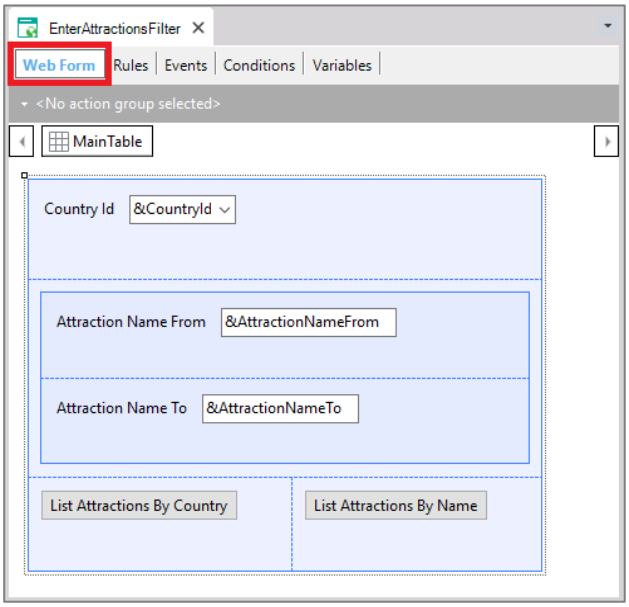
Pantallas interactivas en ambiente web: objeto Web Panel

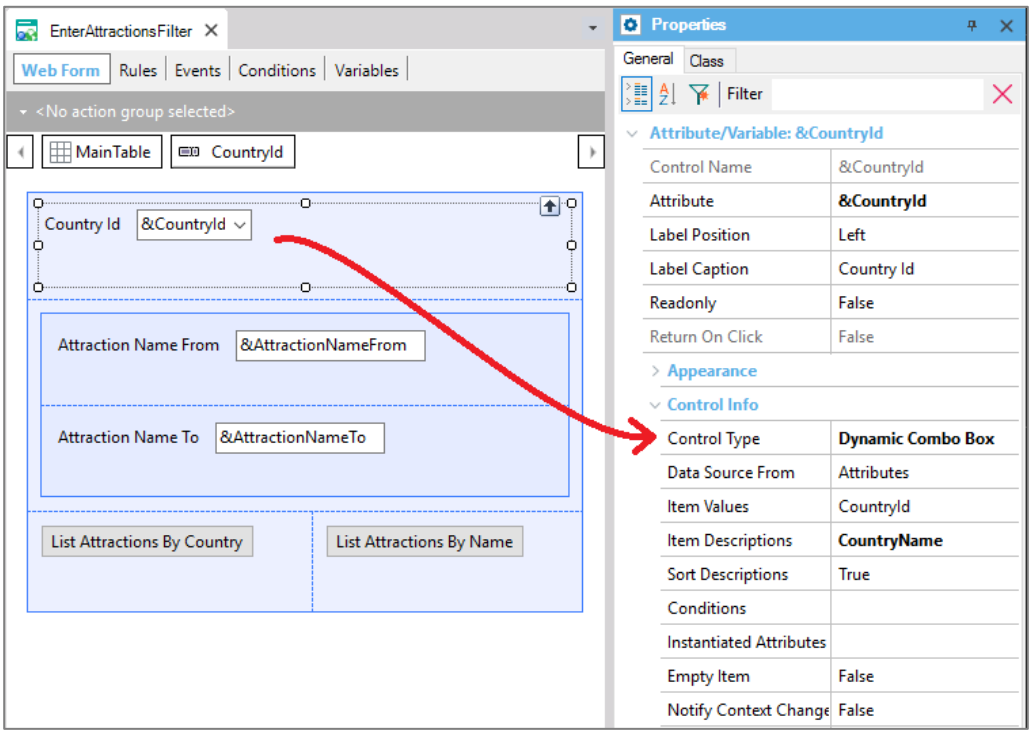
El web panel es el objeto más flexible que provee GeneXus.

Como ya hemos visto en algunos ejemplos que hemos mostrado, todo web panel ofrece un web form, que es una página web que nos permite diseñar y ofrecer variadas funcionalidades.

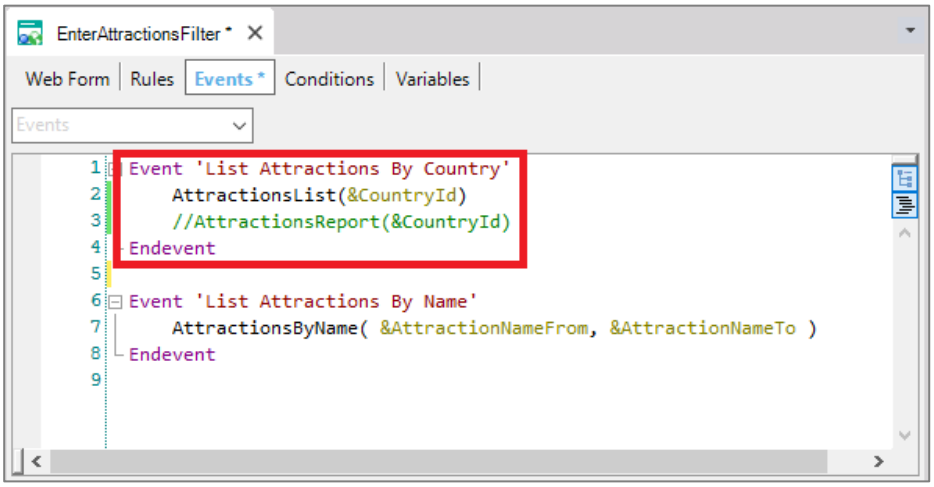


En este ejemplo habíamos visto que por el hecho de incluir variables en el web form éstas quedaban habilitadas para que el usuario les ingresara algún valor. Es decir, eran controles de entrada, o, dicho de otro modo, no readonly.

En particular esta variable de tipo combo dinámico

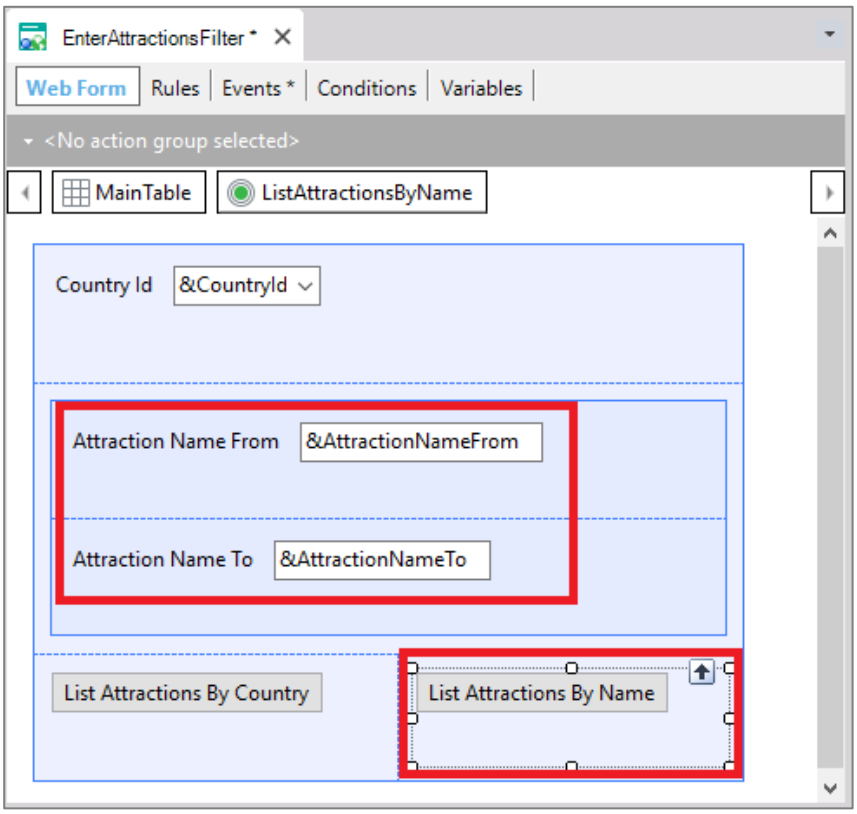


esperaba que el usuario eligiese un país de los cargados en el combo y al presionar el botón “List Attractions By Country”, se ejecutaba el evento asociado…

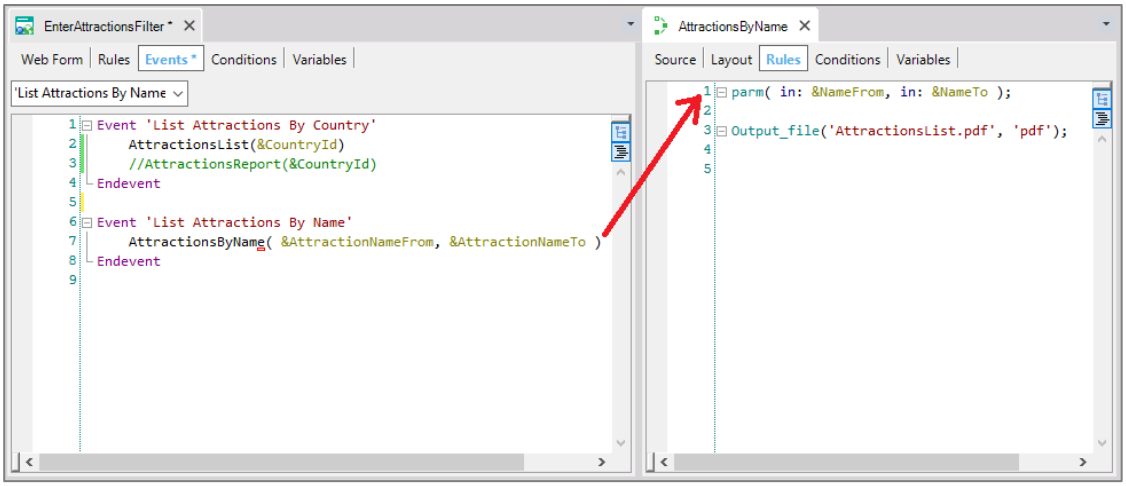


…invocando al pdf que listaba las atracciones de ese país.

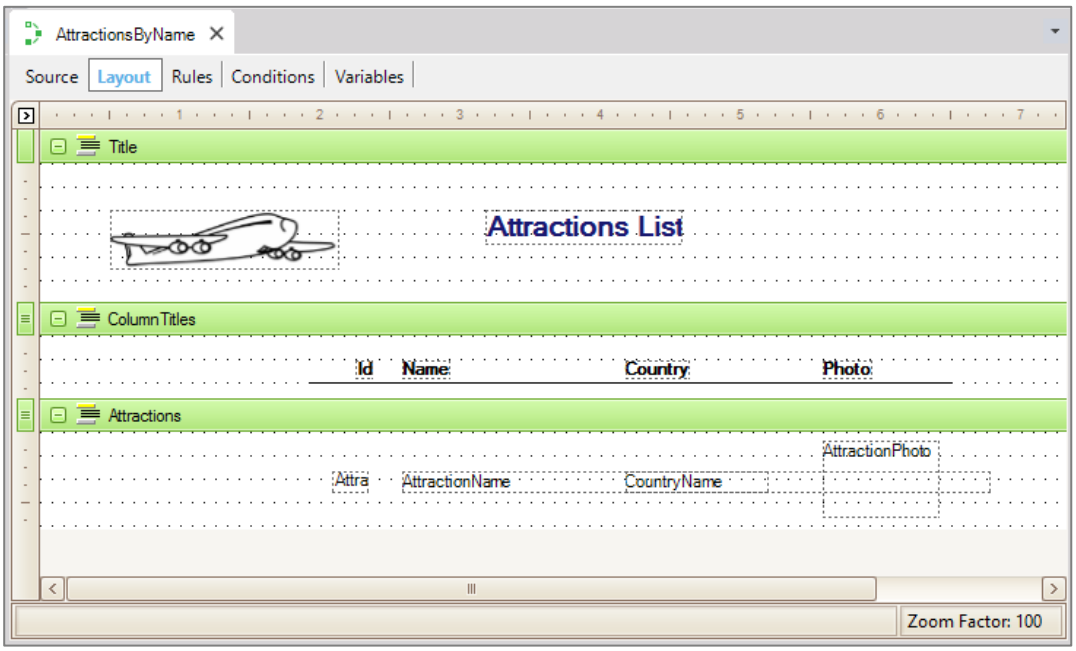
Y en estas otras variables el usuario ingresaba un rango de nombres de atracción para que, presionando este otro botón,



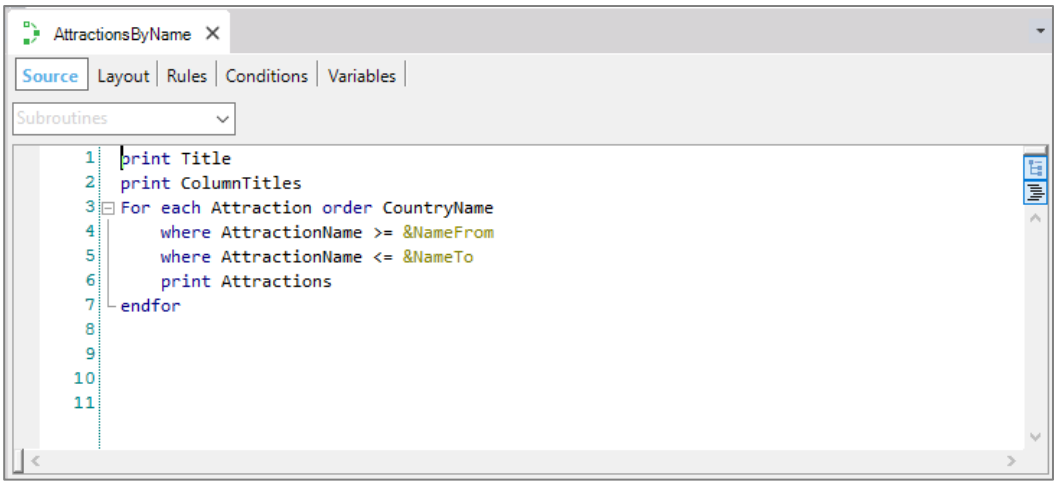
se invocara al listado pdf que mostraba las atracciones dentro de ese rango recibido por parámetro.



Recordemos el layout:

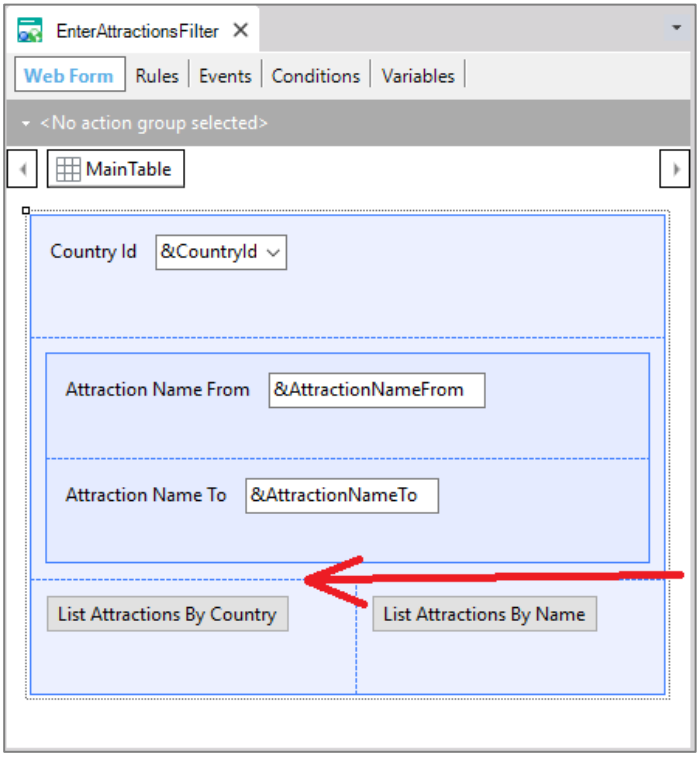


Y en el Source programábamos la consulta a la base de datos con el for each, filtrando por nombre:



Pero ¿por qué definir estas consultas a través de listados pdf y no directamente en la propia pantalla en la que le pedíamos al usuario los datos para los filtros?

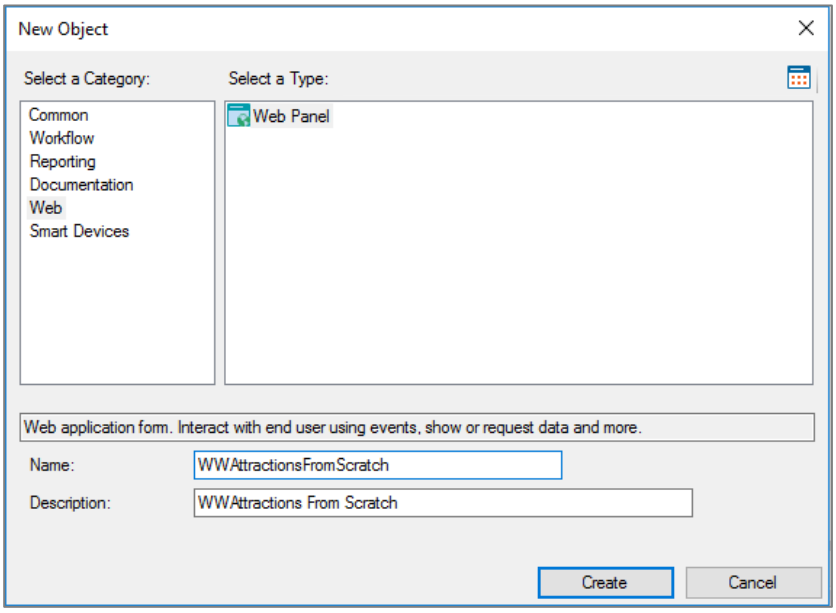
¿Por qué no colocar aquí, en lugar de los botones, una grilla que muestre las atracciones deseadas?



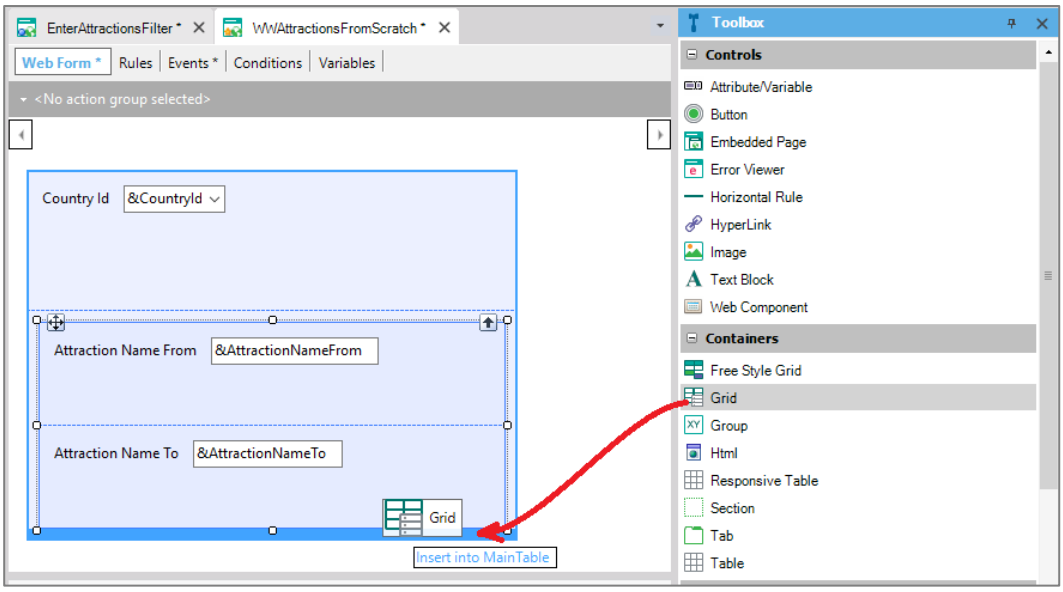
Los web panels, además de permitirnos definir variables para utilizarlas en acciones programadas en botones, nos permiten, -y es su objetivo fundamental- implementar consultas **interactivas** a la base de datos.

El término "**interactivas"** se refiere a que el usuario puede ingresar en la página del web panel una y otra vez distintos valores **– en variables -** y consultar a continuación datos de la base de datos que concuerden con esos valores ingresados, utilizándolos como filtros, como veremos ahora.

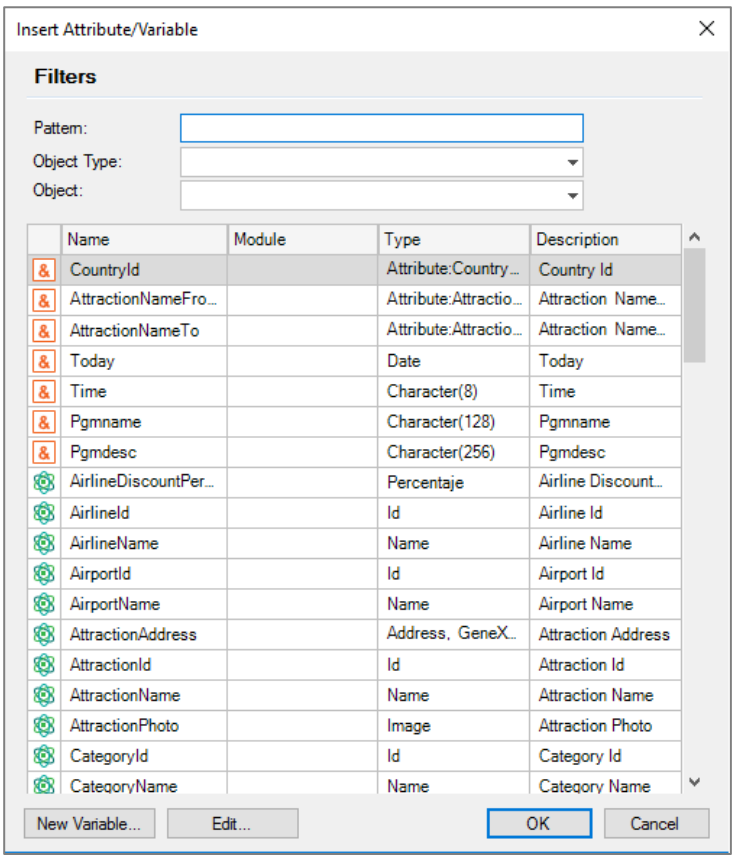
Grabemos este web panel con otro nombre. Como lo que vamos a implementar va a ser similar a un work with, llamémosle:



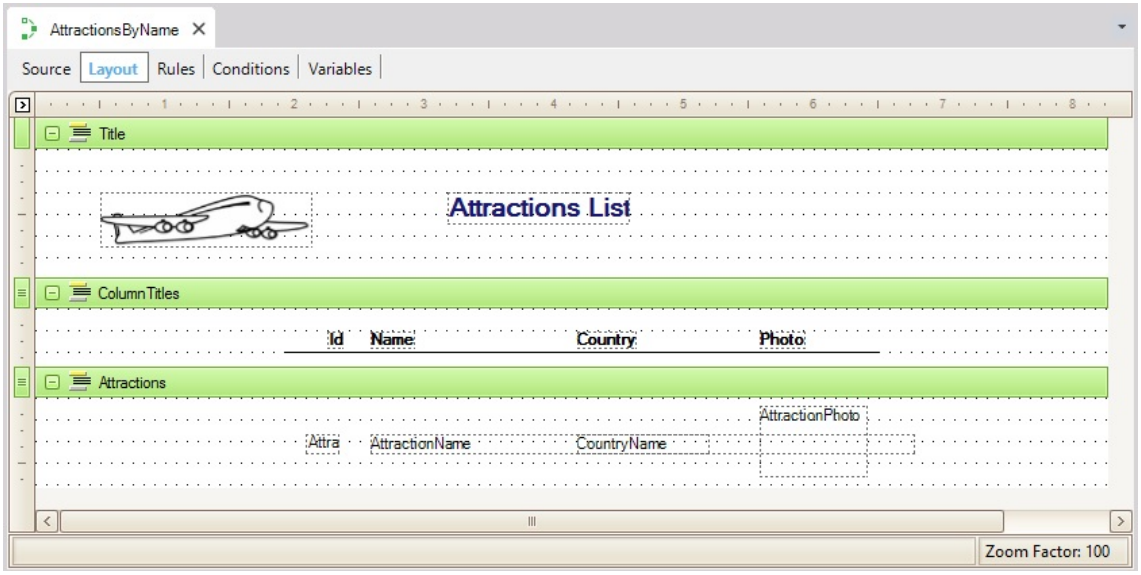
Eliminemos los botones, que ya no serán necesarios, así como los eventos asociados.
Ahora insertemos debajo de las variables un control de tipo grilla (**grid** en inglés):



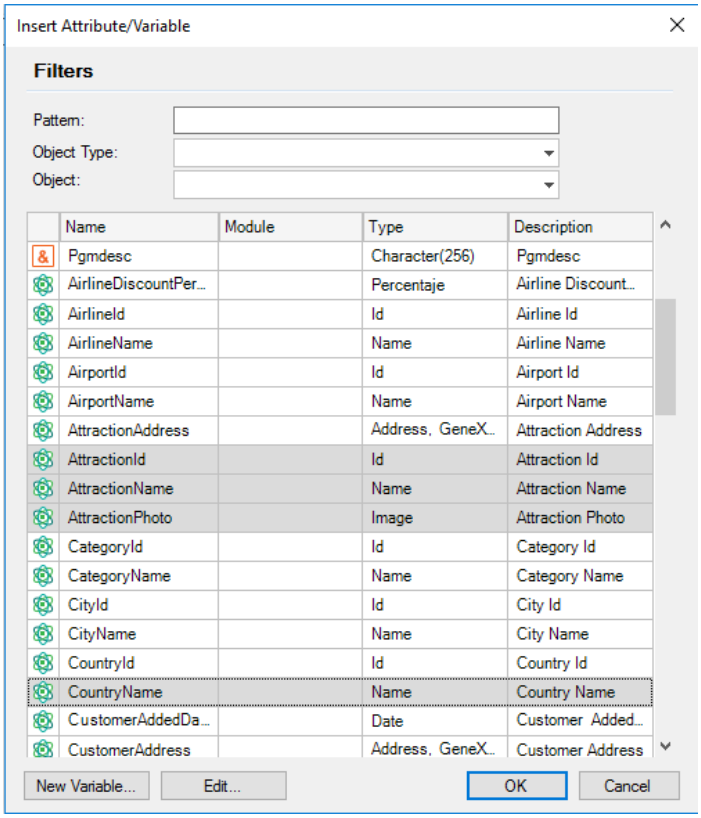
Se nos abre esta pantalla para elegir los atributos y/o variables que serán las columnas de este grid.



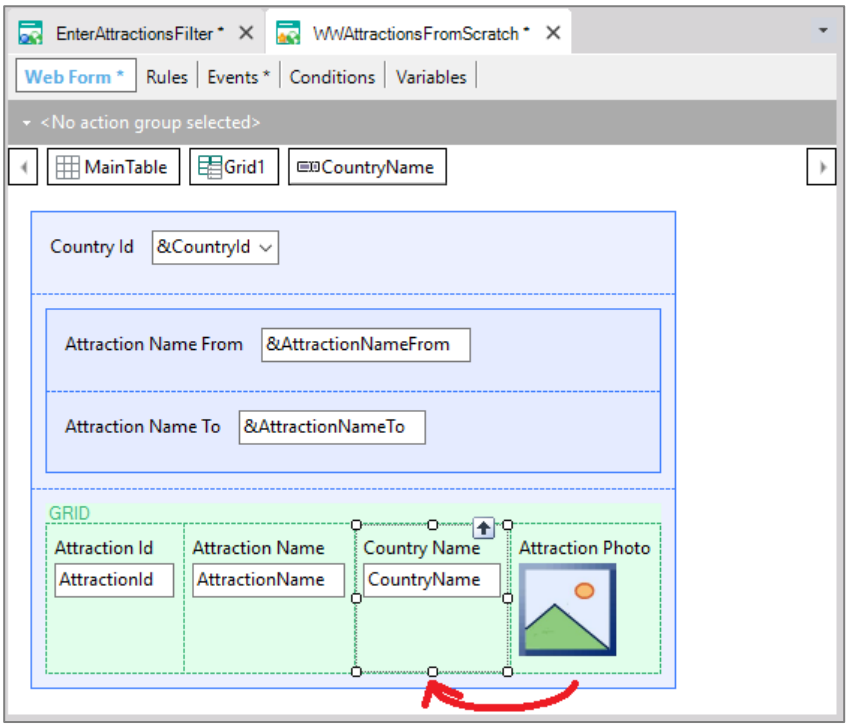
Como lo que queremos mostrar es lo mismo que mostrábamos en los listados pdf…



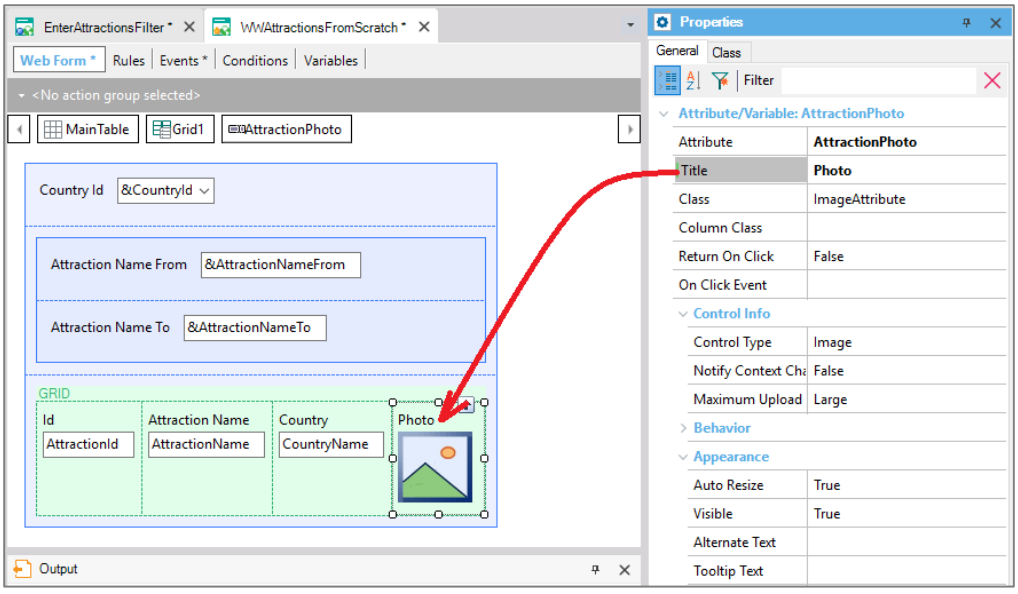
…elegimos los atributos AttractionId, AttractionName, AttractionPhoto y CountryName…



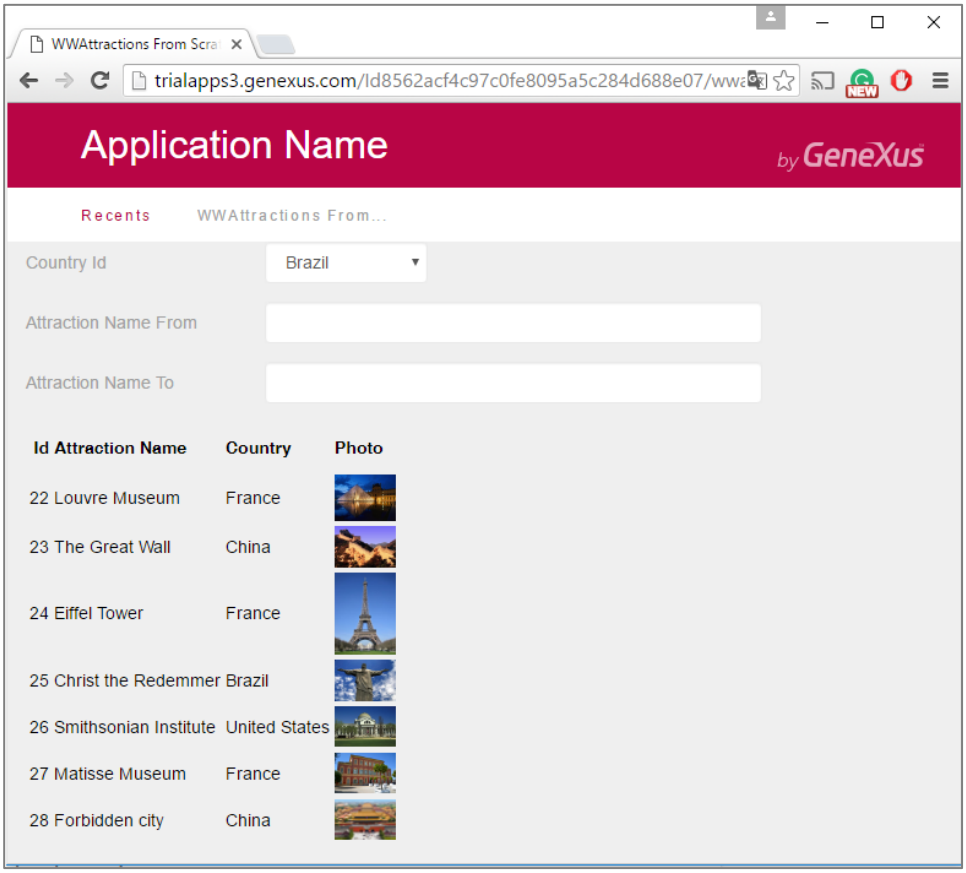
y presionamos OK. Vemos que creó un grid con esas columnas. Movamos CountryName para aquí:



Podemos cambiar los títulos de cada columna, editando las propiedades de cada uno de los atributos que conforman esas columnas del grid.

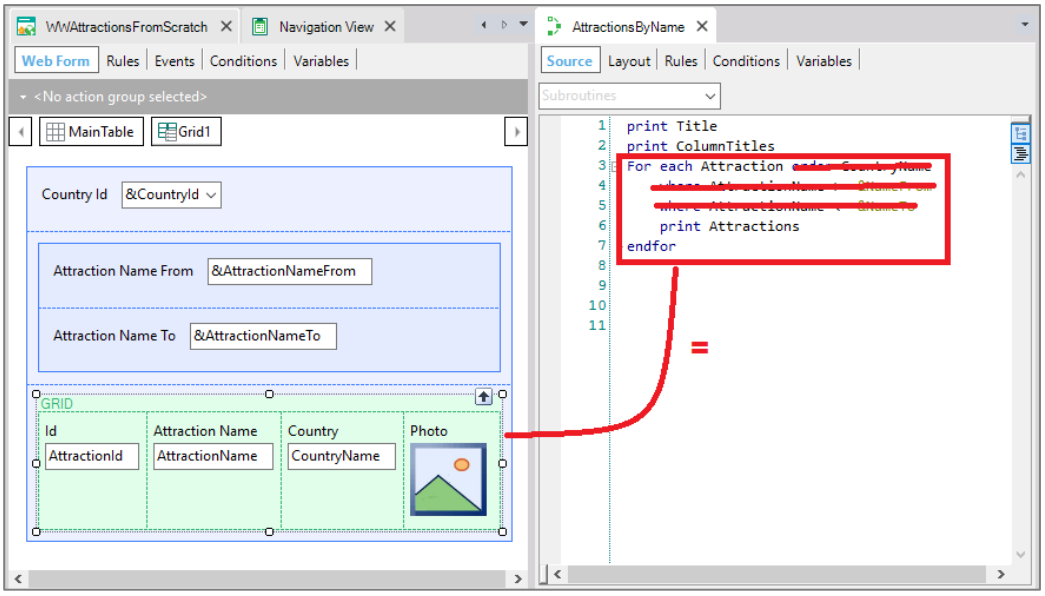


Los **atributos** en el form de un web panel son, por defecto, **de salida**. Es decir, son **readonly**. Esto significa que GeneXus interpreta que debe ir a la base de datos a buscar su valor para mostrarlo al usuario.
Presionemos F5 para ejecutar nuestro nuevo web panel así como lo tenemos hasta ahora.

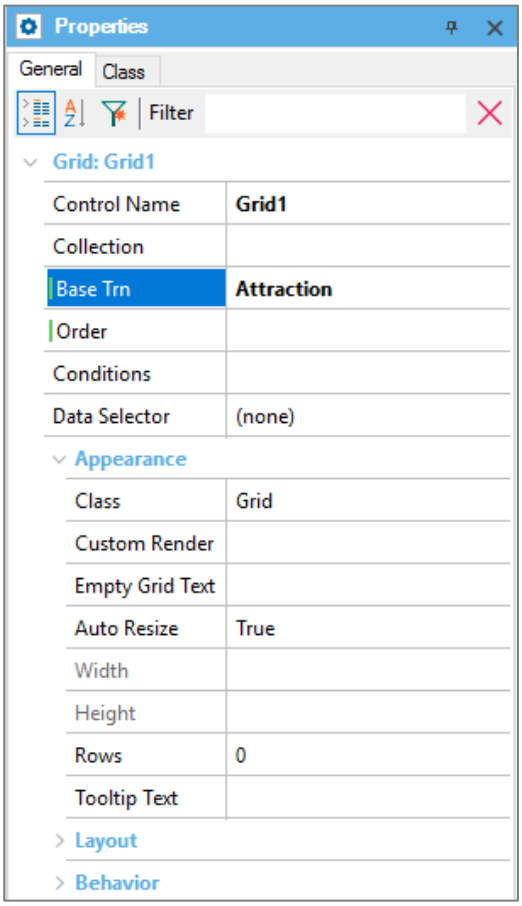


Vemos que salieron impresas todas las atracciones, con los datos que indicamos (el id, nombre, país y foto). Además vemos que salieron ordenadas por AttractionId.

Tan sólo con colocar un grid con esos atributos, GeneXus entendió que debía ir a la base de datos a navegar la tabla Attraction, accediendo a Country para traer el país de la atracción, tal como lo hacíamos con el comando for each (sin estas claúsulas):

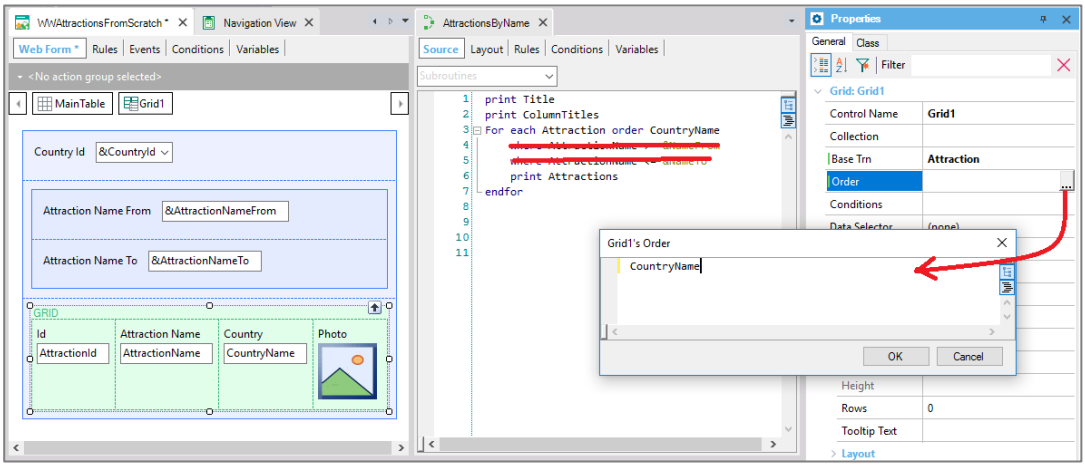


Si observamos las propiedades del grid, vemos que de hecho tiene una de nombre **Base Trn**, que es análoga a la transacción base del for each . De hecho, para asegurarnos de que para el grid se elija la tabla base Attaction, que es la que deseamos, es una buena práctica indicar la transacción base, igual que para el for each.

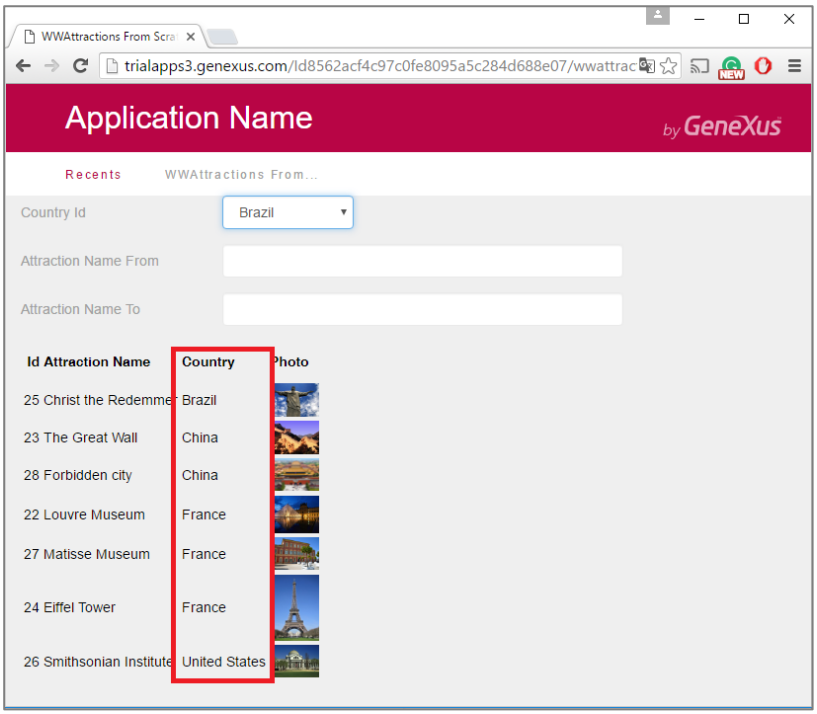


Por otro lado, observemos que tenemos una propiedad **Order** para el grid. Esta propiedad se corresponde con la cláusula Order del for each.

Así, si quisiéramos ordenar por nombre de país, como en el listado:

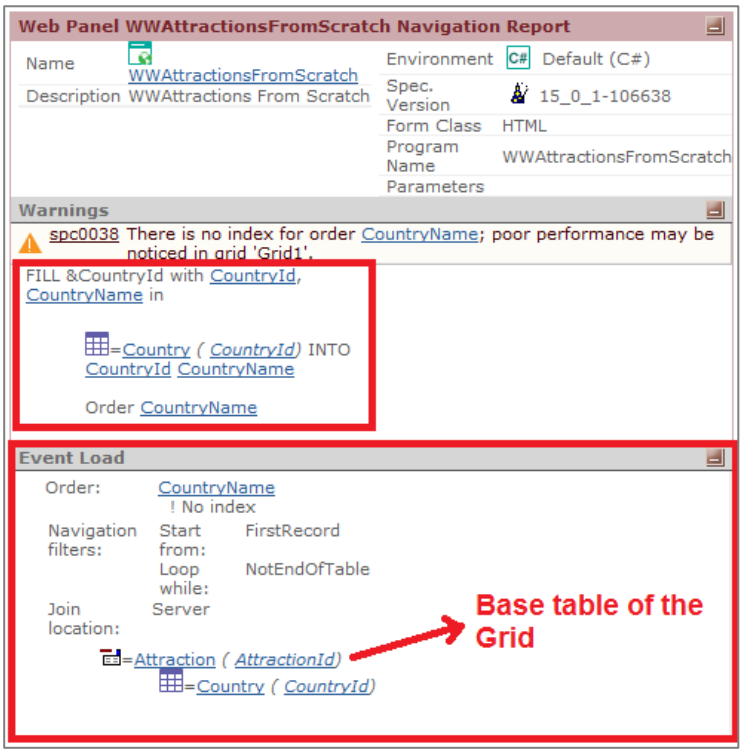


Presionemos F5.



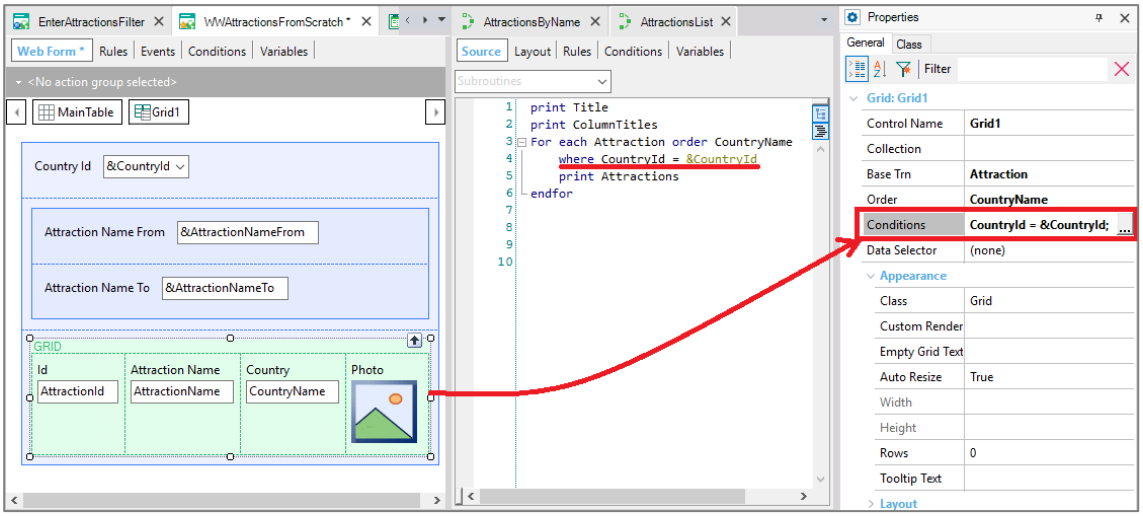
Y vemos que ahora el grid sale ordenado por nombre de país.

Observemos el listado de navegación del web panel:



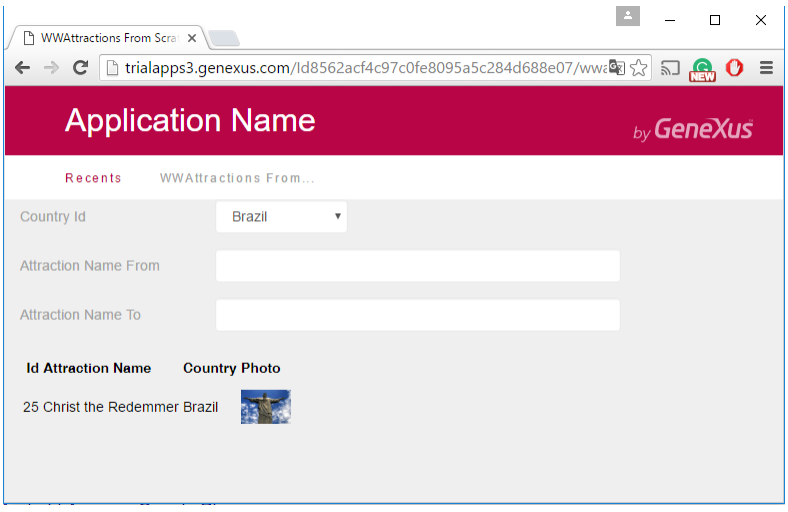
Aquí nos está indicando la navegación que tiene que realizar para cargar el combo box de la variable &CountryId, que por ahora no estamos usando para nada.
Y aquí está indicando la navegación que tendrá que realizar para cargar (Load) el grid. Vemos que lo que lista para esta carga es idéntico a lo que lista para un for each. Vemos que eligió la tabla Attraction, recorriéndola por CountryName, el atributo de la propiedad Order. Recorrerá toda la tabla. Y por cada registro de Attraction a ser cargado, accederá a la tabla Country para mostrar el CountryName de la atracción.

Hasta ahora no hicimos nada con las variables. Pero queríamos utilizarlas para filtrar los datos que se muestran en el grid, tanto por país como por nombre de atracción.
En AttractoinsList filtrábamos por país. ¿Cómo indicamos este filtro para el grid? A través de la propiedad **Conditions**:

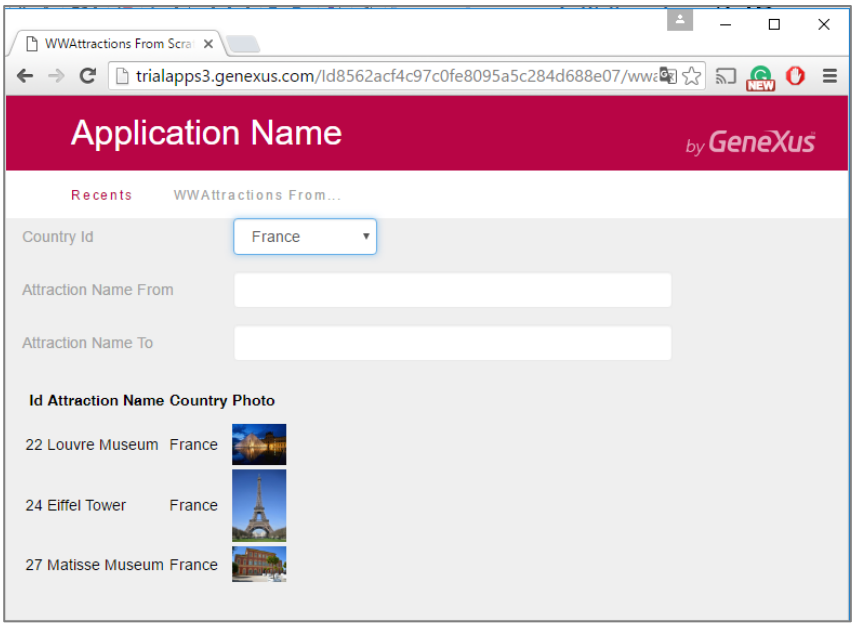


Aquí le estamos diciendo a GeneXus que queremos que cuando recorra la tabla base del grid, Attraction, filtre aquellos registros para los cuales el CountryId de la atracción es igual al valor que el usuario haya elegido en el combo &CountryId.

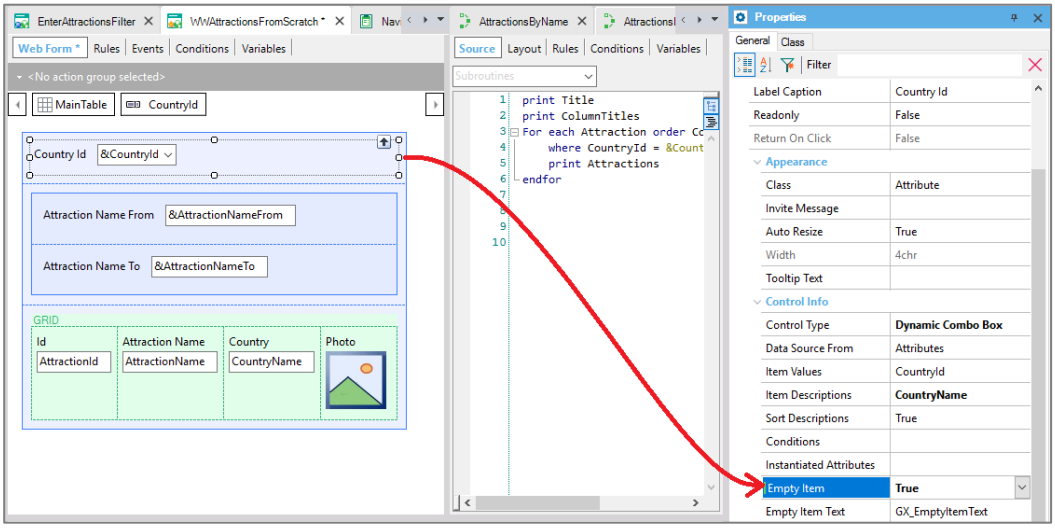
Presionemos F5.



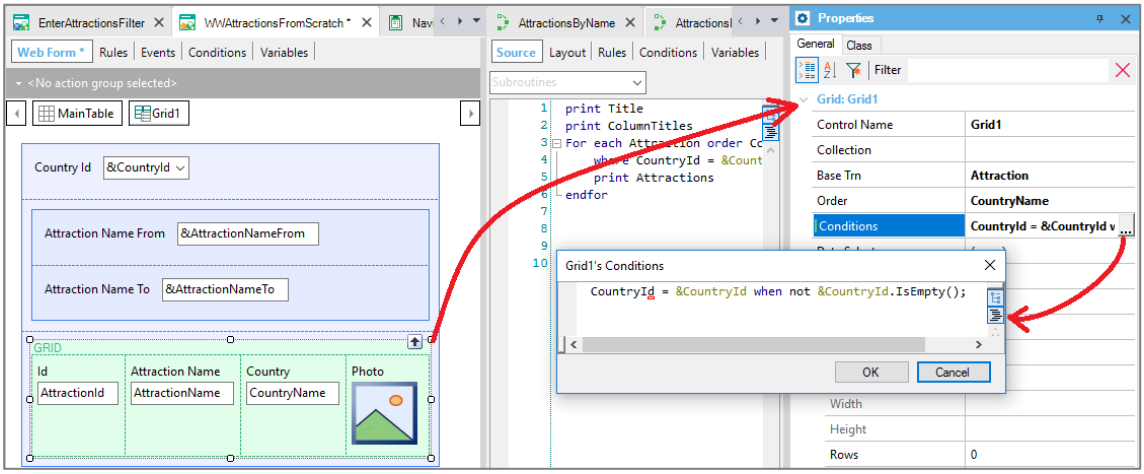
Observemos que por defecto el combo toma el valor Brazil, y que en el grid sólo vemos la atracción de Brazil. Si elegimos Francia vemos que se refresca la pantalla, volviéndose a cargar el grid, ahora con las atracciones de Francia.



Probablemente querremos que el combo aparezca sin valor elegido la primera vez, y que en ese caso se muestren las atracciones de todos los países.
Para ello editemos las propiedades del combo box… y pasemos a True la de nombre Empty Item. Esto hará que se agregue una opción "(none)" al combo. Corresponderá al valor empty, es decir, vacío, cero.

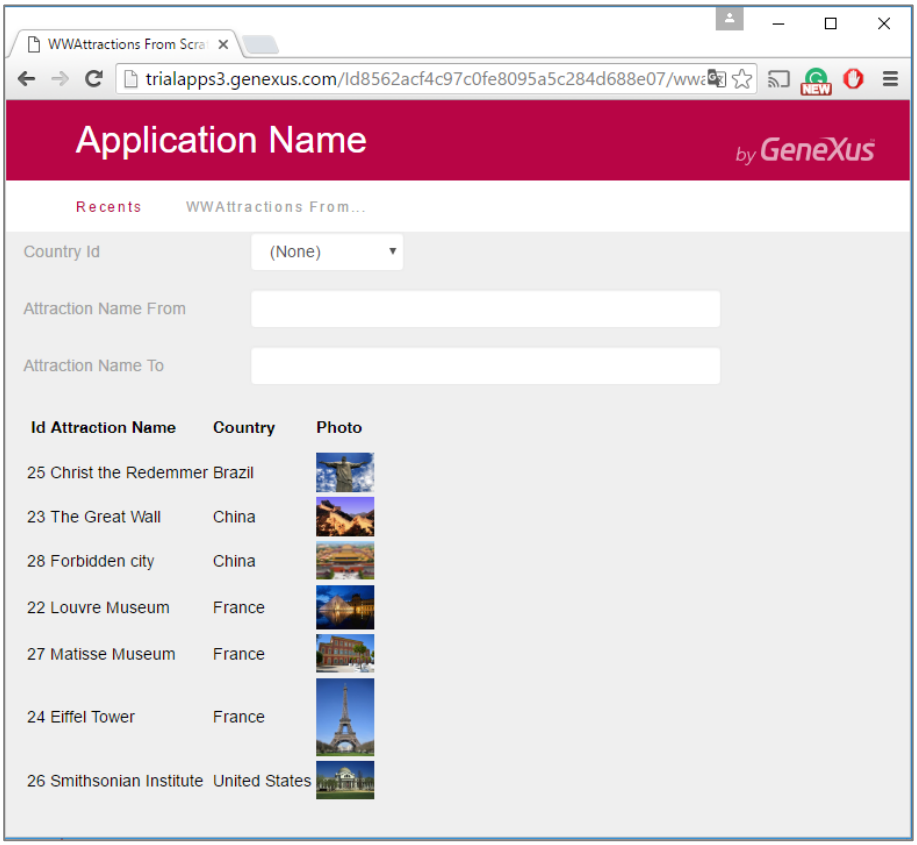


Entonces vayamos a la propiedad **Conditions** y especifiquemos que queremos que se aplique esta condición solamente cuando el combo no tiene el valor vacío. Cuando sí lo está, que no la aplique.

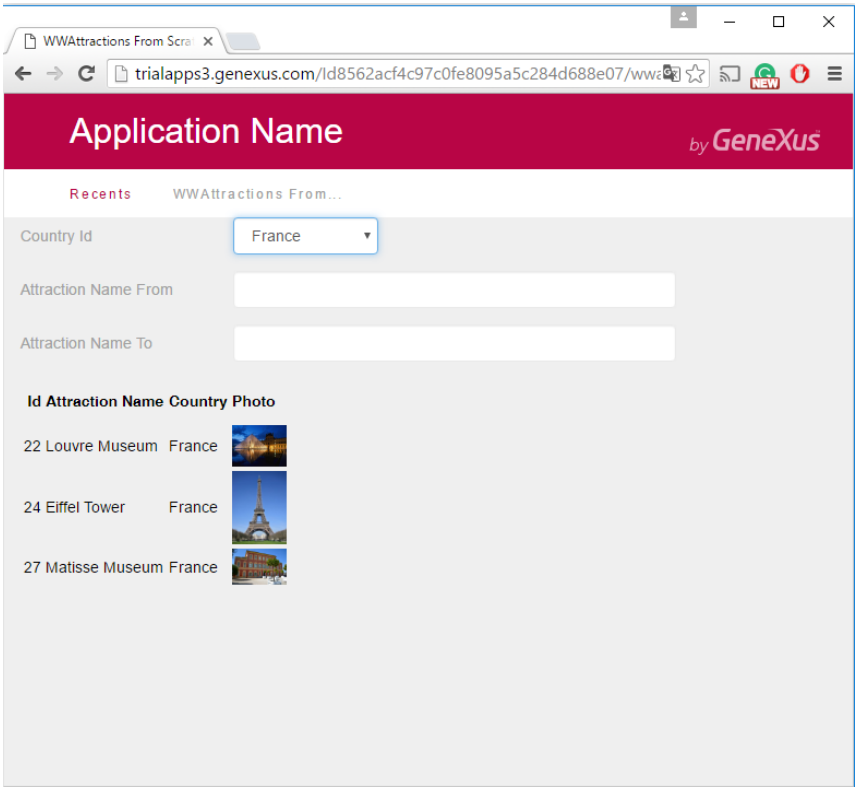


Ejecutemos…

Vemos cómo aparece el valor (None) en el combo, y además cómo para este caso, no se están filtrando las atracciones. Se muestran todas:

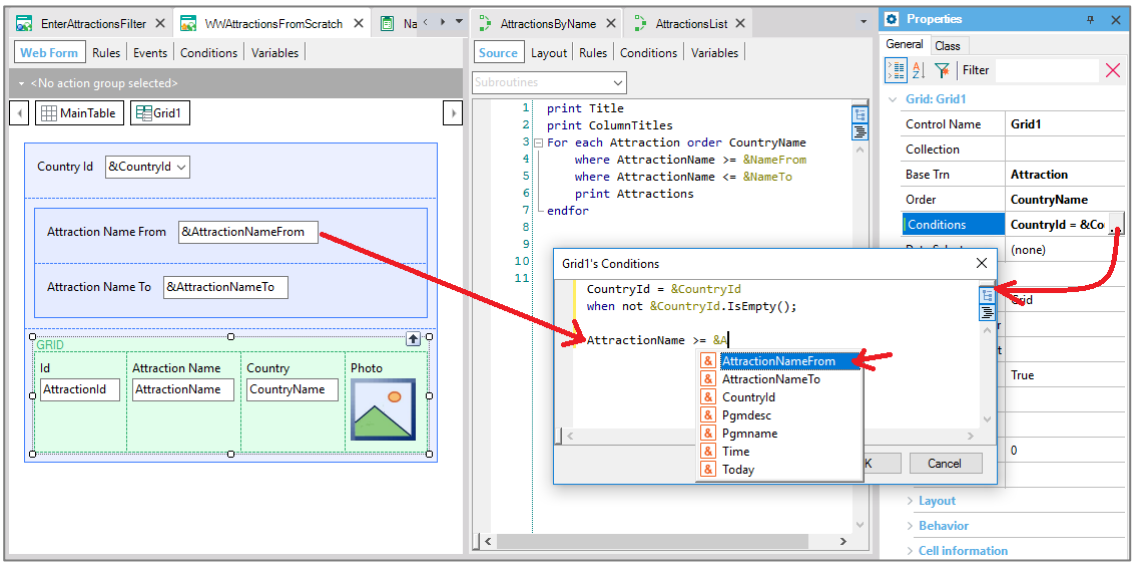


Si ahora elegimos, por ejemplo, Francia:

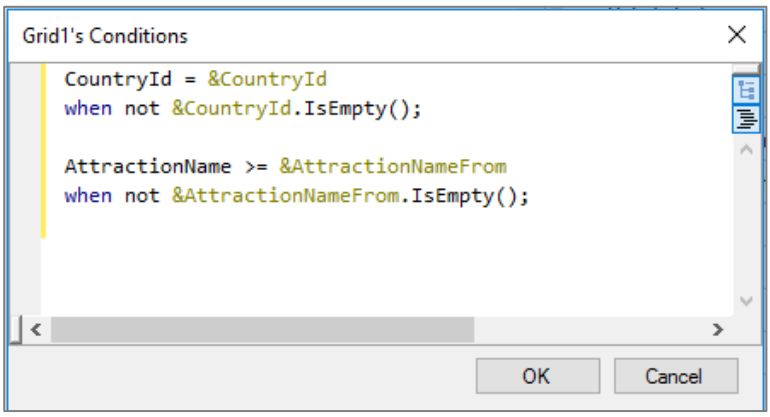


Como la variable no tiene el valor vacío, sí se aplica el filtro, y se muestran las atracciones de Francia.

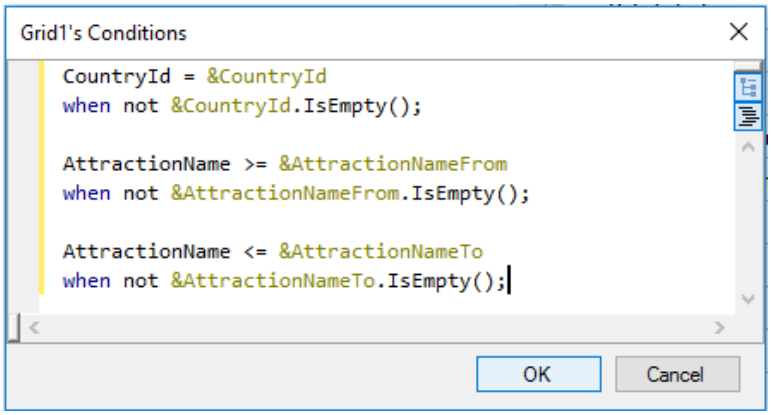
También tendríamos que agregar los filtros por nombre de atracción, que queremos que se sumen al otro filtro. Entonces… si en el listado filtrábamos en el for each con estas dos cláusulas where …en el grid las agregaremos como condiciones:



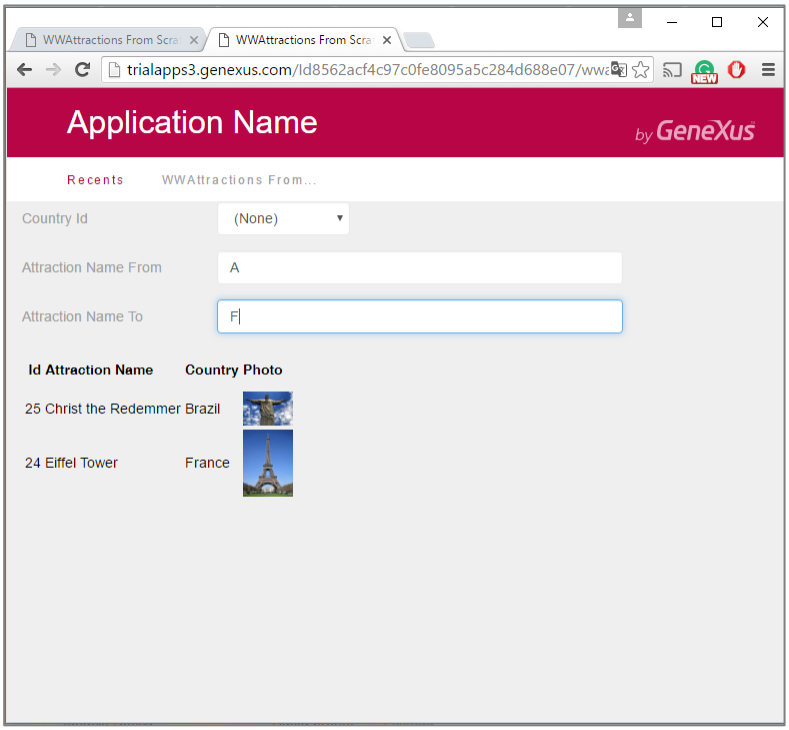
AttractionName mayor o igual al valor de la variable &AttractionNameFrom del form, que el usuario habrá ingresado, de desearlo. Otra vez, si el usuario no ingresa valor en la variable, no querremos que este filtro se aplique. Entonces usamos la cláusula when . Esta cláusula también se puede utilizar en la cláusula where del for each, en forma completamente análoga.



Y agregamos el otro filtro:

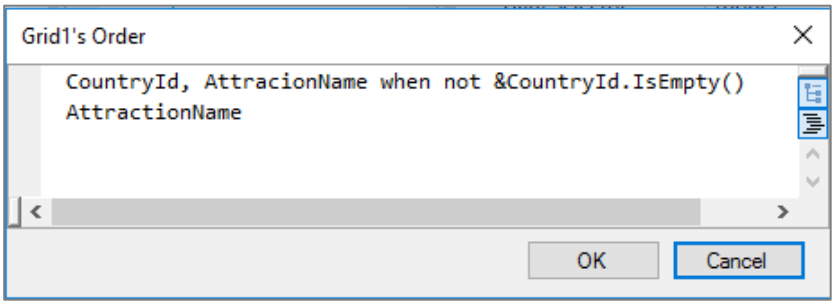


Ejecutemos nuevamente. Y elijamos ver las atracciones entre la A y la F:



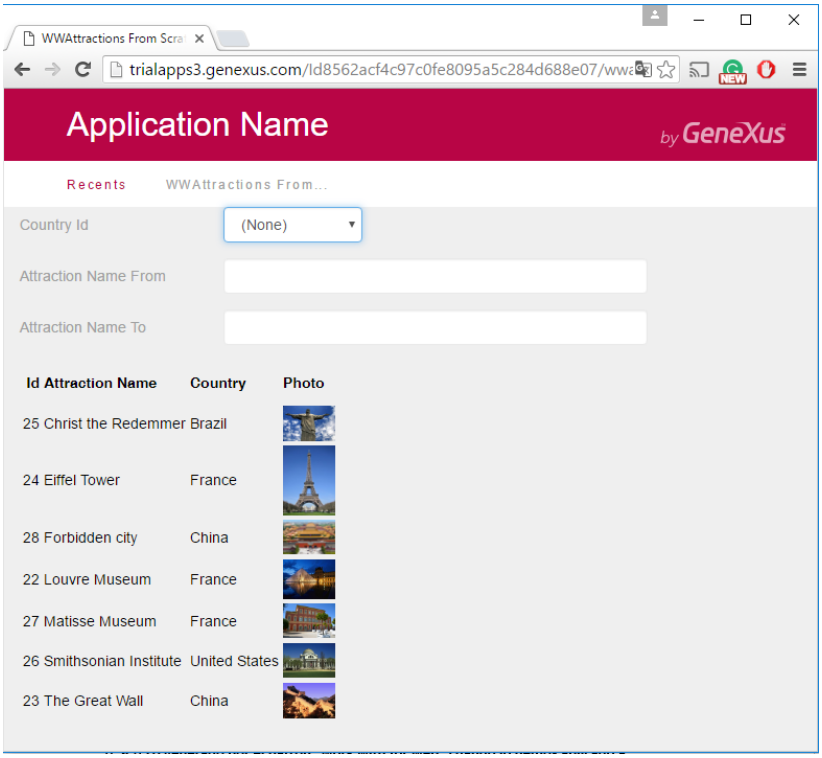
Podemos pedirle al web panel que si el usuario elige un país, entonces ordene la información por CountryId y dentro de CountryId por AttractionName, y que si no, la ordene por AttractionName. Esto pensando en optimizar la búsqueda de los registros de la tabla.

Para ello, editamos del grid la propiedad Order y escribimos primero el orden, condicionado a que el usuario haya elegido un país en el combo. En ese caso se filtrará por ese país, pero además las atracciones saldrán listadas alfabéticamente para ese país. Y si el usuario dejó el combo con el valor “(none)”, es decir, vacío, entonces se elegirá el orden siguiente, que es por AttractionName:

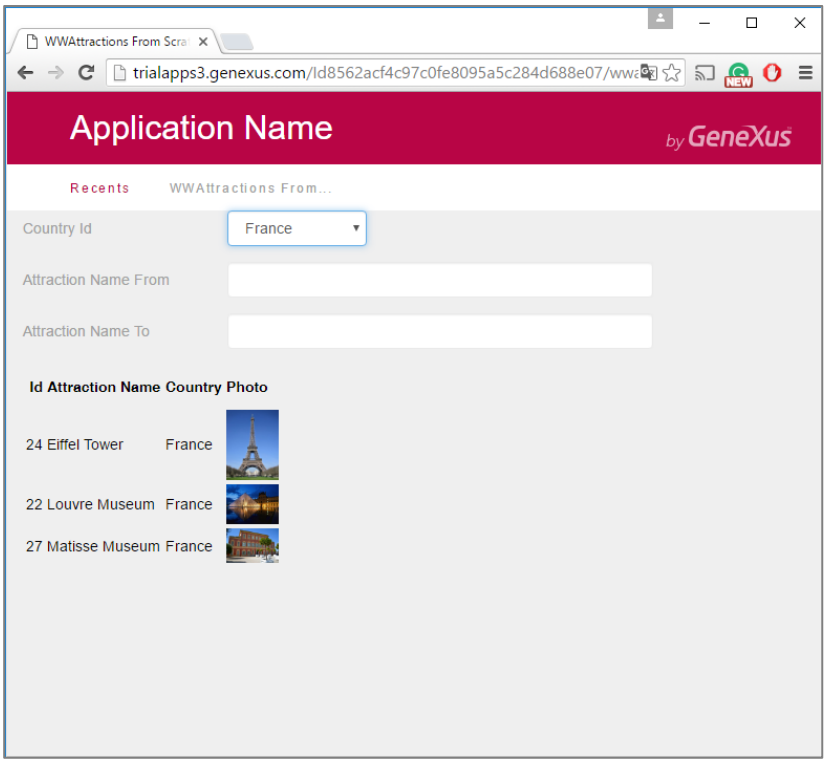


No nos detendremos en esto aquí. Lo dejamos apuntado únicamente para mostrar que también es posible condicionar la forma en que se quiere ordenar la información. Esto es idéntico en un for each.

Ejecutemos:

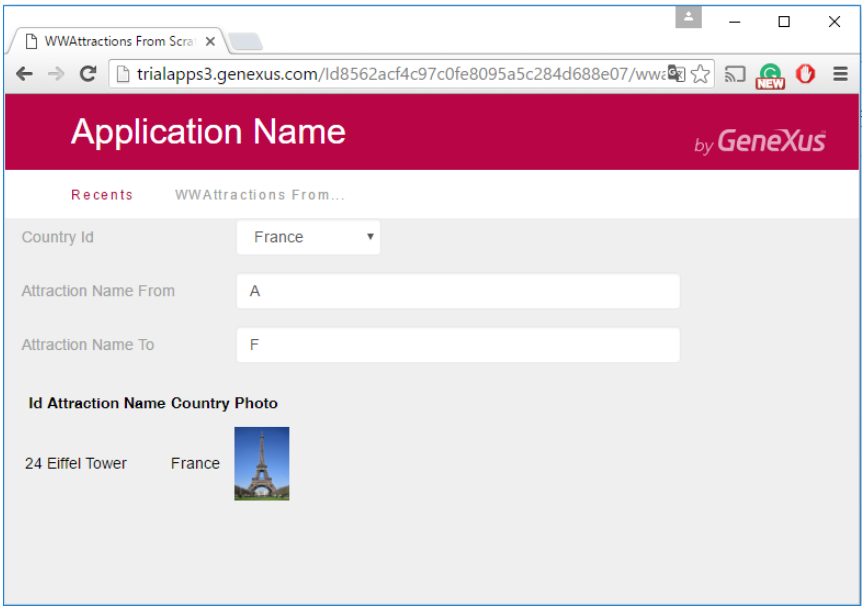


Aquí ordenó por AttractionName. Y si elegimos Francia, ordenará por el Id de Francia y dentro de él por AttractionName.

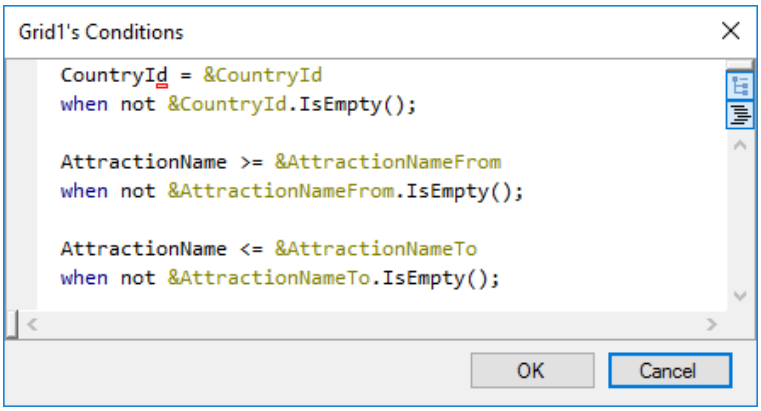


En definitiva, siempre vemos las atracciones ordenadas alfabéticamente.

Si dentro de las atracciones de Francia, queremos las que se encuentran entre la A y la F:



Vemos que el grid se cargó filtrando por las tres condiciones que habíamos escrito:

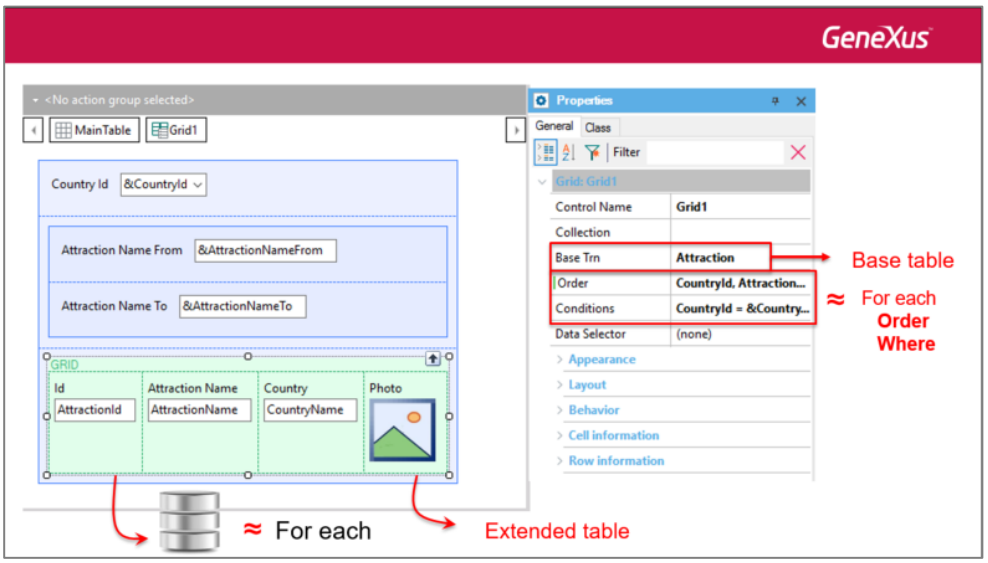


Resumiendo lo hecho hasta el momento:

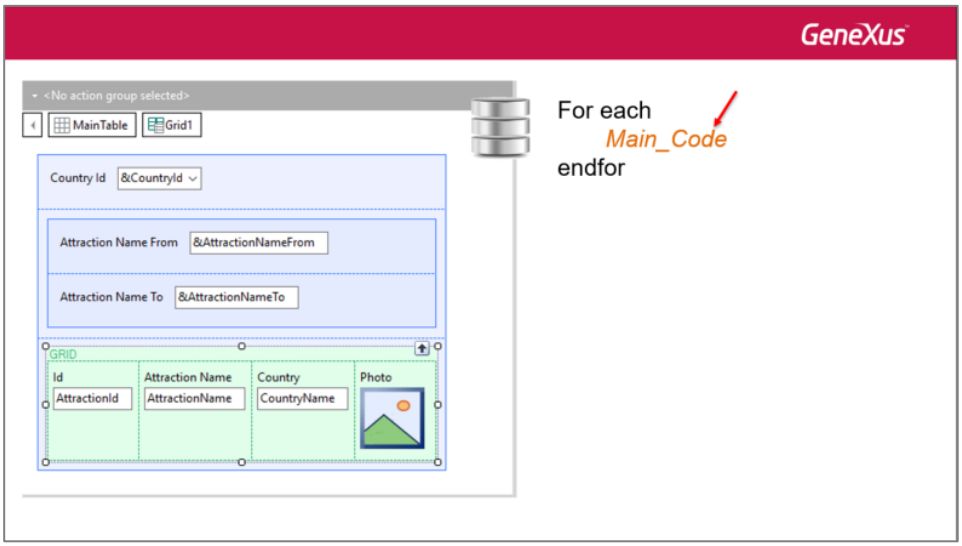
Hemos implementado un web panel en el que incluimos algunas variables para que el usuario les ingrese valor, y en el que insertamos un control Grid con atributos.

Los atributos corresponden a información de la base de datos, por lo que GeneXus entiende que debe ir a buscar esa información. Un grid con atributos es como un for each. Por ello contamos con la propiedad **Base Trn**, como la del for each, para especificar el nivel de la transacción cuya tabla asociada queremos recorrer. Llamamos a esa tabla, **tabla base del grid**. Si no la especificamos, como también sucede con un for each, GeneXus la infiere en base a los atributos que se utilicen. Aunque este caso no lo veremos.

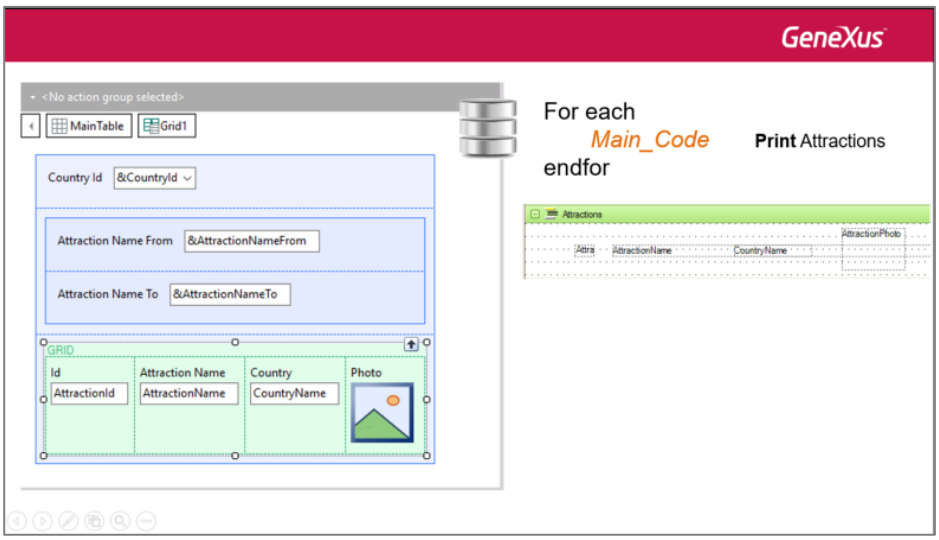
Todos los atributos del grid deberán pertenecer, como en el caso de un for each, a la tabla extendida de esa tabla base.
Así como en un for each ordenamos la información con la cláusula Order y filtramos los datos a ser devueltos por la consulta con una o varias cláusulas where, para hacer lo mismo con el grid tenemos las propiedades Order y Conditions, respectivamente.



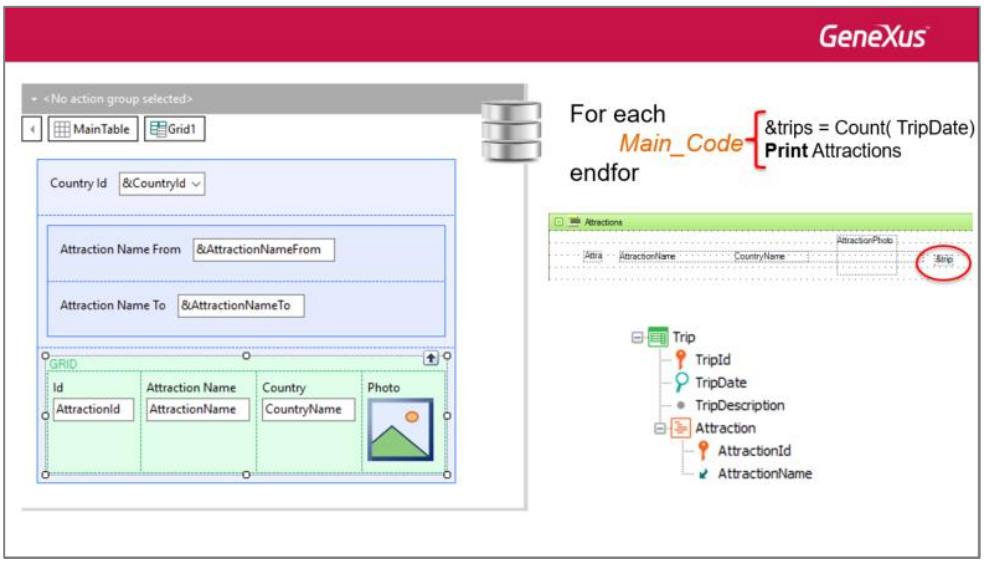
Ahora bien, en un for each programamos lo que queremos que se haga con cada registro que cumpla las condiciones, dentro de su cuerpo.



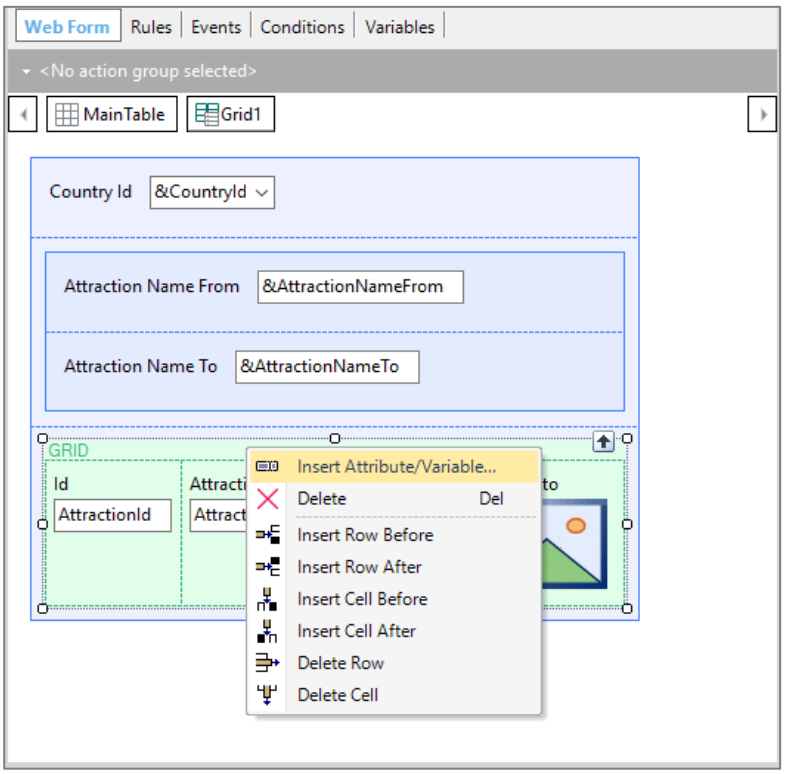
Por ejemplo, en el listado de las atracciones turísticas, el comando print Attraction enviaba a imprimir en la salida lo que en nuestro caso sería la línea del grid. En el caso del grid no hay que especificarlo. Esto se hace automáticamente.



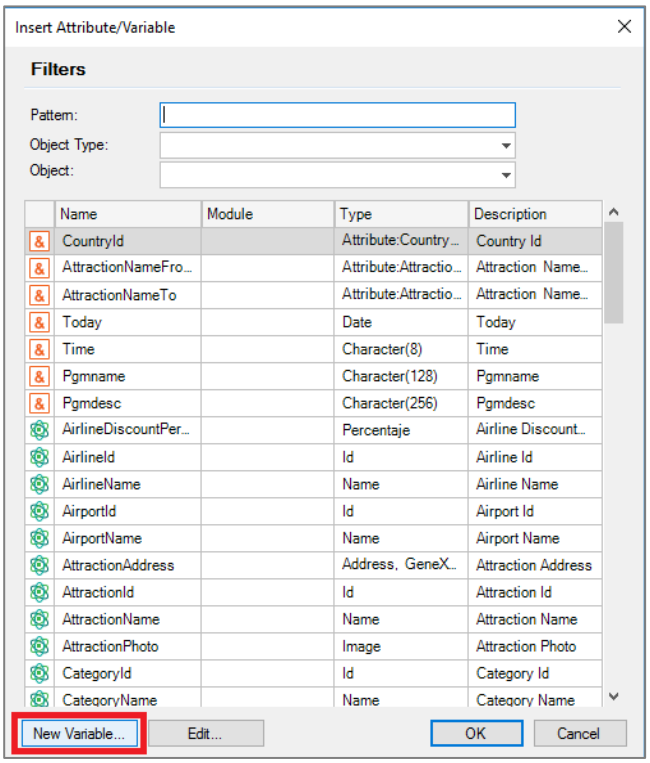
Pero, por ejemplo, imaginemos que tenemos una transacción Trip que registra las excursiones que ofrece la agencia de viajes. En forma muy simplificada, imaginemos que de una excursión sólo se registra la fecha en la que se realizará y la descripción y luego se registran las atracciones turísticas que serán visitadas por esa excursión. Bien, entonces ahora supongamos que en el listado de atracciones turísticas queremos ver también la cantidad de trips que tiene asociados cada atracción. Para ello alcanzaba con definir una variable &trips, numérica, y asignarle dentro del cuerpo del for each, (es decir, cuando el for each está posicionado en el registro de su tabla base que está por procesar) el resultado de contar los trips de esa atracción. Y colocar esa variable en el print block.



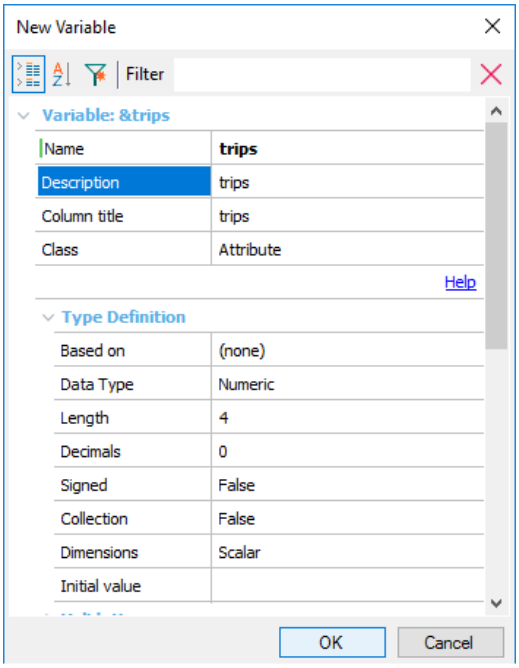
Para hacer lo mismo en el web panel hacemos botón derecho sobre el grid e Insert Attribute/Variable



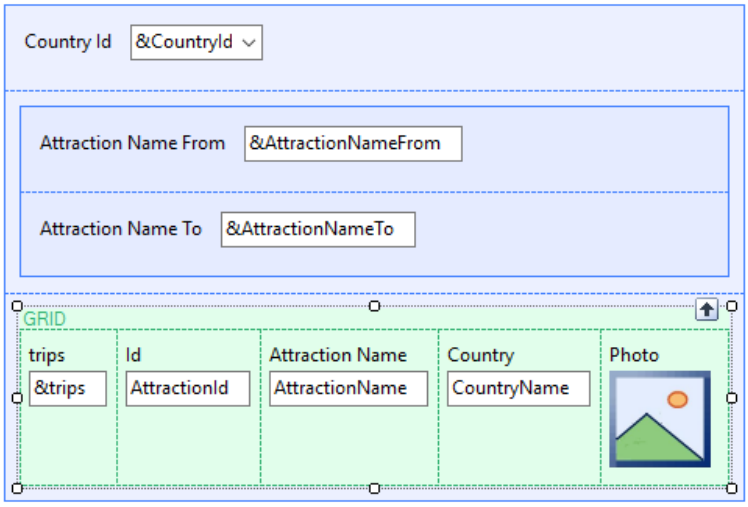
Luego New Variable:



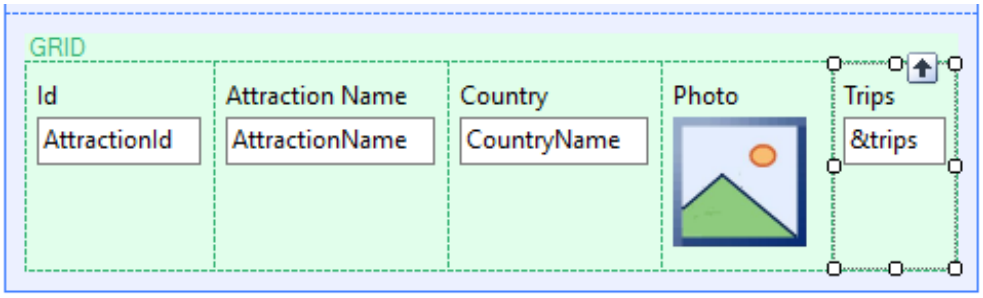
Y definimos la variable trips:



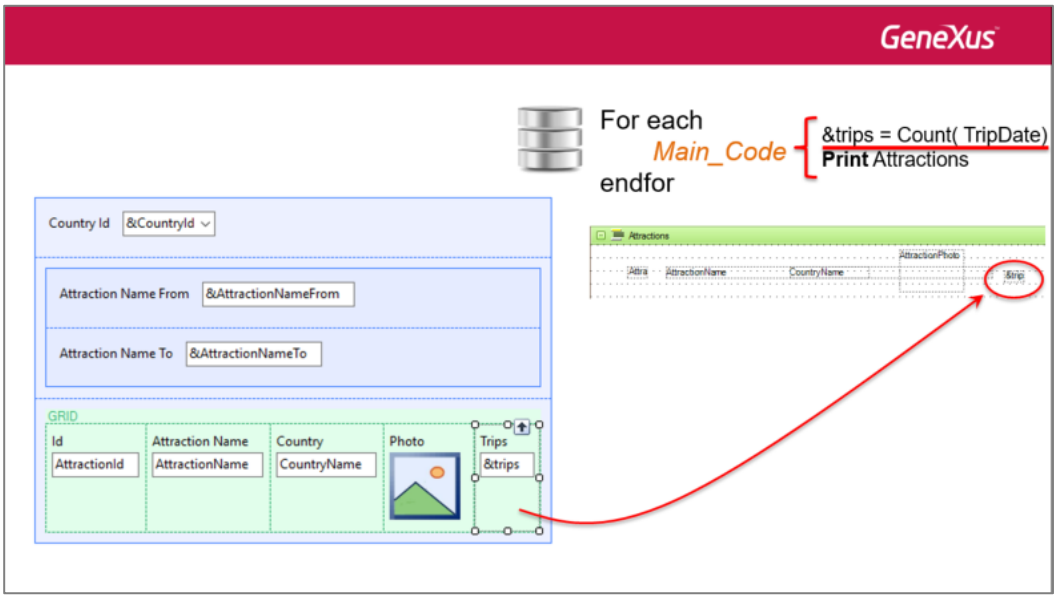
Al presionar OK… vemos que se la insertó como columna del grid. La movemos para que ocupe la posición que nos interese dentro del grid.



Y le cambiamos la propiedad **Title** para que el título de la columna salga en mayúscula.

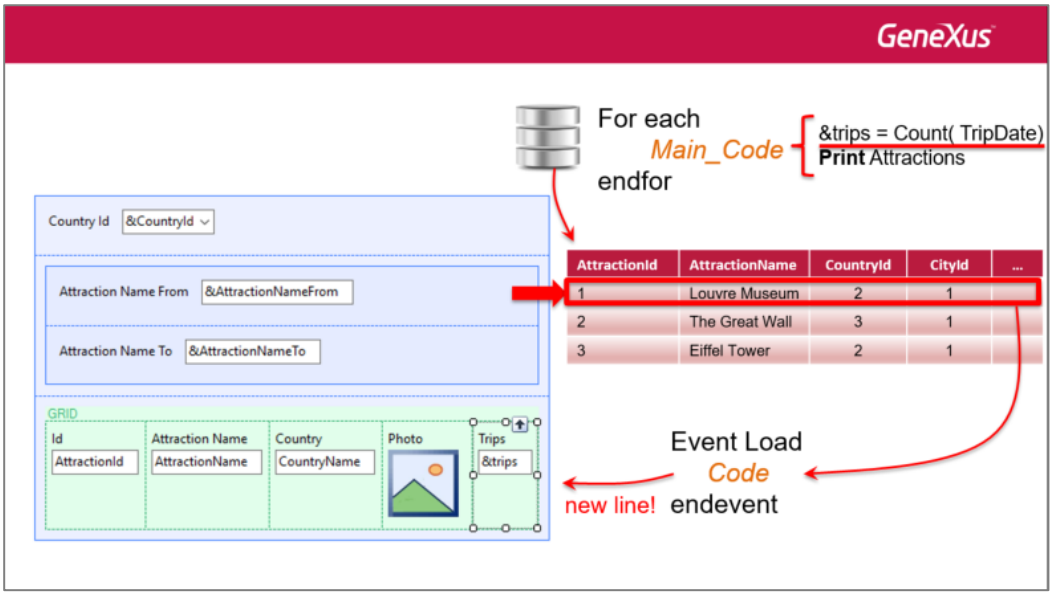


Esto corresponde a haber insertado la variable en el printblock . Pero ¿dónde le decimos cómo se calcula?

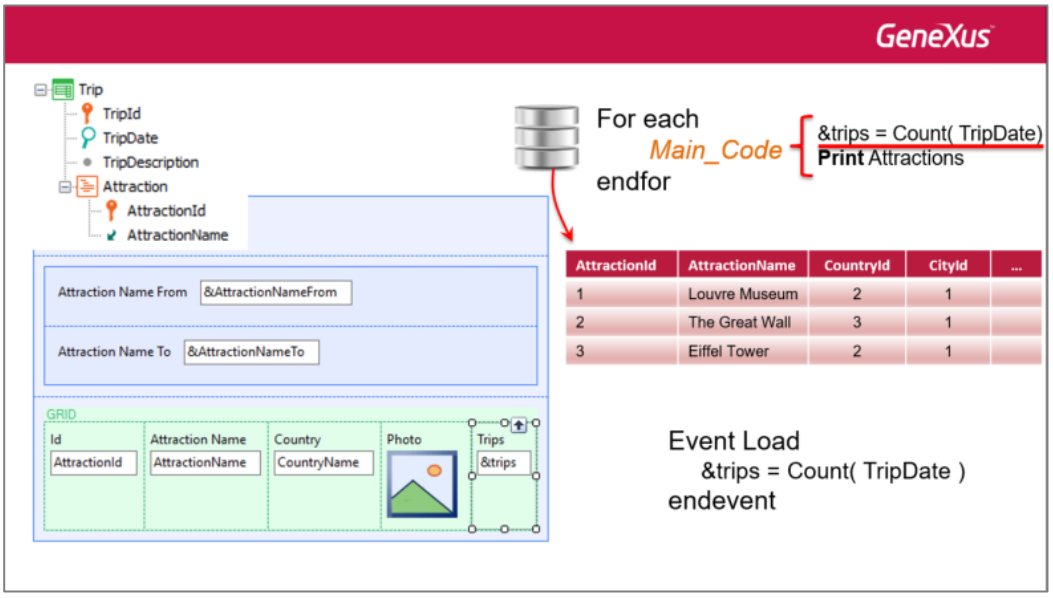


En el for each es dentro de su cuerpo. ¿Aquí?

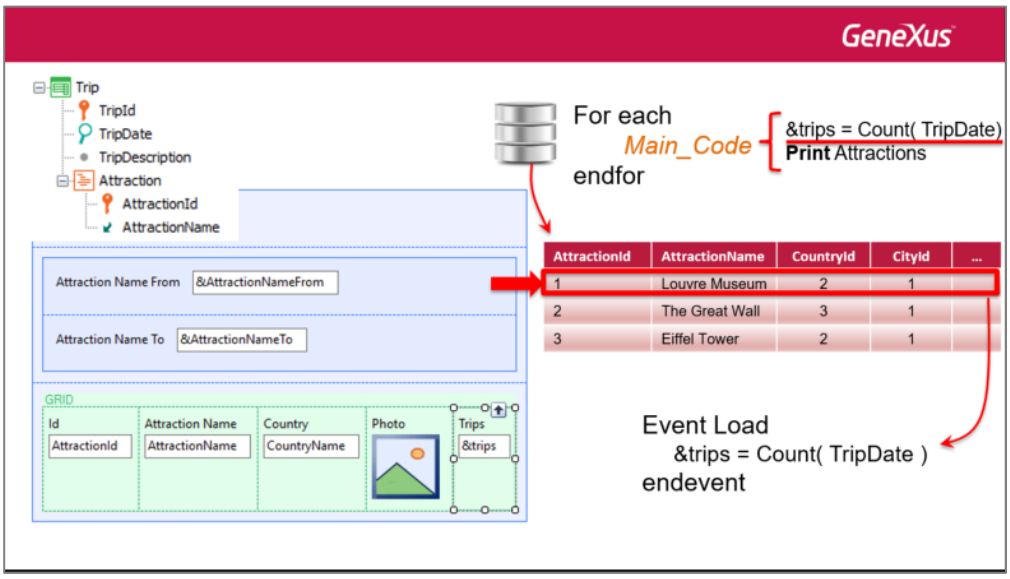
Contamos con el evento **Load**, del sistema. Allí dentro programaremos lo que queremos que se ejecute cuando se está posicionado en un registro de la tabla base del grid, inmediatamente antes de que la línea correspondiente sea cargada en el grid.



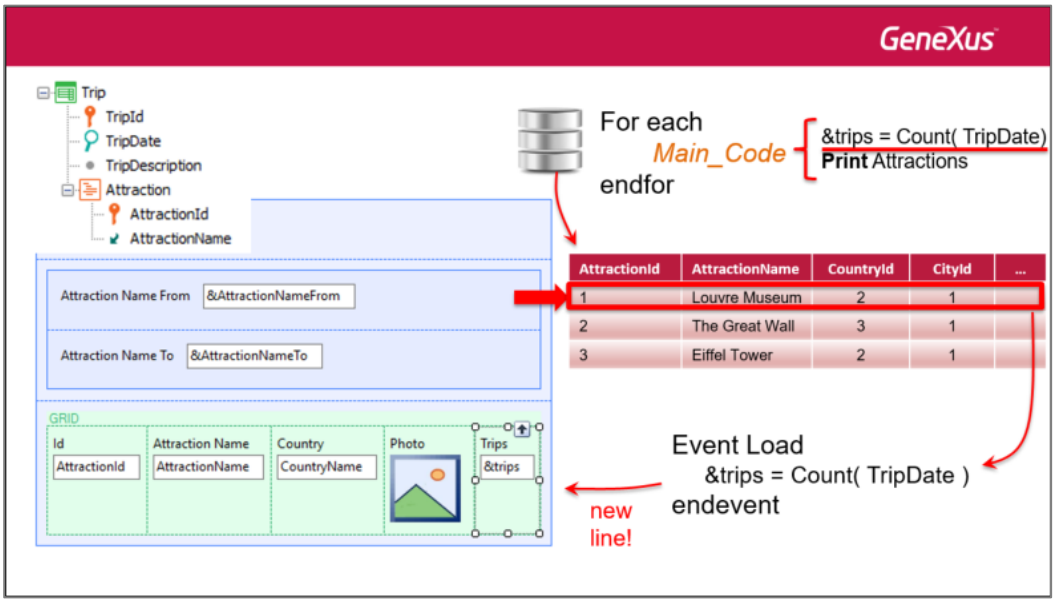
En nuestro caso, allí es donde asignaríamos valor a la variable &Trips:



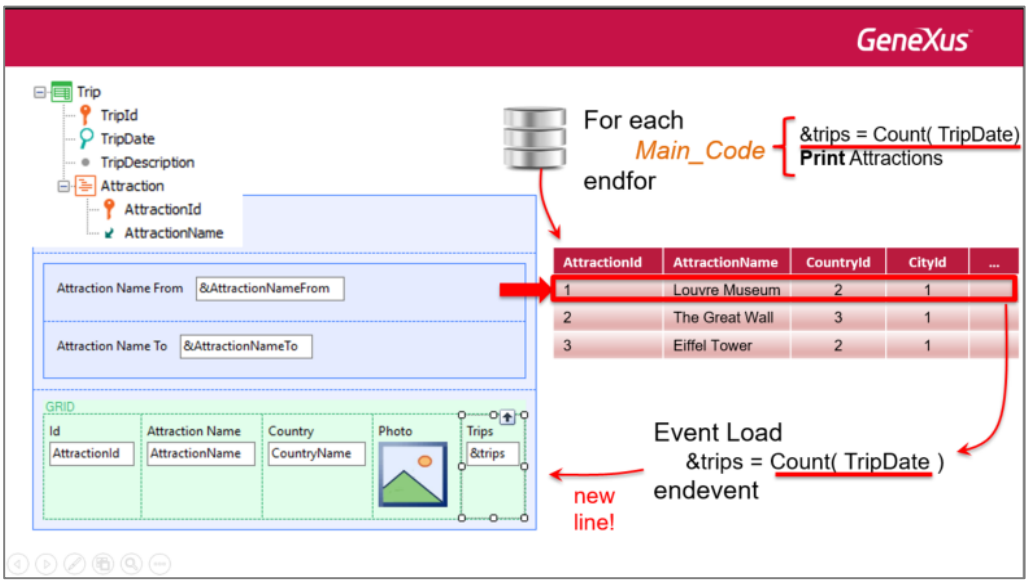
El evento Load se va a ejecutar automáticamente por cada registro



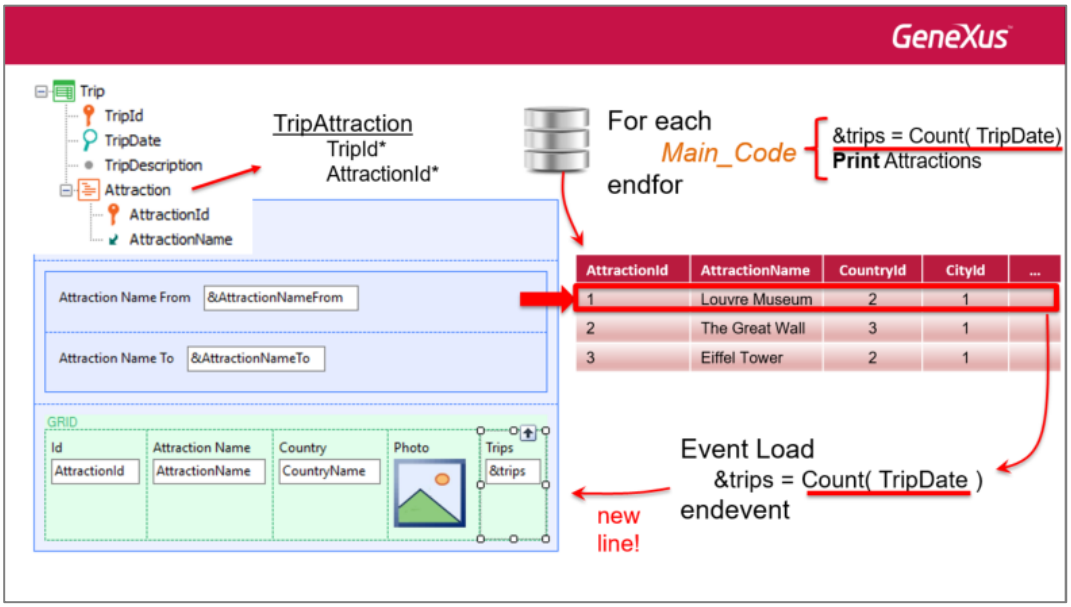
de la tabla base del grid que cumpla con las condiciones de filtro, inmediatamente antes de que se agregue la línea al grid.



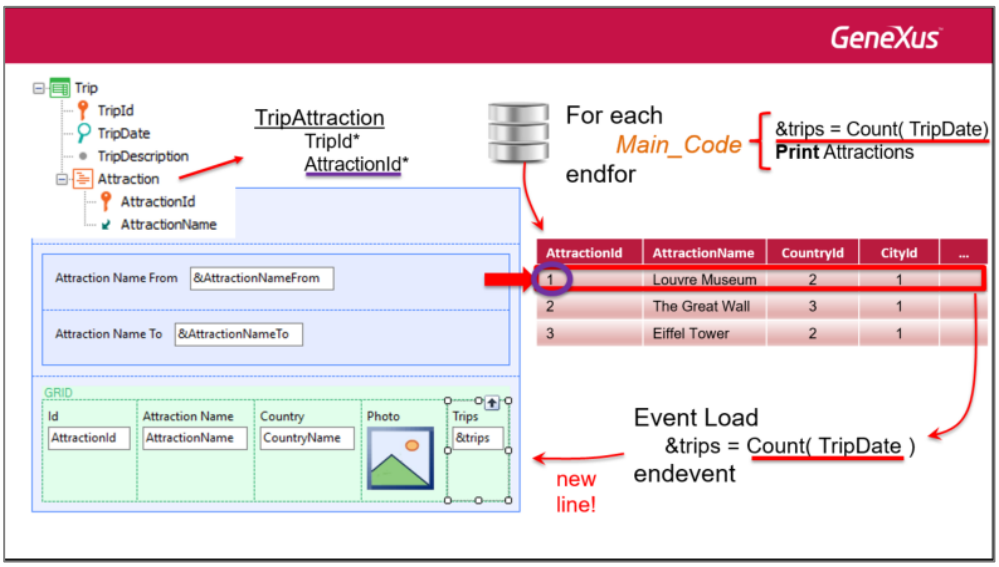
Es por esa razón que al ejecutarse su código se sabe que estamos trabajando con un registro de la tabla base y su extendida, y esta fórmula inline:



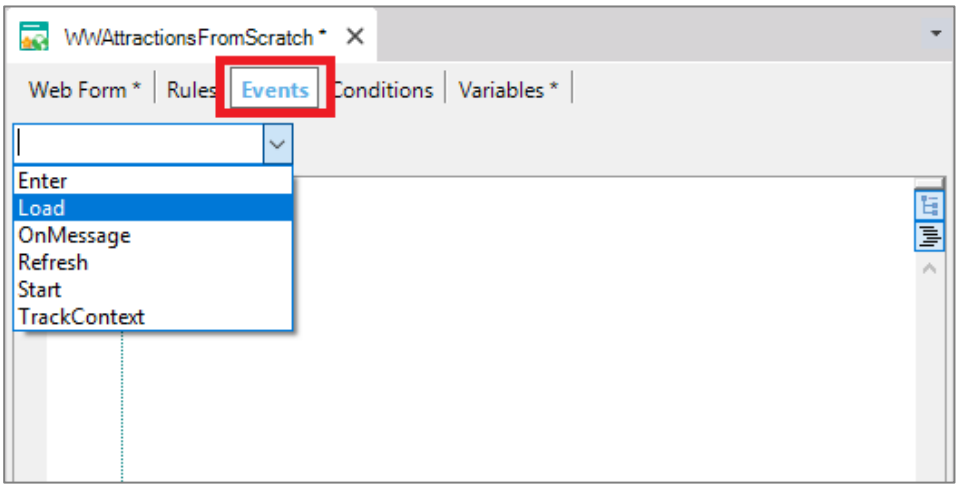
no contará todos los trips



sino solo aquellos de la tabla TripAttraction que correspondan a ese AttractionId, el del registro de Attraction que estamos a punto de cargar en el grid.



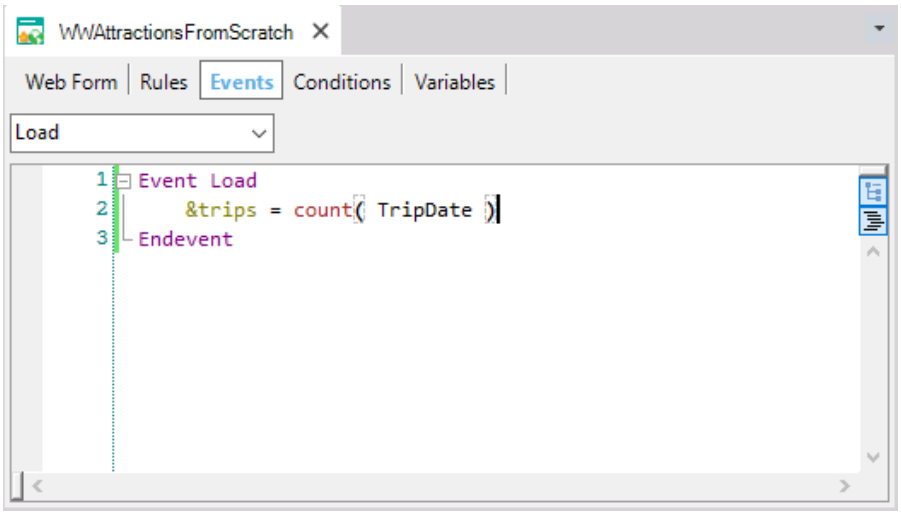
Implementémoslo en GeneXus. Ya tenemos la transacción Trip creada. Vayamos a la sección de eventos del Web panel. En este combo, se nos ofrecen los eventos predefinidos, es decir, eventos del sistema que se producen en momento específicos, y a los que podremos programarles código.



Entre ellos, vemos el evento Load. Al elegirlo, vemos que nos aparece este código:

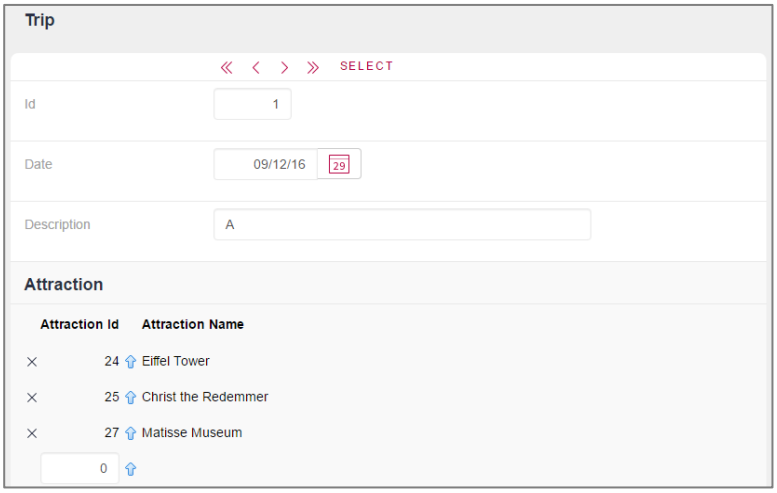
```
Event Load
Endevent
```

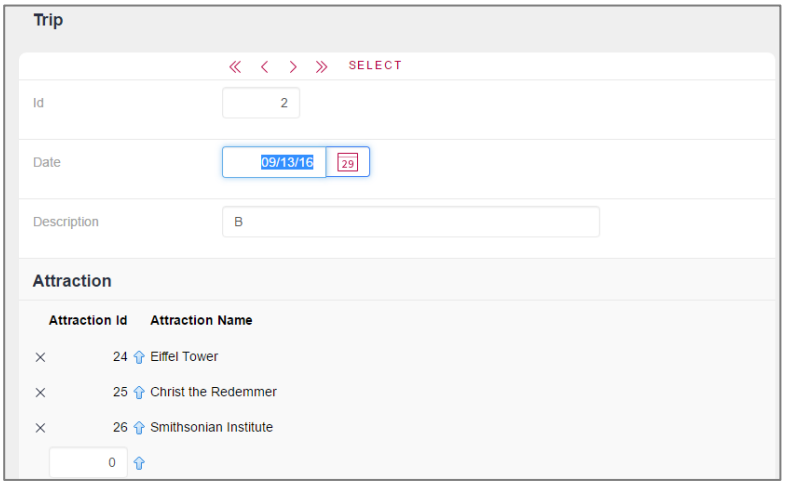
Aquí dentro programaremos lo que queremos que se ejecute cada vez que se esté posicionado en un registro de la tabla Attraction, antes de cargar la línea en el grid. En nuestro caso…



Ejecutemos… Habrá que reorganizar la base de datos para agregar las tablas correspondientes a la nueva transacción Trip.

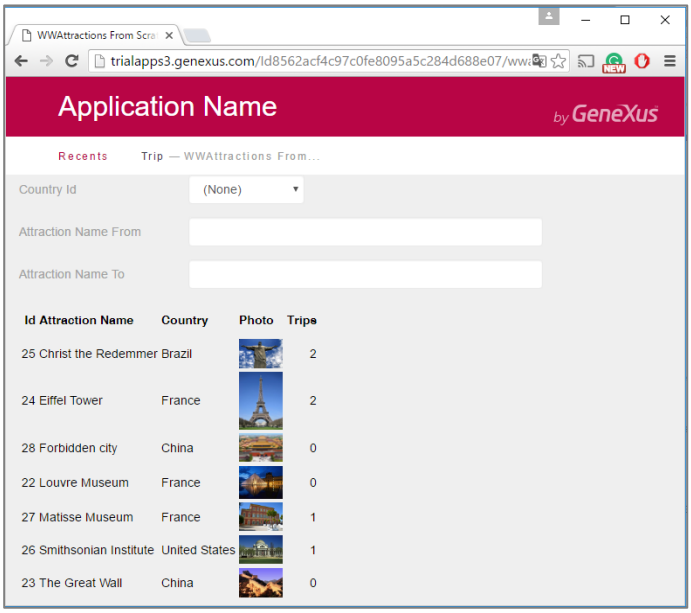
Aquí mostramos la ejecución con un par de trips ya ingresados:





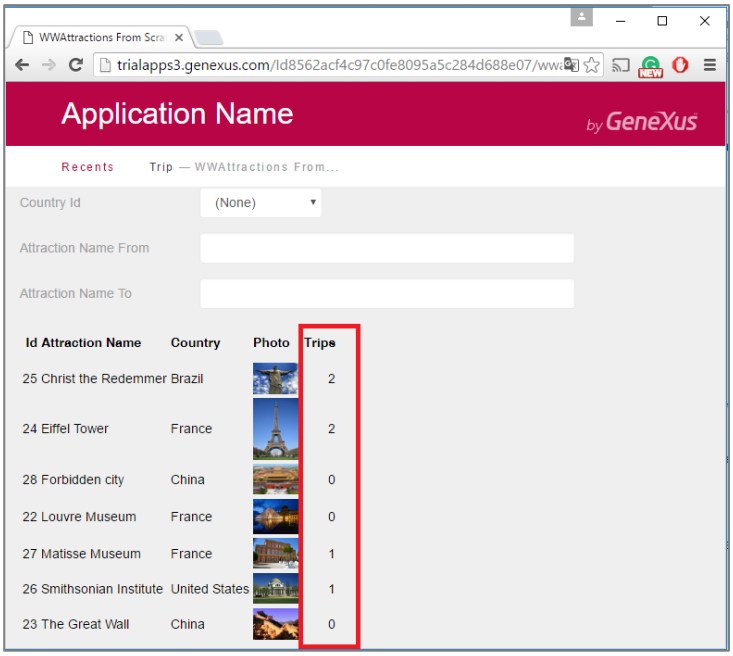
Vemos que la torre Eiffel está en un par de Trips, el cristo redentor también, Smithsonian Institute en uno, y el museo Matisse también en uno.

Ahora ejecutemos nuestro web panel.

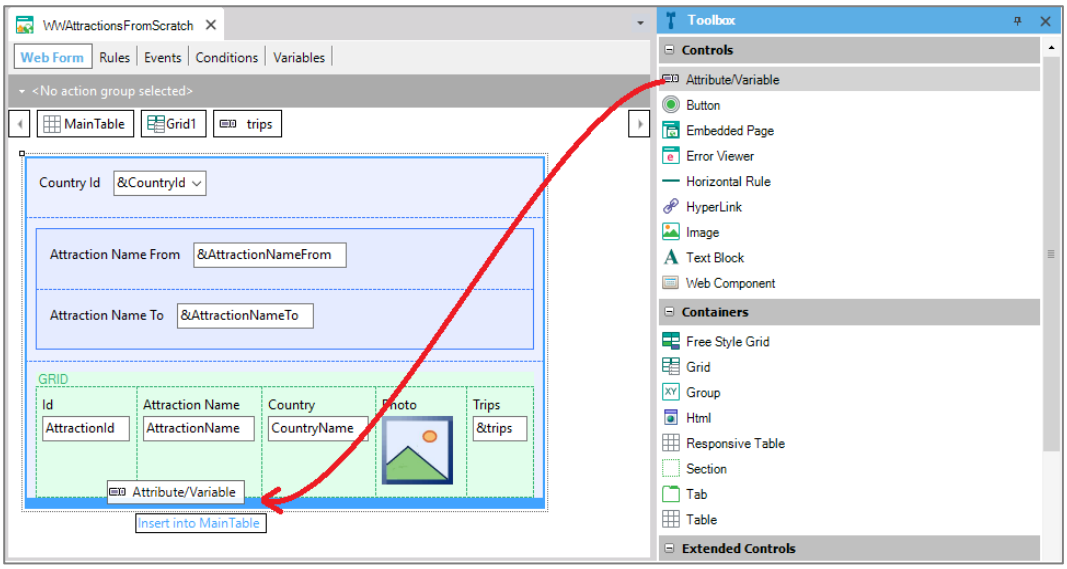


Y vemos que está mostrando correctamente lo que queríamos.

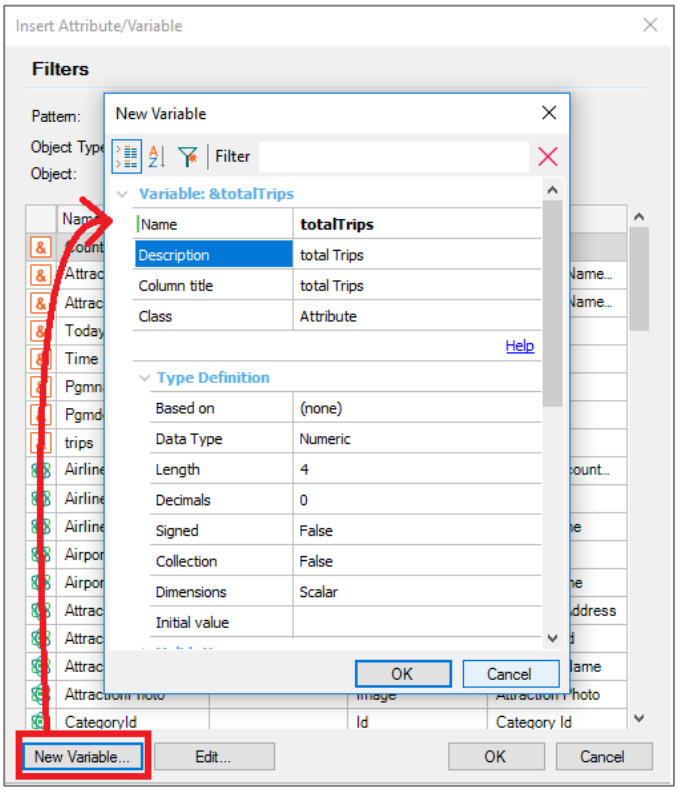
Ahora queremos agregar la suma de excursiones en las que las atracciones que se muestren en el grid en cada oportunidad estén incluidas. Es decir, la suma de los valores de esta columna:



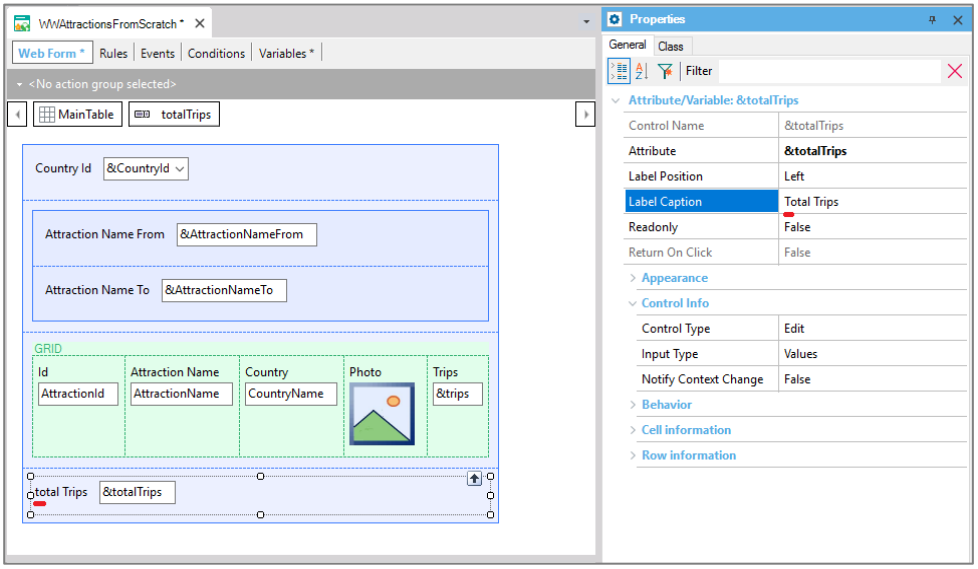
Para ello, agregamos una variable fuera del grid:



Llamémosle totalTrips:



Cambiémosle la propiedad **Label Caption** para que su etiqueta aparezca en mayúscula, ya que el nombre lo pusimos en minúscula:

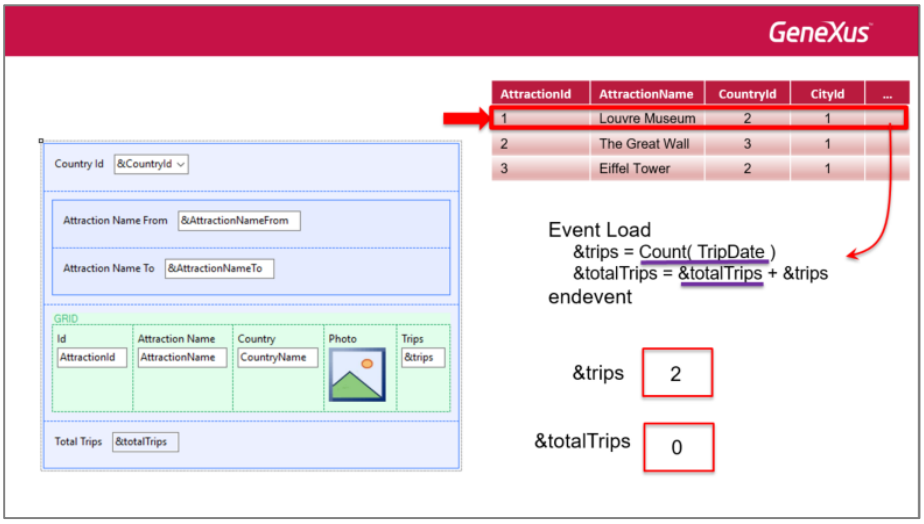


Una forma eficiente de calcular el valor que deberá mostrar la variable es… cada vez que se vaya a cargar una línea en el grid, sumarle el valor de la variable &Trips de esa línea, al valor calculado hasta el momento en &totalTrips.

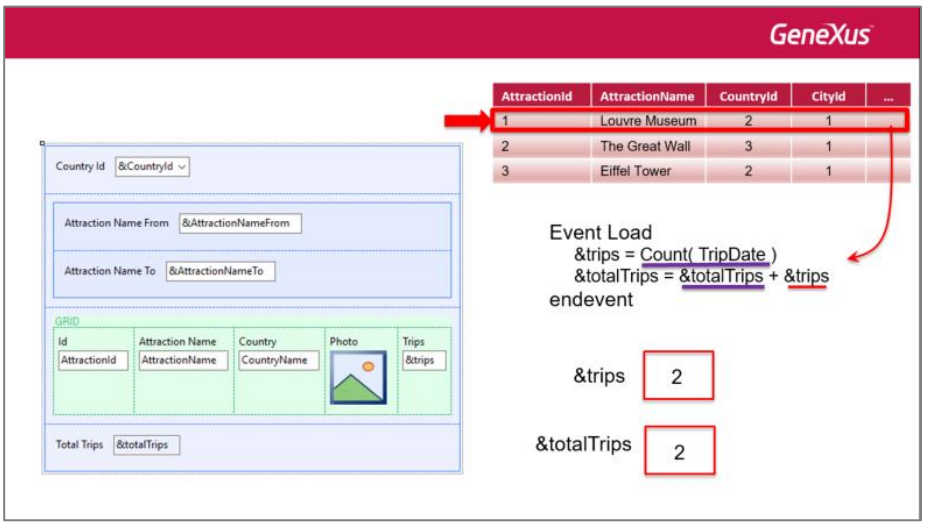
Es decir, en el evento Load, luego de calcular el valor de &trips, a &totalTrips asignarle el valor que contiene hasta el momento más el valor de la variable &trips:

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

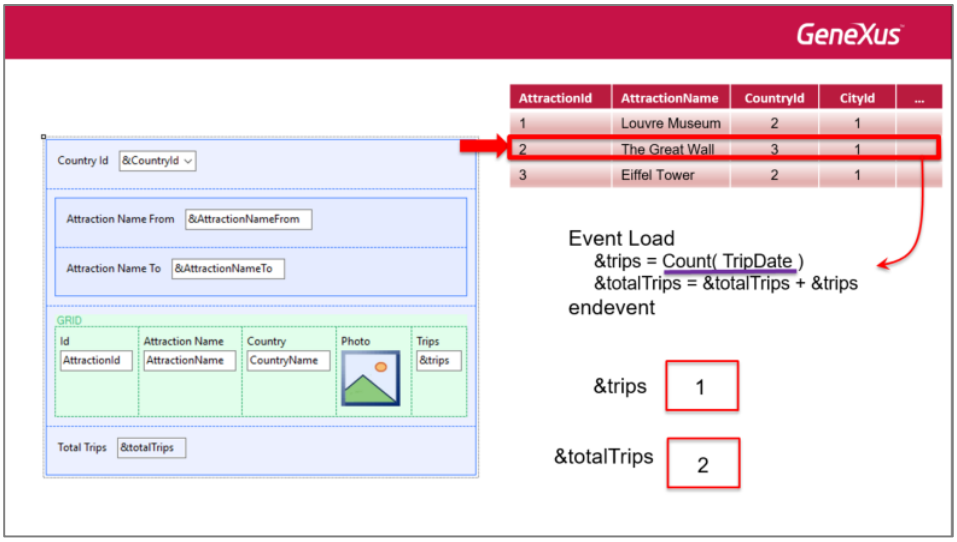
Para la primera línea a ser cargada en el grid, se ejecutará por primera vez el evento Load, se calcula &Trips y cuando se va a calcular &totalTrips, el valor actual estará en cero,



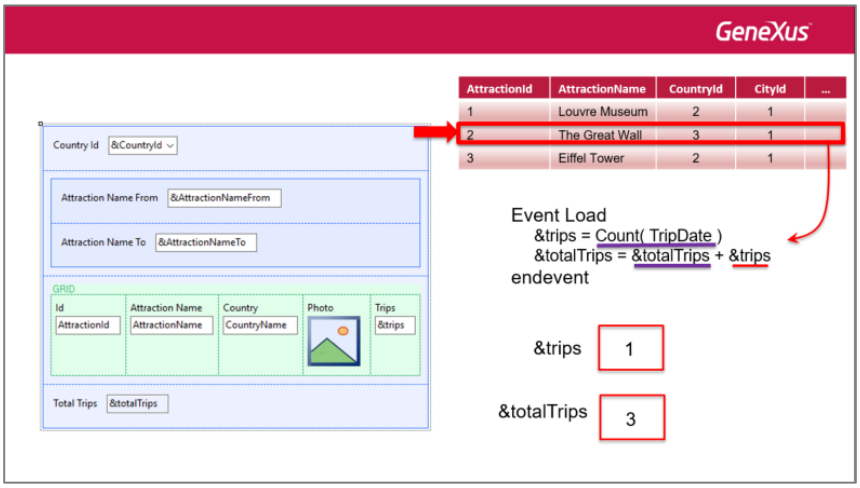
así que en ese caso &totalTrips asumirá el valor de &Trips.



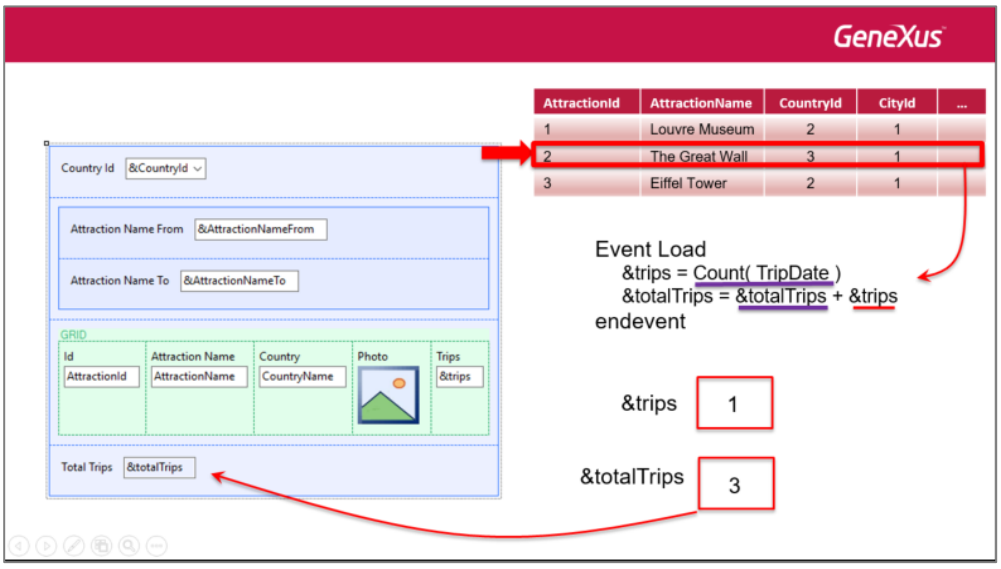
Para la segunda línea a ser cargada, se calcula el valor de &trips y &totalTrips contrendrá el valor anterior



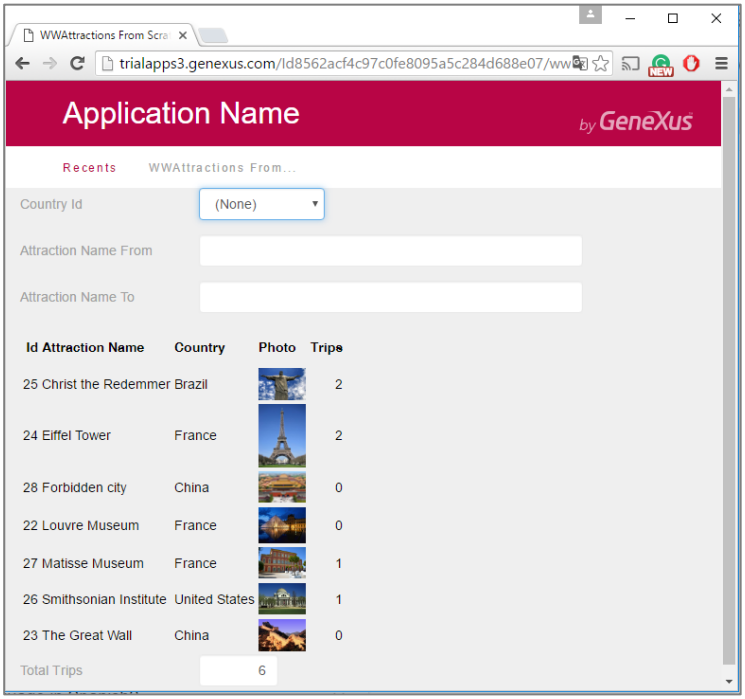
al que se le sumará el de la variable &trips para esta segunda línea, y así sucesivamente.



Luego de que se cargue la última línea, la variable &totalTrips tendrá el valor deseado.

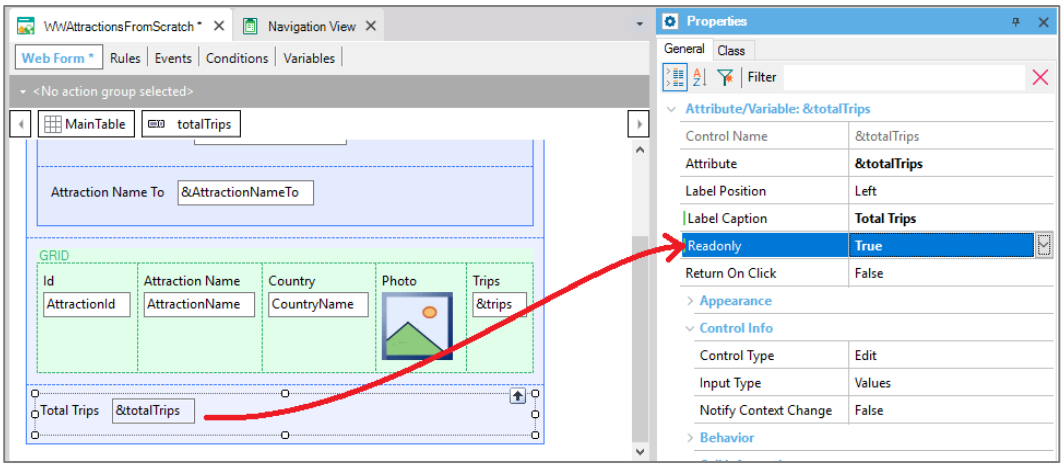


Ejecutemos para probar.



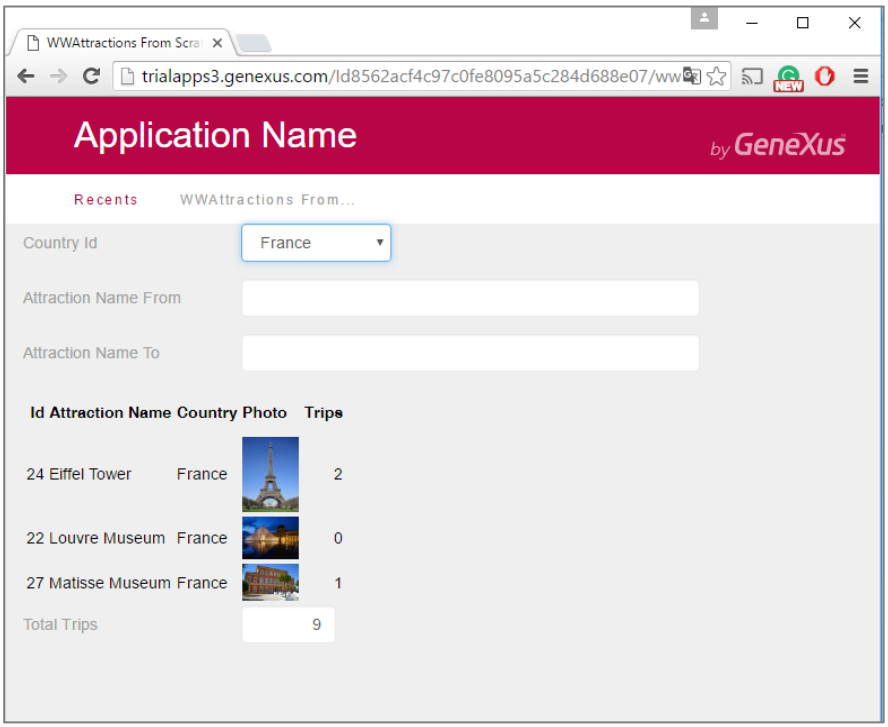
Vemos dos cosas: primero, que está sumando correctamente; segundo: que por tratarse de una variable, es de entrada, por lo que el usuario puede modificarle el valor, lo que no tiene sentido. Así que primero que nada, configurémosla como Readonly.

Para ello nos posicionamos sobre la variable en el form y entre sus propiedades, modificamos la ReadOnly pasándola a True:



Puede llamarnos la atención, también, que &tips también es una variable, está mostrándose como readonly, y sin embargo no hicimos nada para ello. Las variables en los grids, cuando no se han programado eventos a nivel de las líneas, ni se los recorre por código, serán readonly. Volveremos a esto luego.

Pero además, veamos qué pasa si filtramos, por ejemplo, por Francia.



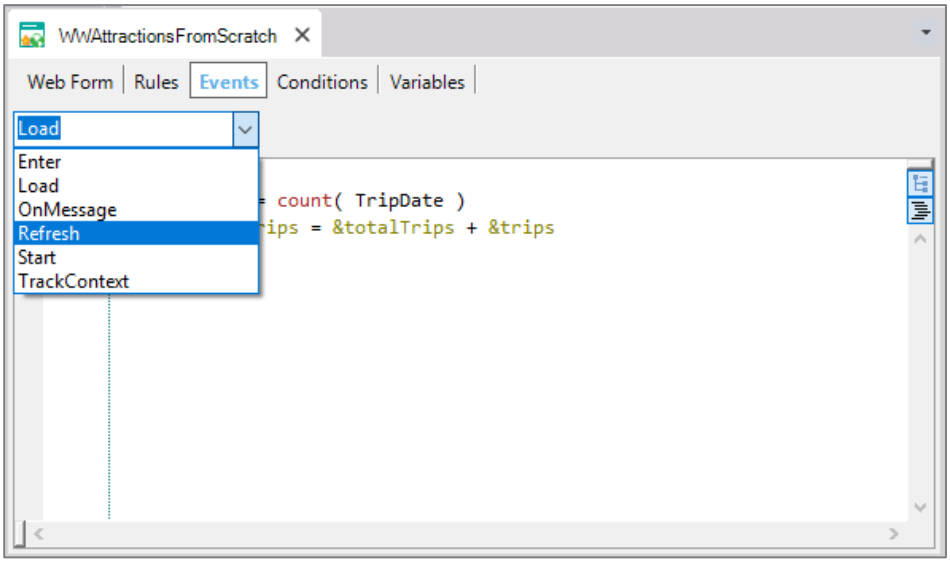
En vez de mostrarnos un total de 3, nos está mostrando 9, que es la suma del valor que nos mostraba antes, 6, más los 3 que ahora debería estar mostrando. ¿Por qué sucedió esto?

Al modificar una de las variables de filtro, el web panel volvió a cargar el grid. Es decir, volvió a consultar la tabla Attraction de la base de datos, y volvió a ejecutar el evento **Load** para cada registro que cumpliera los filtros. El problema es que la variable &totalTrips tendría que haberse reseteado, es decir, vuelto a cero, antes de lanzar la carga del grid.

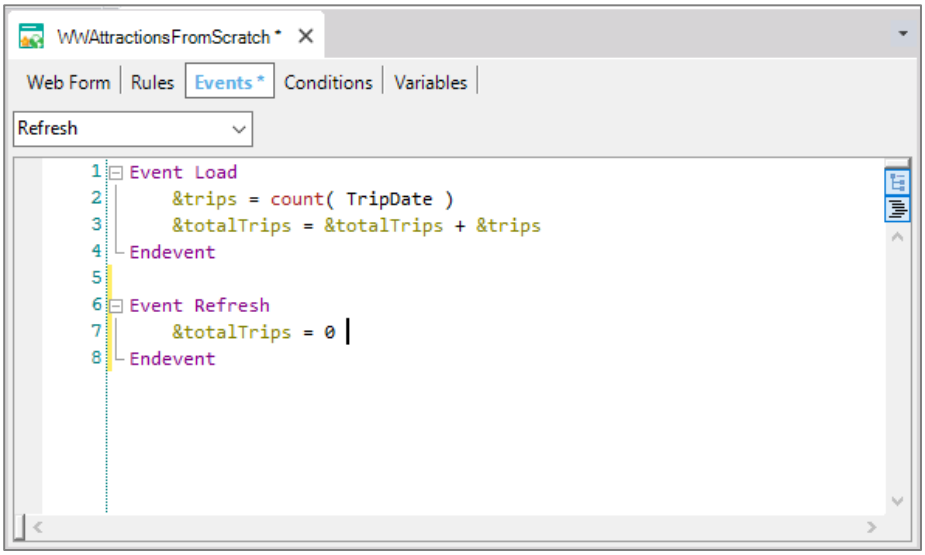
¿Dónde hacemos esto? En el **evento Refresh**.

Este evento del sistema se produce siempre que se va a cargar el web panel, inmediatamente antes de ir a la base de datos a buscar la información para cargar en el grid. Es decir, inmediatamente antes de ejecutarse los eventos Load por cada línea a ser cargada.

Así, vamos a la solapa de eventos, y elegimos en el combo el Refresh…

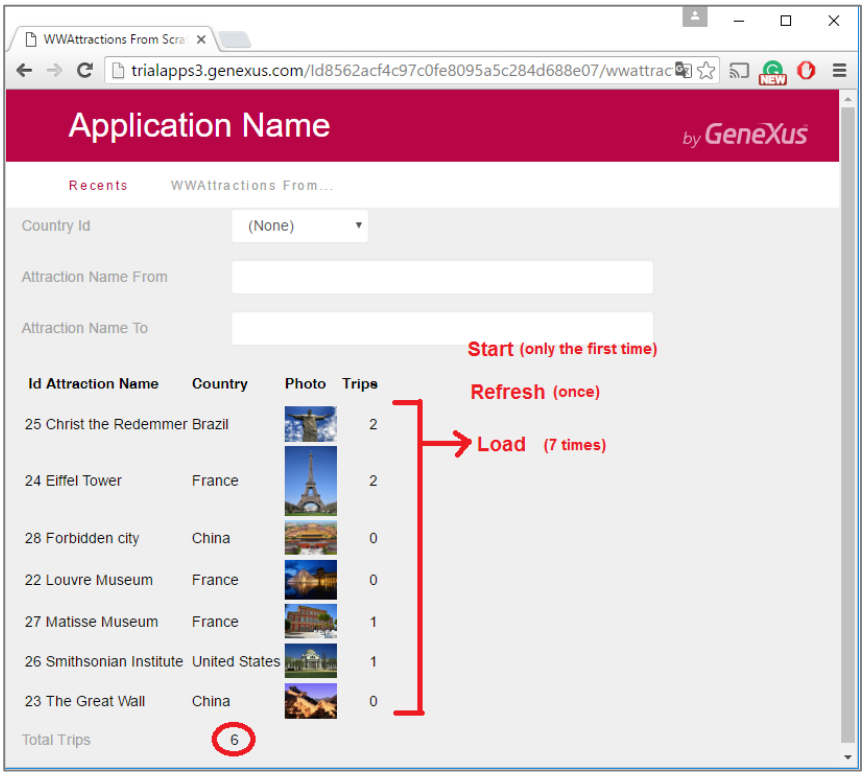


Ese es el momento para poner en cero la variable



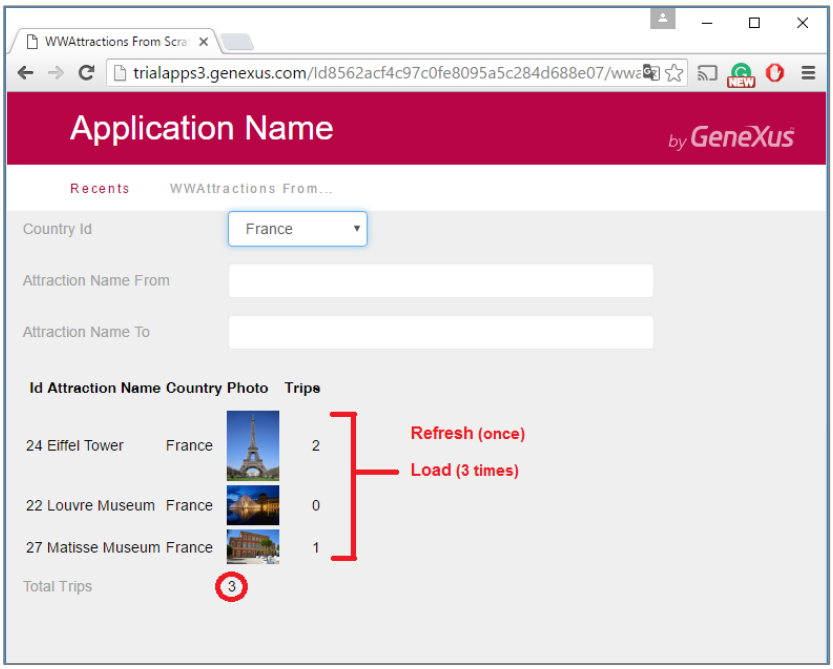
Observemos que el orden en que los eventos quedan escritos no tiene la menor importancia. Aquí solamente se indica el código que se ejecutará al producirse cada uno de ellos.
Presionemos F5.

Podemos ver que ahora el total de tips aparece Readonly. Al ejecutar este web panel por primera vez, se dispararon tres eventos en forma consecutiva: el **evento Start**, que se ejecuta sólo al abrirse el web panel, la primera vez, el **evento Refresh**, que puso la variable en cero, y el **evento Load**, tantas veces como líneas se fueran a cargar en el grid. En este caso, fueron 7.

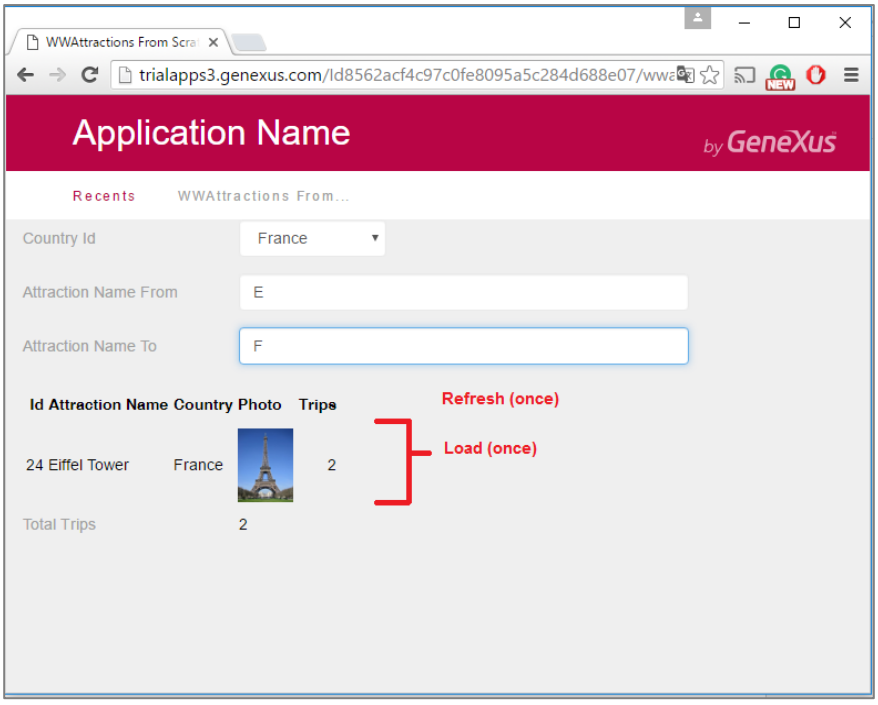


Ahora, si elegimos filtrar por un país, por ejemplo, Francia, vemos que se está calculando bien el número de trips.

Al cambiar el valor de una variable que tiene efectos sobre las condiciones que deben cumplir los registros para cargarse en el grid, se vuelve a disparar el evento **Refresh** (y por tanto la variable &totalTrips se vuelve a poner en cero) y la ida a la base de datos para volver a filtrar y cargar los registros en el grid. Por tanto el **Load** se vuelve a disparar por cada atracción de Francia a ser cargada.

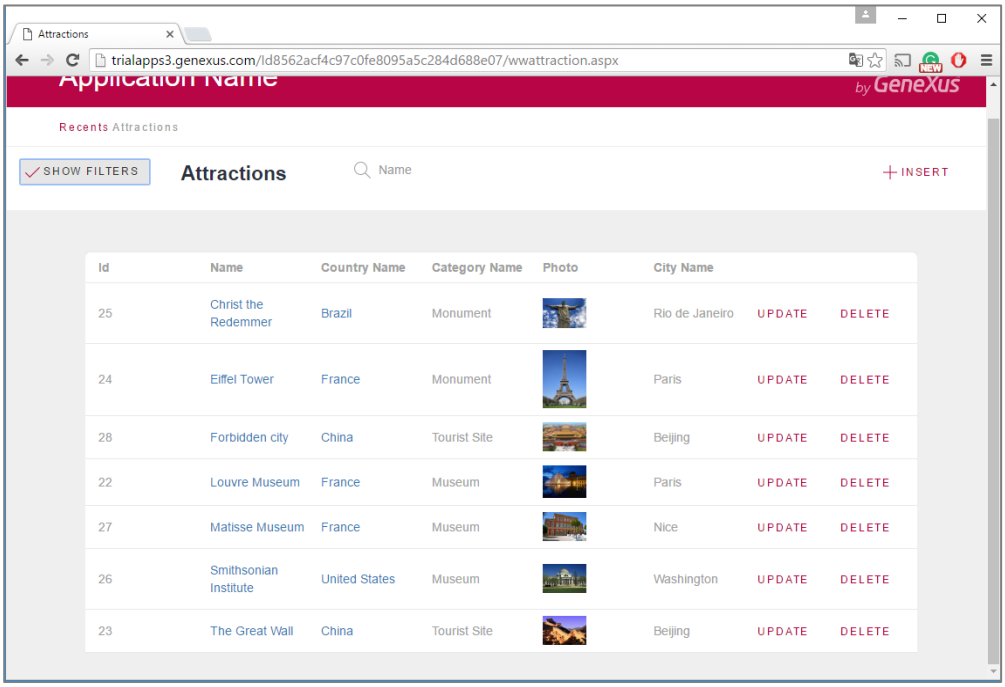


Si ahora elegimos ver las atracciones entre la E y la F, también se vuelve a refrescar la pantalla. Se vuelve a poner en cero la variable &totalTrips en el Refresh, y se vuelve a ir a recorrer la tabla base del grid y su extendida, esta vez recuperándose un solo registro, por lo que el Load se ejecutará una única vez.

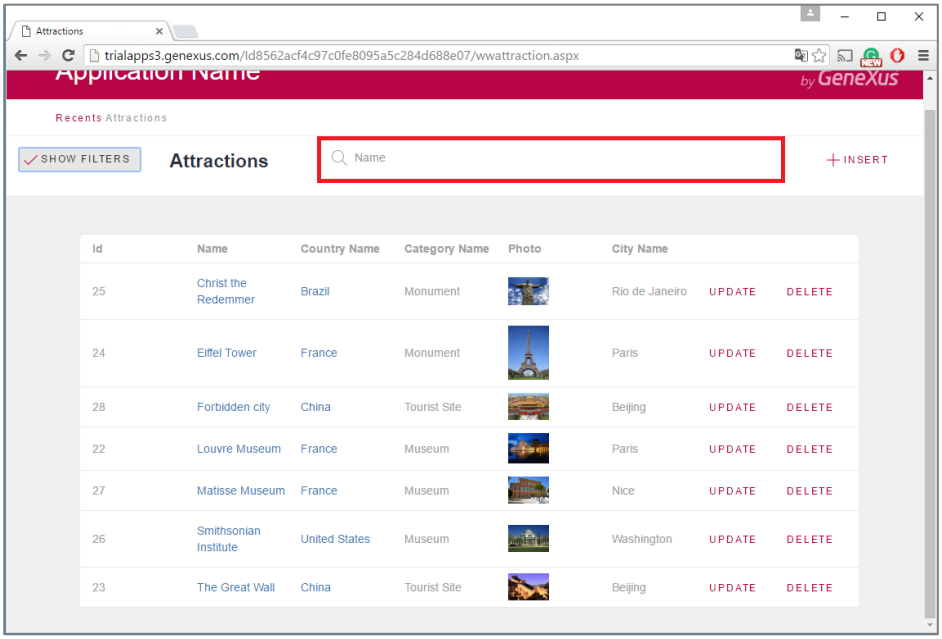


Ahora bien, si observamos el web panel que tenemos implementado hasta el momento, podremos apreciar que se va pareciendo al objeto que había creado el Pattern Work with aplicado a la transacción Attraction.

Este objeto era, evidentemente, un web panel.



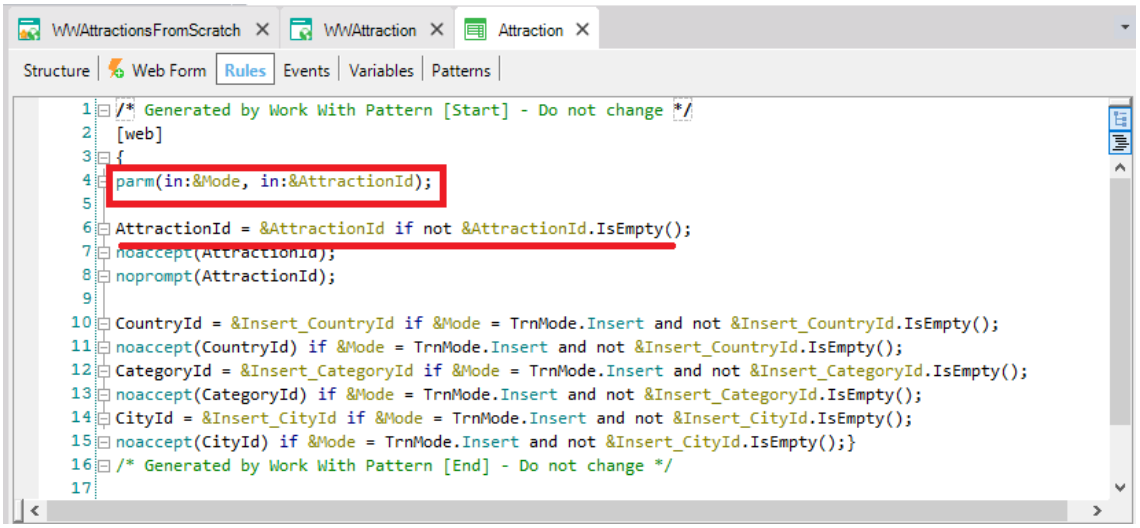
Aquí se filtraba por una única variable:



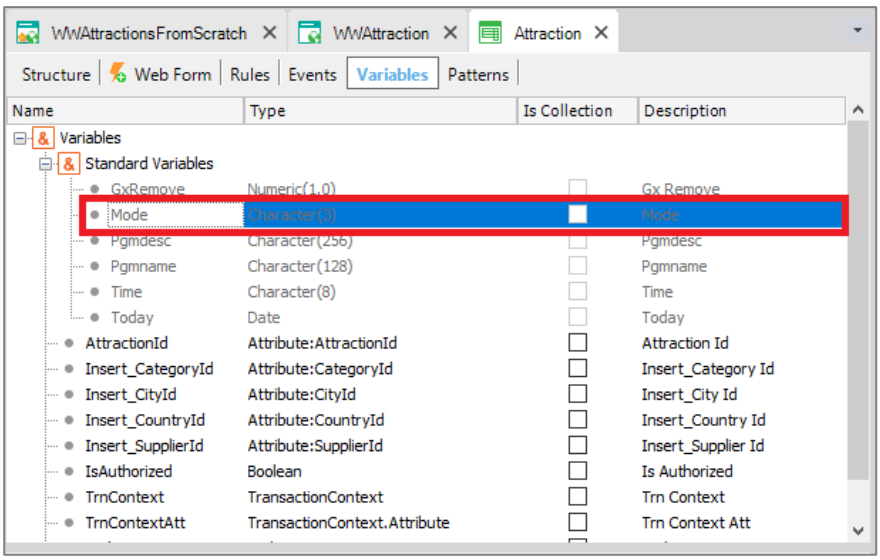
Pero lo más interesante es que además de permitir filtrar los datos del grid, el Work With ofrece ejecutar acciones sobre los datos. Por ejemplo, poder actualizar los datos de una atracción o eliminar la atracción así como insertar una nueva.

Para ello el pattern insertó dos controles a nivel de las líneas del grid, y uno fuera. En cualquiera de los tres casos, la acción asociada a cada control consiste en llamar a la transacción Attraction, enviándole como parámetro el **modo** en que debe abrirse la transacción, esto es, si se la llama para actualizar, eliminar o insertar. En los primeros dos, Update o Delete como corresponderán a eventos de una línea se contará con el id de la atracción de la línea que se le enviará como segundo parámetro a la transacción, de modo de actualizar los datos de **esa** atracción, o eliminar **esa** atracción. En el caso del Insert control fuera del grid, se le enviará 0 como segundo parámetro, ya que las atracciones en Insert se autonumeran.

Es por ello que el pattern modificó la transacción Attraction, agregándole, entre otras cosas, la regla Parm:

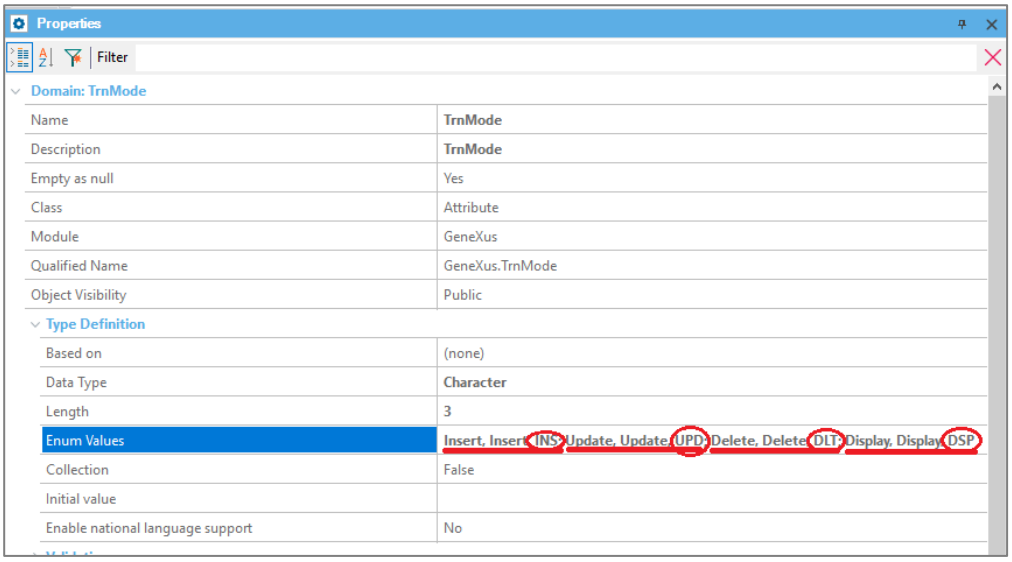


que como vemos, recibe dos variables: la variable &Mode es una variable estándar en transacciones, Character de 3:



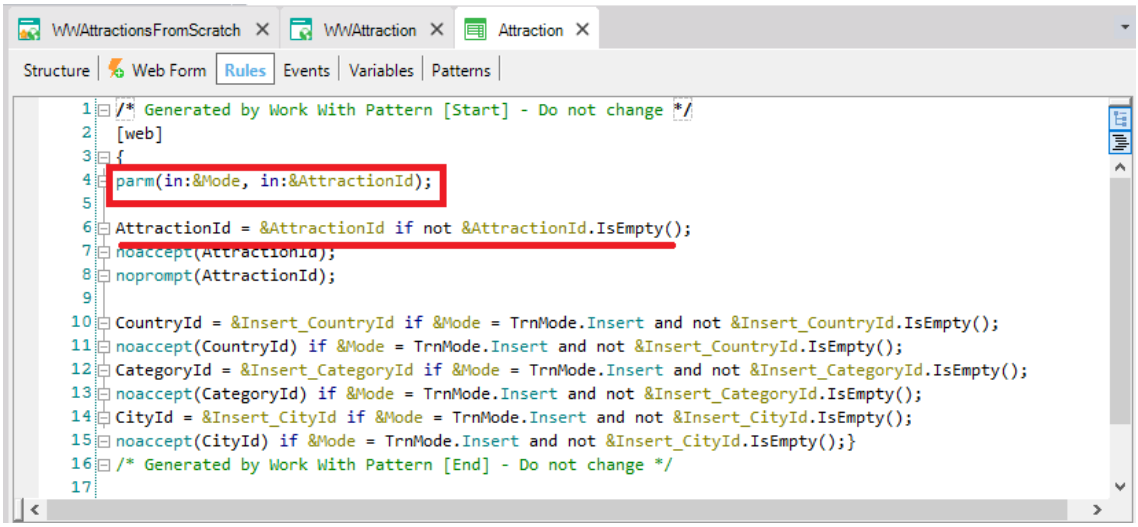
que acepta uno de cuatro valores, especificados en el dominio enumerado TrnMode:

- Insert, Ins
- Update, U Pe De
- Delete, De eLe Te
- Display, De eSe Pe



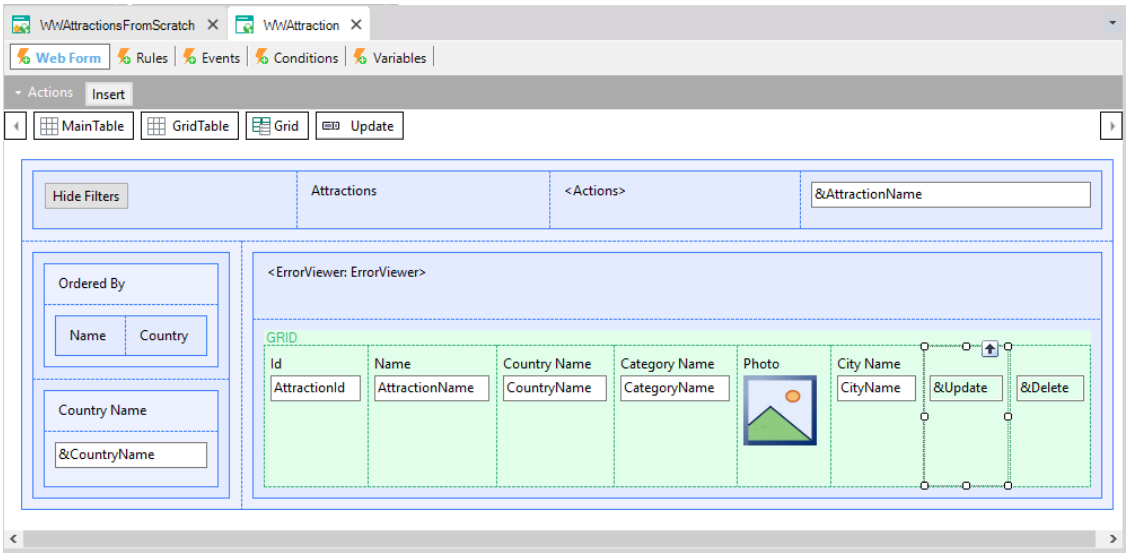
Recibiendo uno de estos cuatro valores, la transacción sabrá en qué modo se le pide que se abra.

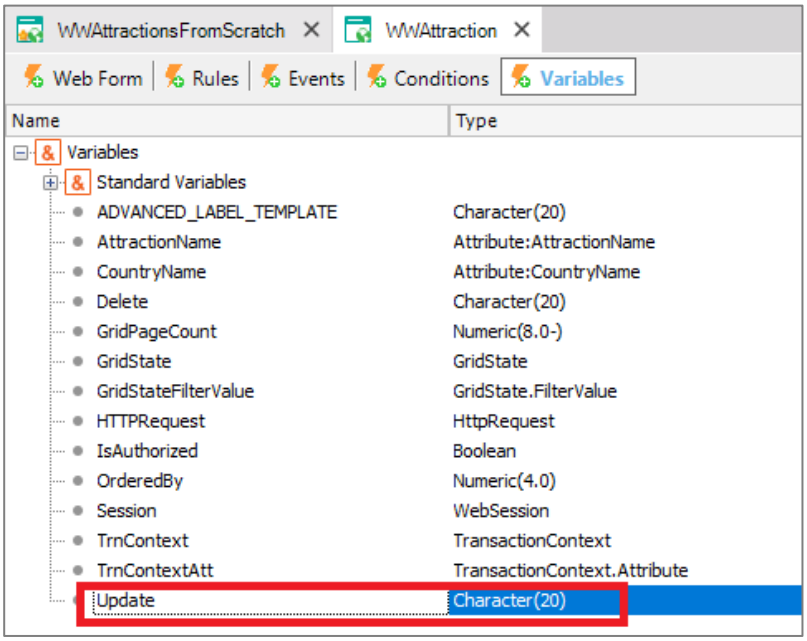
Y por otro lado, recibirá como segundo parámetro el id de atracción, en la variable &AttractionId, para cuando se quiera hacer update, delete o display.



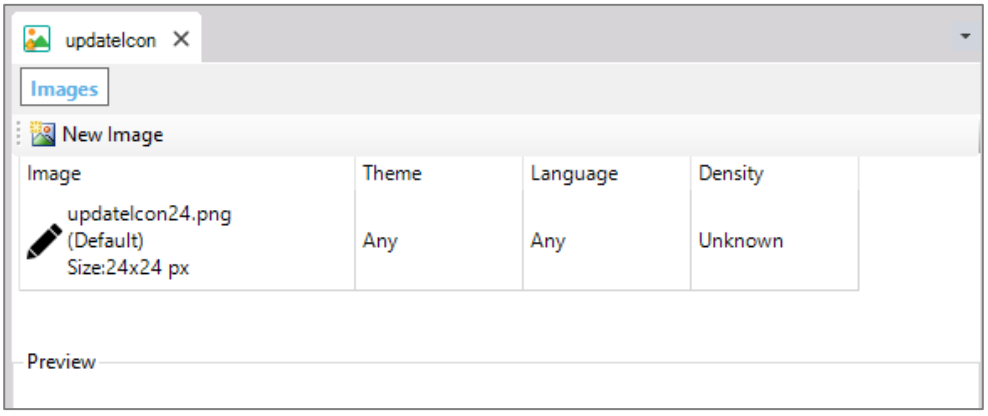
En nuestro web panel implementaremos una de estas acciones sobre las atracciones. Por ejemplo, la de Update. La intención es mostrar un ejemplo de acciones sobre los datos.

Tendremos que insertar un control en el grid. En el caso del pattern se inserta una variable character a la que se ha llamado update,



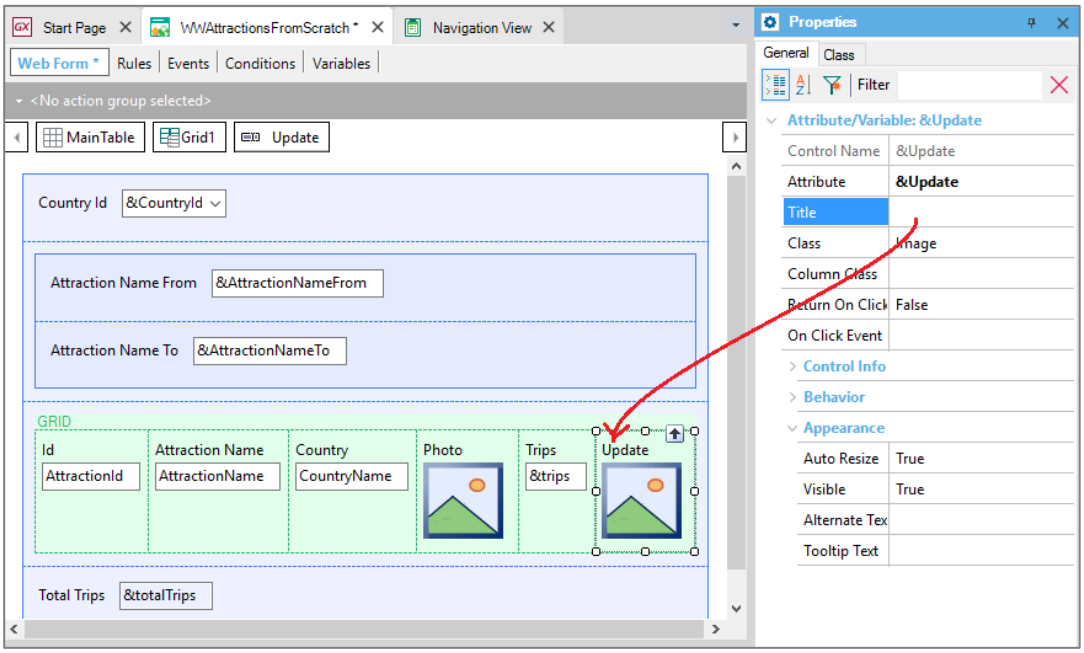


Y a la que se le asigna el texto “UPDATE” que vemos en ejecución. Pero nosotros elegiremos insertar una imagen, que antes que nada debemos insertar en la KB.



La llamamos updateIcon.

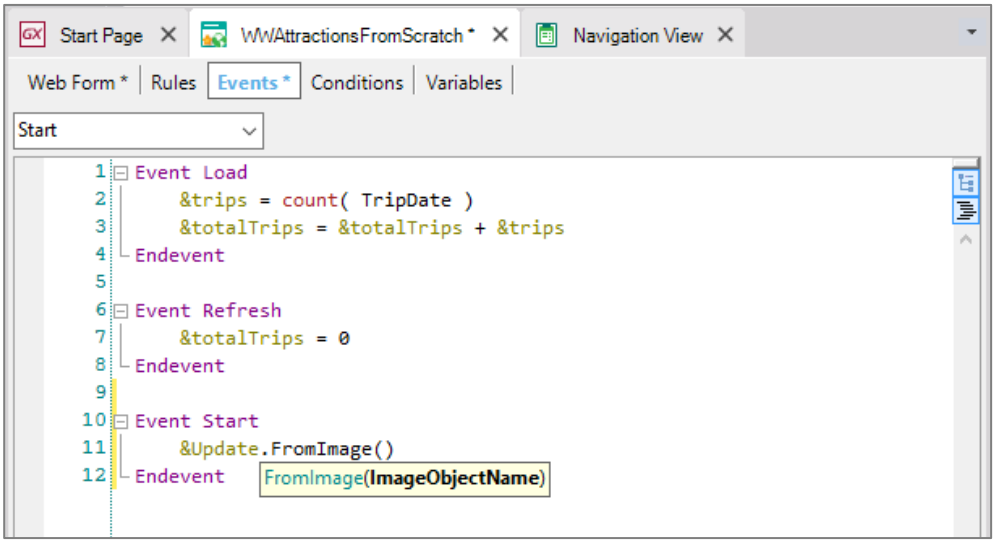
Ahora vamos al web panel y arrastramos de la toolbox el control Attribute/Variable a la última columna del grid, definimos la nueva variable como &Update, pero no de tipo character, sino Image:



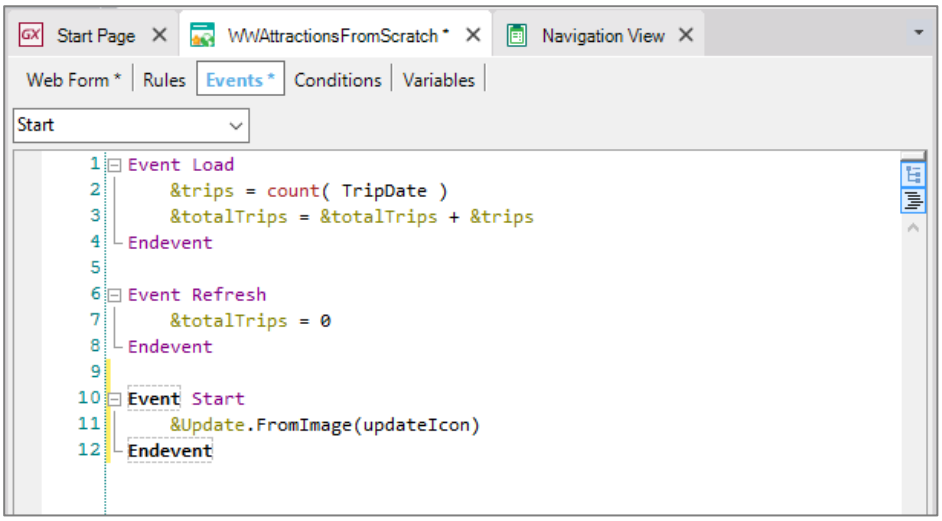
Le quitamos el título para que no figure como título de columna, y nos resta cargar esa variable con la imagen que acabamos de insertar en la KB. ¿Dónde lo hacemos?

Si la imagen variara por línea del grid, lo haríamos en el evento Load, pero la imagen será la misma para cada línea, y no variará nunca, así que una buena opción es hacerlo en el **evento Start**, que se ejecutará una única vez, cuando se abra el web panel, y ya no más.

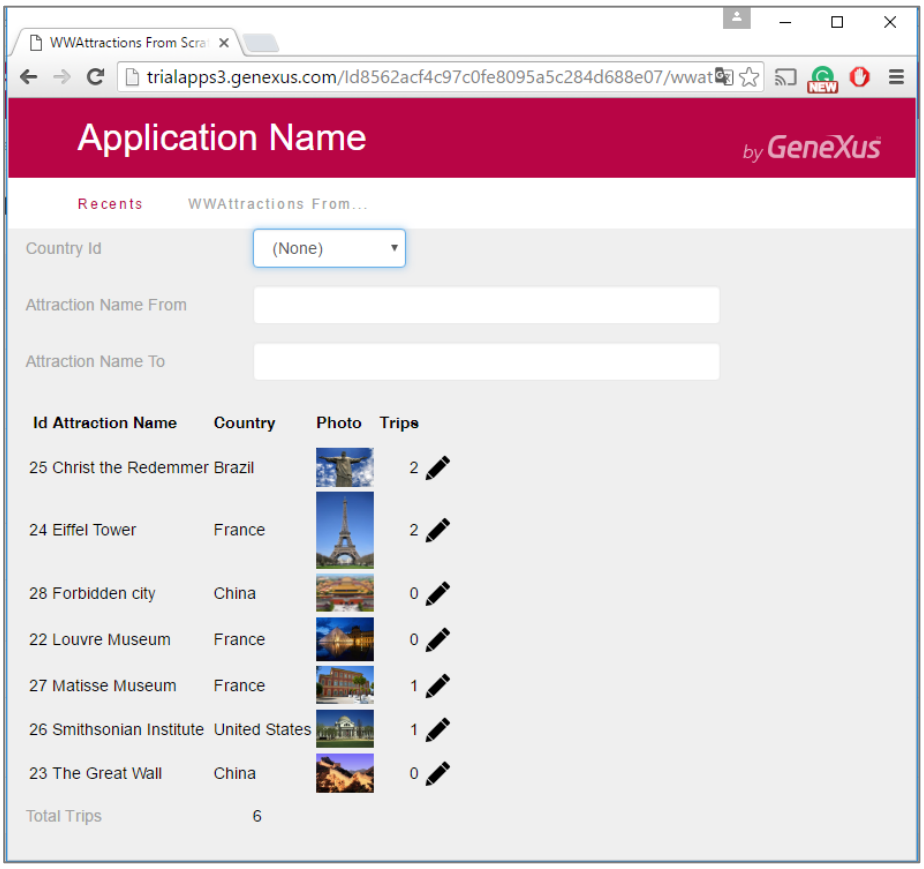
Así que insertamos el evento Start, y allí escribimos:



Y como parámetro del método, escribimos el nombre de la imagen de la KB, esto es, updateIcon.

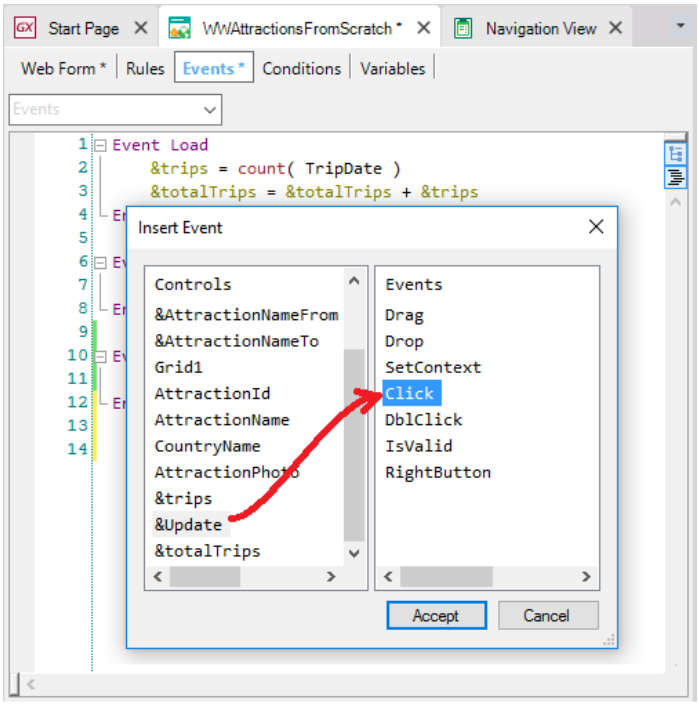


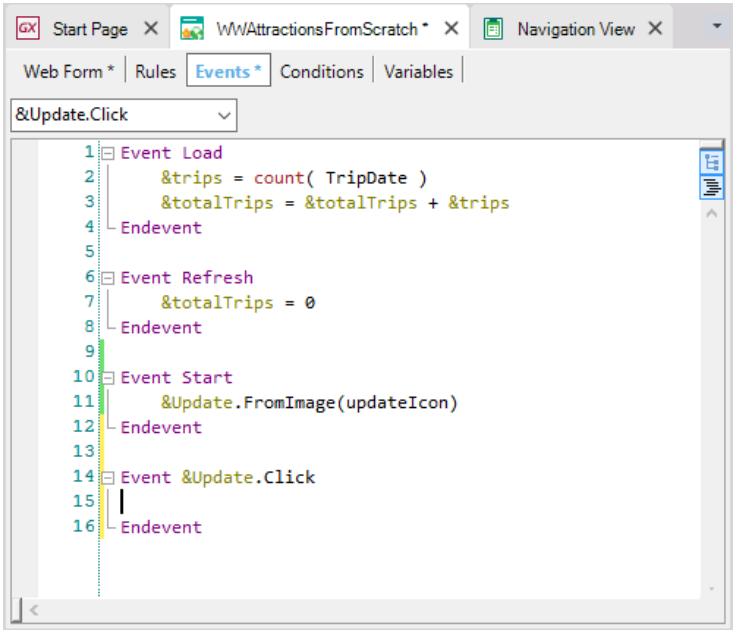
Veámoslo en ejecución.



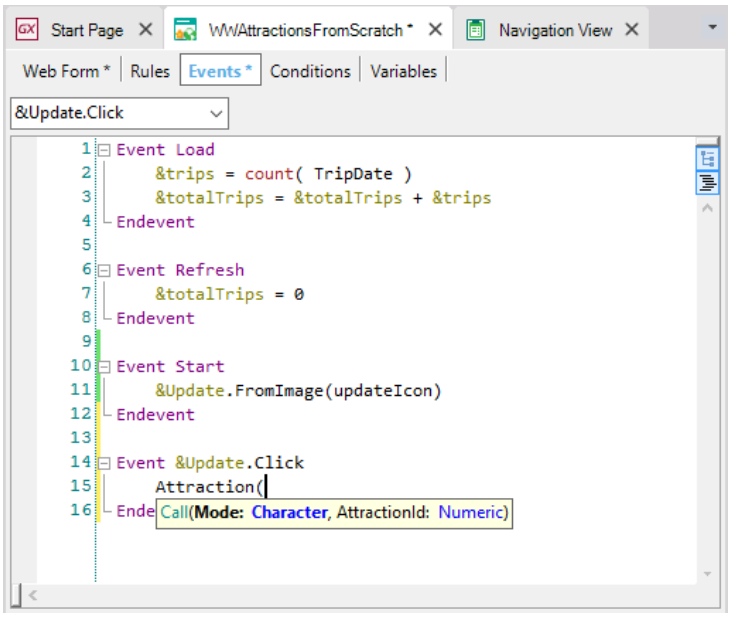
Ahora nos resta asociar un evento a esa imagen, de manera tal de que cuando el usuario haga clic sobre ella, se produzca ese evento y se ejecute su código, en el que invocaremos a la transacción Attraction.

Hay varias alternativas para realizar esto. Una de ellas es ir a la solapa de eventos y con Insert/Event vemos a la izquierda todos los nombres de los controles que hemos insertado en el form. Elegimos el que queremos, la variable &Update, y vemos que a la derecha se muestran los eventos que se le pueden asociar, por ejemplo, el evento click:

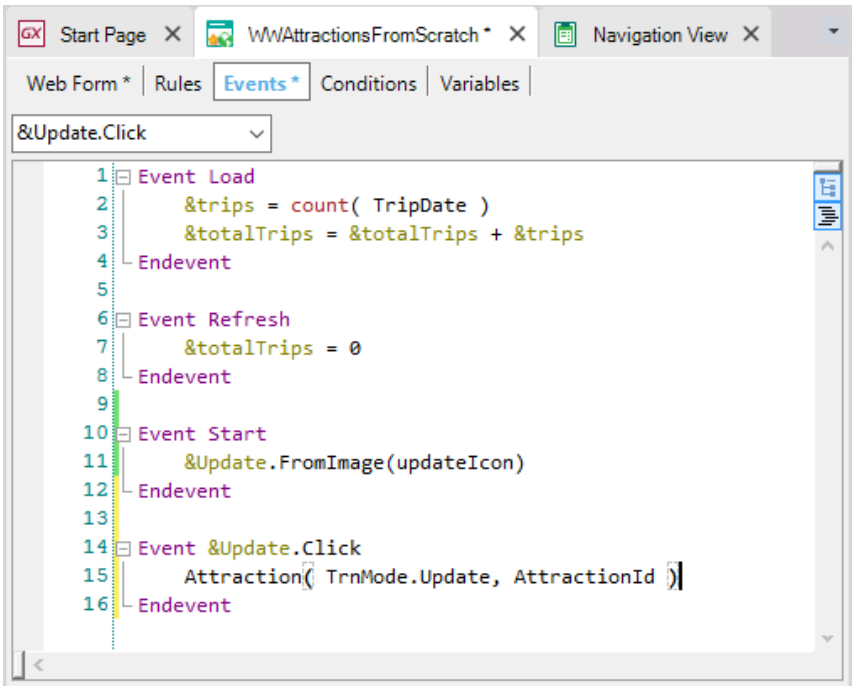




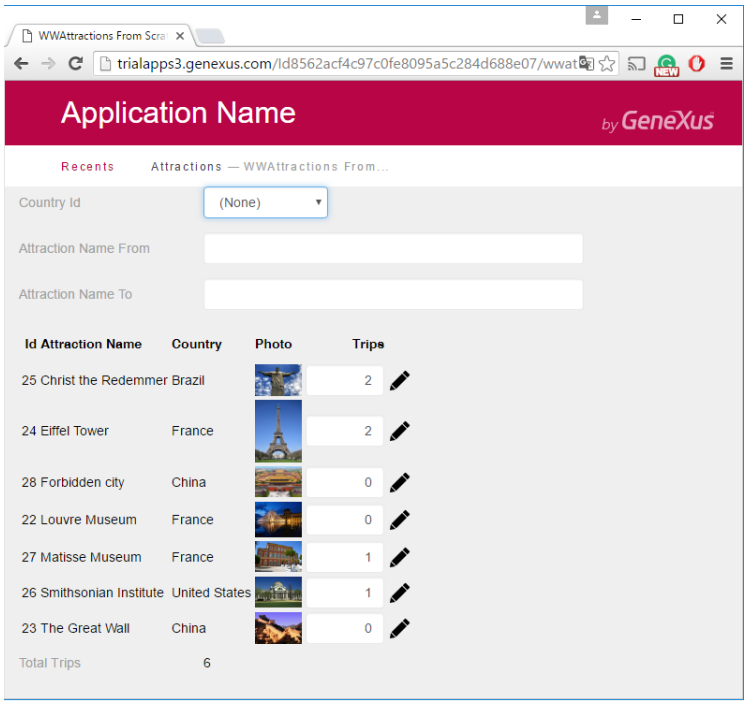
De este modo, cuando el usuario haga clic sobre la imagen para una línea, se ejecutará el código que escribamos dentro de este evento. Lo que queremos hacer, en ese caso, es invocar a la transacción Attraction:



Pasándole el modo Update, es decir, el valor Update del dominio enumeardo TrnMode que vimos antes, y el valor de AttractionId correspondiente a la línea del grid donde se hizo clic:

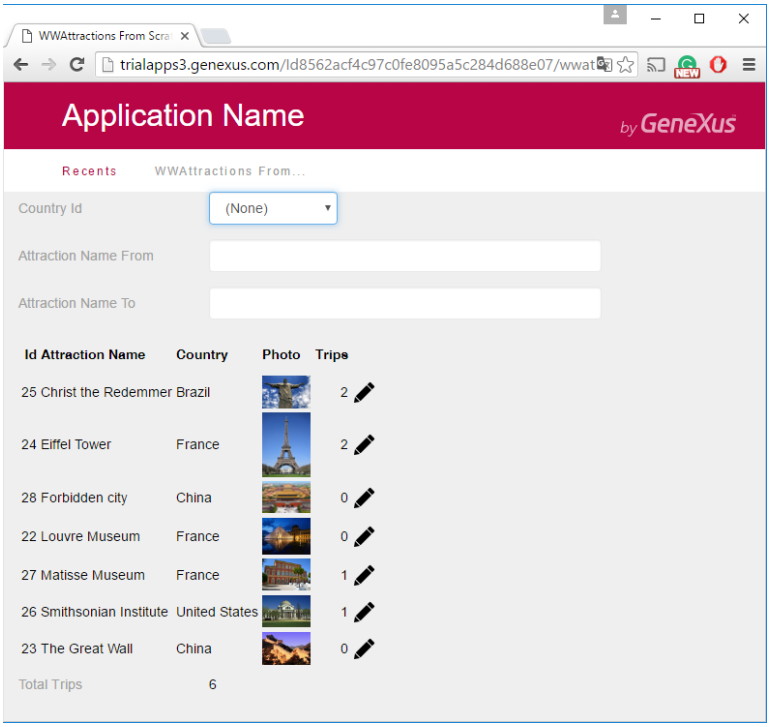


Ejecutemos para probar.

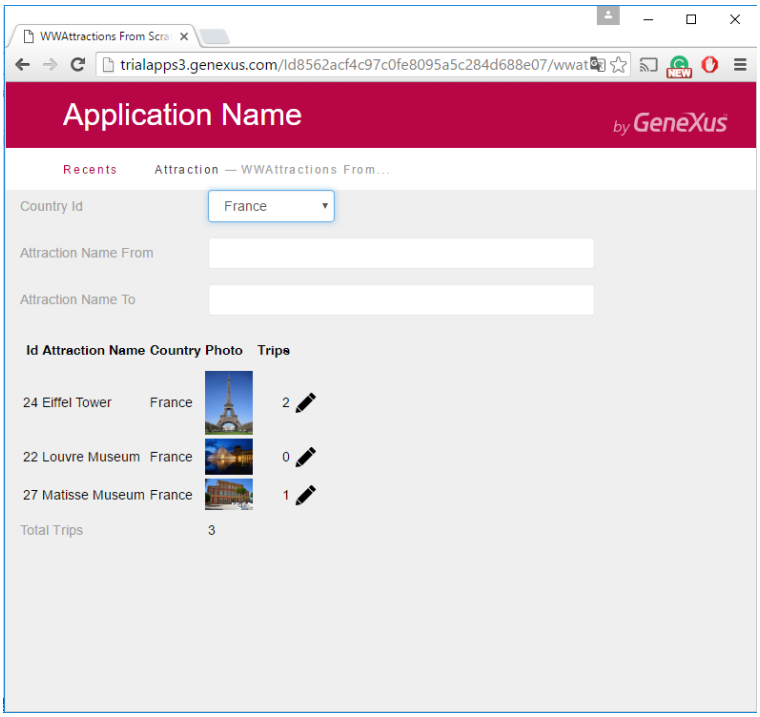


Primero observamos que la columna de la variable &Trips aparece ahora como editable. No la habíamos definido como Readonly explícitamente porque al ejecutar vimos que ya lo estaba. Como dijimos antes, las variables del grid en principio se colocan como readonly, salvo que se defina algún evento a nivel de las líneas, como ha sido nuestro caso, o salvo otras excepciones que ahora no veremos.

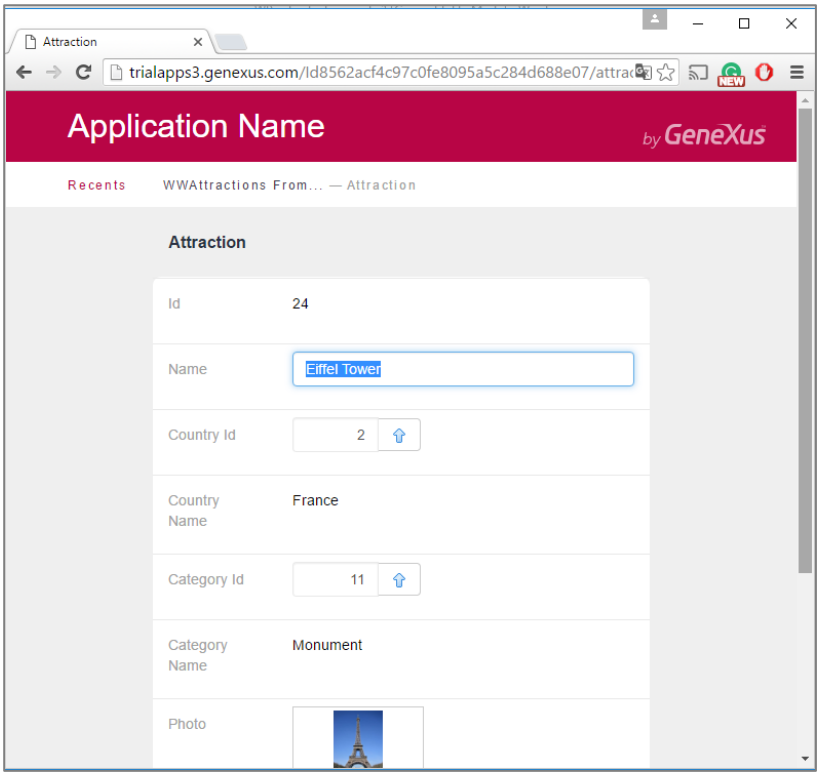
Configurémosla como Readonly . Volvamos a ejecutar.



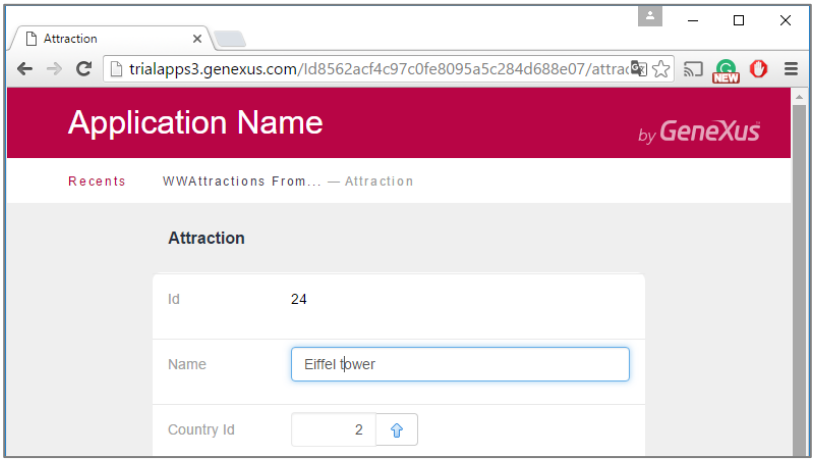
Seleccionemos por ejemplo el país Francia:



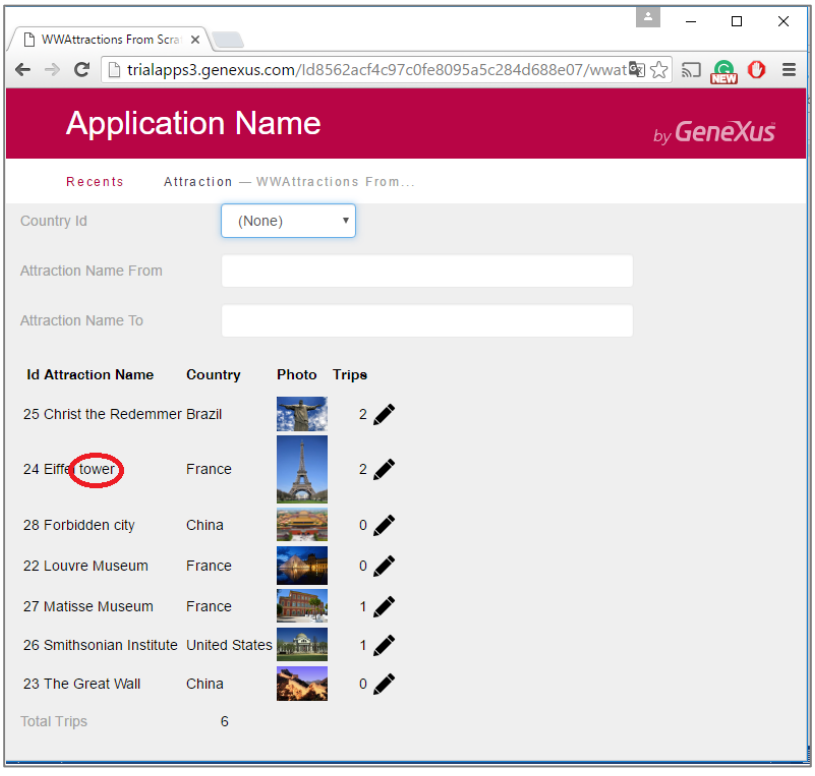
Y ahora hagamos clic sobre la imagen de update para la Torre Eiffel:



Vemos cómo se llama a la transacción en update. Modifiquemos algo… por ejemplo pasemos de mayúscula a minúscula la T de Tower:



Confirmamos… y como el pattern work with, aunque no lo vimos, ha agregado a la transacción un comando Return, retorno, para volver a quien la llamó, es que se vuelve al web panel. Este comando Return es como hacer una invocación al web panel por primera vez, por lo que en éste se ejecutará el evento Start, seguido del Refresh y del Load tantas veces como registros se vayan a cargar:



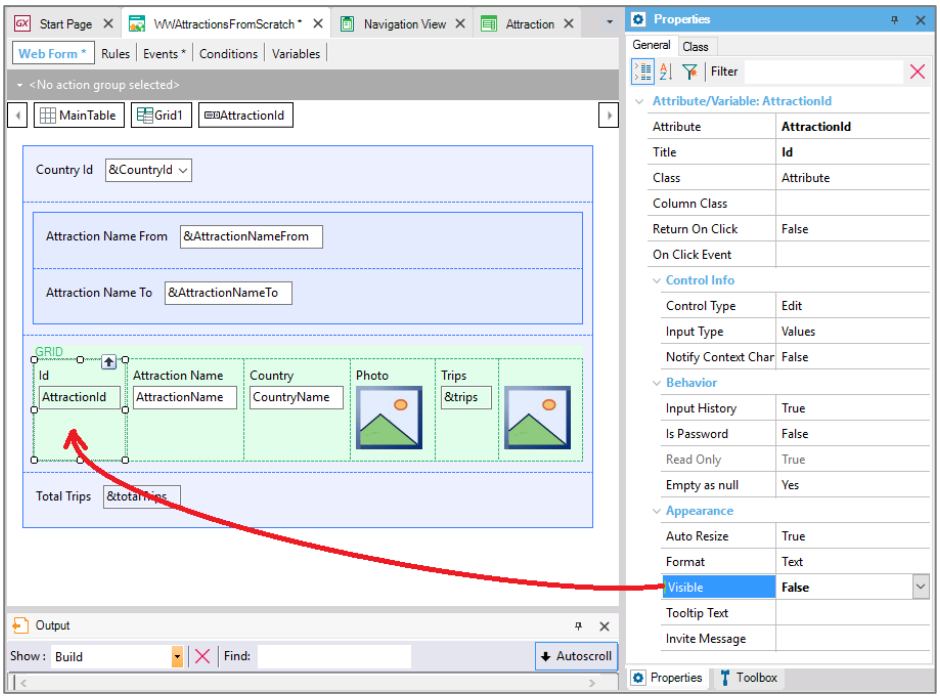
Es por eso que vemos que al volver se cargan todas las atracciones, sin filtros.

¿Qué pasaría si no hubiéramos colocado el atributo AttractionId en el grid? Al hacer clic sobre la imagen para actualizar,¿qué AttractionId se habría enviado como parámetro a la transacción? No tendría ese valor para enviar.

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

Como lo vamos a usar en un evento a nivel de las líneas, que se va a disparar después de que las líneas se hayan cargado, no podemos quitar AttractionId del grid. Es que aquí ya no se está más en la base de datos. El grid almacenó, al cargarse con el Load, todos los valores de sus columnas y nada más. Un evento posterior trabajará únicamente sobre los datos cargados en el grid. Entonces, lo que podemos hacer si no queremos ver esa columna en el grid, es ocultarla. Seguirá estando presente, pero invisible.

Para ello utilizamos la **propiedad Visible con el valor False**:



Ejecutamos

En el siguiente video veremos qué sucede con un evento a nivel de las líneas que modifica el valor de una variable del grid, veremos un web panel sin grid pero con atributos en el form, haremos un resumen conceptual de todo lo visto, veremos web panels sin tabla base, y mencionaremos más casos avanzados de uso de web panels y sus características.