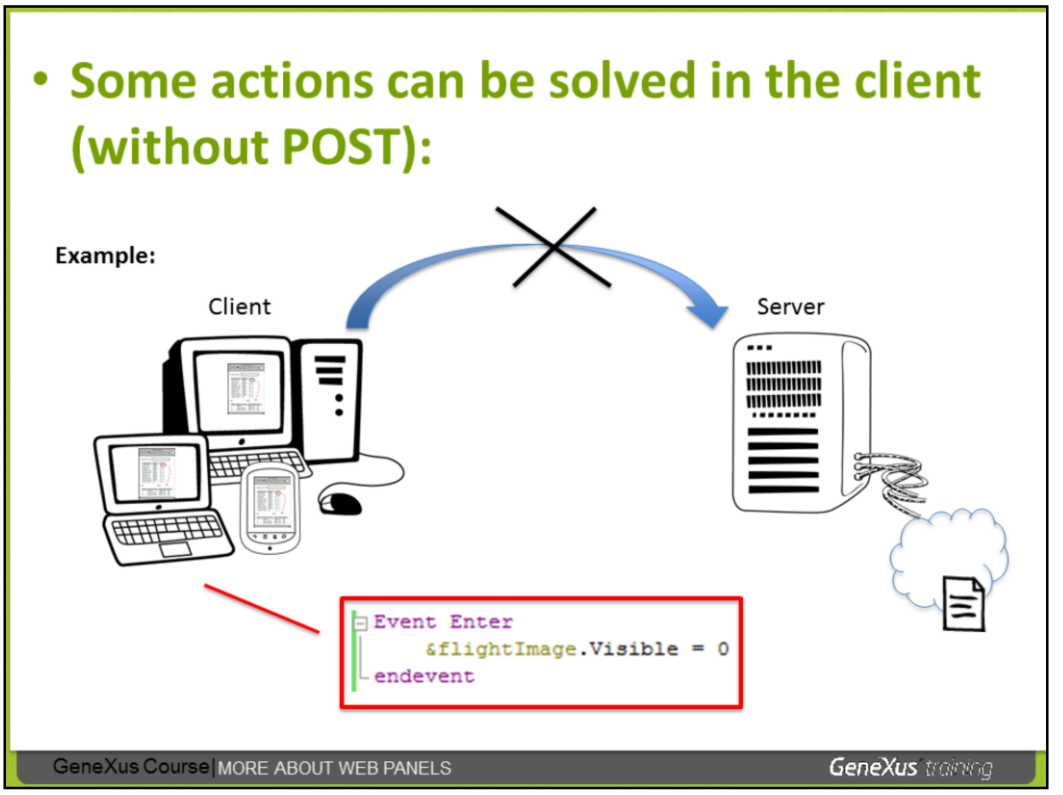
In the TripId.click event, the selected trip identifier is stored in the &TripId variable, and the Refresh of the 2nd grid is then explicitly triggered (executing the command: CustomersGrid.Refresh()).

Note that the CustomersGrid has a condition defined indicating the filtering by the trip assigned to the &TripId variable.

So when the command CustomersGrid.Refresh() is executed, the filter defined in the CustomersGrid conditions will be applied and the Load event of the CustomersGrid will then be triggered N times, filtering the clients registered for the trip clicked in the TripsGrid.

CustomersGrid is updated, while the TripsGrid is not reloaded.

As for the Start event, it was run only once, when the GET of the web panel occurred.

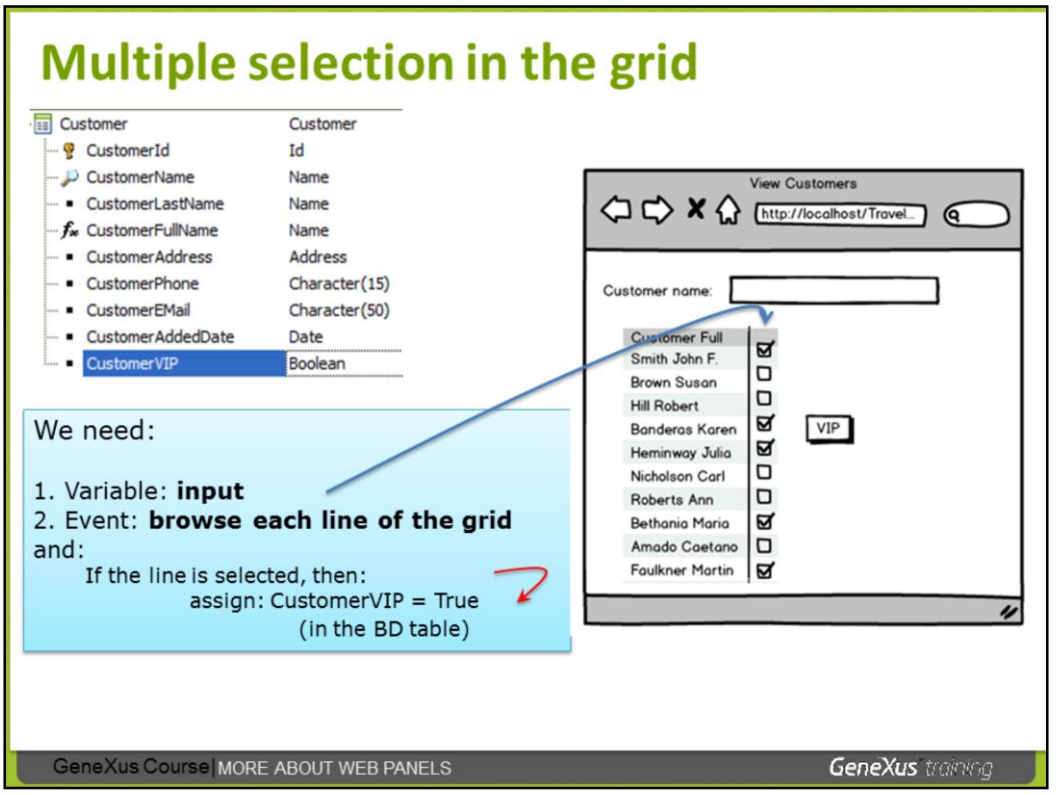


Finally, it should be noted that not every action performed by the client and associated with an event will produce a POST to the server. Some can be solved in the client itself.

For example, if a control becomes invisible in an enter event, either a user event or an event associated with a control, this can be solved in the client itself.

With this we have seen the highlights about the events run in the web panels.

GRID
multiple selection
how to browse it



Let's see an example that includes a variable in the grid, presented as checkbox so the user can mark lines with the purpose of being able to do a multiple selection of lines and then run the action. The CustomerVIP attribute we have added in the Customer transaction allows to distinguish preferential clients.

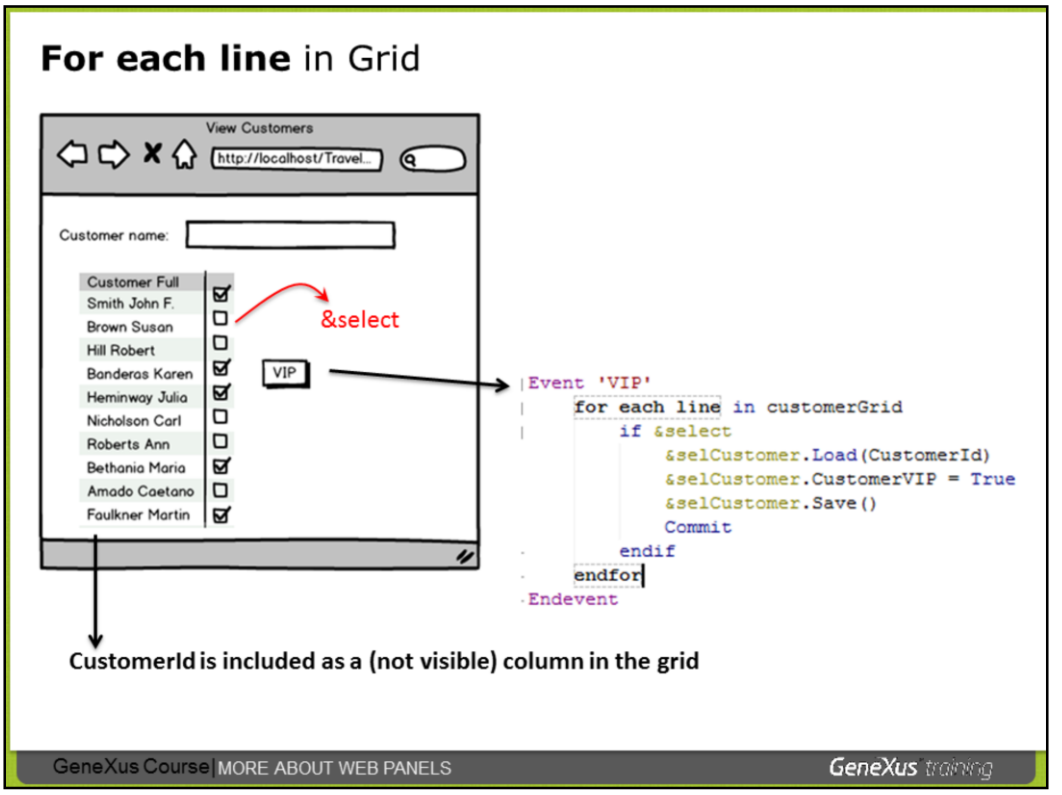
We need the web panel displaying the clients to enable the selection of various clients and by pressing the VIP button to be able to assign them VIP status.

For that, we added a Boolean variable to the grid of flights, &select, that will allow the user to select the line (by default, every Boolean variable will be displayed as a checkbox), and we also added the button in the form with a specially created associated user event named VIP.

We need two things:

1.That the variable added to the grid as a column is not read only (default behavior of the variables in web panels grids) for the user to be able to decide whether to mark it or not, for each line.
2.That when pressing the VIP button, we are able to program a scroll through the lines of the grid and that every line has a marked variable (that is to say, with True value), to assign True to the CustomerVIP attribute of this client and physically save this change.

We know how to solve item 2 by using a business component-type Customer variable (&selCustomer). But let's see how to browse the lines of the grid…



The **for each line** command allows running through the lines of a grid. The iteration ends with endfor.

It's important to differentiate that while the **for each** command runs through records of a (**base**) table in the database, the **for each line** command runs through the lines of a grid.

If the web panel has only 1 grid, you can only write: **For each line**. On the other hand, if it has more than 1 grid, you need to specify which grid to run through, so the following has to be specified: **"for each line in" + name of the grid.**

By default, the variables inside web panel grids are read-only. They change from read-only to read-write in the following cases:

1) When a web panel event uses a specific command to run through the lines of a grid, that is t say: **for each line in GridName**.

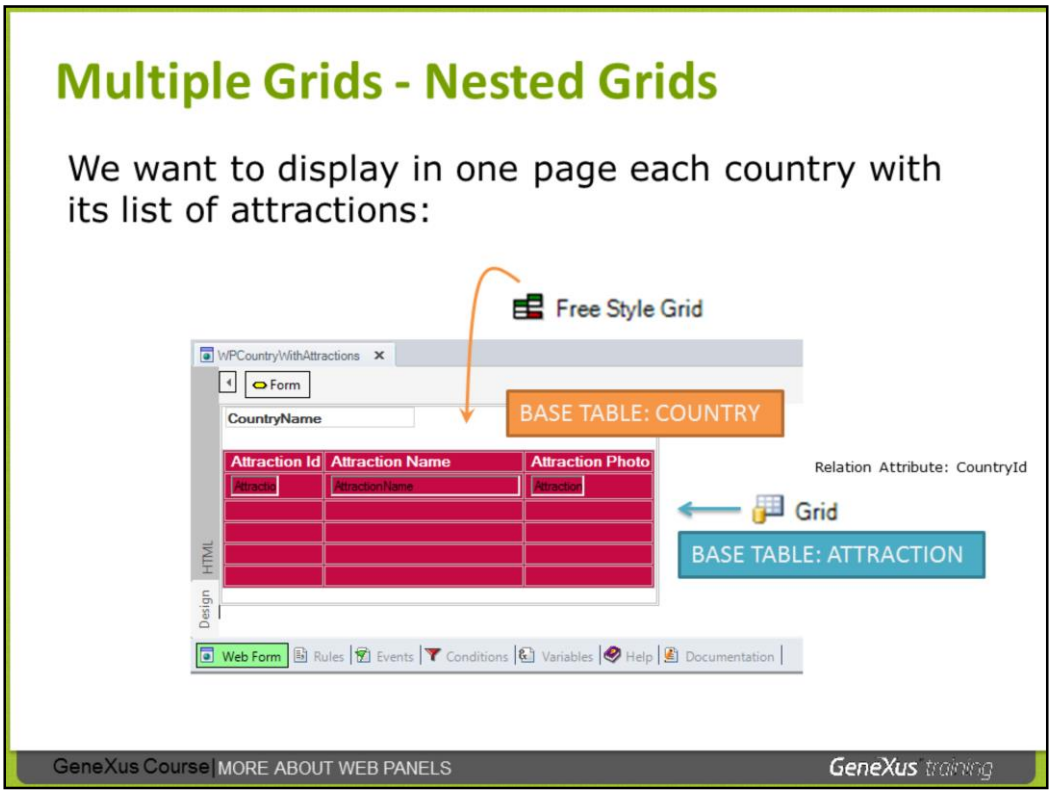
2) When the click event is programmed associated with an image located inside the grid.

In the example shown in the slide, the code associated with the 'VIP' event runs through the lines of the grid. For each line, it first evaluates if the line is selected; if so, we code the assignment of the client as VIP, using the Business Component concept. That is to say: The Customer transaction is configured as a Business Component and the web panel has the &selCustomer variable, which is based on the Business Component data type: Customer.

The Load method is applied to the &selCustomer variable. To do so, the value of the CustomerId attribute that is available (hidden) in the grid to load the customer's record into memory is passed by parameter.

Next, the customer is set as VIP, the assignment is recorded and a Commit is executed.

MULTIPLE GRIDS

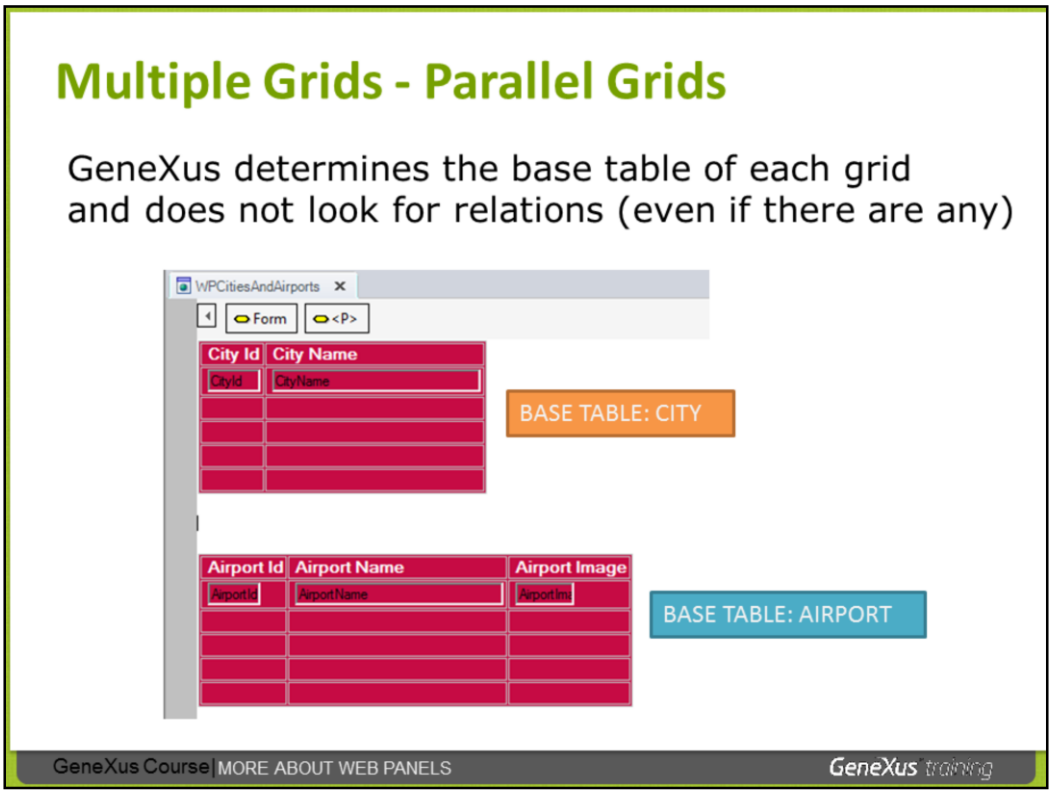


There is a type of grid different from the standard, called **free style grid**, that allows to design 1 line with a more free and colorful design not containing structured columns and that line would be repeated showing that design for every registry browsed.

1 free-style grid can contain another free-style grid or standard.

In the example shown in the slide, we have included in the form of the web panel 1 **free style grid**, and inside it we put the CustomerName attribute and under it 1 standard grid with attraction attributes.

For the nested grid we did not establish conditions at the level of the grid. Why? Because as the grid is inside another grid, GeneXus nests the browsing, as if they were a pair of nested for eachs. The automatic filter is by: CountryId.



In the web panel of the top slide, we have included 2 standard grids.

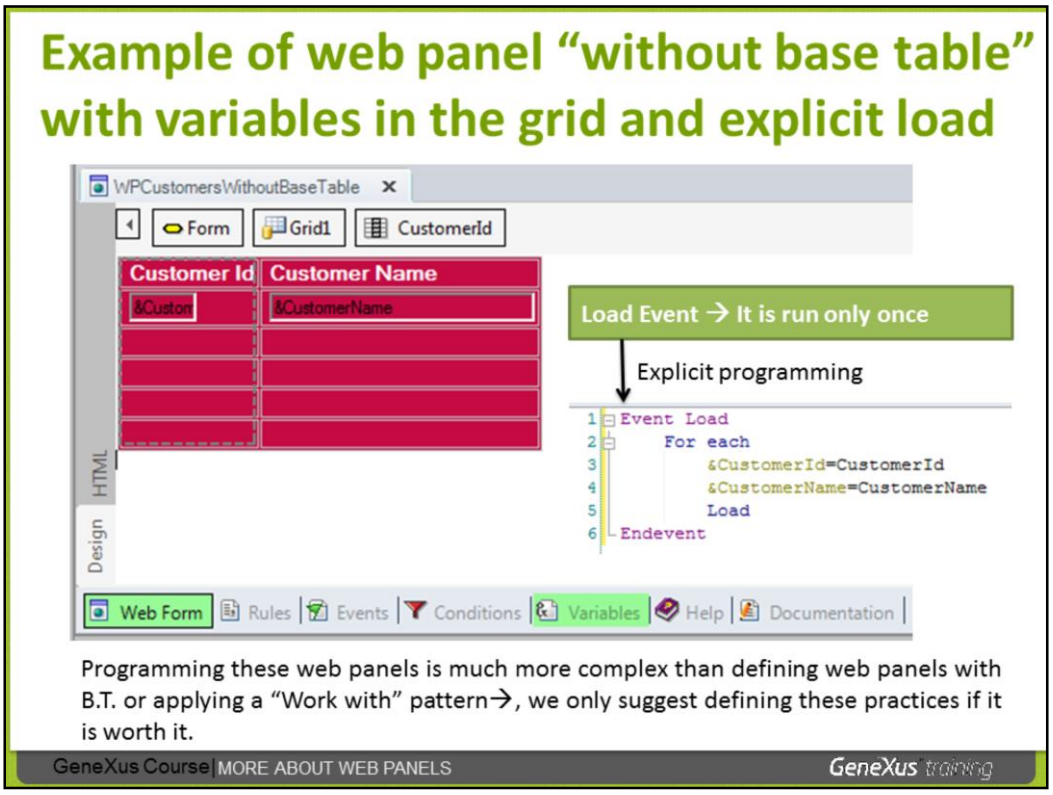
For the 1st grid, GeneXus will determine that its table base is CITY, and for the bottom grid, it will determine that its base table is AIRPORT.

So, the web panel will show, in the first grid, all the cities registered, and in the second grid, all the airports registered. But it will not look for or establish relations.

It is possible to program the possibility that the user can select 1 city, press 1 button and load all the airports of that city in the 2nd grid (as we have already seen in a similar example).

If not, a solution similar to the previous one could be implemented with nested grids, and browsing is nested and contains implicit filters.

MORE ABOUT WEB PANELS
“WITHOUT BASE TABLE”

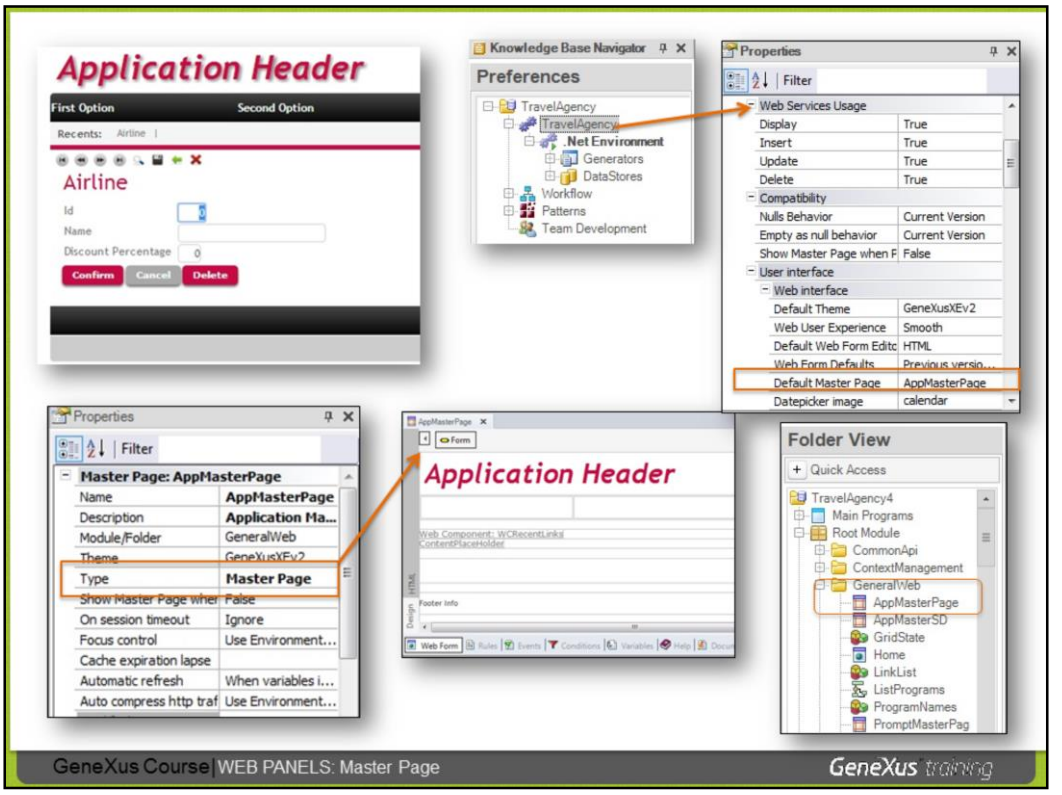


We have already explained that apart from defining web panels "WITHOUT BASE TABLE" with variables to request data from the user, there are also use cases of web panels "without base table" with variables in the grid (variables being also possible in the flat part of the form).

As the Load event in the web panels "without base table" is run only once, it is possible to include within said event an explicit programming with For each, assigning values to the variables on the grid (for example, with the values of the browsed attributes, formulas, etc.), and once all the necessary variables have been initialized to add a line, we have the Load command, whose use only applies within the Load event, to load a line on the grid.

Programming this type of web panel is much more complex than defining web panels "with base table" or applying the “Work with” pattern, so we suggest only defining these practices if it is worth it.

MASTER PAGE



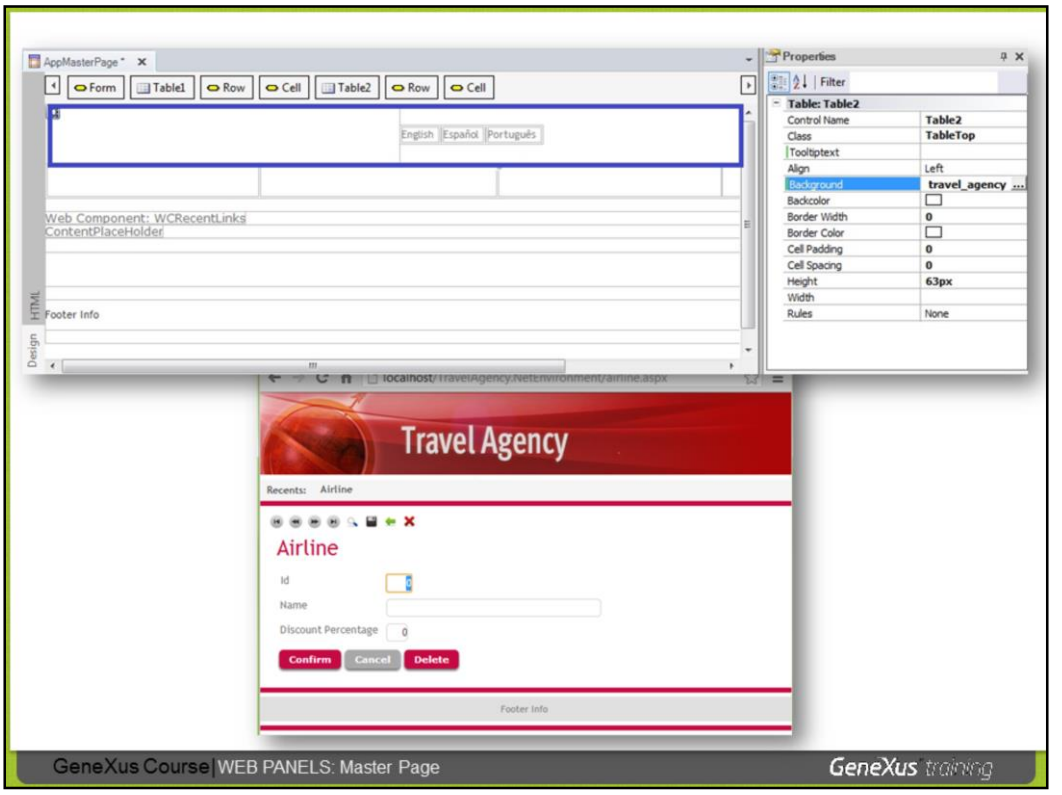
There are three types of web panels: Web Page, Component or Master Page, the first being the one by default.

When creating a KB with GeneXus, we can automatically see that the AppMasterPage object is created inside the GeneralWeb folder. It designs the general frame and the behavior all web pages of our application will have by default. So, for example, if we run any transaction or web panel, we will see that its form will appear in a central space of a page that also has a heading, Application Header, and a final band, Footer.

This object, AppMaterPage, is a Master Page-type web panel. For his reason, a ContentPlaceholder can be placed in its form, which is where all web objects (transactions and web panels) having that page specified as master page will be loaded.

Even though it is possible to specify the default Master Page at the version level, it is also possible to specify it at the object level (for example, tell the Airline transaction that its master page is a different one). For that, search among its properties the one called Master Page.

We can also create our own Master Page web panel, and change the Default by ours. Or simply personalize the one contained in the KB, as it is shown in the next page.



Here, we have modified the AppMasterPage object, deleting the image it contained and adding a new image to the KB, specified as the background of the corresponding table.

GeneXus training
training.genexus.com