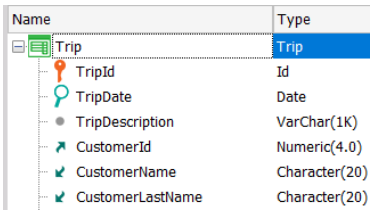
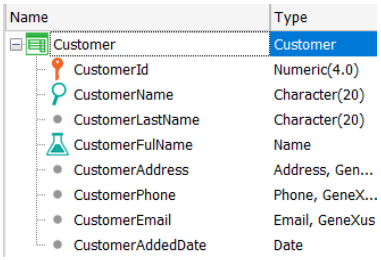
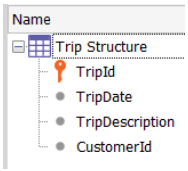
Atributos redundantes y su mantenimiento

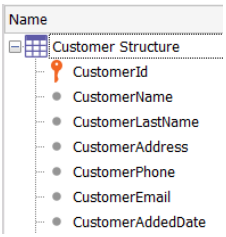
GeneXus™

En este video veremos cómo definir atributos inferidos o fórmulas, que por definición no están almacenados, como redundantes y que pasen a ser atributos de una tabla de la base de datos.









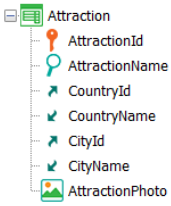
Como sabemos, GeneXus normaliza automáticamente la base de datos en Tercera Forma Normal, lo que implica que los únicos atributos que pueden estar en más de una tabla son los atributos que son clave primaria, cumpliendo funciones de clave foránea.

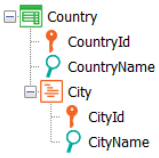
El resto de los atributos, a los que llamamos secundarios, se almacenan en una única tabla, determinados por la clave primaria y en caso de que se agreguen a una transacción distinta, GeneXus se encarga de inferirlos, recuperando por medio de la clave foránea, su valor desde la tabla donde están almacenados.
Además cuando definimos un atributo como fórmula en una transacción, deja de estar almacenado y pasa a ser un atributo virtual.

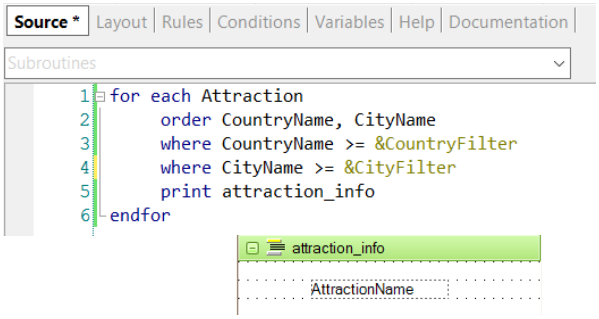
Sin embargo, muchas veces por razones de performance, en algunos casos queremos permitir que un atributo inferido se almacene en la tabla asociada a la transacción donde es inferido, o que un atributo fórmula que debe realizar muchos cálculos y consume mucho tiempo cada vez que se obtiene su valor, se almacene en su tabla asociada para poder obtener el valor más rápidamente.

GeneXus permite que podamos almacenar un atributo que por defecto no está almacenado en una tabla, definiéndolo como redundante.

Referencial redundancy







Cuando redundamos un atributo inferido, se denomina **redundancia referencial**.

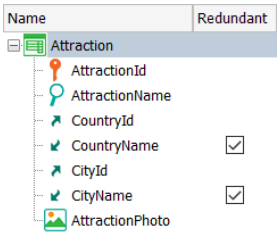
La razón principal por la que surgió esta alternativa de redundar un atributo era mejorar la performance.

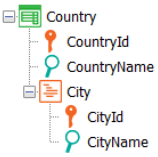
Imaginemos que la tabla de atracciones turísticas tuviera millones de registros, y que quisiéramos recuperar aquellos que correspondan a países cuyo nombre sea alfabéticamente posterior a un valor dado, y cuyo nombre de ciudad sea posterior a otro.

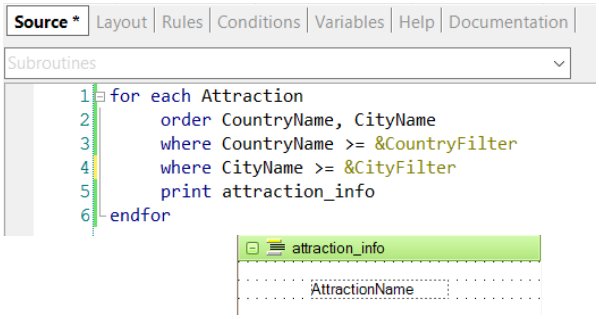
Para optimizar sabemos que nos conviene ordenar por los atributos de los filtros.

Si observamos el listado de navegación, por un lado vemos que la búsqueda está optimizada en cuanto a los filtros de navegación, pero observemos que tendrán que hacerse dos joins dado que CountryName y CityName no están en la tabla Attraction que se recorre. En cambio, si estuvieran en la tabla...

Referencial redundancy





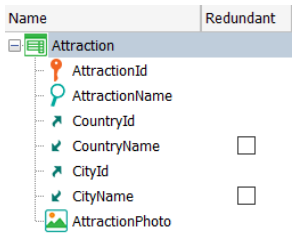


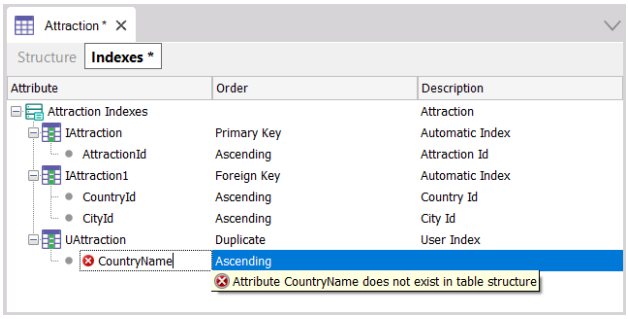
...no habría join que realizar, por lo que la performance mejorará.

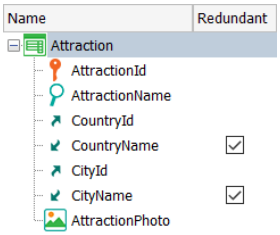
En ambos casos se nos informa que por el hecho de no existir un índice por el orden que definimos podríamos notar problemas de performance. Dependiendo del DBMS que estemos utilizando, su inteligencia y capacidades. Sabemos que hoy en día los DBMSs son mucho más inteligentes que antaño y tienen estrategias para optimizar las búsquedas. Sin embargo en algunos casos para resolver la consulta será preciso crear un índice temporal que se elimina tras la consulta. Y así cada vez que la consulta se ejecute.

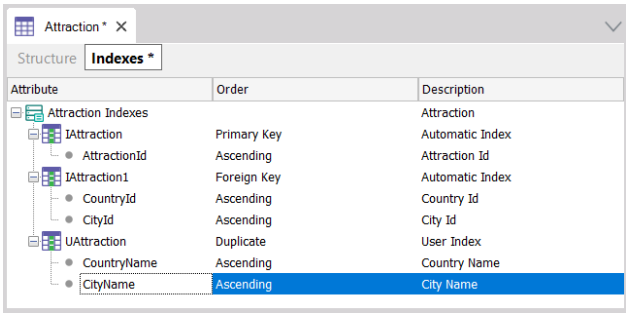
Si fuera el caso, y necesitáramos, justamente para evitar esa creación continua de índice y su posterior eliminación, crear un índice de usuario por esos atributos...

Referencial redundancy

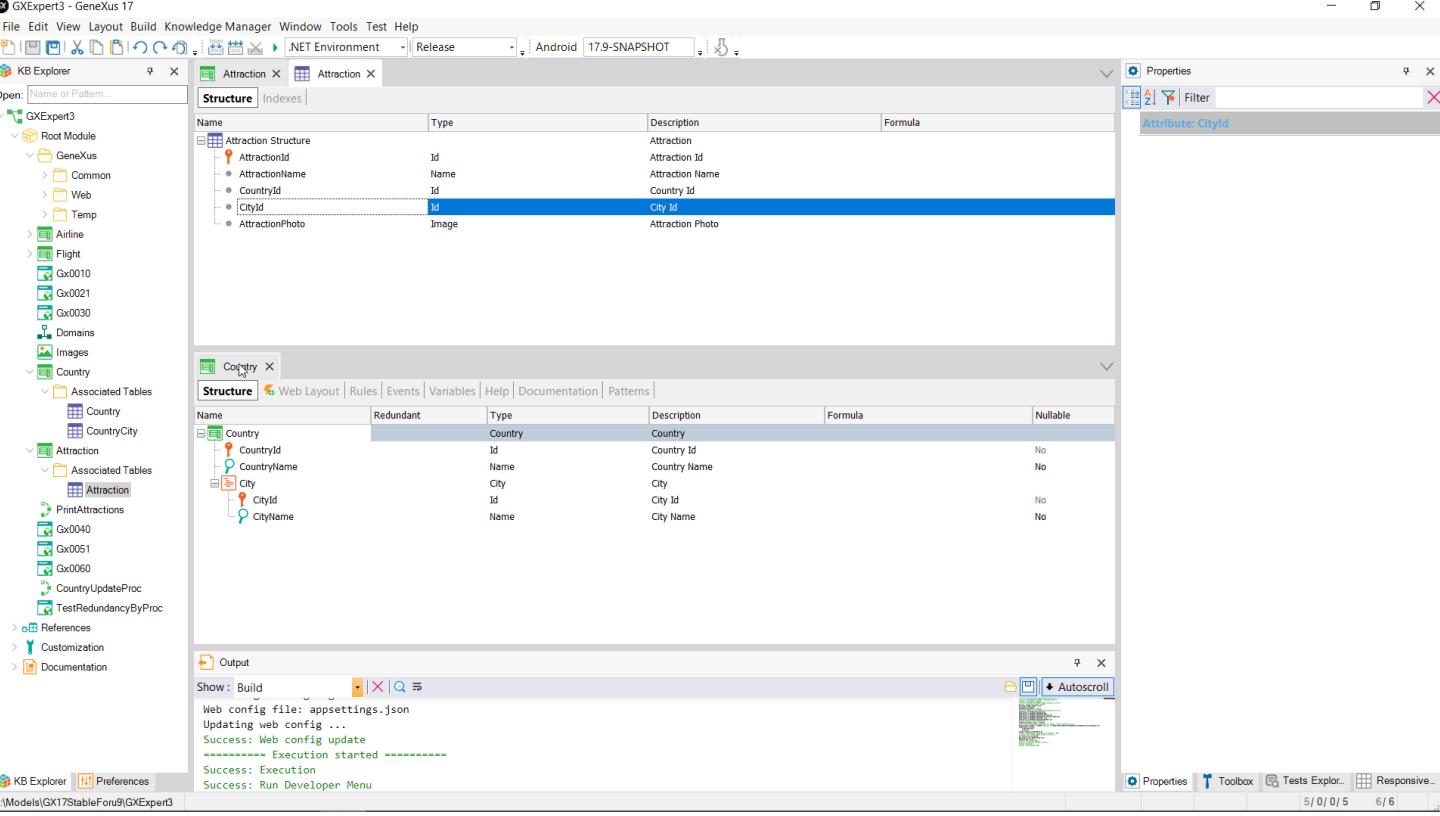






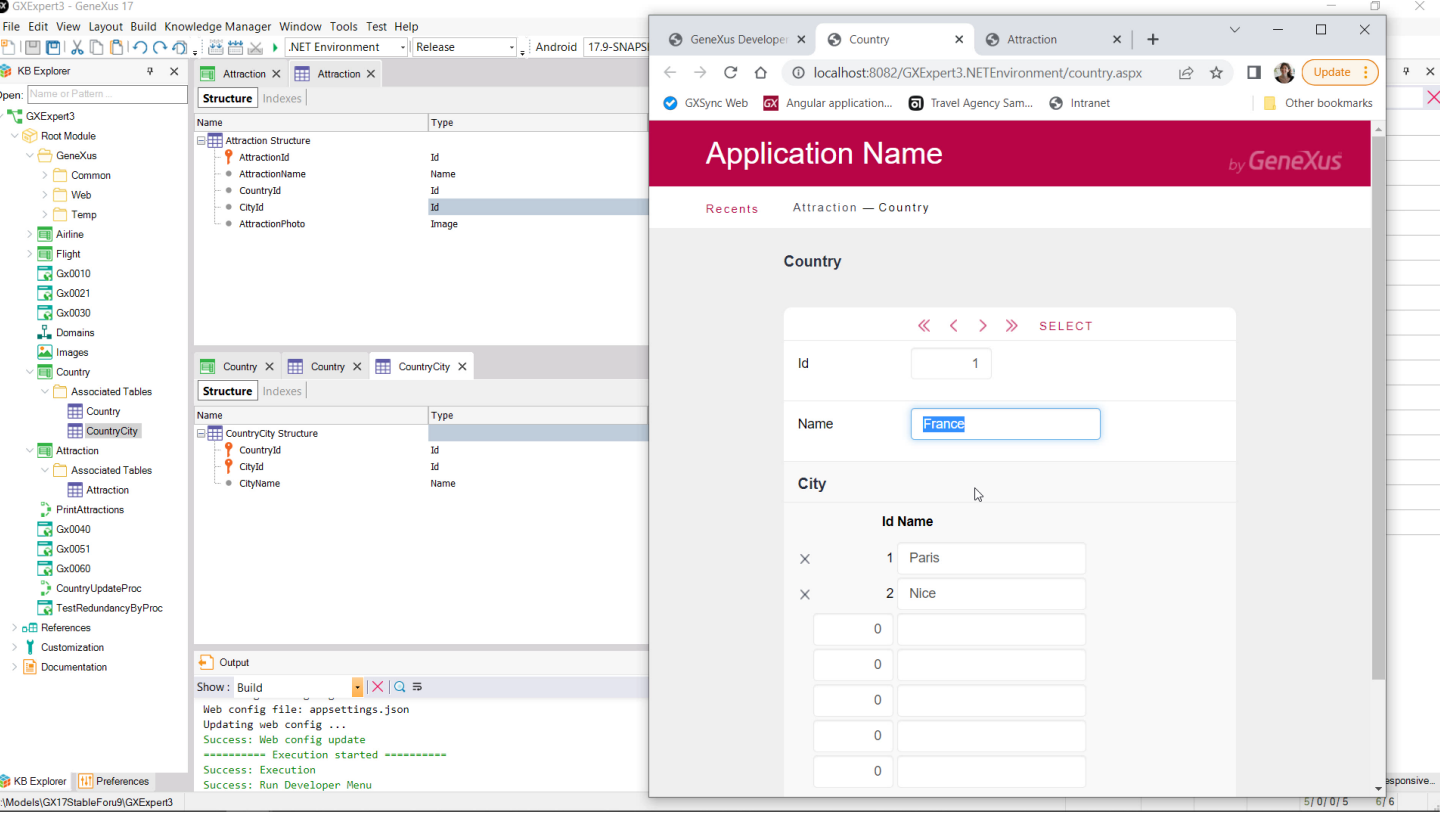


...solo podremos hacerlo si ambos están en la tabla **Attraction**.

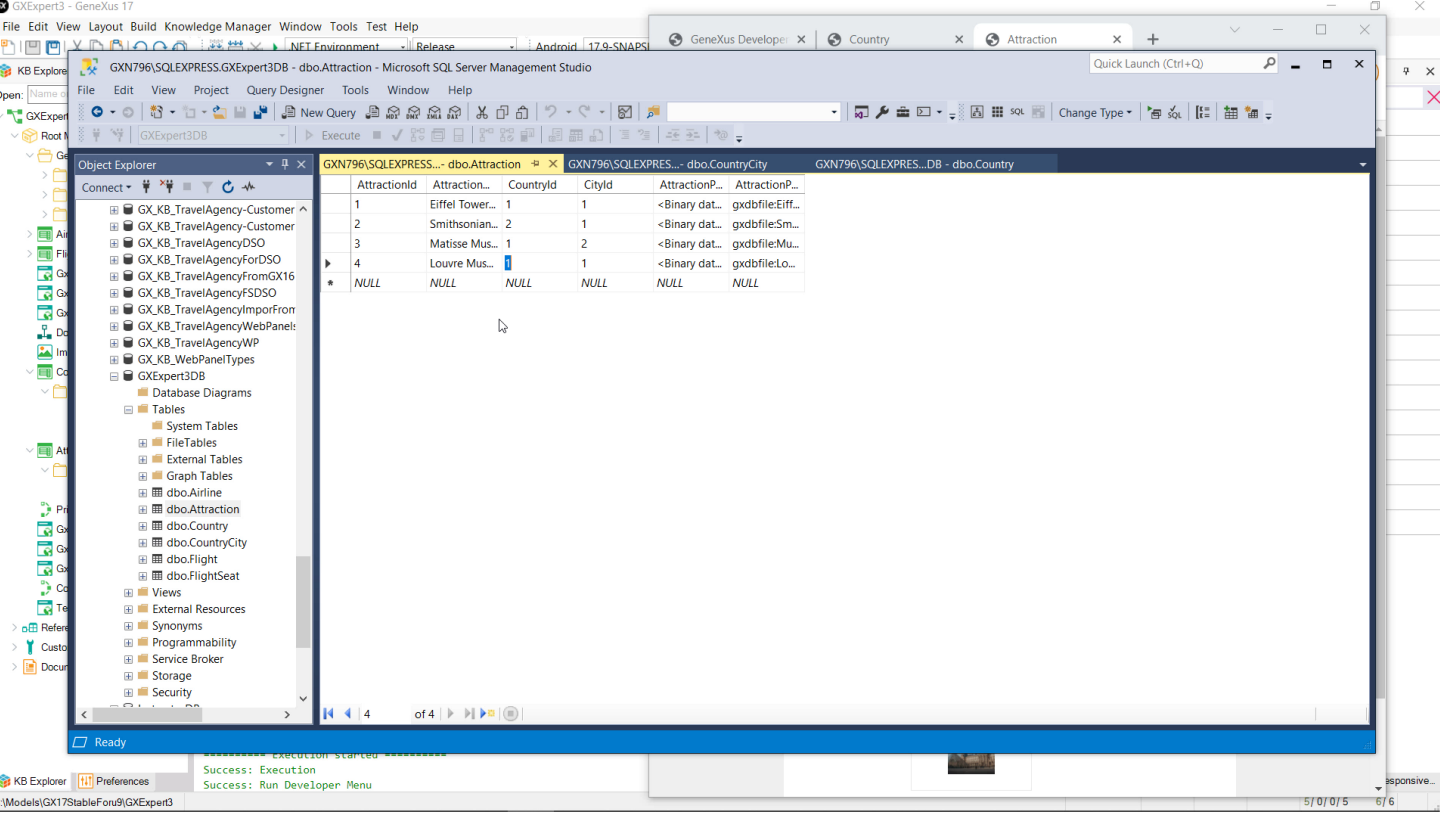


Veamos en GeneXus cómo declarar la redundancia y sus efectos.

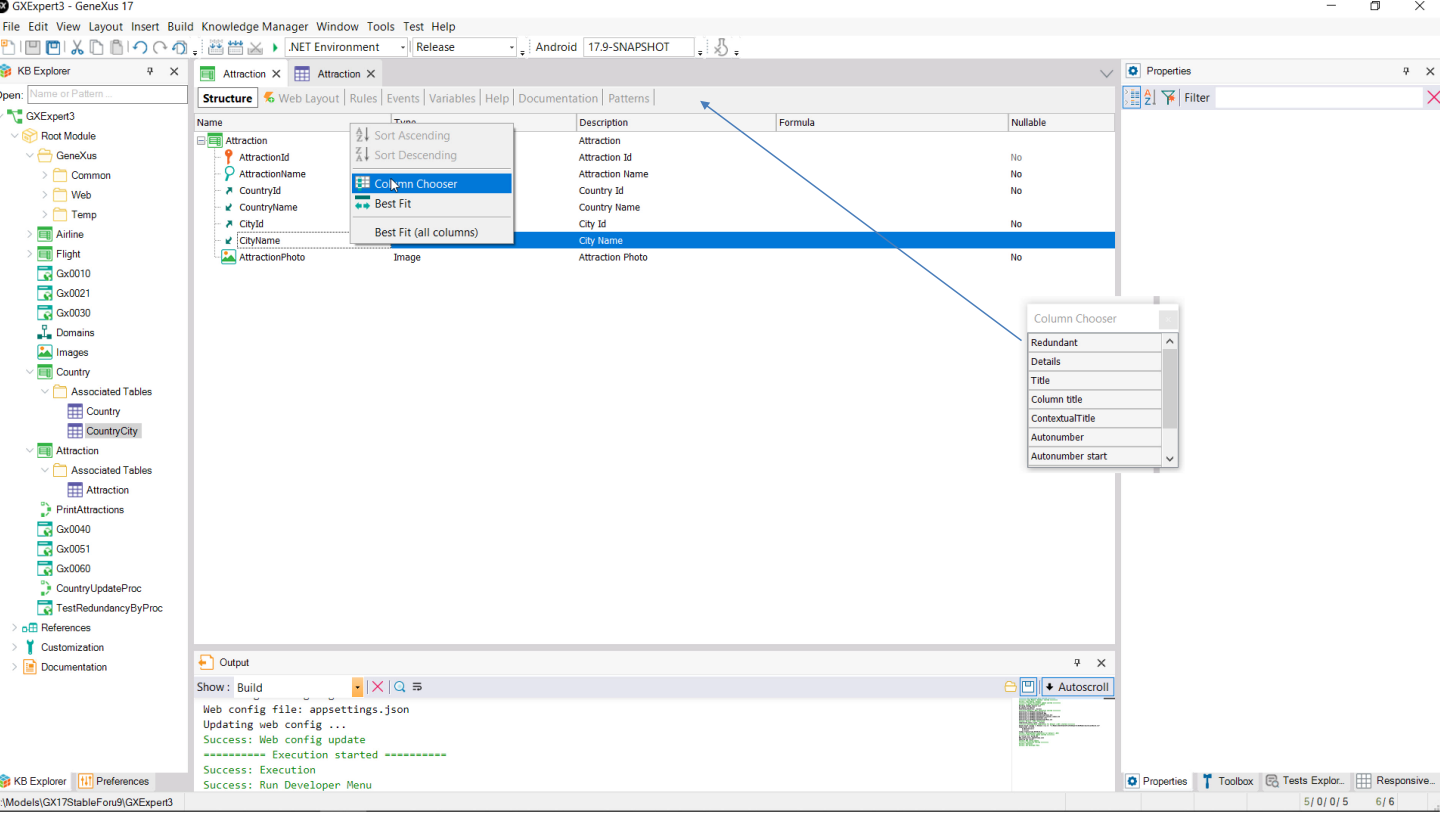
Tenemos la transacción de Attracciones infiriendo, como es lo habitual, los valores de CountryName y CityName a partir de las claves foráneas. De hecho, si observamos la estructura de la tabla de atracciones, vemos que no están esos atributos, sino solo las claves foráneas.



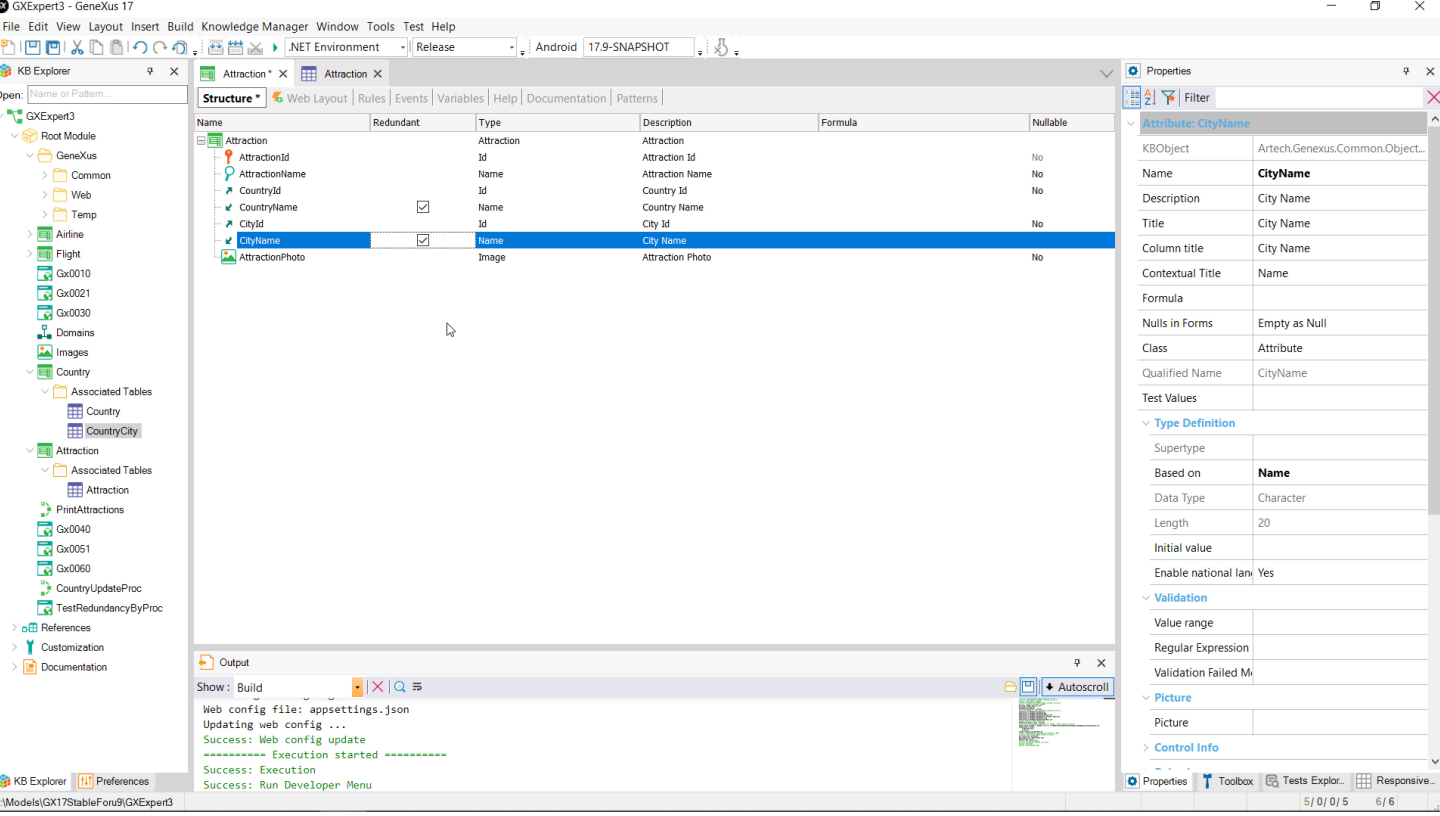
Ya tenemos algunos datos cargados en ejecución. enemos dos países con sus ciudades. Prestemos especial atención al 1, France, con sus dos ciudades Paris y Nice. Luego vemos que tenemos 3 atracciones de Francia: dos de París y una de Nice.



Si buscamos los datos de las tablas en SQLServer... vemos que en la de atracciones solo están las claves foráneas.

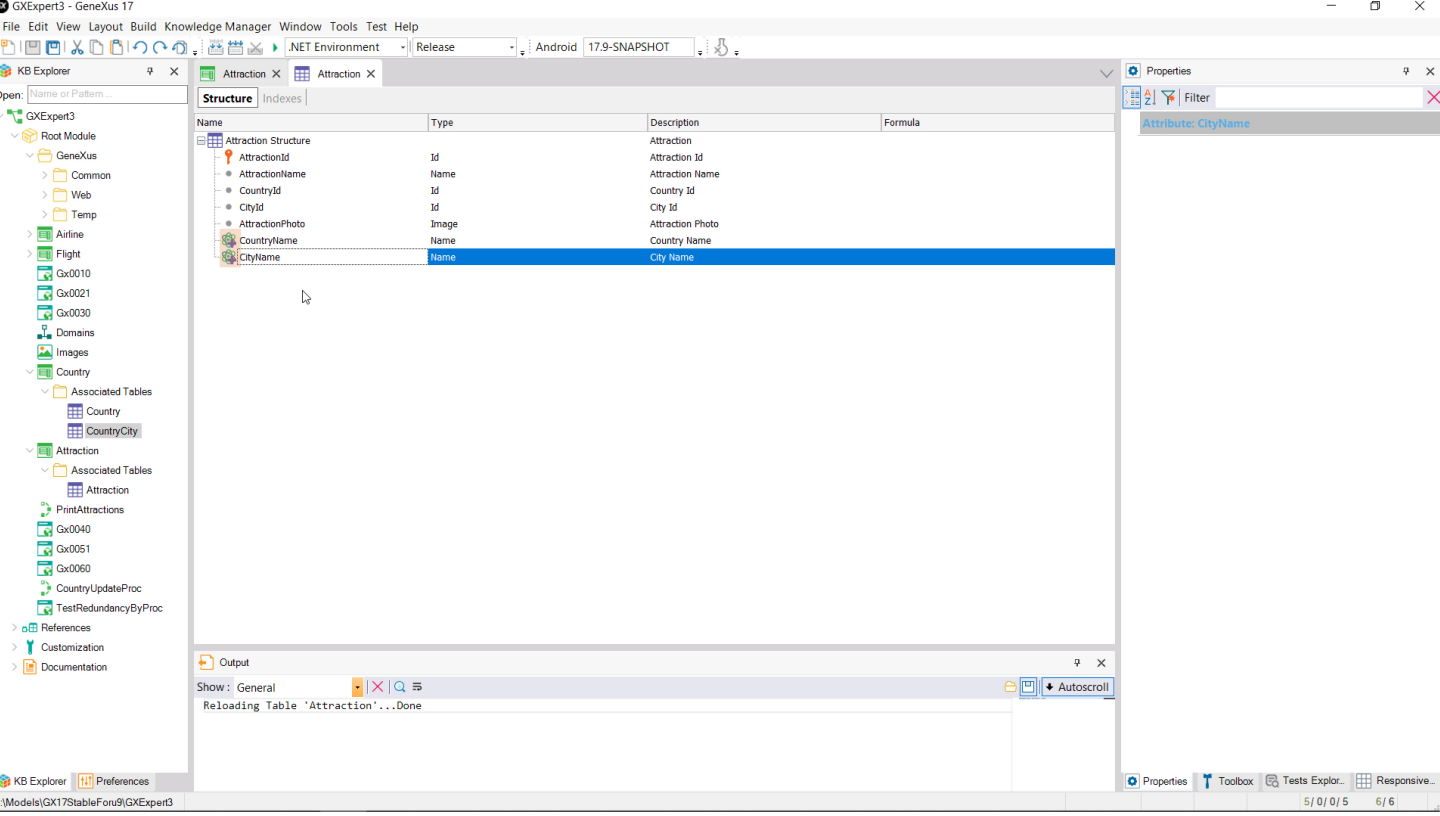


Bien, ahora queremos definir en Attraction los dos atributos inferidos como redundantes. Para ello agregamos en el editor de la estructura de la transacción esta columna...

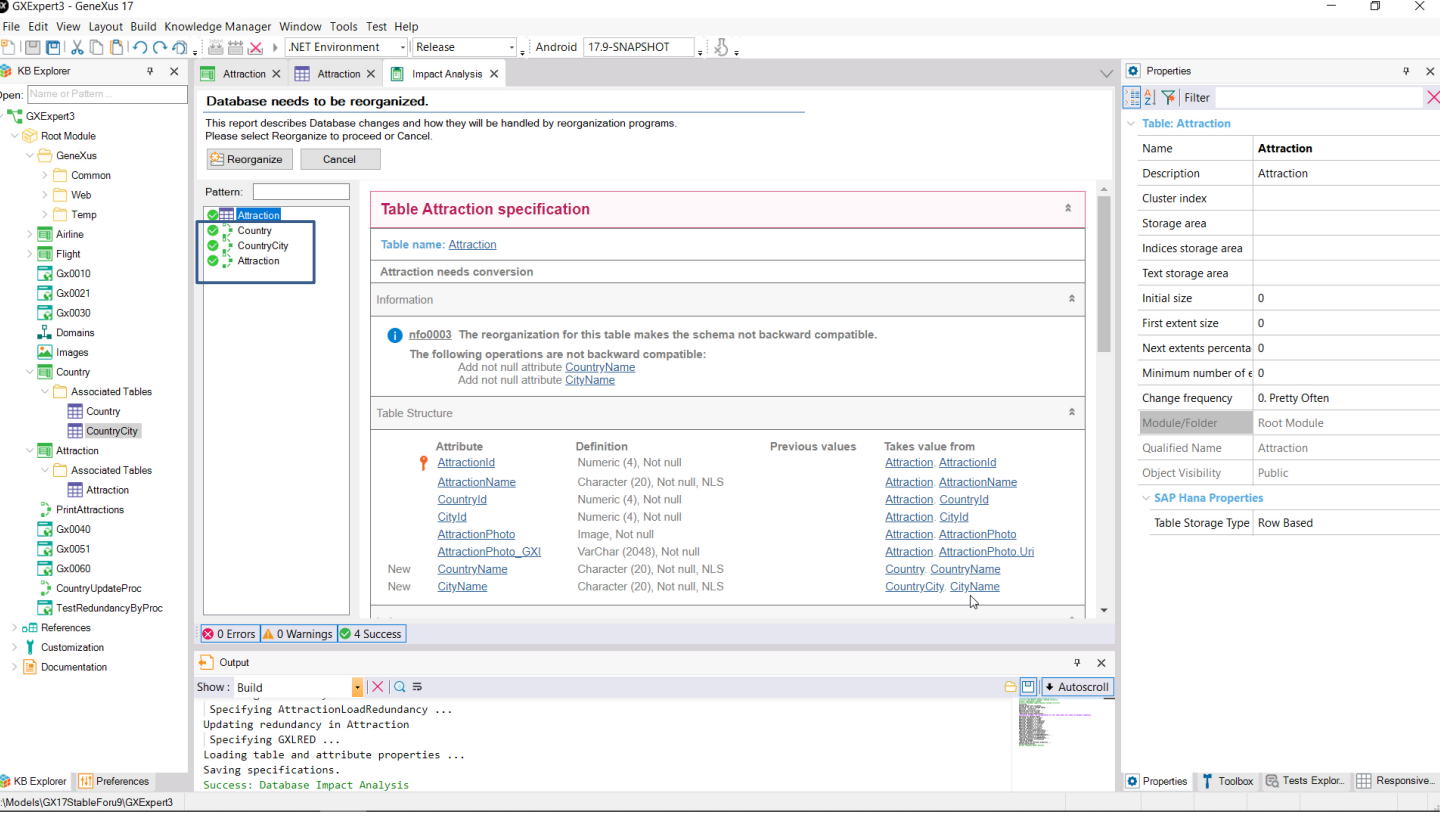


... que nos ofrece **check boxes para indicar cuáles atributos de la estructura de la transacción queremos redundar. Nos ofrece únicamente redundar los atributos inferidos. Si hubiera atributos fórmula también nos ofrecería redundarlos, como veremos.**

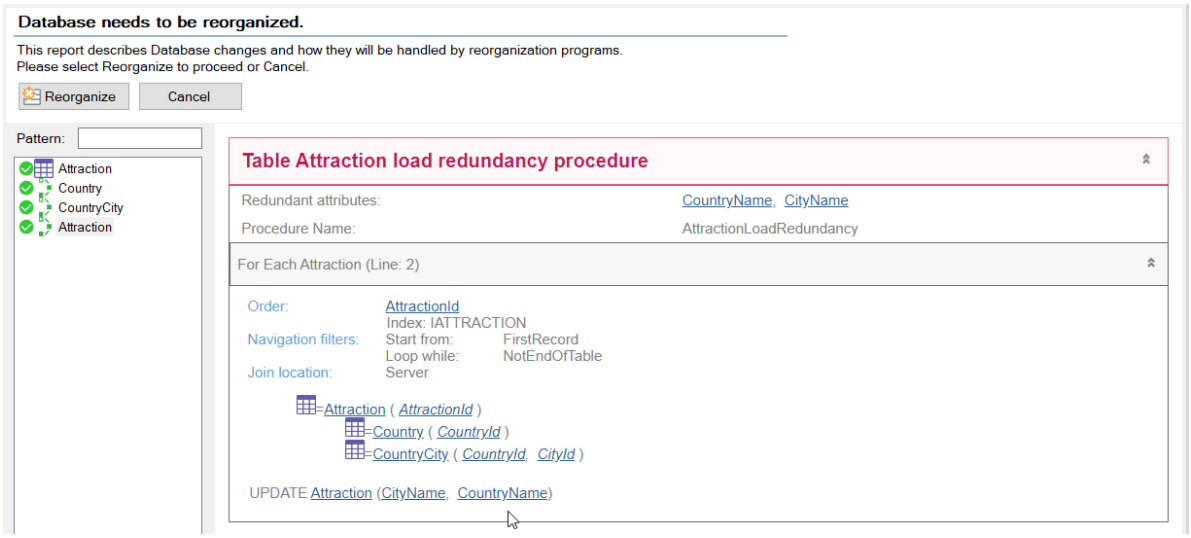
Vamos a marcar ambos, porque queremos que ambos se redunden en la tabla Attraction.



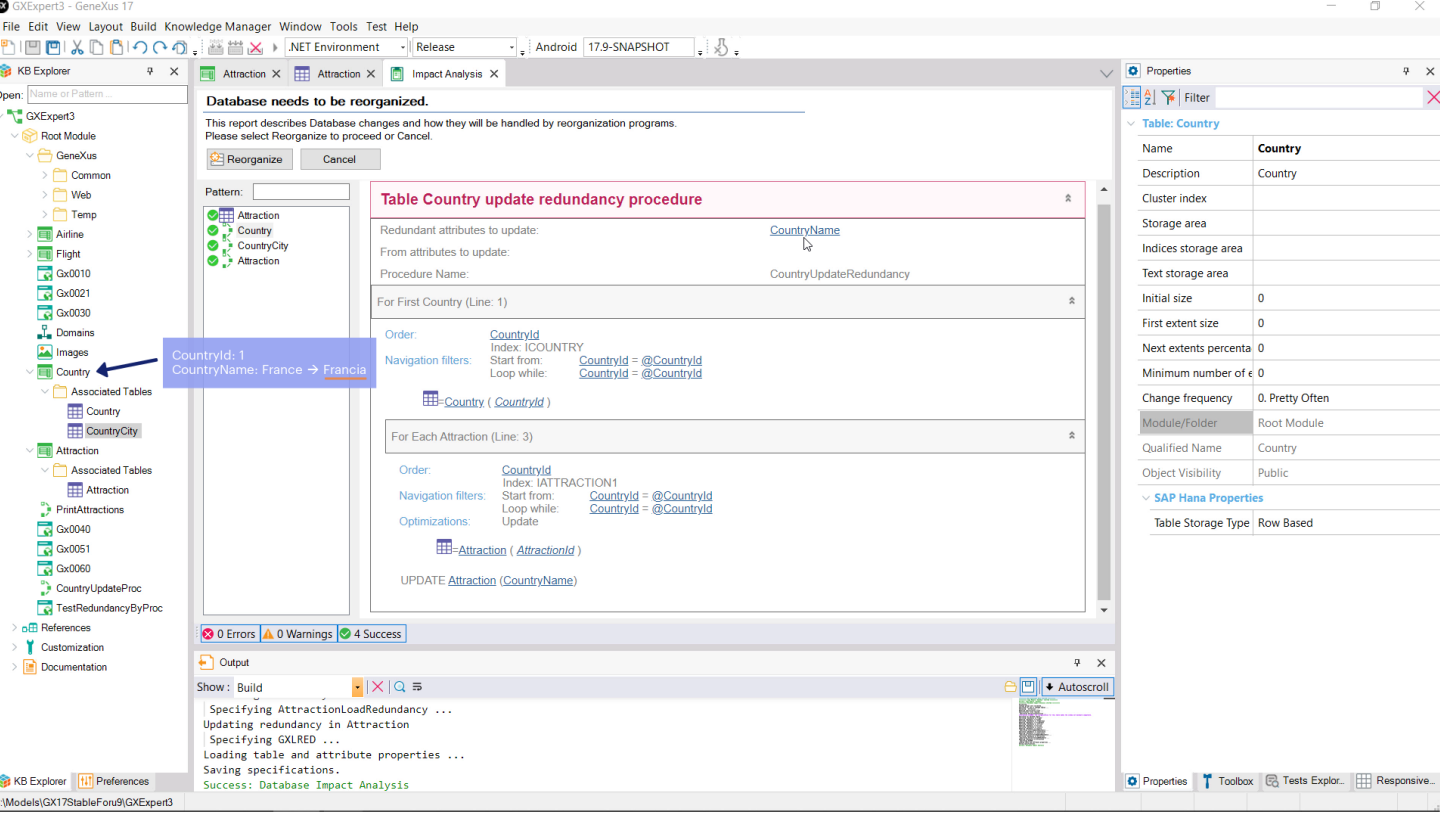
Grabemos. Vemos que se agregaron los atributos a la estructura de la tabla y se indica de esta manera que son redundantes.



Si ahora pedimos para ejecutar, claramente habrá que reorganizar la tabla Attraction para agregar esos atributos que hasta el momento eran inferidos pero ahora queremos que estén almacenados.
Observemos que se indica de dónde se tomarán sus valores.
Bien, esto lo esperábamos. Pero, ¿qué son estos tres procedimientos internos que creará GeneXus?

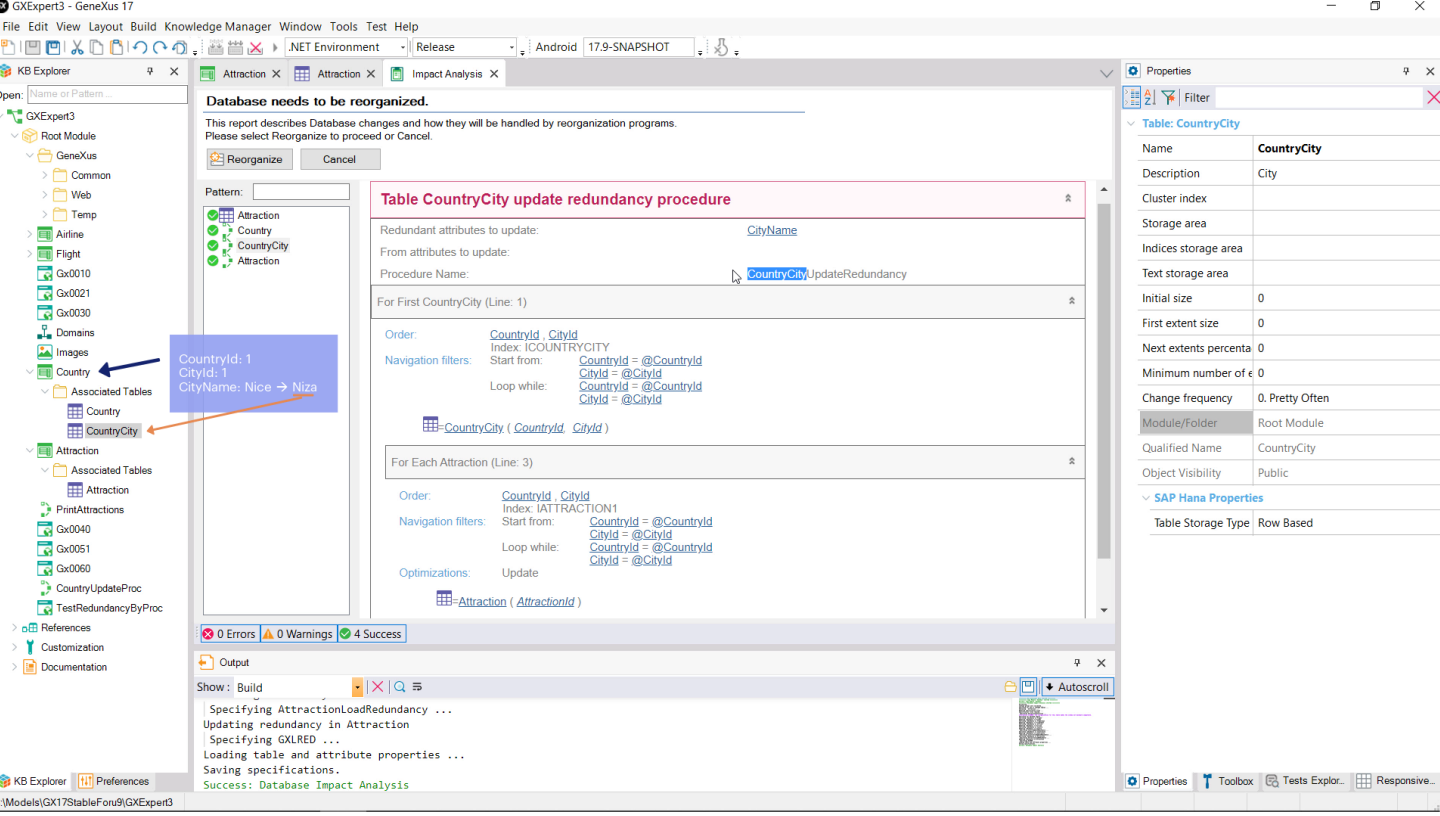


Empecemos por analizar el último. Vemos que el nombre es AttractionLoadRedundancy. Aquí nos dice que es el procedimiento de carga de las redundancias de la tabla Attraction. Y nos indica cuáles son los atributos redundantes que debe cargar. Lo que hará este procedimiento es recorrer toda la tabla de atracciones y para cada una ir a buscar a Country el valor del atributo CountryName para almacenarlo en el atributo redundante de esta tabla Attraction, e ir a buscar a la tabla de ciudades el valor del atributo CityName para almacenarlo en el nuevo atributo CityName redundante en esta tabla Attraction. En definitiva deberá ejecutar automáticamente este procedimiento tras reorganizar la tabla Attraction para que los atributos redundantes sean cargados con los valores apropiados.



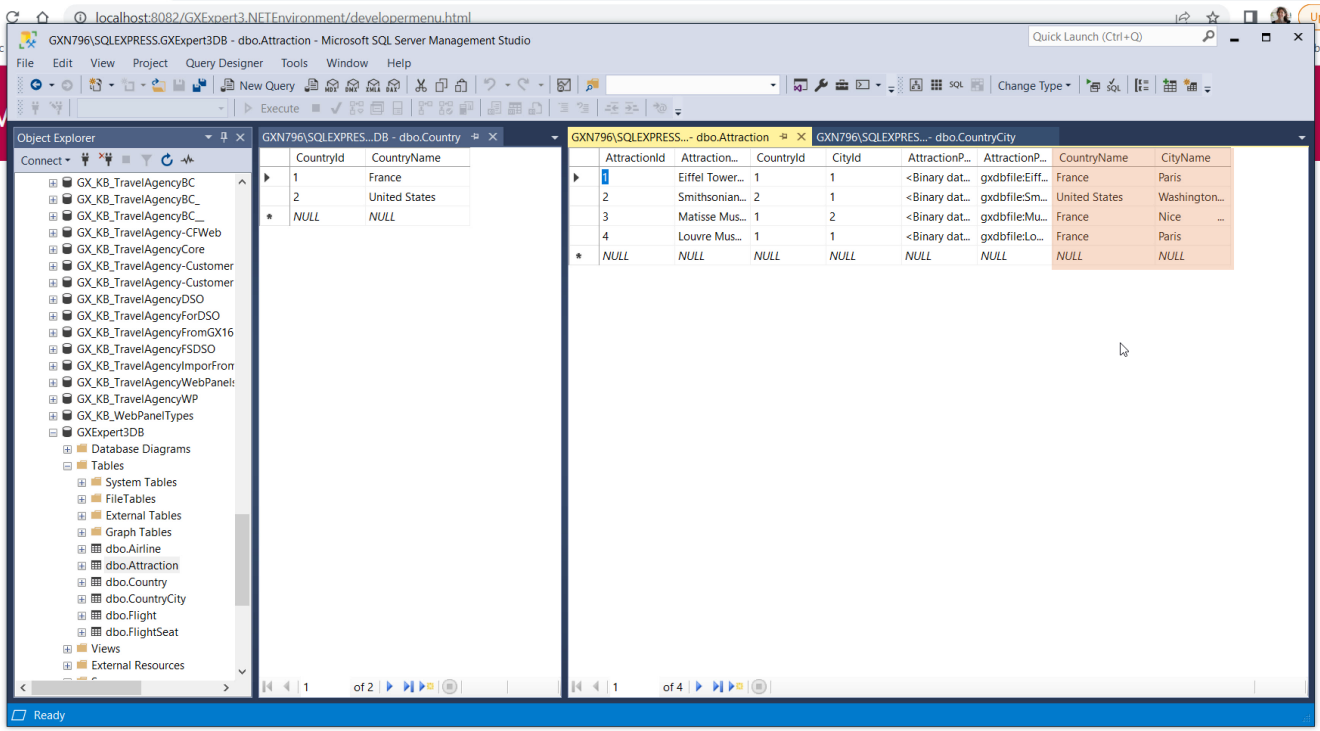
Bien, pero, ¿y estos dos procedimientos?

Si analizamos el primero, vemos que es un procedimiento de actualización de las redundancias de la tabla Country. Sabemos que el atributo CountryName de esta tabla está redundado en Attraction. Esto significa que si ejecutamos la transacción y cambiamos para un país su nombre, ese cambio debería ser efectuado también para todas las atracciones que correspondan a ese país, para mantener actualizada la redundancia. Es lo que hará este procedimiento, que será invocado de manera transparente cada vez que el usuario modifique a través de la transacción Country (o de su business component) el valor de CountryName para un CountryId. Aquí podemos apreciar que el procedimiento recibe como parámetro el valor de la clave primaria para instanciar ese registro, y luego recorre la tabla de atracciones filtrando por ese valor y para cada atracción actualiza el CountryName en esa tabla.



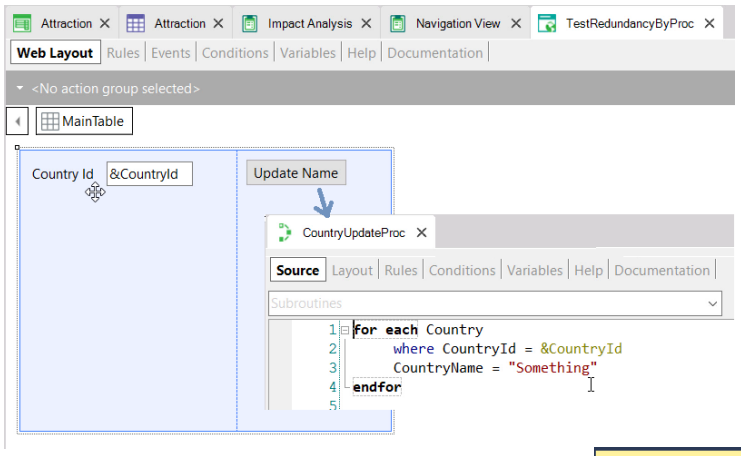
¿Y qué pasa con este otro procedimiento? Lo mismo, pero para mantener actualizadas las redundancias relativas a la tabla de ciudades. Es decir, cuando desde la transacción Country se modifique el nombre de una ciudad de un país, automáticamente y de forma transparente se ejecutará este procedimiento, al que se le enviarán por parámetro el id de país y el id de ciudad, se accederá a ese registro de la tabla y para cada registro de Attraction que matchee se modificará en Attraction el valor del atributo redundante CityName.

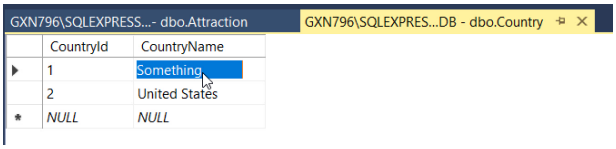
Estos procedimientos se crearán entonces en esta reorganización, y se agregará a las transacciones la lógica que acabamos de explicar. Reorganizamos.



Veamos ahora la tabla en el SQLServer. Se agregaron los atributos redundantes y se les asignó el valor que les correspondía.
Vamos ahora a cambiar el nombre del país France por su nombre en español, Francia. Lo haremos a través de la transacción.
Vamos a ver los datos de la tabla... cambió el nombre en la tabla de la transacción... y a ver qué sucedió con el atributo redundante... tal como esperábamos lo actualizó.
De igual manera, si vamos a modificar un nombre de ciudad, por ejemplo el de Niza, escribiéndolo ahora en español... vemos que en la tabla de las ciudades está lógicamente cambiado, y ahora en la de Atracciones... también.

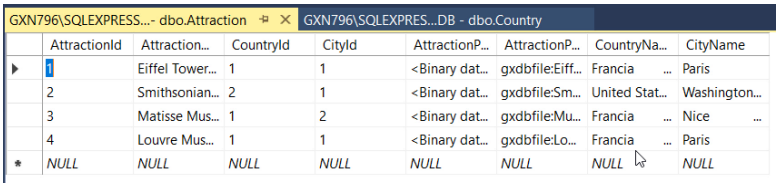
Eso fue por hacer la modificación a través de la transacción Country.





GXN796\SQLEXPRESS...- dbo.Attraction | GXN796\SQLEXPRES...DB - dbo.Country

	CountryId	CountryName
▶	1	Something
	2	United States
*	NULL	NULL

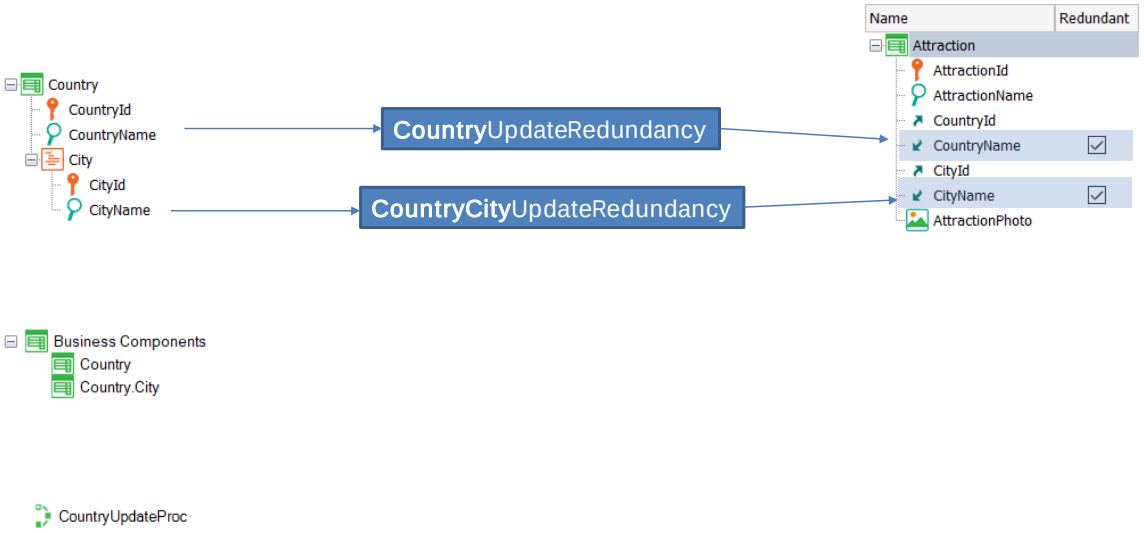


GXN796\SQLEXPRESS...- dbo.Attraction | GXN796\SQLEXPRES...DB - dbo.Country

	AttractionId	Attraction...	CountryId	CityId	AttractionP...	AttractionP...	CountryNa...	CityName
▶	1	Eiffel Tower...	1	1	<Binary dat...	gxdbfile:Eiff...	Francia	Paris
	2	Smithsonian...	2	1	<Binary dat...	gxdbfile:Sm...	United Stat...	Washington...
	3	Matisse Mus...	1	2	<Binary dat...	gxdbfile:Mu...	Francia	Nice
	4	Louvre Mus...	1	1	<Binary dat...	gxdbfile:Lo...	Francia	Paris
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Veamos qué pasa si lo hacemos a través de un procedimiento... Tenemos un web panel para que el usuario ingrese un id de país, y al presionar el botón invocamos a un procedimiento que recibe ese id y va a la tabla Country para modificar para el país con ese id el valor del atributo CountryName por este "Something". Ejecutemos para probar.

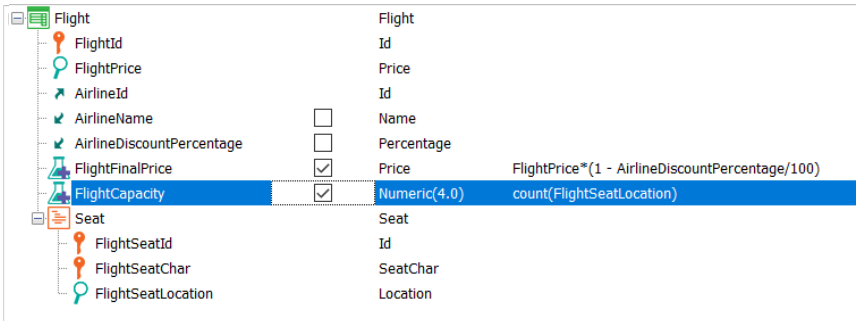
Elegimos el país de id 1, que era Francia. Vemos que en la tabla Country efectivamente se modificó su valor por el que asignó el procedimiento. Pero ahora vayamos a ver si el procedimiento mantuvo actualizada la redundancia en Attraction. No, no lo hizo.

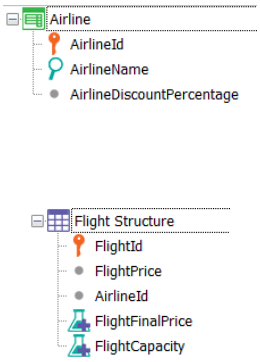


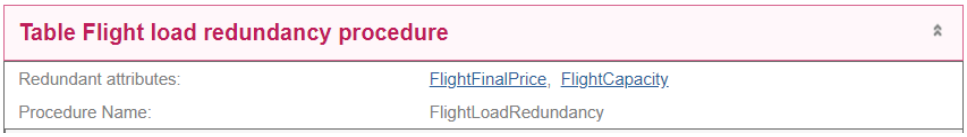
GeneXus invoca a esos procedimientos que vimos cuyos nombres terminaban en UpdateRedundancy únicamente cuando las modificaciones se realizan a través de la transacción (o el business component, lógicamente). No los invoca cuando las modificaciones se realizan de otro modo.

Así que hay que tener cuidado y en caso de modificar atributos que están redundados en otras tablas por otra vía distinta a la de la transacción o el business component, deberá ser el desarrollador quien se encargue de mantener los atributos redundantes actualizados. No lo hará GeneXus.

Formulas redundancy



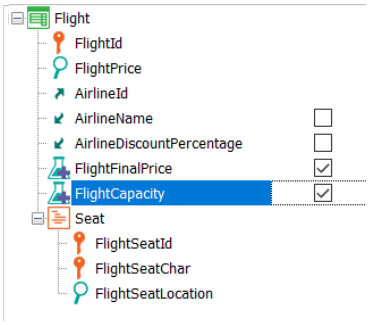


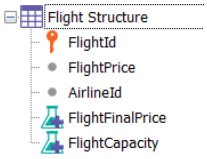


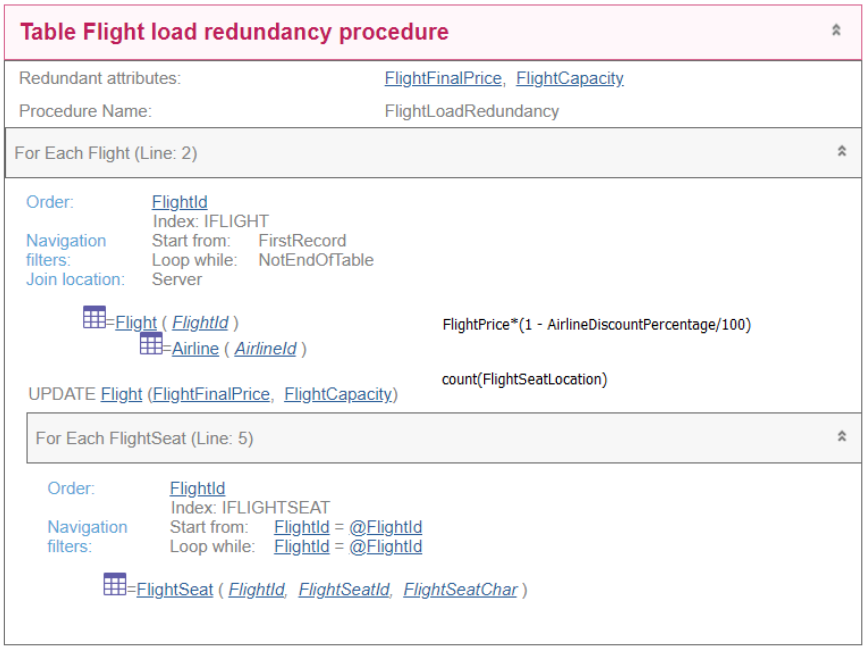
El otro caso de redundancias que ya habíamos mencionado era la de fórmulas. En este ejemplo vemos que además de ofrecernos redundar los atributos inferidos, también nos ofrece redundar las dos fórmulas definidas para la transacción: la horizontal que aplica un descuento al precio del vuelo de acuerdo al porcentaje de descuento fijado por la aerolínea, y la aggregate que cuenta la cantidad de asientos del vuelo.

Al definir estas fórmulas como redundantes se agregarán los atributos a la tabla. Pero además, como es evidente, en la reorganización GeneXus deberá crear también un procedimiento para cargarlos con los valores correspondientes.

Formulas redundancy

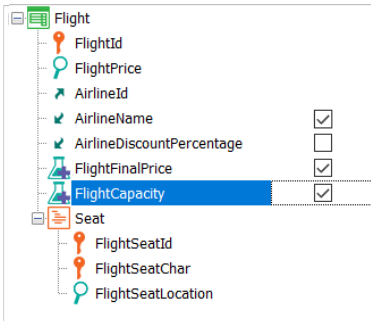


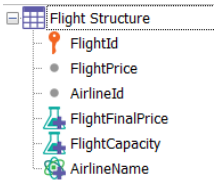


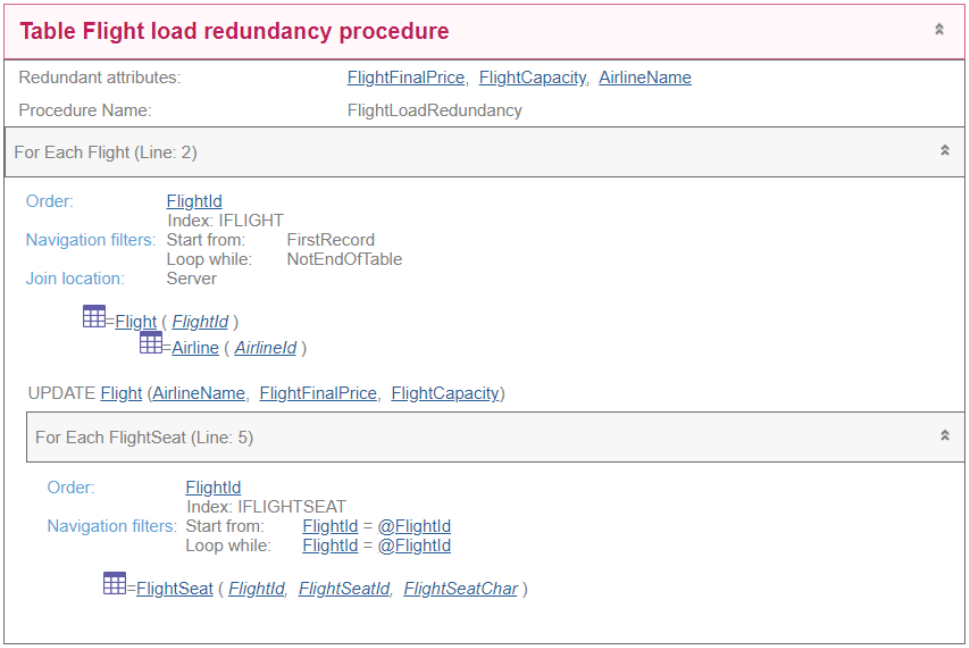


Es decir, un procedimiento en el que para cada registro de la tabla Flight disparará el cálculo de cada fórmula... para FlightFinalPrice deberá ir a buscar a Airline el valor de AirlineDiscountPercentage, y almacenará su valor en el atributo redundante; y para FlightCapacity deberá ir a contar los registros asociados en la tabla FlightSeat, y almacenar el resultado en el atributo redundante.

Formulas redundancy

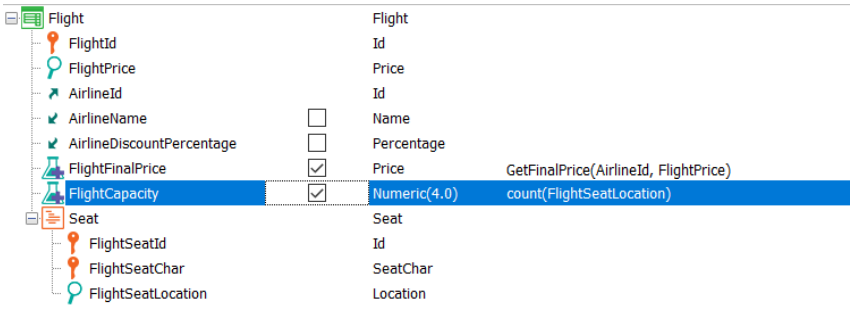


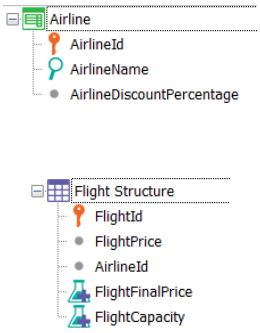


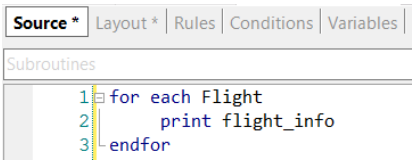


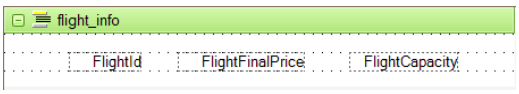
Si además definiéramos, por ejemplo, AirlineName como redundante, es decir, una redundancia referencial, en este mismo procedimiento es donde se cargaría su valor en la tabla. Por eso el nombre del procedimiento siempre se compone de la concatenación de el nombre de la tabla donde están las redundancias, en este caso Flight, y LoadRedundancy (carga de redundancias). Es decir, en este procedimiento se cargarán todas las redundancias de la tabla: las referenciales y las de fórmulas.
Y será el procedimiento invocado automáticamente en la reorganización, luego de agregarse los nuevos atributos a la tabla de la base de datos.

Formulas redundancy







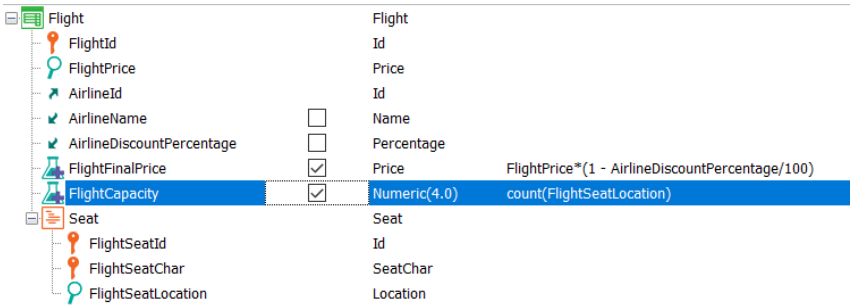


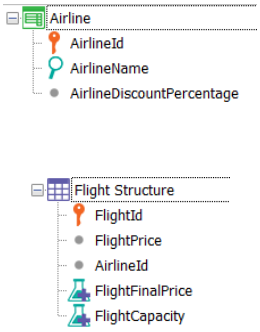
Volvamos a concentrarnos solo en las fórmulas. Por supuesto, una vez redundadas, cuando se necesite la información de precio de vuelo o capacidad, no se volverá a disparar la fórmula, sino que se traerá el valor almacenado, y esa es justamente la gracia.

Si ejecutáramos millones de veces este listado, y aunque no suceda en la realidad hubiera miles de asientos para cada vuelo, entonces ejecutar cada vez el cálculo de la fórmula FlightCapacity podría tornarse un problema de performance. Tenerla redundante nos evita el costo del cálculo por cada consulta. Solamente tenemos el costo del cálculo para cargar el atributo redundante la primera vez, y para mantenerlo actualizado luego.

Para la fórmula horizontal del ejemplo no parece demasiado clara la utilidad de definirla redundante, por lo menos en cuanto a la performance, salvo que se necesite, por ejemplo, un panel o web panel donde el usuario quiera filtrar los vuelos mostrados en un grid por precio de vuelo.
El caso se volvería más interesante si el cálculo horizontal involucrara muchas tablas de la extendida. O, más aún, si se tratase de una fórmula horizontal de las que se resuelven invocando a un procedimiento que sí realiza un cálculo complejo.

Formulas redundancy






```
For each Flight
     FlightPrice *= 1.1
endfor
```

```
For each Flight.Seat
     where FlighId = 4
     Delete
endfor
```

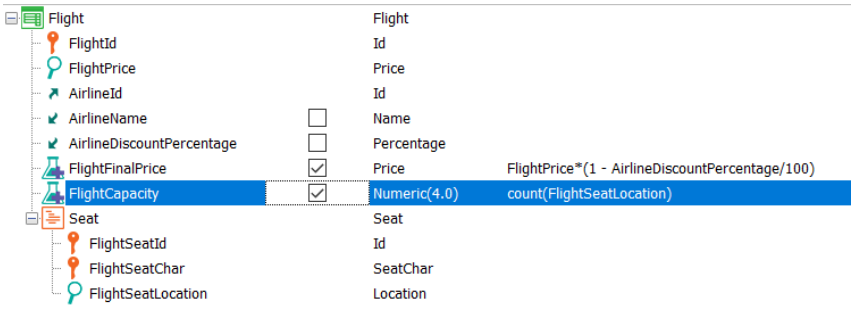
```
new
     FlightId = 12503
     FlightPrice = 500
     AirlineId = find(AirlineId, AirlineName = "Air Europe")
     new
          FlighSeatId = 1
          FlightChar = 'B'
          FlightSeatLocation = Location.Window
     endnew
endnew
```

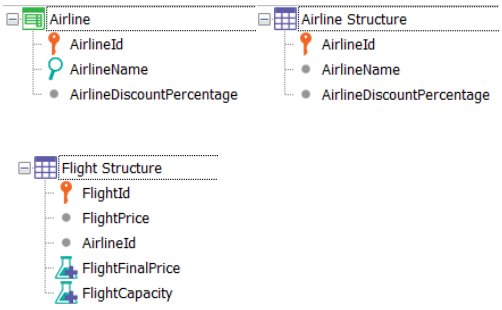
Si la ventaja de redundar atributos fórmula es clara, ¿cuál es su desventaja?

Que deberían mantenerse siempre actualizados, y esa actualización tiene un costo. Si a través de la transacción Flight (o de su business component) se modifica el valor de FlightPrice, como siempre, la fórmula horizontal que lo involucra se redispara. Lo mismo si se agrega una línea o se elimina una, la fórmula FlightCapacity se redispara. En ambos casos se almacena su valor en los atributos redundantes y el desarrollador no debe preocuparse por hacerlo.

Sin embargo, como en el caso de las redundancias referenciales, si la modificación se realiza por procedimiento, como en estos ejemplos, GeneXus no hará nada. Aquí el desarrollador deberá preocuparse por actualizar los atributos redundantes, agregando código.

Formulas redundancy



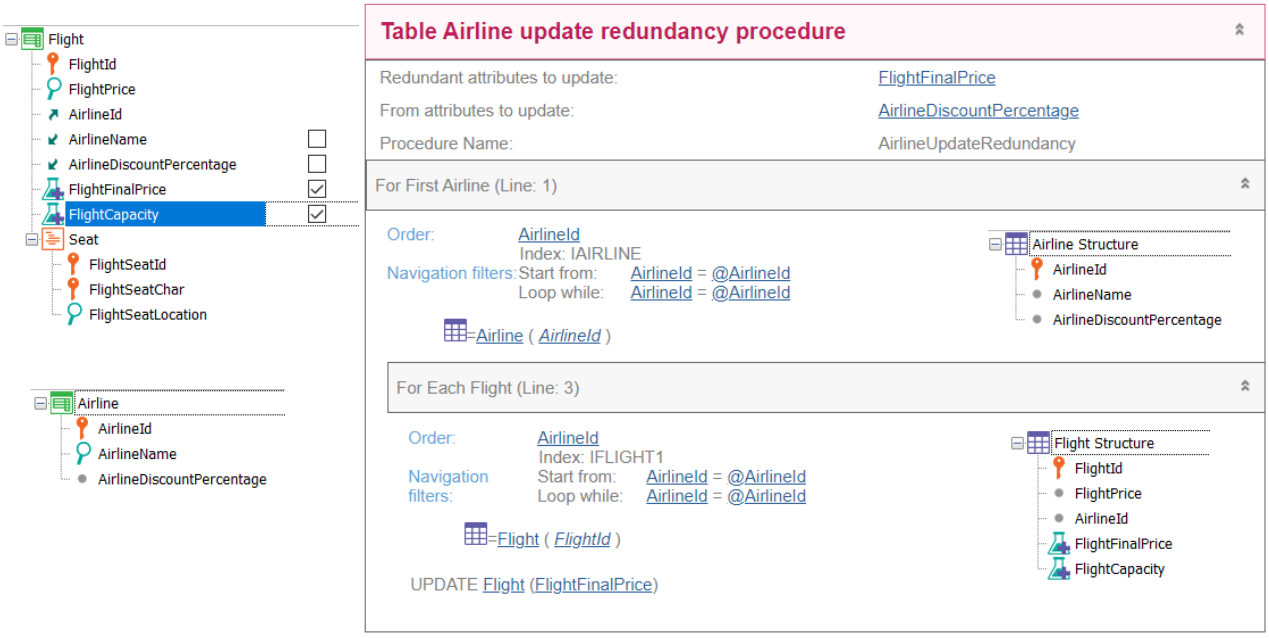


¿Y qué pasa si el usuario ingresa a la transacción Airline y modifica para una aerolínea el porcentaje de descuento? (o si lo hace a través del business component).

En Flight la fórmula redundante FlightFinalPrice depende de ese valor, por lo que debería actualizarse el valor redundante almacenado para todos los registros de Flight que pertenezcan a esa aerolínea.

GeneXus creará en la reorganización el procedimiento cuyo nombre es la concatenación del nombre de tabla, en este caso Airline, y UpdateRedundancy, tal como lo hacía con las redundancias referenciales.

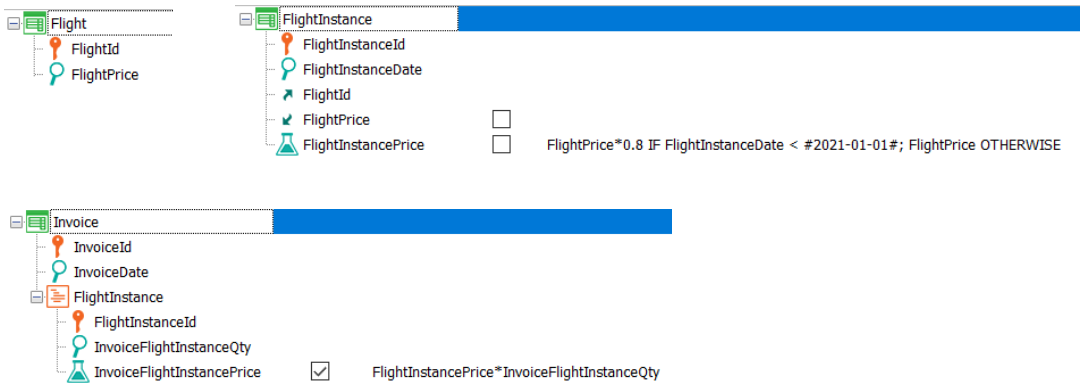
Formulas redundancy



Este procedimiento se invocará desde la transacción Airline pasándole el Id y el procedimiento accederá al registro correspondiente, y luego recorrerá la tabla Flight filtrando por esa aerolínea y redisparará la fórmula FlightFinalPrice, almacenando su resultado en la tabla.

Todo esto significa un costo de performance, por lo que hay que evaluar muy bien cuándo conviene redundar una fórmula y cuando no.

Formulas redundancy: limitations



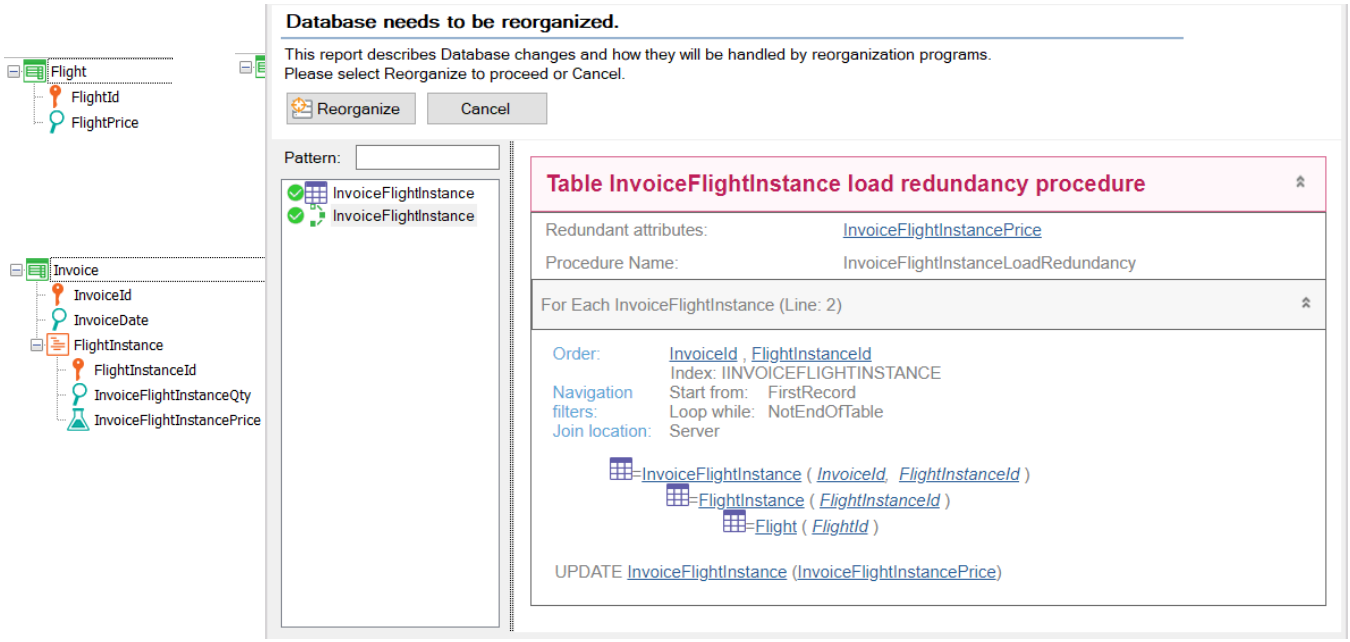
El mantenimiento de fórmulas redundantes por el momento tiene sus limitaciones.

Por ejemplo, si tenemos estas tres transacciones relacionadas, donde la transacción Flight registra la información genérica de un vuelo, como su precio, pero es la transacción FlightInstance la que corresponde al vuelo real, en una fecha determinada. En esta el precio del vuelo real se obtiene con una fórmula que toma en cuenta el precio de lista del vuelo según la fecha.

Y luego tenemos una transacción para registrar la facturación de vuelos. Las líneas registran los vuelos para los que se están comprado pasajes, y cuántos pasajes para cada uno. Entonces esta fórmula calcula el precio de cada línea, utilizando para ello el atributo fórmula inferido de FlightInstance, que a su vez es una fórmula, como dijimos.

Supongamos que queremos hacer redundante a esta fórmula en la tabla InvoiceFlightInstance. Si solo la redundamos a ella...

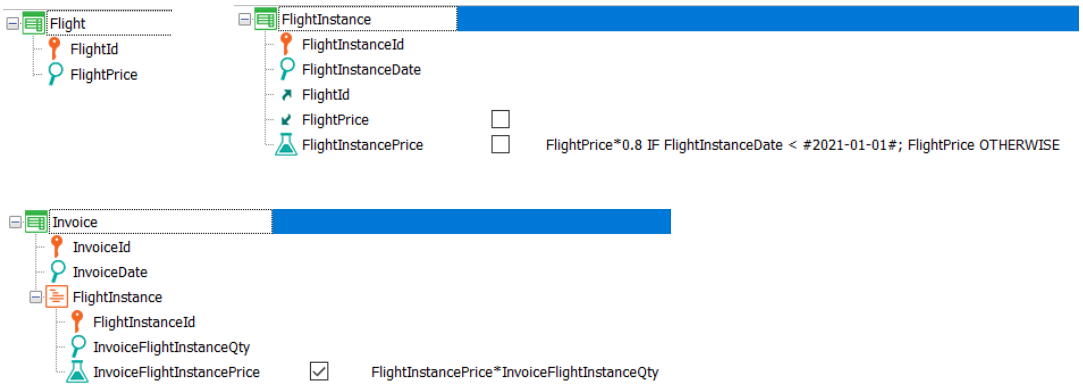
Formulas redundancy: limitations



...el procedimiento de carga de la redundancia se resolverá correctamente, yendo a la tabla InvoiceFlightInstance y disparando la fórmula horizontal para cada registro, y almacenándola.

Pero lo que no sucederá es que se creen procedimientos de actualización de las redundancias.

Formulas redundancy: limitations

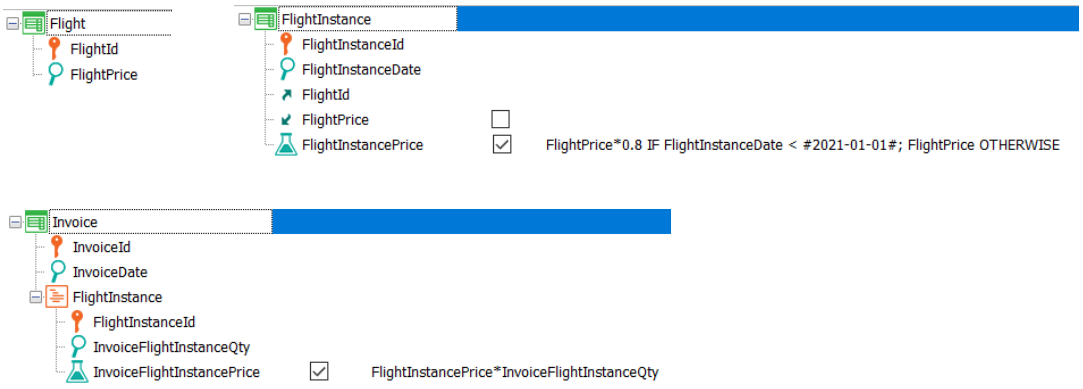


¿Qué queremos decir?

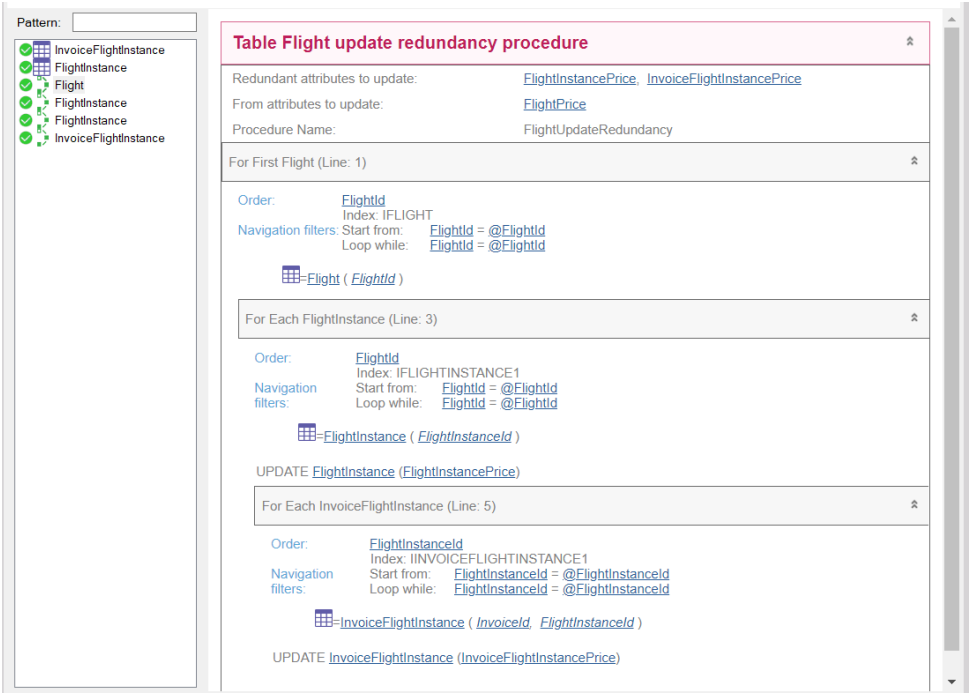
Esto significará que si el usuario ejecuta la transacción FlightInstance y cambia por ejemplo el valor de FlightInstanceDate (supongamos que de una fecha menor a esta, a una mayor), como la fórmula horizontal se redisparará y cambiará su valor, esperaríamos que la transacción invoque a un procedimiento, FlightInstanceUpdateRedundancy, que vaya a la tabla FlightInstance y actualice las redundancias correspondientes. Pero no lo hará.

De la misma manera, esperaríamos que si el usuario modifica el valor del atributo FlightPrice a través de la transacción Flight también exista un procedimiento FlightUpdateRedundancy que vaya a la tabla InvoiceFlightInstance a recalcular y realmacenar las redundancias del atributo fórmula que deban modificarse por la modificación de FlightInstancePrice, cuando corresponda. Tampoco lo hará.

Formulas redundancy: limitations

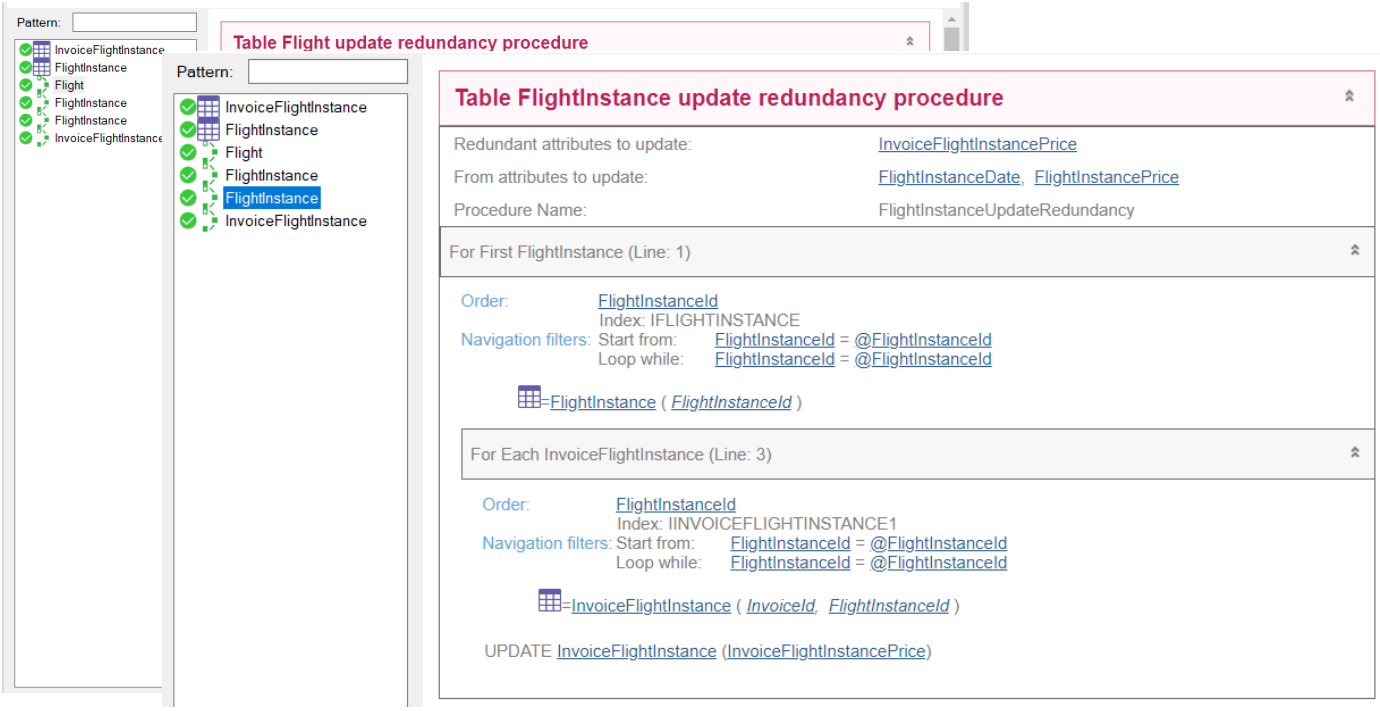


¿Pero, y si también redundamos la fórmula horizontal involucrada?



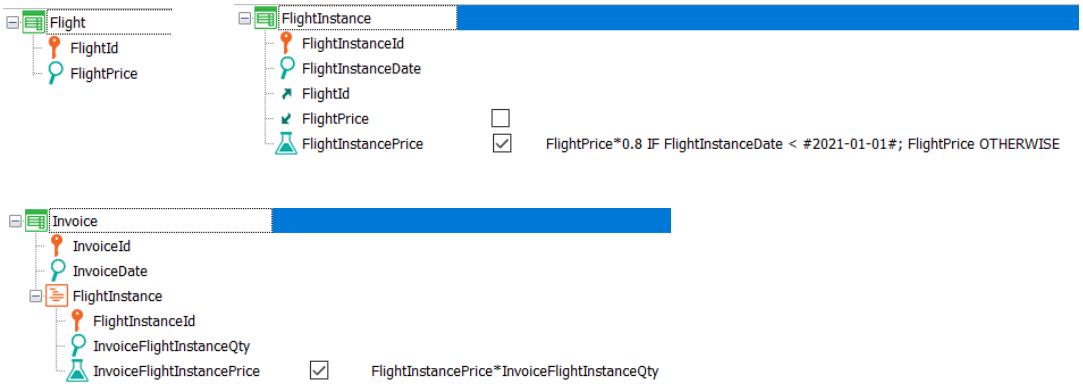
Aquí veremos al reorganizar que además de informar los cambios en las dos tablas para agregar las dos fórmulas como redundantes, y los dos procedimientos de carga de las redundancias, se crearán ambos procedimientos de UpdateRedundancy.

El primero que se dispara desde la transacción Flight cuando se cambia el precio, que va a actualizar la primera fórmula redundante en FlightInstance, y luego, a partir de ella, la segunda, en InvoiceFlightInstance.



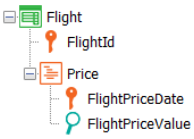
Y el segundo procedimiento, que se ejecutará cuando desde la transacción FlightInstance se modifica la fecha del vuelo, que va a actualizar el atributo redundante en InvoiceFlightInstance.

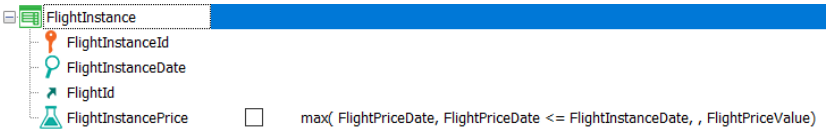
Formulas redundancy: limitations



Por lo que, si tenemos una fórmula que en su cálculo involucra a otra, para que el mantenimiento de su redundancia se mantenga automáticamente necesitamos definir a la fórmula interior como redundante también.

Formulas redundancy: limitations





FlightId	FlightPriceDate	FlighPriceValue
1	01/01/2022	800
1	02/02/2022	1000
1	04/04/2022	950

FlightId: 1

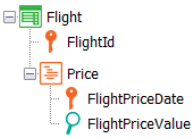
FlightInstanceDate: 03/03/2022

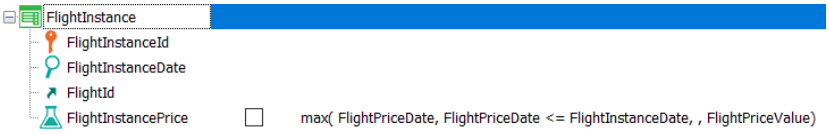
FlightInstancePrice: ?

Pero hay más limitaciones. Por ejemplo, las fórmulas aggregate/select la mayoría de las veces no podrán ser mantenidas a través de esos procedimientos de actualización.

Por ejemplo si mantenemos en Flight una lista de precios del vuelo por fecha, en FlighInstance tendremos que calcular el precio de un vuelo concreto de acuerdo al precio que le corresponda a la fecha del vuelo real. Es decir, calculamos el precio de acuerdo a una fórmula max.
Así, teniendo estos datos, si se está ingresando una instancia del vuelo 1, con esta fecha, la fórmula max se quedará con este registro...

Formulas redundancy: limitations





FlighId	FlightPriceDate	FlighPriceValue
1	01/01/2022	800
1	02/02/2022	1000
1	04/04/2022	950

FlightId: 1

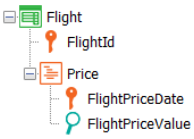
FlightInstanceDate: 03/03/2022

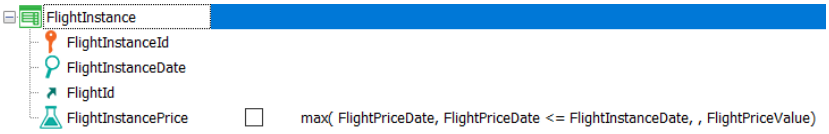
FlightInstancePrice: 1000

LIST

... por lo que devolverá el valor 1000 para el precio del vuelo. Si se estuviera imprimiendo en un listado, se mostraría 1000.

Formulas redundancy: limitations





FlightId	FlightPriceDate	FlighPriceValue
1	01/01/2022	800
1	02/02/2022	1500
1	04/04/2022	950

FlightId: 1

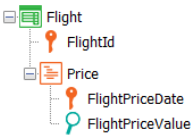
FlightInstanceDate: 03/03/2022 **LIST**

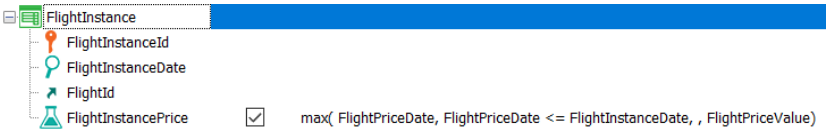
FlightInstancePrice: 1500

FlightPriceUpdateRedundancy?

Si luego el usuario ingresa a la transacción Flight y para la línea correspondiente a este registro, cambia 1000 por 1500, como la fórmula es virtual, al volverse a ejecutar el listado se disparará el max y se listará 1500.

Formulas redundancy: limitations





FlightId	FlightPriceDate	FlighPriceValue
1	01/01/2022	800
1	02/02/2022	1500
1	04/04/2022	950

FlightId: 1

FlightInstanceDate: 03/03/2022

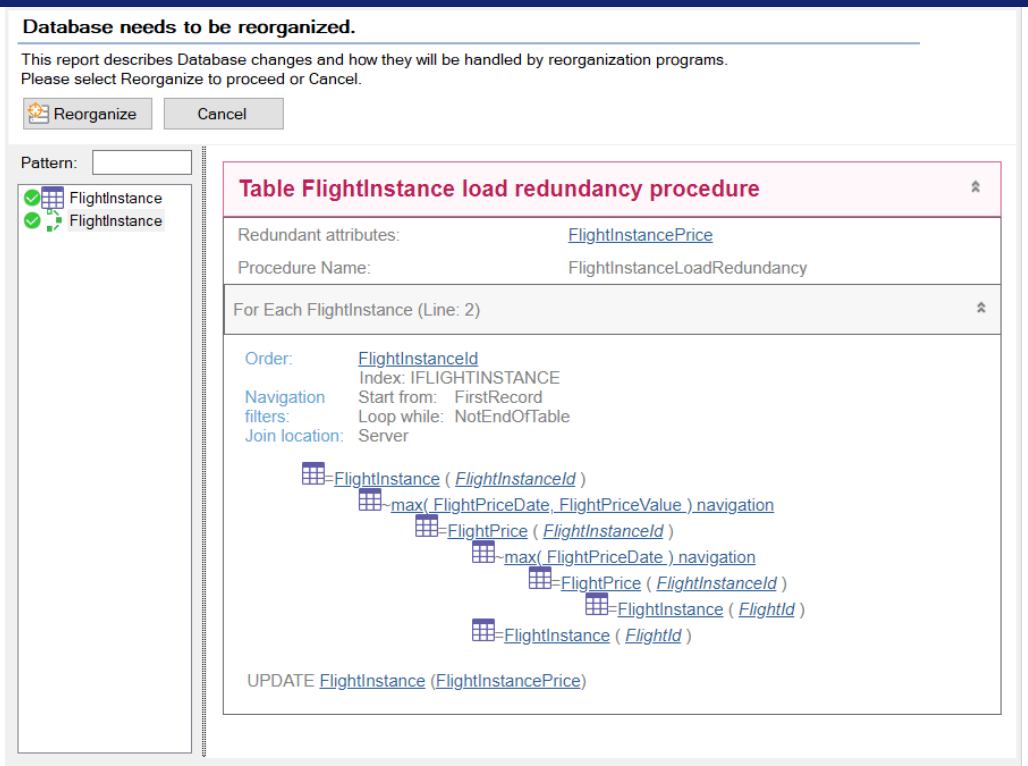
FlightInstancePrice: 1500

LIST

FlightPriceUpdateRedundancy?

Pero, ¿y si hacemos a la fórmula max redundante? Esperaríamos que se creara un procedimiento de actualización de la redundancia para la tabla FlightPrice, de modo que cuando se modifique el valor de FlightPriceValue a través de la transacción Flight, se acceda a la tabla FlightInstance buscando qué registros estarían afectados por el cambio para modificar el valor de FlightInstancePrice. Pero cuando uno intenta pensar cuáles serían esos registros que estarían afectados vemos lo dificultoso que resulta saberlo.

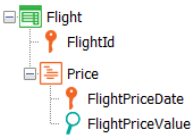
GeneXus, por tanto, no creará en este caso ese programa de mantenimiento de la redundancia. Lo vemos claramente cuando definimos el atributo como redundante y vemos el informe de reorg.

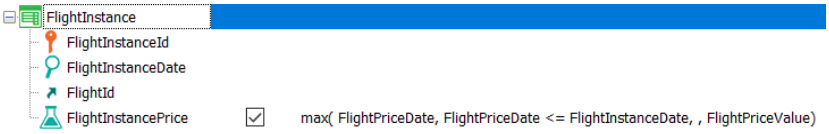


Solamente creará el procedimiento de carga de la redundancia, pero no el de actualización.

Por tanto la recomendación es siempre inspeccionar este informe de análisis de impacto para saber cuáles fórmulas redundantes van a ser mantenidas y cuáles no.

Formulas redundancy: limitations





FlightId	FlightPriceDate	FlighPriceValue
1	01/01/2022	800
1	02/02/2022	1500
1	04/04/2022	950

FlightId: 1

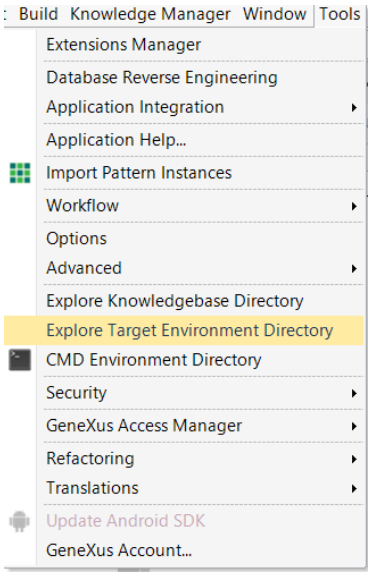
FlightInstanceDate: 03/03/2022

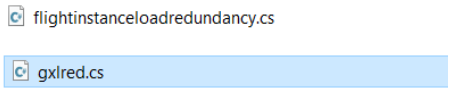
FlightInstancePrice: 1500

LIST

Como observación lateral: por supuesto que si el usuario ingresa a la transacción FlightInstance y modifica el valor de FlightInstanceDate allí sí se mantendrá actualizado el atributo redundante, dado que dentro de la transacción la fórmula se disparará normalmente, y se almacenará su valor. El problema surge cuando un atributo involucrado en el cálculo de la fórmula se modifica desde otra transacción, no la de la fórmula. En este caso, desde la transacción Flight.

Rebuild redundancy






```
1  Event 'Update redundant attributes'
2      call("gxlred")
3      Commit
4  Endevent
```

Si necesitáramos tener ese atributo redundante de todos modos, sabiendo que no se mantendrá actualizada automáticamente la redundancia desde Flight, lo que siempre podemos hacer es invocar al procedimiento de carga de las redundancias creado por GeneXus.

Si vamos a buscar los archivos en el directorio del environment, encontraremos por cada tabla con atributos redundantes el procedimiento de carga de las redundancias de la tabla. Ese que termina con LoadRedundancy, nombre de tabla load redundancy. En nuestro último caso el flightinstanceloadredundancy, que podemos invocar como una llamada a un procedimiento externo.

También, aunque no aparezca listado en el reporte de análisis de impacto, GeneXus crea un procedimiento de nombre gxlred, por GeneXus Load Redundancy que lo que hace es invocar a cada uno de los procedimientos LoadRedundancy de cada tabla con redundancias.
Por lo que si en un momento queremos asegurarnos de que todas las redundancias definidas en la KB se actualicen, podemos por ejemplo invocar a este programa desde un evento.

Other limitations in defining redundancies

- To define as redundant a formula that is already a formula, we must first define that other formula as redundant
- Redundant aggregate formulas that add more than one level of nested redundant formulas will not be properly maintained
- Subtypes cannot be defined redundant
- To change the definition of a formula that is redundant, you must first remove the redundancy

Hay algunas otras limitaciones para definir atributos redundantes, que debemos considerar.

Aquí dejamos algunas listadas que se podrán consultar en más detalle en el wiki.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications