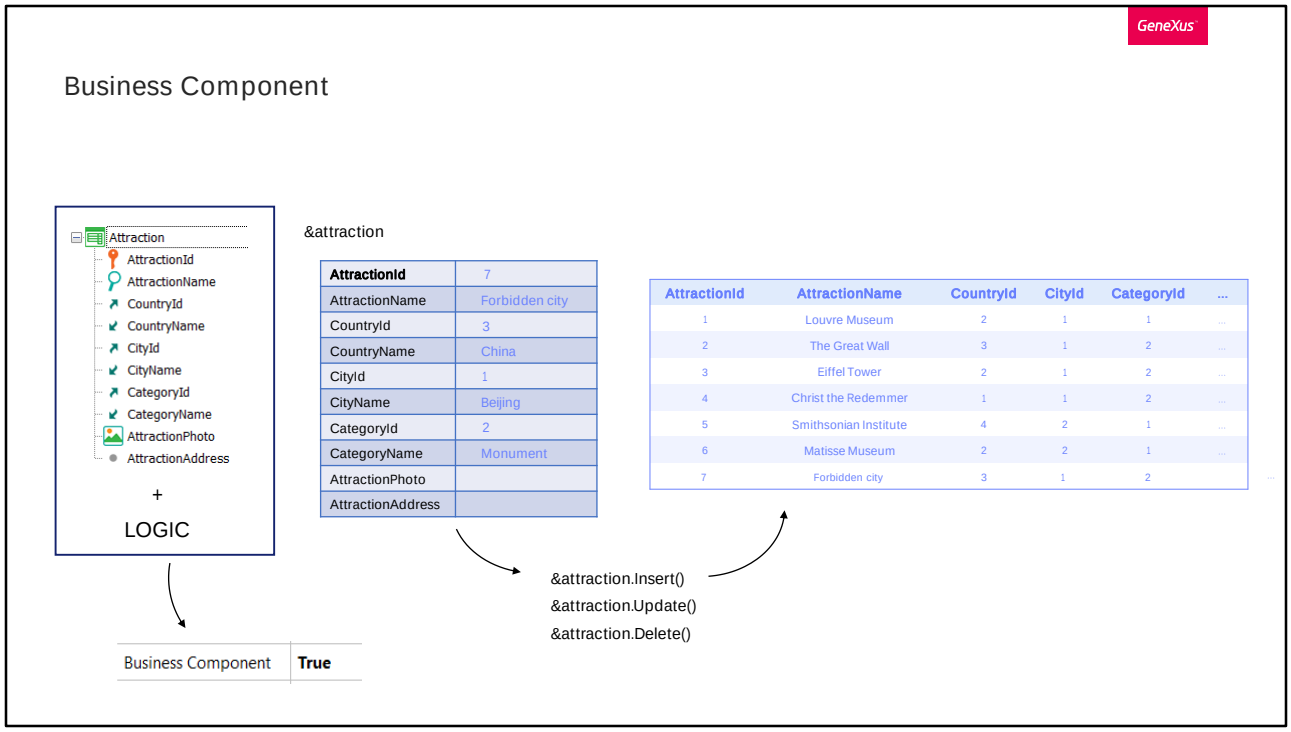
Actualización de la base de datos

Usando Business Components

GeneXus™



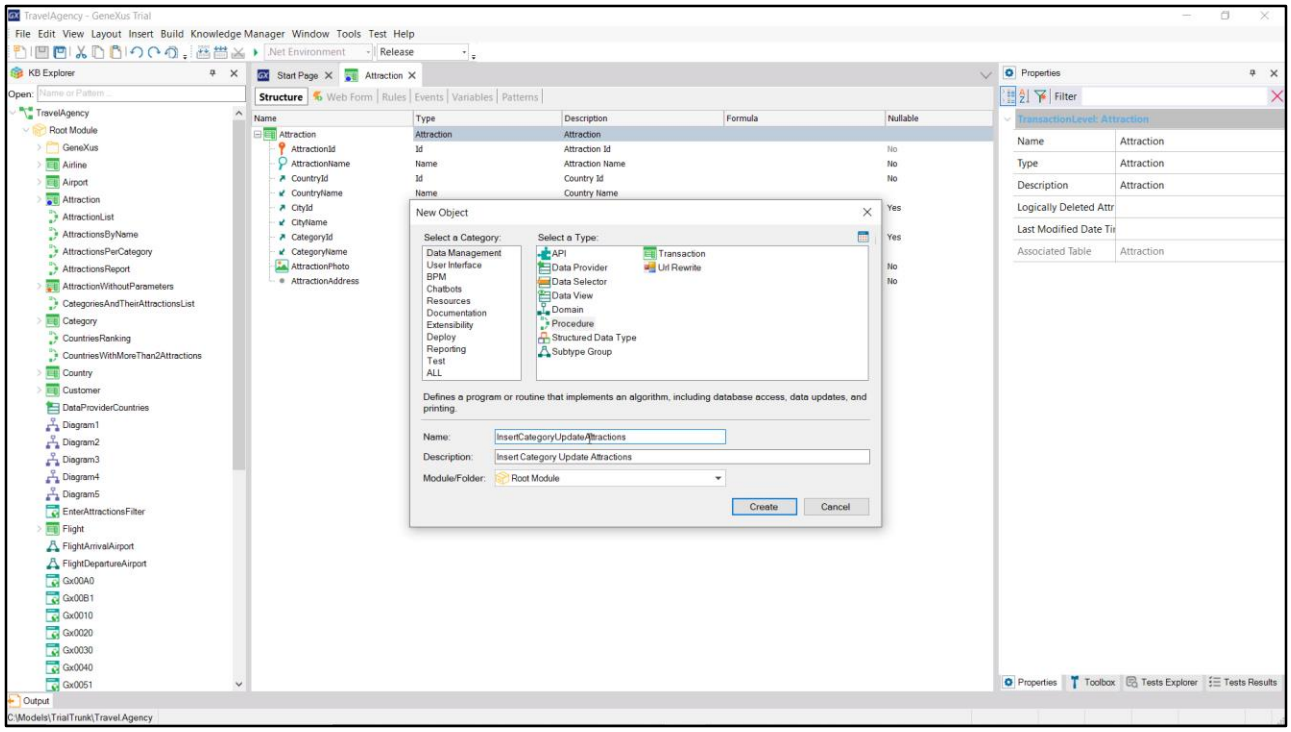
En el video anterior hicimos un recorrido detenido por lugares conocidos para allanarnos el camino en la comprensión y el uso de los business components.

De la estructura de la transacción con su lógica (y por lógica entendemos los controles de duplicados –no sólo de clave primaria, sino también de claves candidatas–, de integridad referencial, la mayoría de sus reglas y algunos de sus eventos), se obtiene una suerte de tipo de datos similar a un SDT, pero mucho más potente.

Luego, alcanzará con definir en casi cualquier programa una variable de ese tipo de datos y manipularla, que es lo que veremos a continuación.

Esa variable, en su estructura, se manejará de modo similar a un SDT. Pero además ofrecerá métodos para realizar lo que es específico de una transacción: cargar de la base de datos, insertar, modificar, eliminar; todo eso ejecutando la lógica de la transacción, y luego obtener los mensajes generados y los resultados de éxito o fracaso.

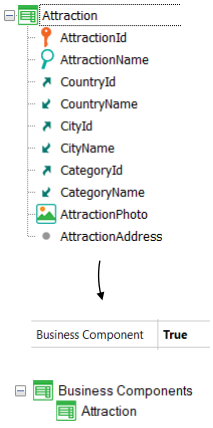
La forma de obtener ese tipo de datos en la base de conocimiento es simplemente prendiendo la propiedad Business Component de la transacción.

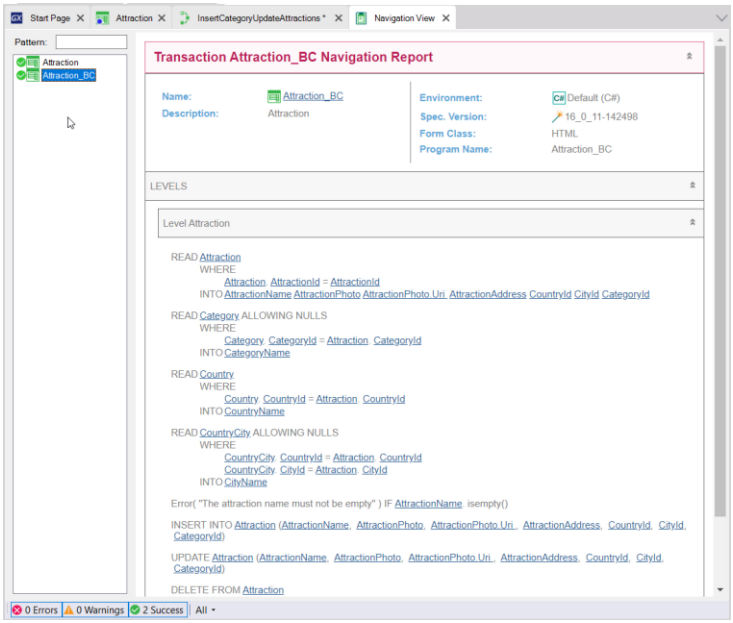


Vamos a empezar por crear un procedimiento simple que iremos modificando para implementar un requerimiento más completo más adelante. No le prestemos ahora atención al nombre.
Empecemos por algo bien simple: supongamos que queremos insertar una nueva atracción turística.

GeneXus

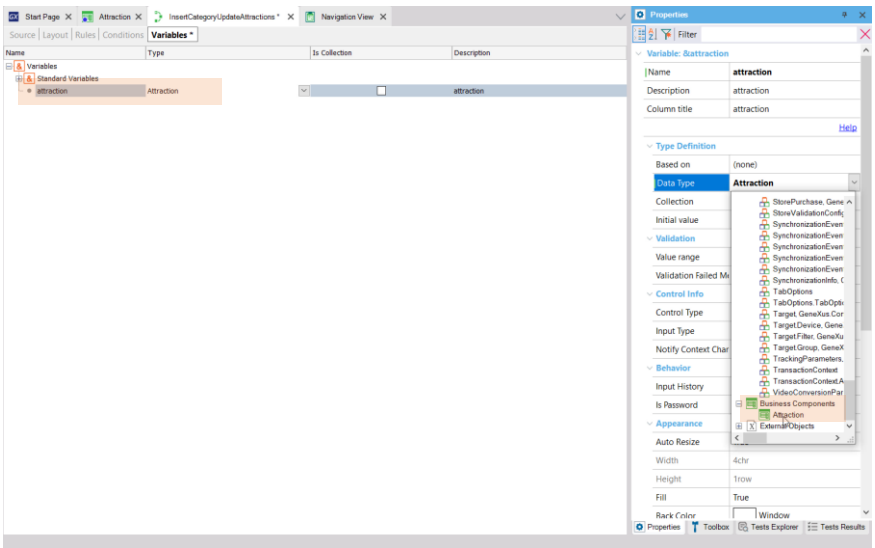
Insert through BC



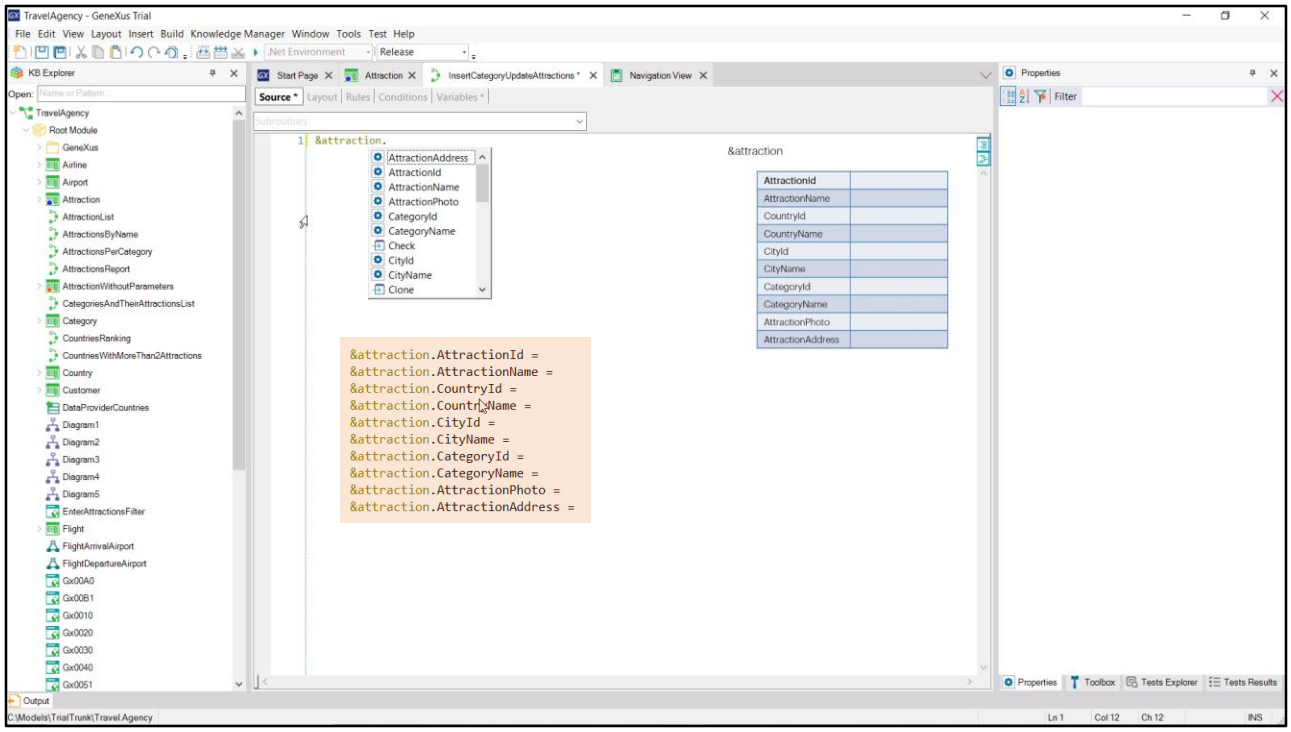


Lo primero que debemos hacer es crear el Business Component de la transacción Attraction, para que pueda ser utilizado en cualquier objeto de la base de conocimiento. Por tanto vamos a la transacción y entre sus propiedades ubicamos la de nombre Business Component. Como vemos, por defecto está en False. La cambiamos a True. En principio no vemos ningún efecto. Sin embargo, si grabamos... vemos esto en el listado de navegación.

Insert through BC

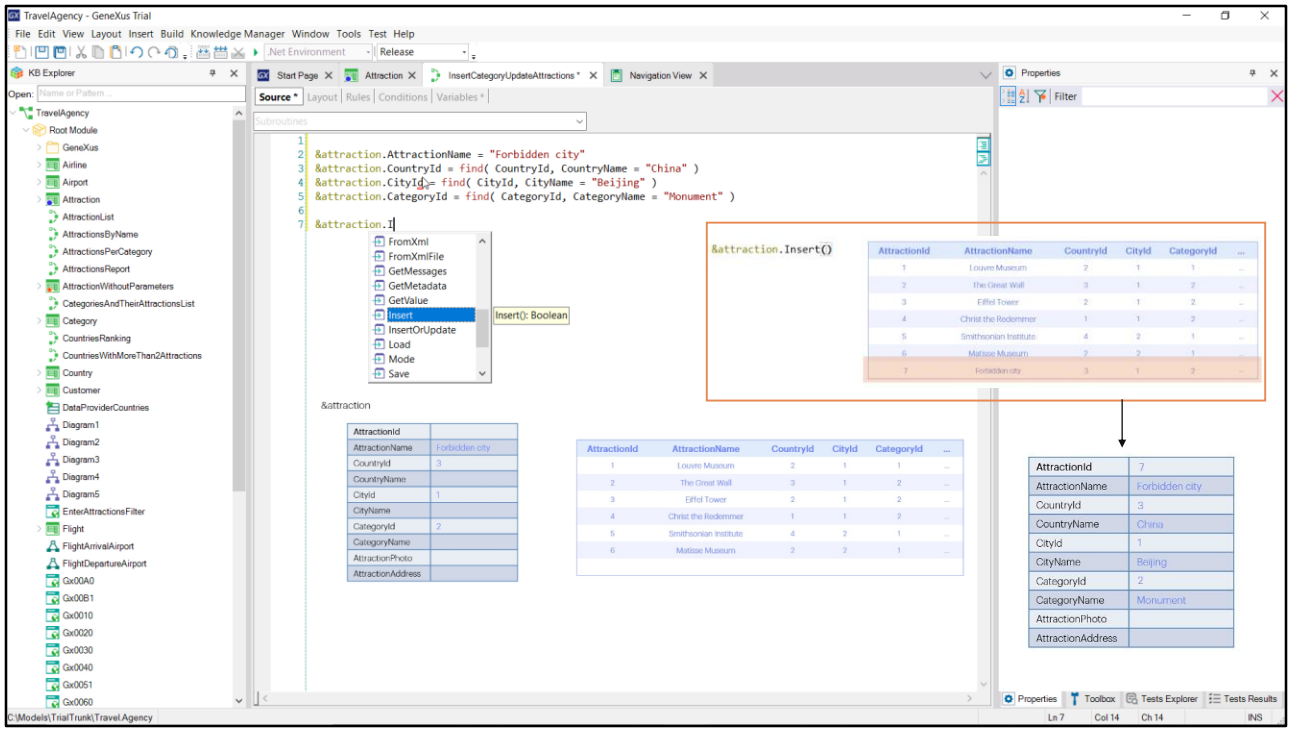


Si ahora vamos al procedimiento y definimos en él una variable de nombre attraction, vemos que automáticamente, por haber elegido ese nombre que coincide con el de la transacción, le eligió automáticamente un tipo de datos de ese nombre. ¿Qué tipo de datos es este?
Si vamos a las propiedades de la variable y abrimos el combo.... vemos que tenemos un grupo Business Component que por ahora solo ofrece un valor: no casualmente es Attraction. Aquí es donde vemos el efecto de haber prendido la propiedad de la transacción. Quedó creado en la KB, disponible, el tipo de datos Business Component Attraction, que se le puede asignar de ahora en más a cualquier variable.



Al haber definido esta variable en el procedimiento, automáticamente se le reserva el espacio de memoria para poder cargar como un SDT todos sus elementos.

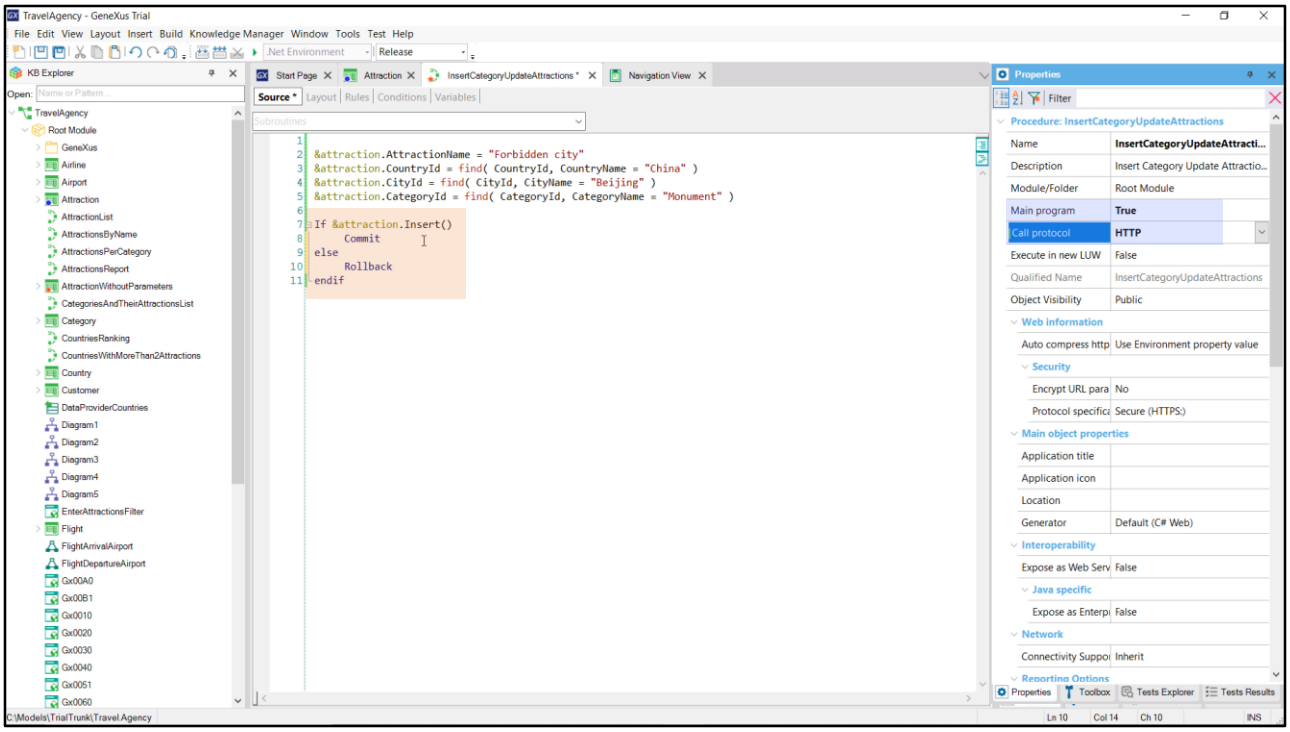
Así, si insertamos la variable y digitamos punto, vemos que en esta ventana nos muestra los nombres de todos los atributos de la transacción, entre otras cosas, para que podamos utilizarlos en la estructura, por ejemplo asignándoles valor. Así, podríamos hacer...



Si queremos insertar la atracción turística la ciudad prohibida, como vimos antes cuando usamos la transacción, tendremos que completar los datos.
Al identificador no necesitamos asignarle valor, dado que será auto-numerado.
El nombre de atracción será...que sabemos está en China, que tiene el identificador 3, pero ¿y si nos equivocamos? Mejor buscar ese identificador con la fórmula find, porque de lo que sí estamos seguros es del nombre de país, China.
Ese nombre es un atributo inferido en la transacción, así que no tenemos que asignarle valor aquí para insertar la atracción.
La ciudad sabemos que es Beijing, así que buscamos su identificador y lo asignamos al elemento del business component y también quitamos el elemento inferido.
Asignamos valor para el id de categoría, buscando la de nombre "Monument".
Y a la foto y dirección no les asignaremos valor.

Tenemos entonces la variable cargada con los datos de la atracción que queremos insertar en la tabla. Nos resta dar la orden de insertar. Para ello contamos con el método Insert de la variable Business component, que, como vemos, devolverá un booleano: True si pudo insertar, False en caso

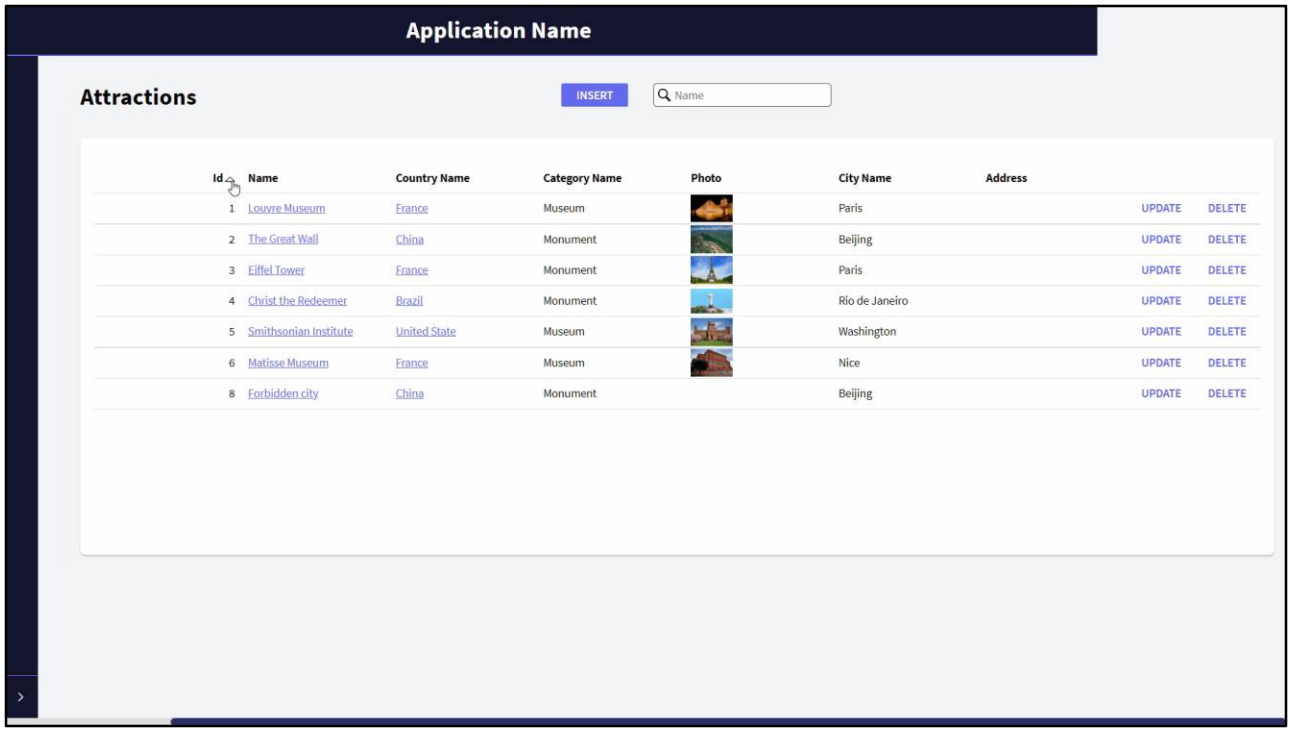
contrario. Podemos invocarla sin más e intentará insertar un nuevo registro exactamente igual que como lo hace la transacción, es decir, ejecutando todas las reglas y validaciones. Si lo consigue, entonces la variable nos quedará cargada con el id que le dio la base de datos y todos los atributos propios e inferidos.



Sin embargo, debemos saber que si bien el registro ya está en la tabla, está en un estado inestable. Si llegara a haber un apagón o caerse el sistema por cualquier otra razón, cuando la base de datos se reestablezca ese registro ya no estará. Es que las bases de datos nos permiten ir insertando, modificando, eliminando registros como si fuera a un nivel lógico y cuando queramos, en el momento en que nos parezca conveniente, debemos indicarle que todas esas operaciones que estuvimos realizando ya se pueden dar por buenas. Esto se hace con el comando Commit. Por tanto, si la inserción fue exitosa, commiteamos. Le llamamos así a la acción de realizar un commit sobre la base de datos, es decir, indicarle que realmente impacte todas esas operaciones, que las deje fijas, permanentes, en la base de datos. Cuando se ejecuta un Commit todas las operaciones que se hayan realizado entre el commit anterior y el actual serán dadas por buenas.

Así como tenemos el Commit para dar por buenas todas esas operaciones, tenemos el opuesto, el Rollback, para deshacer todo lo que se hubiera hecho después del último Commit. Aquí podríamos especificar un Rollback si la operación no fue exitosa... pero no tendrá mucho sentido, dado que si la operación no fue exitosa, podemos suponer que el registro no llegó a insertarse siquiera, por tanto no habrá nada que deshacer.

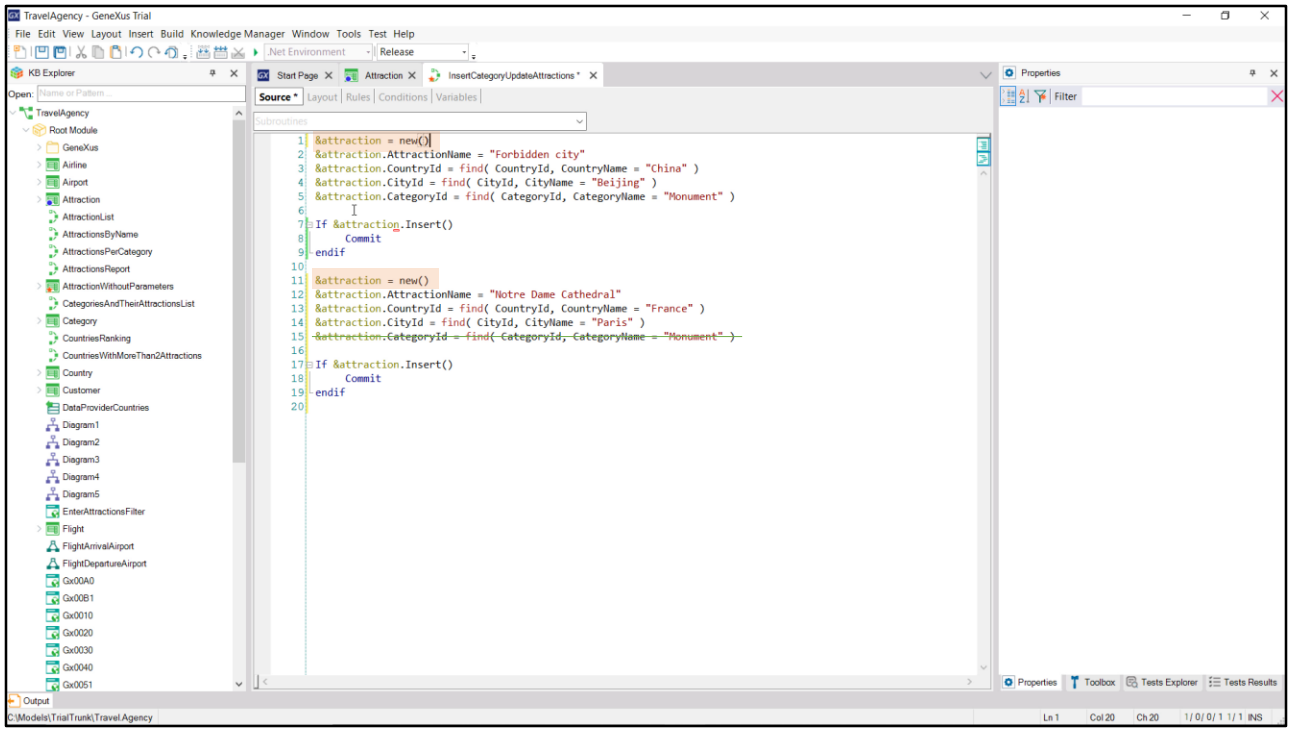
Para ejecutar este procedimiento rápidamente desde el Developer Menu lo especificamos como objeto principal y con protocolo de invocación HTTP.
Presionemos F5.



Si vamos al trabajar con atracciones, veremos que ya teníamos la atracción 7 para la ciudad prohibida, que habíamos insertado antes a través de la transacción. Eliminémosla (el valor 7 de identificador quedará perdido, no se volverá a dar a la siguiente atracción que se ingrese, que quedará con 8, como veremos ahora, si vamos a ejecutar nuestro procedimiento).

Como no tiene ninguna salida, no vemos nada en el navegador, pero si volvemos al trabajar con atracciones... vemos que insertó efectivamente a la ciudad prohibida.

Borrémosla nuevamente, para volver a ejecutar el procedimiento, pero al que le agregaremos otra atracción más, además de esa. Por ejemplo, la catedral de Notre Dame.

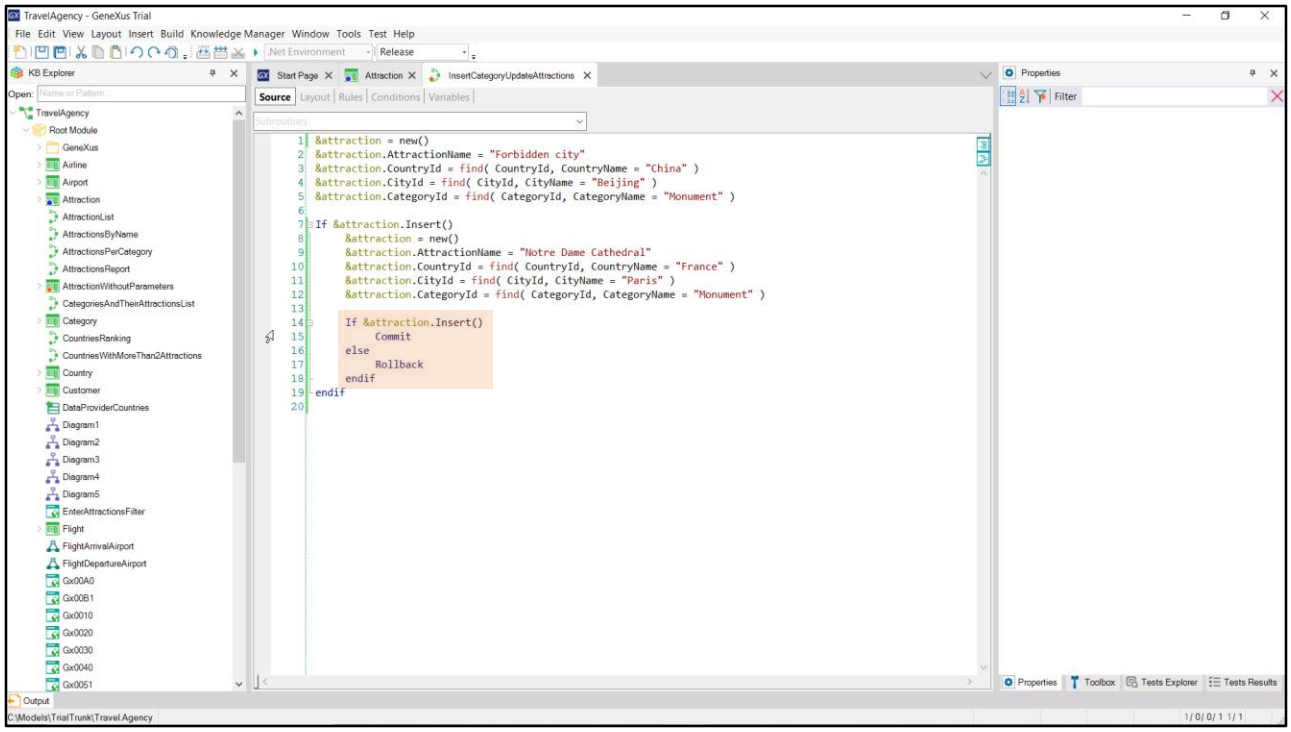


Debemos insertarla entonces, aquí. Ya no necesitamos la información que tenía la variable. De hecho, necesitamos, al menos, limpiarla, para asegurarnos de que no quede nada viejo cuando carguemos los datos de la catedral. Una manera es pedir memoria nueva para la variable, y esto se consigue así...

Ahora sí. Y luego otra vez pedimos que inserte.

Pedir memoria nueva es importante, dado que si, por ejemplo, hubiésemos omitido asignar valor al identificador de categoría, porque la queremos dejar vacía para la Catedral de Notre Dame, habría quedado cargado con el valor que asignamos más arriba. Por tanto, siempre es buena práctica antes de completar los datos de un business component que vamos a insertar, pedir memoria para él. Incluso aquí arriba.

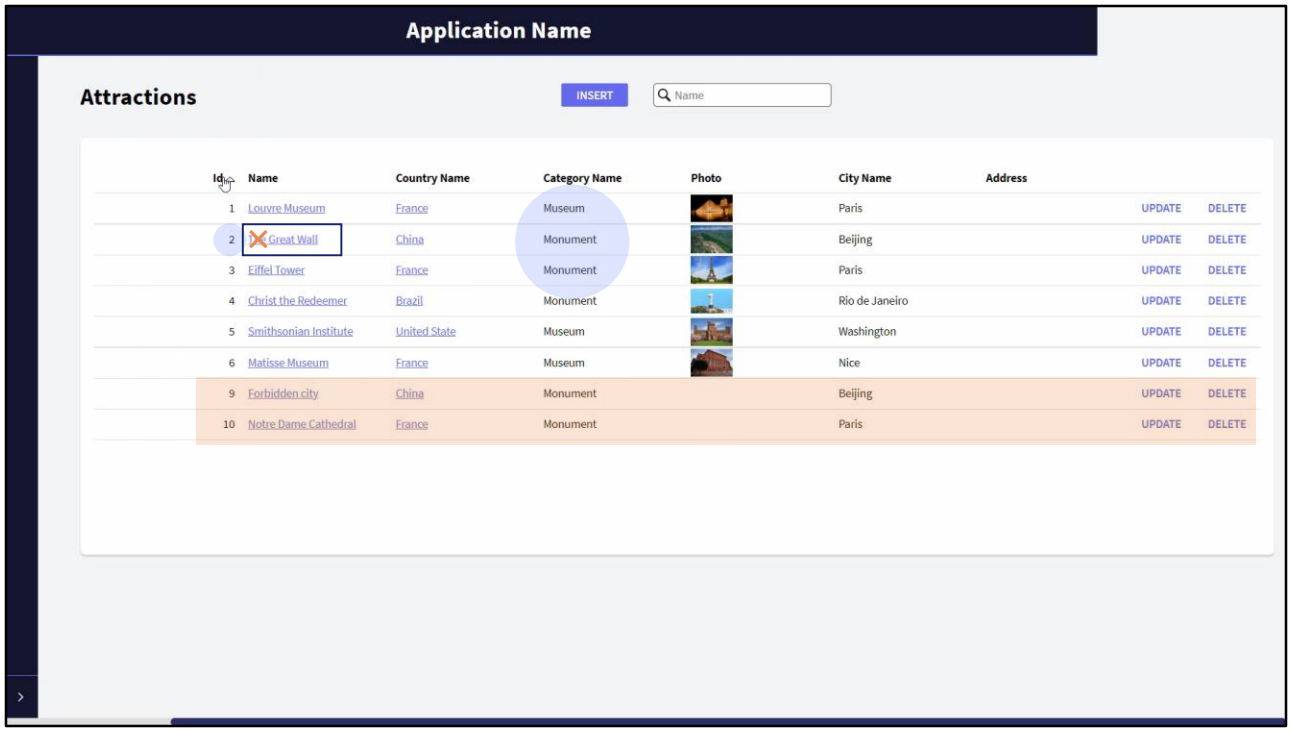
Por último, antes de ejecutar, veamos que en este caso insertamos la primera atracción y commiteamos si fue exitosa esa inserción. Y hacemos lo mismo con la segunda, y commiteamos.



Pero podríamos querer commitear una sola vez, después de haber insertado las dos atracciones. O incluso podríamos hacer el intento de insertar la segunda atracción solamente si la primera fue exitosa. Por ejemplo, en ese caso, sería de este modo.

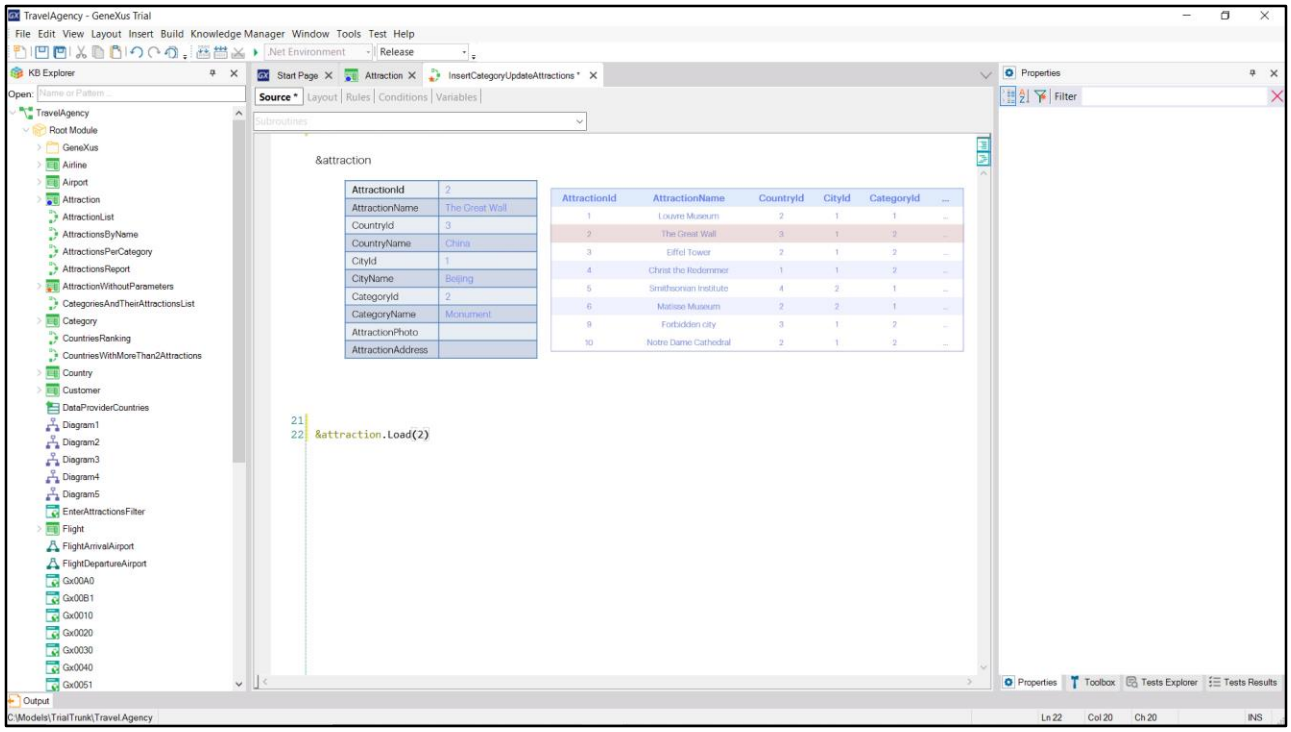
Aquí solamente commiteamos cuando la primera inserción fue exitosa y la segunda también. Por lo que nos podría pasar que la primera haya sido exitosa y la segunda no. Entonces nos quedará grabado el registro de la primera en la base de datos, pero no quedará commiteado. Por tanto, tal vez en ese caso, cuando la segunda no es exitosa, no queremos que el primer registro quede en la base de datos; y ese es un caso donde colocaríamos el comando Rollback.

Ejecutemos para probar. F5.



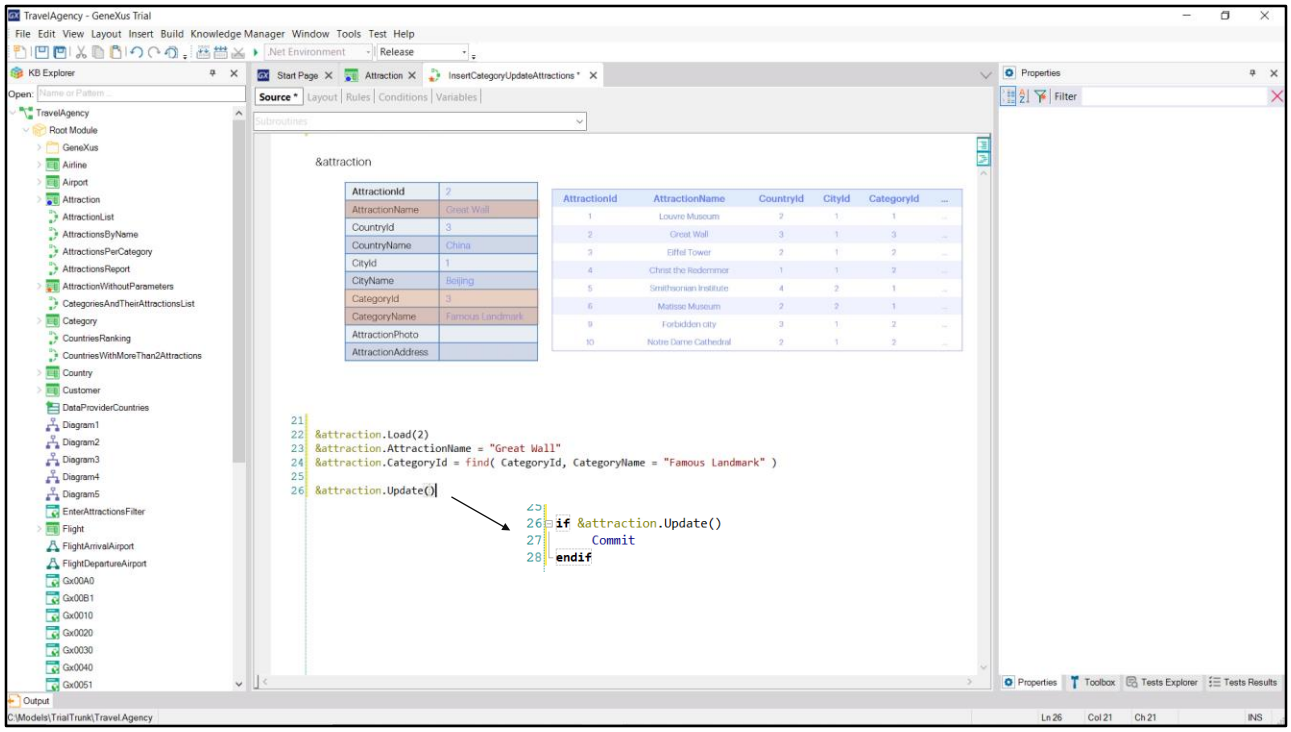
Veamos las atracciones. Ejecutemos el procedimiento. Y volvamos a ver el work with de atracciones. Ahora tenemos la 9 y la 10.

¿Y si ahora quisiéramos cambiar la categoría de la muralla china para que no sea más “Monument” y pase a ser “Famous Landmark”, y por ejemplo quisiéramos también cambiarle el nombre, para quitarle este “The”?



Dejemos el código anterior comentado para que no vuelva a insertar otra vez estas dos atracciones (si sus identificadores no fueran auto-numerados, entonces las inserciones fallarían por clave duplicada, pero no es el caso)

Necesitamos hacer un Update, como lo haríamos con la transacción. Primero que nada debemos cargar la variable con los valores de la atracción que queremos modificar. Sabemos que su id es el 2. Entonces lo que hacemos es invocar al método Load de la variable, pasándole por parámetro la clave, 2.
Al ejecutar esto, se irá a la tabla a buscar si existe un registro 2, y en caso afirmativo, se cargará la variable BC con todos los valores: los propios, inferidos y fórmulas, tal como sucede al ejecutar la transacción.

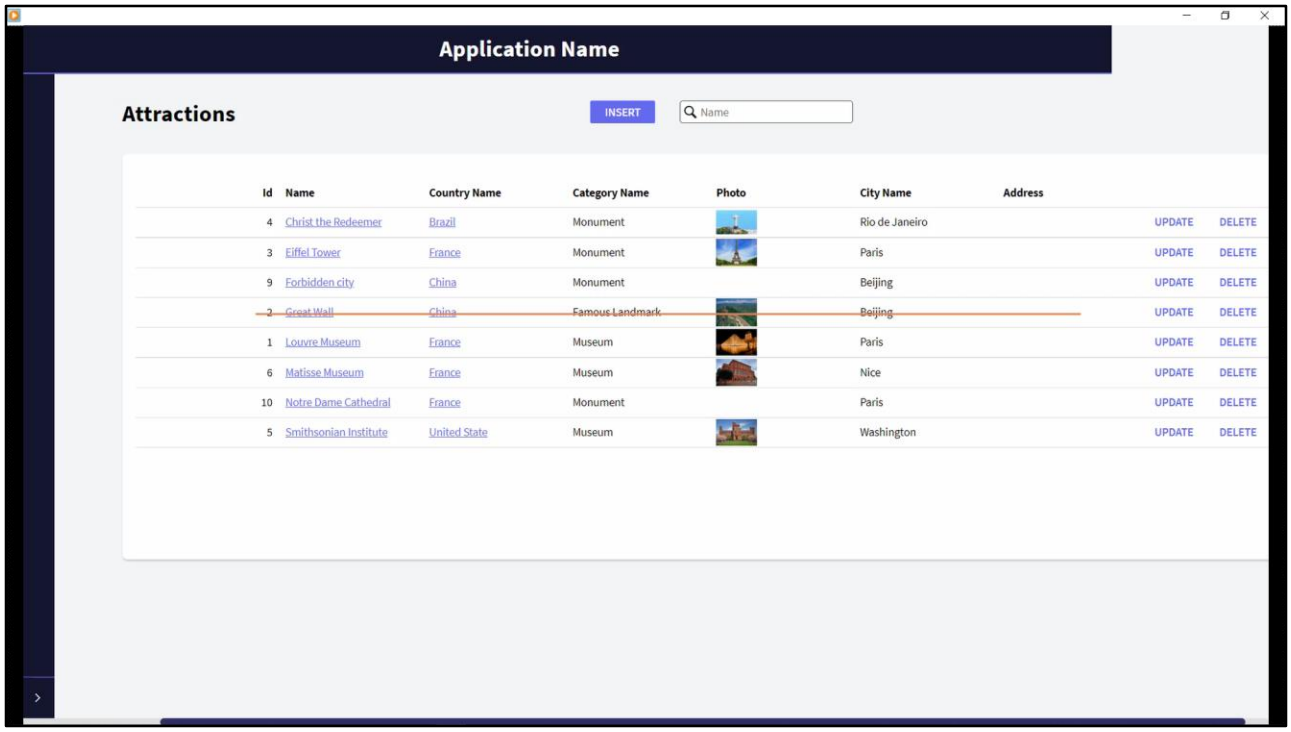


De todos estos valores, necesitamos cambiar dos: el nombre de la atracción... Y su categoría...

Nos resta únicamente pedirle a la variable que actualice esos datos en la tabla. Y para ello, contamos con el método Update.
Al invocarlo, exactamente igual que como sucede en la transacción, se ejecutarán todas las reglas, fórmulas, controles de unicidad de claves candidatas e integridad referencial; y si todo anduvo bien, se actualizará el registro en la tabla. Y, por supuesto, la variable quedará cargada con los datos actuales.
Otra vez, podríamos preguntar por el resultado de la actualización para, por ejemplo, ya realizar un Commit.

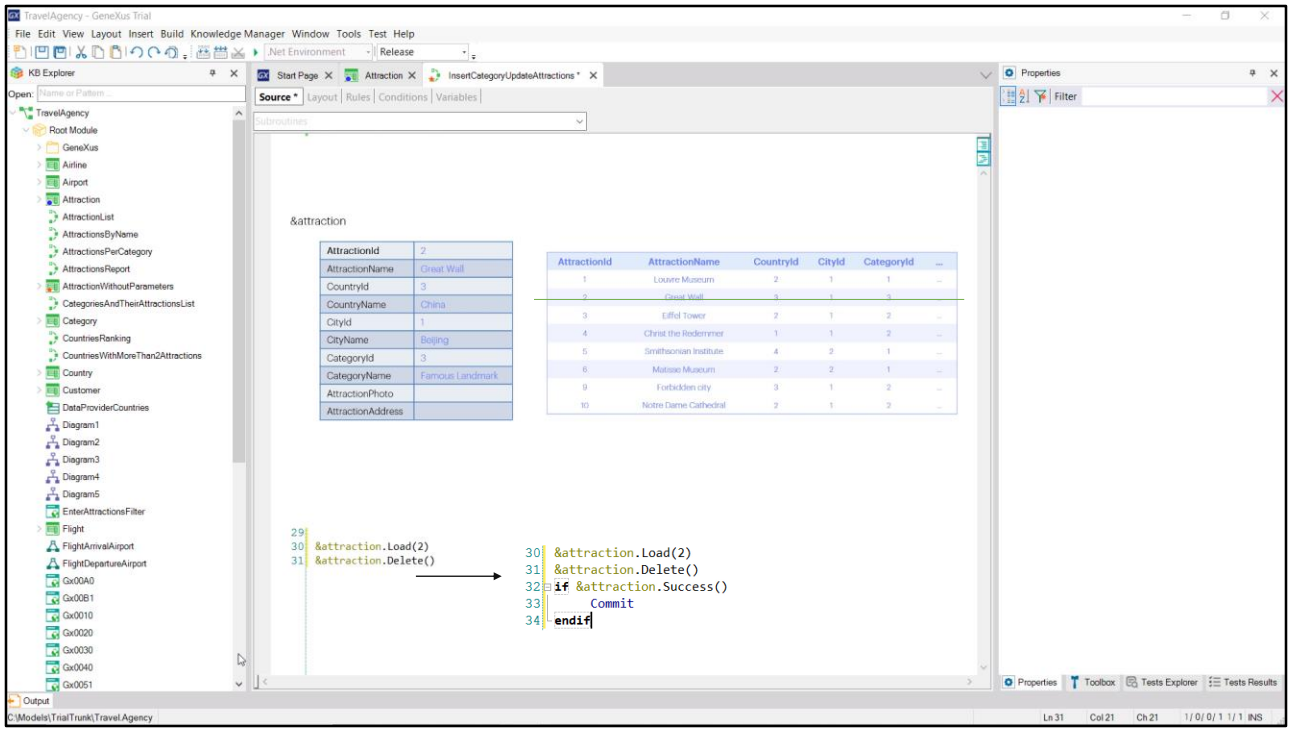
Ejecutemos para probar.

Ejecutamos el procedimiento, y vamos a ver las atracciones. Vemos que quedó modificada.



Para completar las operaciones, ¿y si ahora quisiéramos eliminar una atracción por código?

Por ejemplo, eliminar la gran muralla.



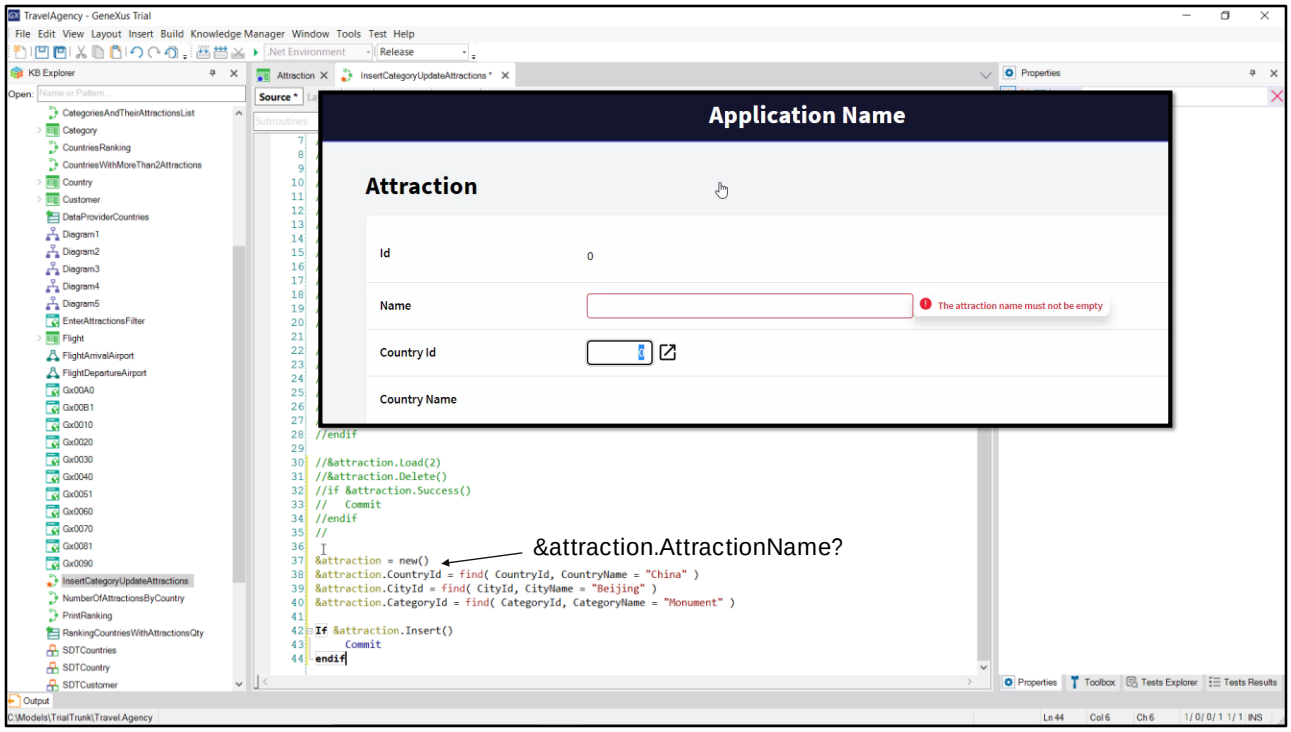
Al igual que como hacemos en la transacción, primero tenemos que cargar la estructura de la variable con la atracción 2, y luego simplemente dar la orden de eliminar.

Esto lo hacemos con el método Delete. No nos cansamos de repetirlo: al ejecutarse este método se ejecutará toda la lógica de la transacción en modo Delete, incluyendo los controles de integridad referencial. Por tanto, por ejemplo, si hubiera información relacionada, imaginemos que hubiera city tours que contuvieran a esta atracción que deseamos eliminar, entre sus datos, entonces el control de integridad referencial que realiza la transacción –y por tanto también el Business Component–, lo impedirá. Se producirá un error y el registro no se eliminará de la tabla. En cambio, si no hay ningún error el registro será eliminado. La variable de todos modos quedará cargada con los datos, por si queremos hacer algo con ellos.

El método Delete no devuelve el resultado de su operación, por lo que para saber si fue o no exitosa, deberemos consultar en forma explícita, con el método Success (su opuesto es el método Fail). Y por ejemplo, allí realizar el Commit de la eliminación, es decir, dejar permanentemente eliminado el registro. Este método puede ser utilizado después de cualquier operación, por ejemplo, luego del Load, para saber si encontró

en la tabla el registro pedido.

Probemos lo visto. Ejecutamos el procedimiento y vemos que efectivamente eliminó el registro 2.



¿Y qué pasará si ahora volvemos a ejecutar el procedimiento? No pasó nada. ¿Por qué?

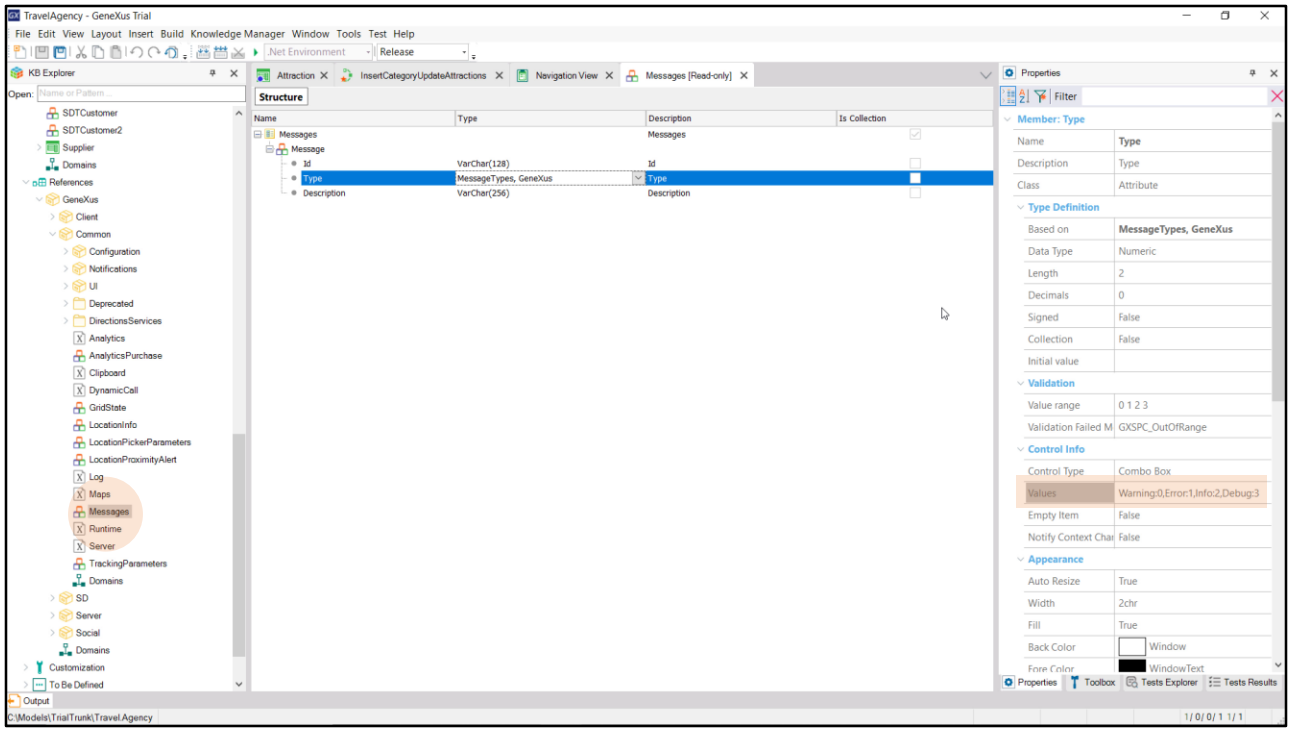
En esta segunda ejecución la atracción 2 ya no existe. Por tanto no habrá podido ni cargarla ni eliminarla.

¿Y si ahora probamos insertar nuevamente la muralla china, pero olvidamos colocar el nombre? Probemos.

Ejecuto el procedimiento. Vamos al Work with de Attraction, y no vemos la nueva atracción insertada. ¡Claro, olvidamos poner el nombre y teníamos una regla de error que se está disparando y está impidiendo que se inserte ese registro en la tabla!

Si esto lo hubiéramos realizado a través de la transacción, al usuario le hubiera aparecido el mensaje que especificamos en la regla de error.

¿Dónde quedó ese mensaje al intentar la inserción a través del business component?



La transacción tiene una pantalla para mostrar los errores al usuario, pero el Business component no. Por tanto el manejo de errores también queda a cargo del desarrollador.

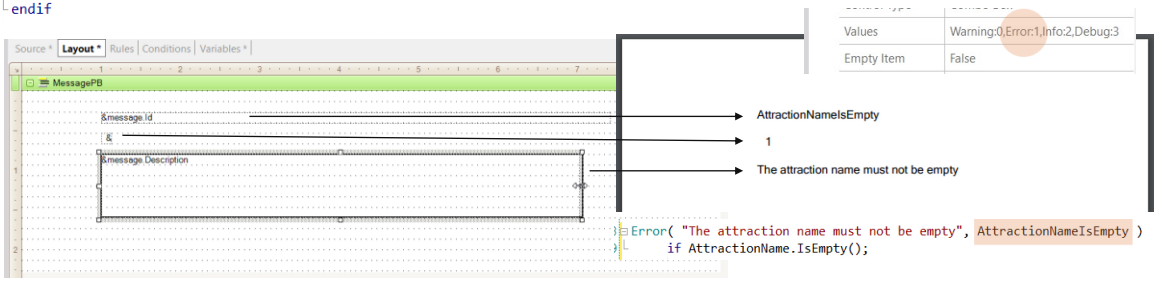
¿Cómo los obtenemos?

En el KB Explorer contamos con un grupo References, del que ya hablaremos en otra oportunidad, pero que siempre contendrá un módulo GeneXus, con funcionalidades ya implementadas para poder utilizar en nuestra KB. En particular allí encontraremos un SDT de nombre Messages. Podemos ver que es una colección de ítems conformados por un Id, un tipo y una descripción. El tipo también viene predefinido en el módulo. Se trata de un enumerado que señala el tipo de mensaje del que se tratará en cada caso: advertencia, error, información, debug.

GeneXus

Error handling

```
&attraction = new()
&attraction.CountryId = find( CountryId, CountryName = "China" )
&attraction.CityId = find( CityId, CityName = "Beijing" )
&attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )

If &attraction.Insert()
    Commit
else
    &messages = &attraction.GetMessages()
    for &message in &messages
        print MessagePB
    endfor
endif
```



Cada vez que se ejecute una operación sobre una variable Business Component, se puede obtener la colección de mensajes que esa ejecución haya producido, con el método GetMessages. Tendremos que definir una variable del tipo de datos devuelto por ese método, para poder manipular su resultado.

Por ejemplo, por ahora solo por practicidad mostremos el resultado en un pdf. Recorreremos la colección de mensajes generados con el comando for in que permite recorrer, entre otras cosas, colecciones. Definimos para ello una variable del tipo de datos de los ítems de la colección de mensajes. Y para cada mensaje de esa colección, lo imprimimos en la salida.

Especifiquemos la regla outputfile, y probemos.

Ejecutamos el procedimiento. Y aquí vemos su salida...

Solamente se obtuvo un ítem en la colección de mensajes, con id vacío, tipo 1 –que es error–, y como descripción exactamente lo que especificamos en la regla de error de la transacción.

Si vamos a ver la sintaxis de la regla de error, vemos que permite un

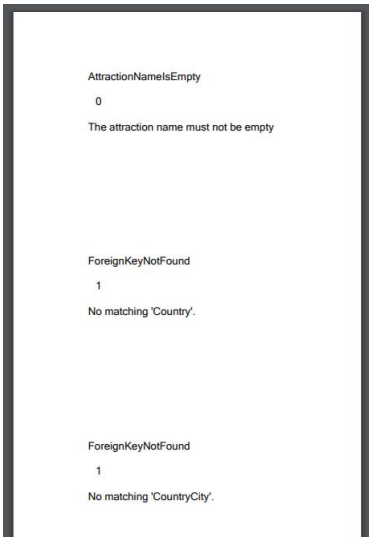
segundo parámetro, opcional. Ese parámetro permitirá especificar el Id del error para el mensaje del business component. Por ejemplo, le daremos este valor. Si ahora probamos... allí lo vemos.

GeneXus

Error handling

```
Msg( "The attraction name must not be empty", AttractionNameIsEmpty )
    if AttractionName.IsEmpty();
```

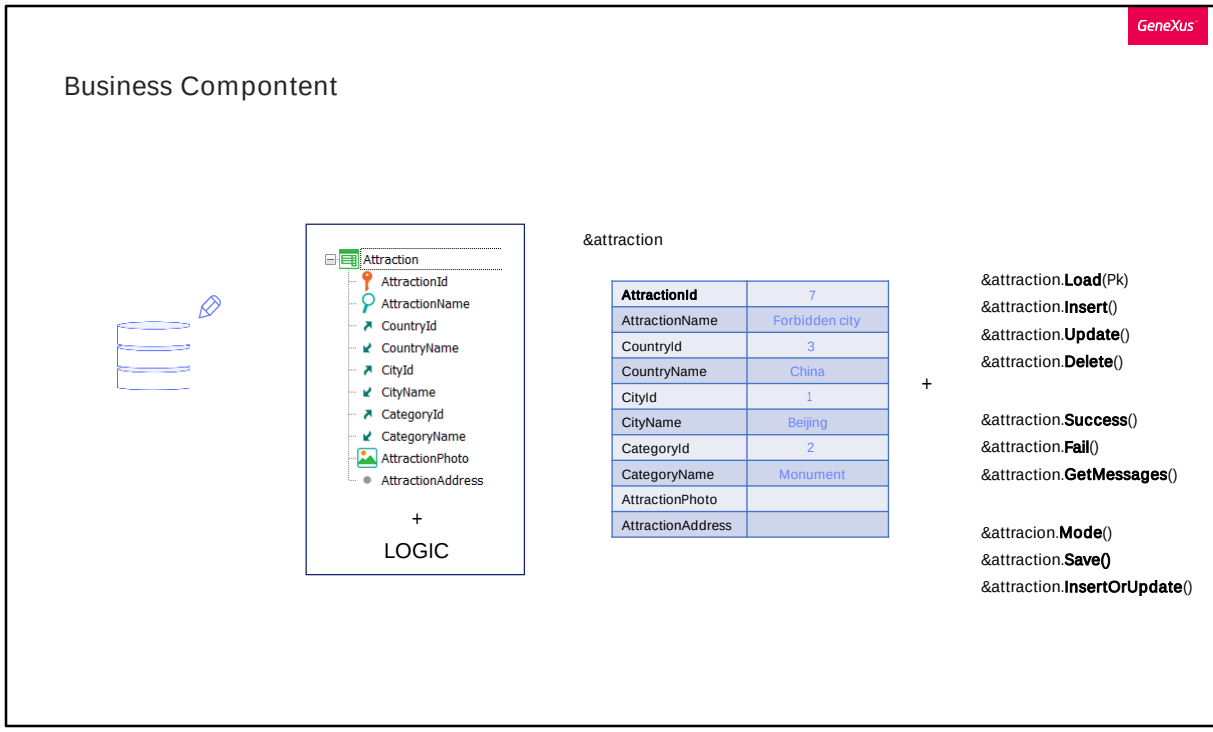
```
&attraction = new()
&attraction.CountryId = 22
//&attraction.CountryId = find( CountryId, CountryName = "China" )
&attraction.CityId = find( CityId, CityName = "Beijing" )
&attraction.CategoryId = find( CategoryId, CategoryName = "Monument" )

If &attraction.Insert()
    Commit
else
    &messages = &attraction.GetMessages()
    for &message in &messages
        print MessagePB
    endfor
endif
```



Hagamos una última prueba: modifiquemos la regla para que ya no sea de error, sino un mensaje.
Y además especifiquemos como CountryId un identificador de país inexistente, para ver cómo falla la inserción por control de integridad referencial. Probemos.

Al intentar insertar se produjeron tres mensajes: el primero es de tipo Warning, y no hubiera provocado por sí solo la falla de la inserción. En cambio el segundo y el tercero, sí, pues son de tipo Error. El Id interno es este, y el mensaje que ve el usuario de la transacción en la pantalla cuando falla la integridad por país, es, NO CASUALMENTE, No matching ‘Country’.

También arrojará un error de integridad referencial al intentar insertar la ciudad, que depende del país.



Con esto, podemos decir que hemos visto lo más relevante relacionado a la actualización de la base de datos con Business Components.

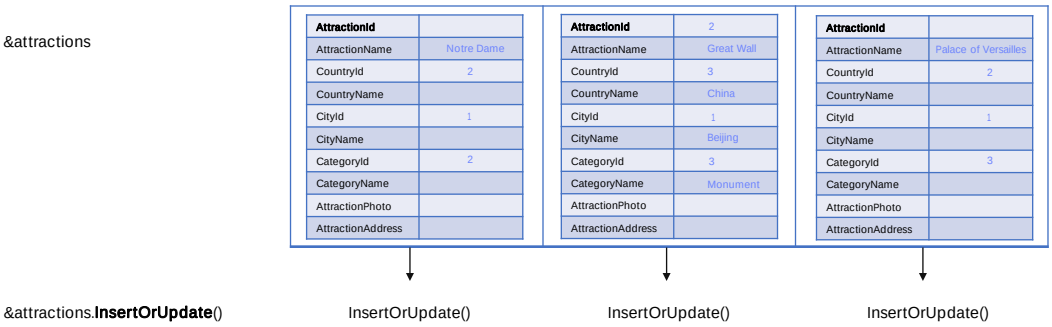
Así, vimos que a partir de la transacción es posible crear un tipo de datos que es como un SDT, pero que permite realizar operaciones sobre la base de datos a partir de métodos.
Estas ejecuciones conservan la lógica de la transacción. Si bien no entramos en ello, claramente no todas las reglas y eventos de la transacción serán incorporadas al business component. Las que tengan que ver con la interfaz obviamente no. Tampoco la regla Parm es tomada en cuenta.
Además de permitirnos estas operaciones sobre la base de datos, podemos consultar el resultado de la última operación realizada, así como obtener los mensajes producidos, en una colección.

Hay más métodos para estudiar. Por ejemplo, existe el método Mode que devuelve el modo en el que se encuentra el business component: si es Insert, Update, Delete.
Un método Save que, dependiendo del modo de la variable intentará Insertar si el modo es Insert o actualizar si el modo es Update.
Y el método InsertOrUpdate que siempre intentará insertar, y si falla por clave duplicada, entonces intentará actualizar.

Los métodos Insert, Update, Delete e InsertOrUpdate pueden ser aplicados también a colecciones de Business Components.

GeneXus

Business Compontent

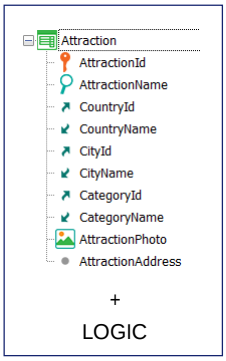


Así, si la variable &attractions fuera una colección del Business Component Attraction, –en este ejemplo una colección de tres items Attraction–, aplicarle, por ejemplo, el método InsertOrUpdate es equivalente a recorrer la colección e irle aplicando el método individualmente a cada ítem.

El resultado será True si los resultados individuales fueron todos True.

GeneXus

Business Compontent



Only in Procedures?

&attraction

AttractionId	7
AttractionName	Forbidden city
CountryId	3
CountryName	China
CityId	1
CityName	Beijing
CategoryId	2
CategoryName	Monument
AttractionPhoto	
AttractionAddress	

+

&attraction.**Load**(Pk)
&attraction.**Insert**()
&attraction.**Update**()
&attraction.**Delete**()

&attraction.**Success**()
&attraction.**Fail**()
&attraction.**GetMessages**()

&attracion.Mode()
&attraction.**Save()**
&attraction.**InsertOrUpdate**()

Lo último importante que mencionaremos es que una variable de tipo Business Component puede utilizarse en cualquier objeto que tenga alguna sección de código, no solo procedimientos. Esto significa que podemos actualizar la base de datos por ejemplo desde el evento de un panel que pide o muestra datos al usuarios, como veremos.

También se podrá hacer desde eventos en transacciones, aunque en ese caso existen alguna limitaciones que aquí no veremos.

GeneXus

More

- InsertCategoryUpdateAttractions proc

Insert "Tourist site" category
Update Attractions of Beijing

- CategoriesAndAttractions web panel

Do:
Insert "Tourist site" category
Update Attractions of Beijing

Undo:
Delete "Tourist site" category
Update Attractions of Beijing

- MassiveInsertRemove panel

Remove Data:
From Category
From Attraction

En el siguiente video aplicaremos en un ejemplo todo lo visto aquí. Podría salteárselo, salvo la parte final.

Allí:

Re-implementaremos el procedimiento que aquí creamos, para que inserte una nueva categoría de atracción turística: "Tourist Site" y modificaremos todas las atracciones de Beijing que tenían la categoría "monument" pasando a asignarle la nueva.

Veremos cómo hacer esto mismo a través desde un panel web interactivo, que ofrezca hacer lo anterior, pero también deshacer lo hecho, dejando todo como estaba.

Y por último crearemos otro web panel que nos permita eliminar la información de ambas tablas, Category y Attraction. Esto será retomado después, por lo que recomendamos ver al menos esta parte final.

GeneXus

To be continued...

Por supuesto hay mucho más para ver, pero lo dejamos para el siguiente nivel. Si está interesado, puede buscar los videos relacionados a este tema en el curso de Analista GeneXus.

GeneXus™

training.genexus.com
wiki.genexus.com