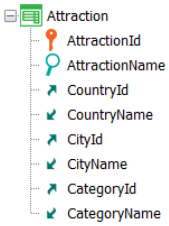
Actualización de la base de datos

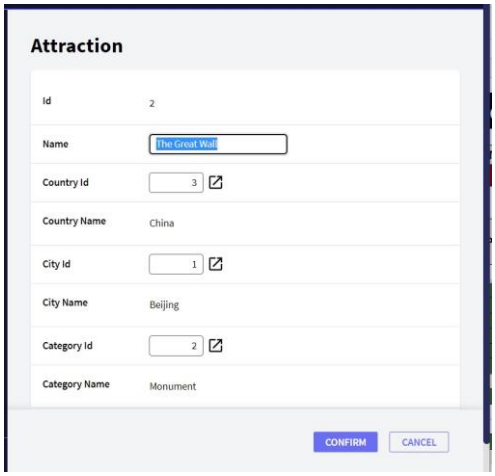
¿Qué sucede cuando una clave foránea acepta nulos?

GeneXus™

1. Update through a form

Insert, Update, Delete



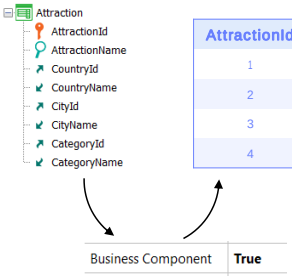


2. Update by code

Tenemos dos formas de actualizar la base de datos: hacerlo interactivamente a través del objeto transacción, utilizando su pantalla, o hacerlo por código.

1. **Business Compontent**: Insert(), Update(), Delete()

Insert, Update, Delete



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

2. **Procedure**: New, For each, Delete

Para actualizar por código teníamos dos posibilidades:

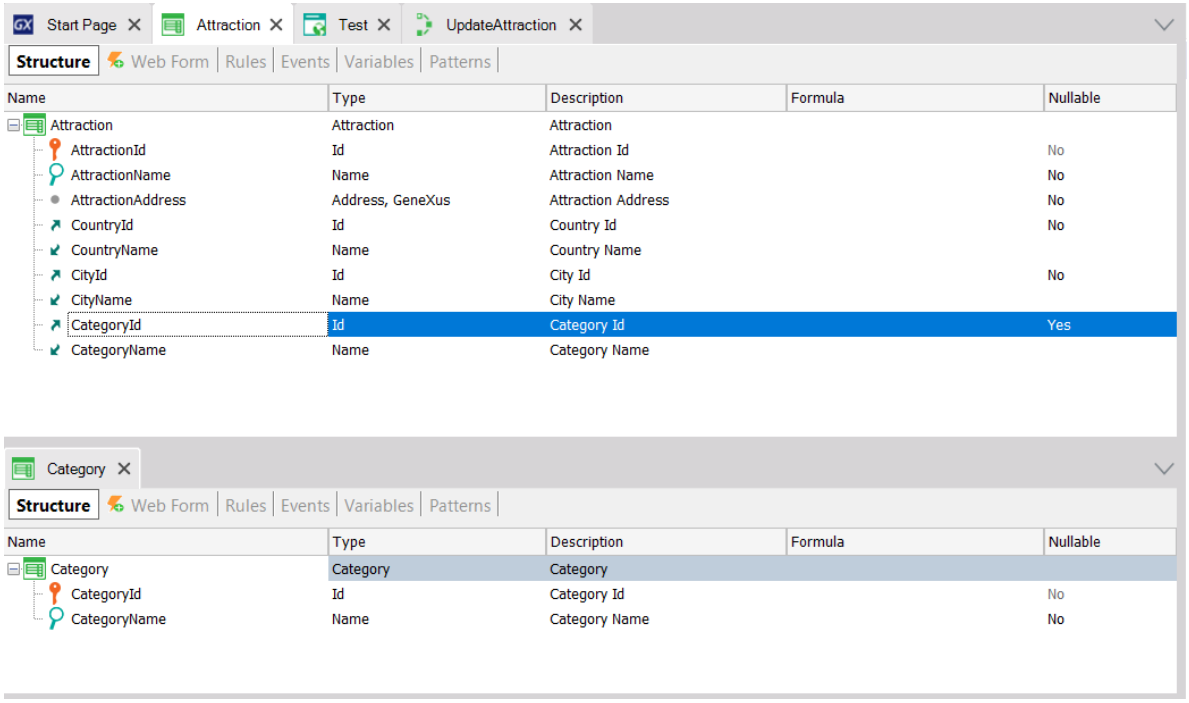
Hacerlo utilizando el business component de la transacción, a través de sus métodos, o hacerlo exclusivamente dentro de un procedimiento, a través de los comandos New, For each con asignación directa de los atributos a ser modificados, y el comando Delete.

	Uniqueness controls	Referential Integrity controls	Rules/Events execution
Business Component	✓	✓	✓
Proc (new, for each, delete)	✓		

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

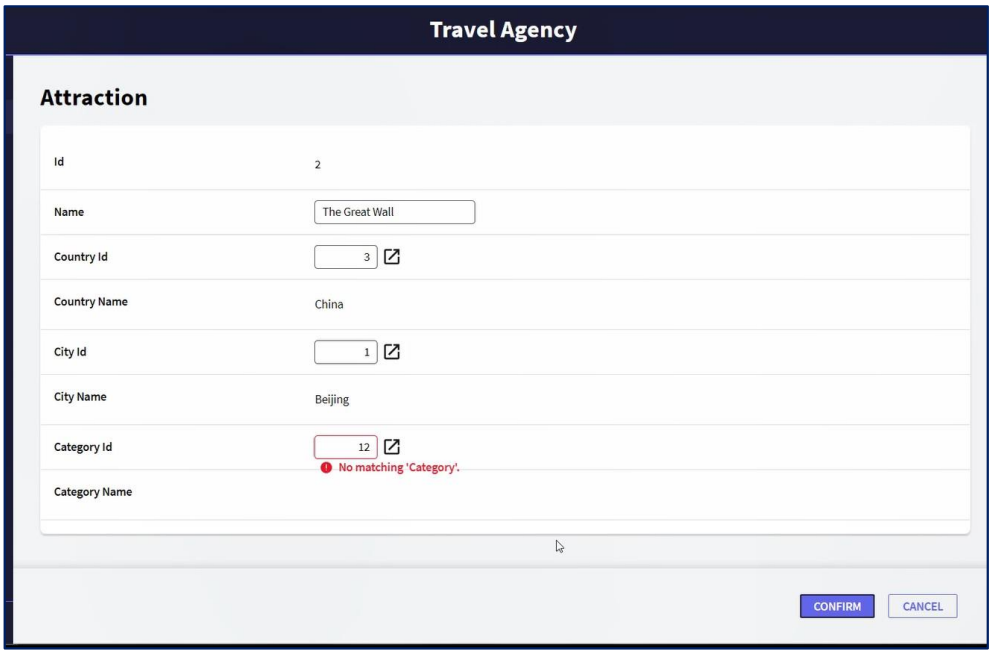
Como sabemos, con ambas opciones se controla la unicidad de registros, es decir, nunca se permite dejar en la base de datos registros con clave primaria o candidata repetida. En el ejemplo, nunca se permitirán dos atracciones con el mismo identificador. Y si, por ejemplo, AttractionName fuera clave candidata, tampoco se permitirían dos atracciones con el mismo nombre.

Pero la gran diferencia entre actualizar vía Business Component y hacerlo vía comandos de actualización directa en un procedimiento es, no solo que únicamente los Business Components ejecutarán la lógica que proviene de las reglas y eventos de la transacción, sino que solo vía Business Components se controla la integridad referencial por programa. ¿Por qué decimos “por programa”?

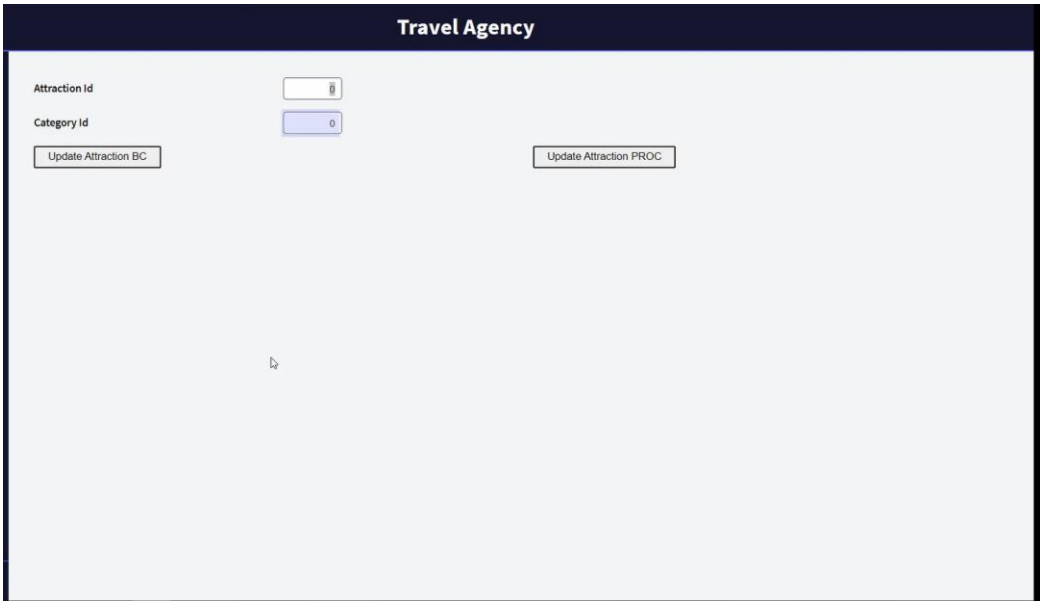


Si tenemos la transacción Attaction que registra las atracciones turísticas que siempre pertenecen a un país y ciudad y a alguna categoría (como monumento, museo, etc.), entonces claramente en la tabla asociada tendremos claves foráneas. En particular, el atributo CategoryId será clave foránea a la tabla Category.

El desarrollador GeneXus no tendrá que encargarse de programar en la transacción el control de integridad referencial para cada clave foránea porque ya estará incluido automáticamente en el programa generado. Es parte de la lógica de la transacción.

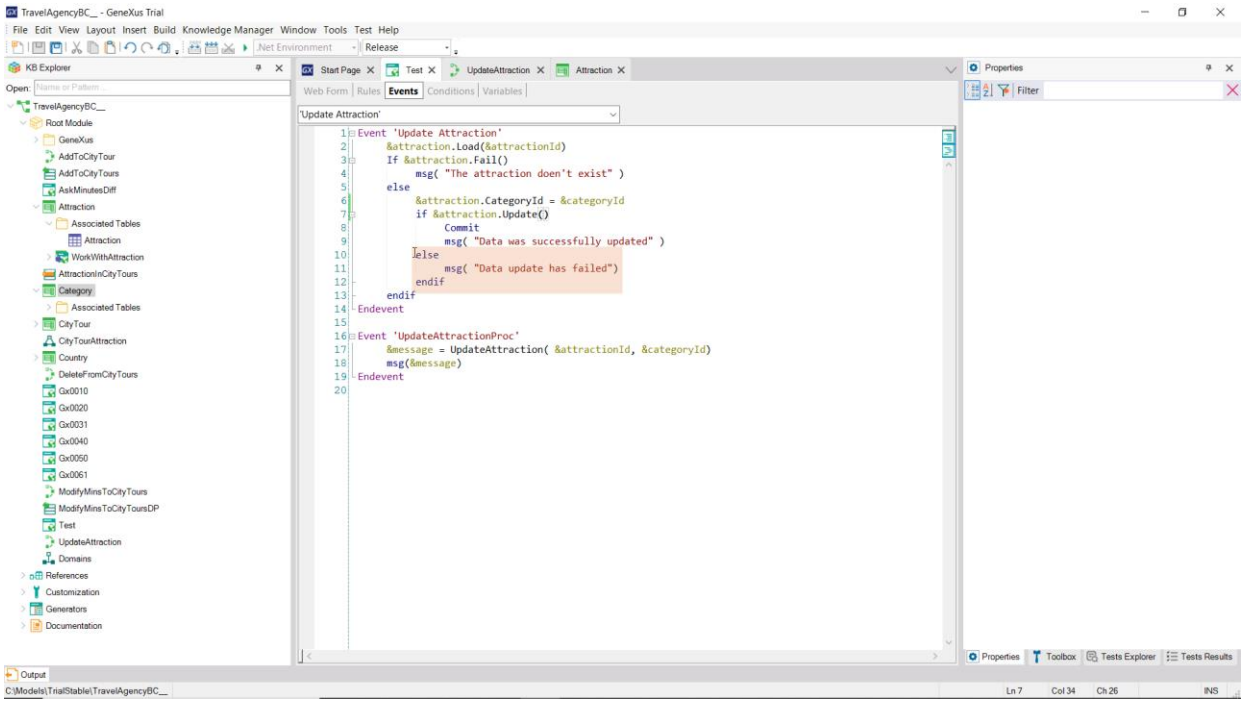


Así, cuando el usuario quiere asignarle a una atracción existente una categoría inexistente, la transacción nos informa que eso no es posible. Está haciendo el control de integridad referencial, antes de que lo haga la propia base de datos.



Supongamos que implementamos un web panel que permite que el usuario ingrese un id de atracción y un id de categoría, para cambiarle a la atracción de ese id su categoría por esta otra.

Tenemos dos alternativas para hacerlo por código: una es a través del Business Component Attraction, y la otra será a través de un procedimiento.

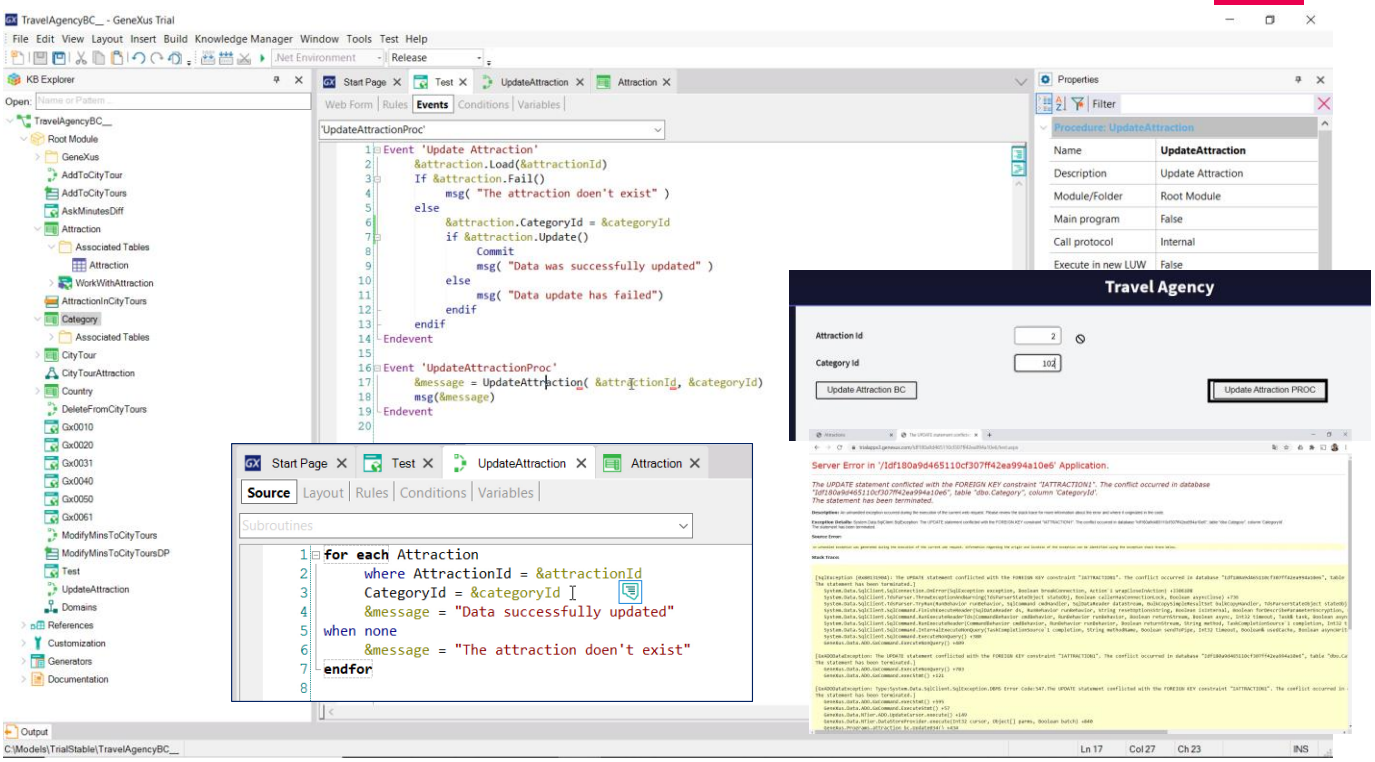


Veamos cómo implementaríamos la primera.

Habiendo prendido la propiedad Business Component, definimos una variable de ese tipo, y en el evento asociado al botón la cargamos a partir de la variable &AttractionId que ingresa el usuario a través de la pantalla.

Si existe, es decir, no falla el Load, entonces le cambiamos la categoría por la ingresada por el usuario, e intentamos actualizar. Esto funcionará igual que la transacción, puesto que el código asociado a ese método Update chequeará la existencia de esa categoría para recién después enviar la orden de actualización a la base de datos. Y como ese código encontrará que no existe esa categoría, no llegará a hacer el intento de actualización en la base de datos; devolverá False y se ejecutará el "else".

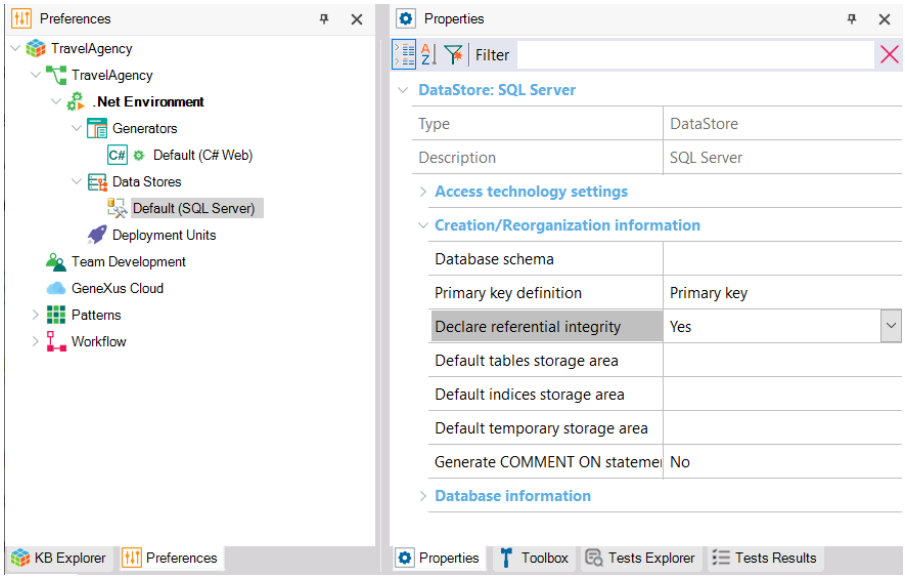
Probémoslo en ejecución.



En cambio, si esa misma actualización queremos hacerla vía procedimiento, es decir, como vemos aquí, a este procedimiento le enviamos identificador de atracción y de categoría, y en él se busca la atracción y se actualiza directamente la categoría.

Cuando ejecutemos con una categoría inexistente, ¡auch! Se dará una excepción de base de datos y el usuario tendrá que enfrentarse a esta pantalla tan poco amigable. Es que aquí el programa no está haciendo los controles para asegurar la integridad referencial. Pero sí los está realizando la base de datos. Y por eso este error.

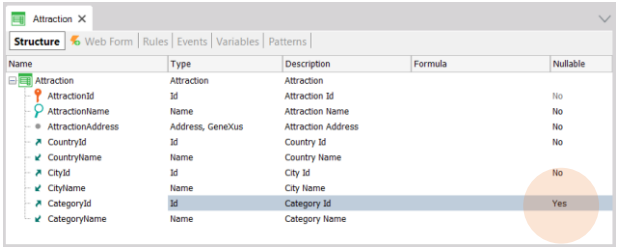
Podríamos capturar el error para hacer algo más amigable y que el programa no cancele abruptamente. Para ello contamos con la regla Error_handler. Como parámetro debemos dar el nombre de una subrutina que debemos definir en el objeto y que será donde programaremos el comportamiento en caso de error de base de datos, preguntando primero por ese error. Puede ver detalles sobre esto en nuestro wiki.



También tenemos la posibilidad de eliminar las declaraciones de integridad referencial en la base de datos, para que ésta no realice los controles. No es muy aconsejable, pero para aquellos casos que lo necesiten, contamos con esta propiedad a nivel del Data Store.

If select CateogryId from
Category where CategoryId =102
...

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	102
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

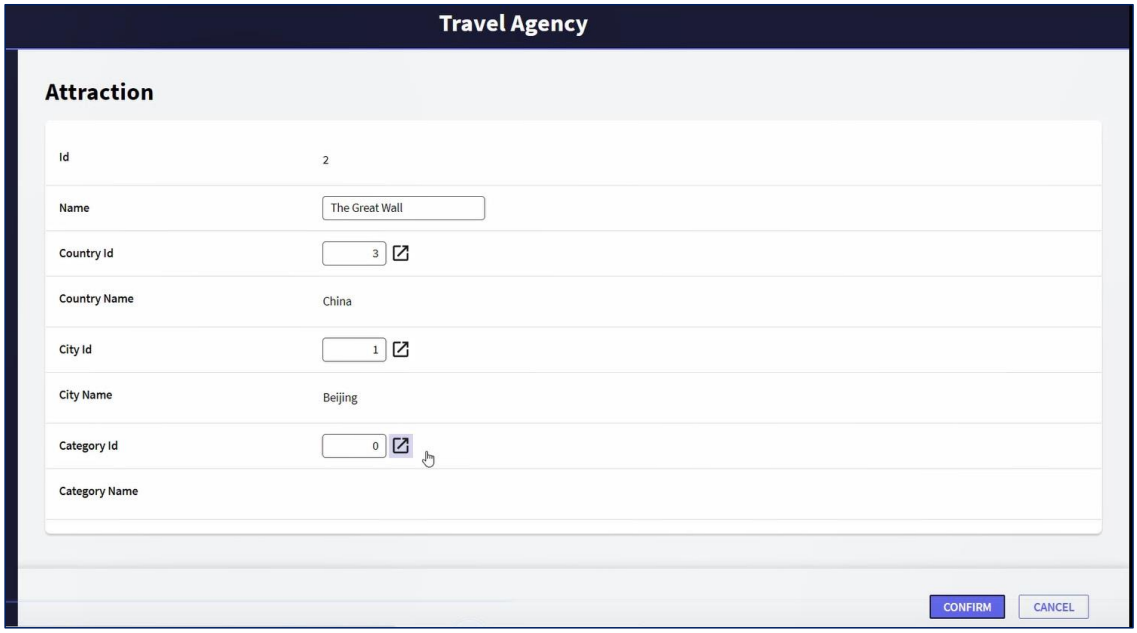


AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	null
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

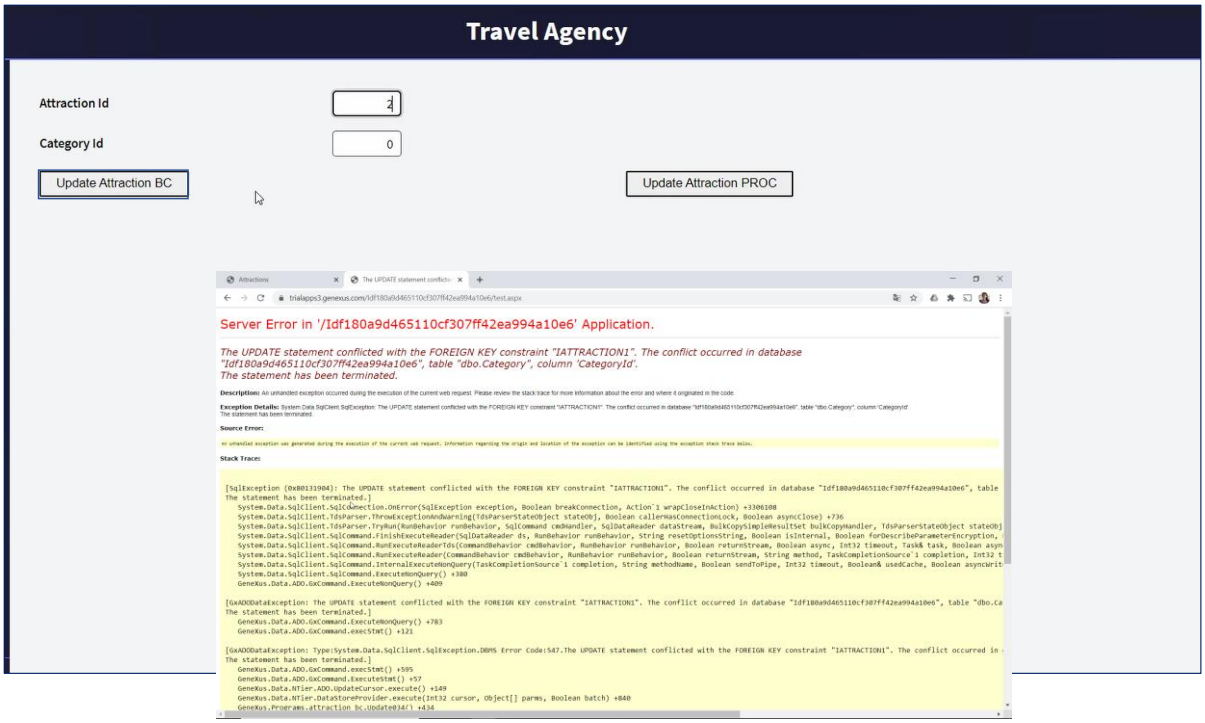
Por defecto, entonces, si no modificamos esa propiedad, la integridad referencial podrá o no ser controlada programáticamente, pero siempre lo va a ser por base de datos.

Sin embargo habíamos visto un caso en el que se permitía un valor de clave foránea inexistente. Es que estrictamente hablando, no se trataba de un valor.
Era cuando permitíamos que la clave foránea fuera nula. ¿Cómo le indicábamos a la base de datos que para una clave foránea íbamos a permitir esto?

En la estructura de la transacción teníamos la columna Nullable, que por defecto estaba en No, pero que podíamos pasar a Yes.



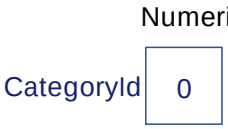
Esto hacía que cuando ejecutábamos la transacción, y dejábamos vacío el identificador de categoría, no sólo el programa no controlara la integridad referencial, sino que tampoco lo hiciera la base de datos.

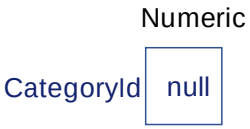


Sin embargo, si intentamos hacer esto mismo, pero a través del Business component, ¡auch! Nos está dando básicamente la misma excepción de base de datos que antes, por falla de integridad referencial. ¿Qué está pasando?

¿Por qué usando la transacción pudimos pero usando su Business Component no?

empty ≠ null





Es que el valor vacío y el nulo son dos cosas distintas, muy distintas.

El valor vacío es un valor. Dependiendo del tipo de datos, será uno de los valores posibles de ese tipo. Así, si el tipo es numérico, el valor vacío es el cero. Las bases de datos tienen especificados cuáles son los valores vacíos de acuerdo a cada tipo de datos. Pero el nulo, o null, estrictamente hablando no es un valor. De hecho es un NO VALOR. Decir que CategoryId es Null es lo mismo que decir que no está especificado su valor, que no se conoce. No es lo mismo que decir que es vacío.

empty ≠ null

For AttractionId = 2 → CategoryId = 0

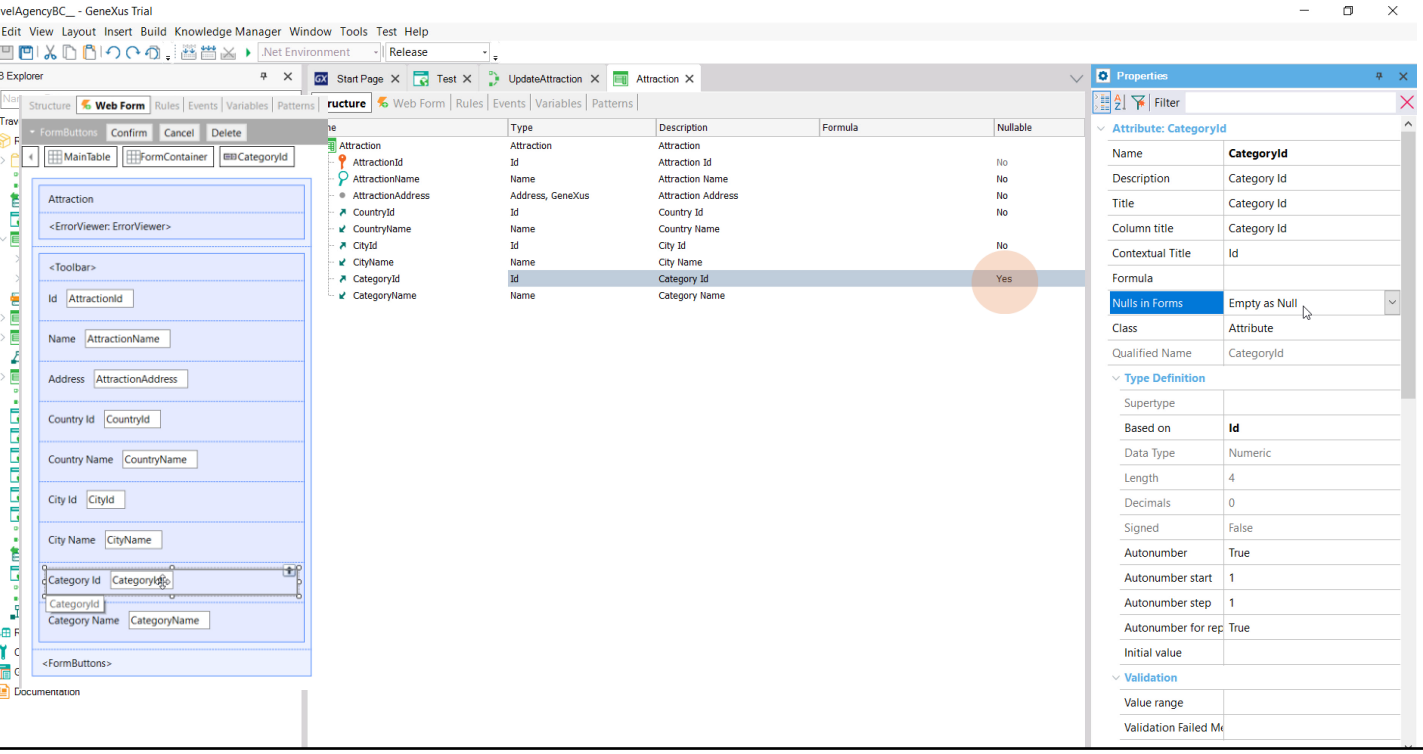
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

For AttractionId = 2 → CategoryId = null

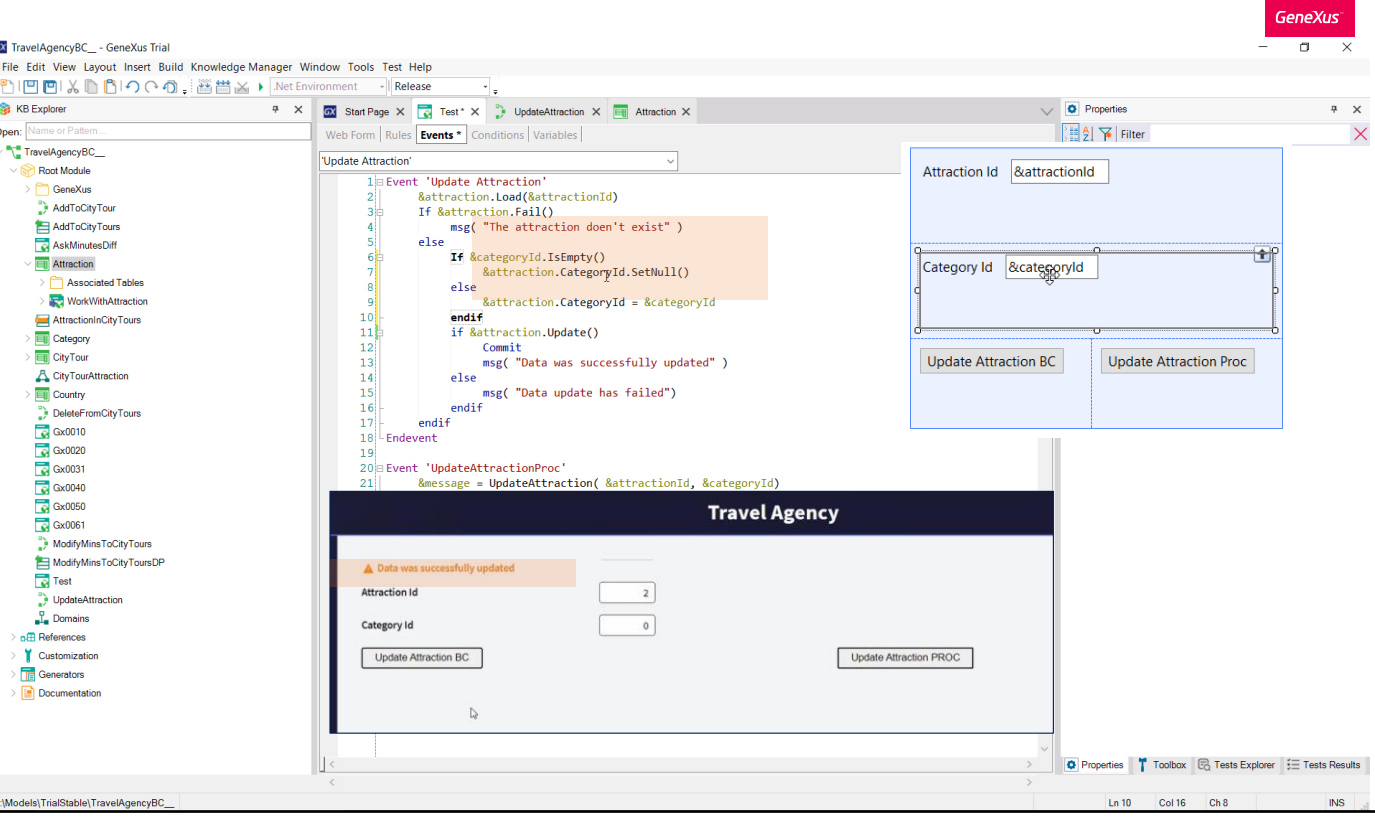
AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	null
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

Por tanto, para las bases de datos vacío y nulo son dos cosas muy distintas. Si estamos implementando un programa y enviamos a la base de datos la orden de modificar la categoría de la atracción 2 para que sea 0 en lugar de 2, lo que obtendremos será ni más ni menos que la pantalla con al excepción de base de datos por falla de integridad referencial. En cambio, si le decimos que lo cambie por null, no habrá ningún problema, tal como sucedió cuando lo hicimos a través de la transacción.

Entonces la pregunta es: ¿por qué la transacción está pidiéndole a la base de datos que modifique por Null y en cambio el Business component está enviando el valor vacío, cero?



Es que si observamos las propiedades del atributo clave foránea, CategoryId, vemos que hay una de nombre “Nulls in Forms” que tiene por defecto el valor “Empty as Null”. Esto significará que si el atributo acepta nulos, como es el caso, entonces toda vez que se deje vacío el valor en el form, se considere que es Null cuando se envíe a la base de datos y no vacío.



Pero esto no vale cuando la actualización la hacemos a través del Business component, que no tiene un form.

Por tanto si esta variable &CategoryId está vacía, este elemento quedará con valor vacío y no nulo, y por ello es que falla la integridad referencial al intentar la actualización. Tenemos que especificar el nulo si la variable está vacía.

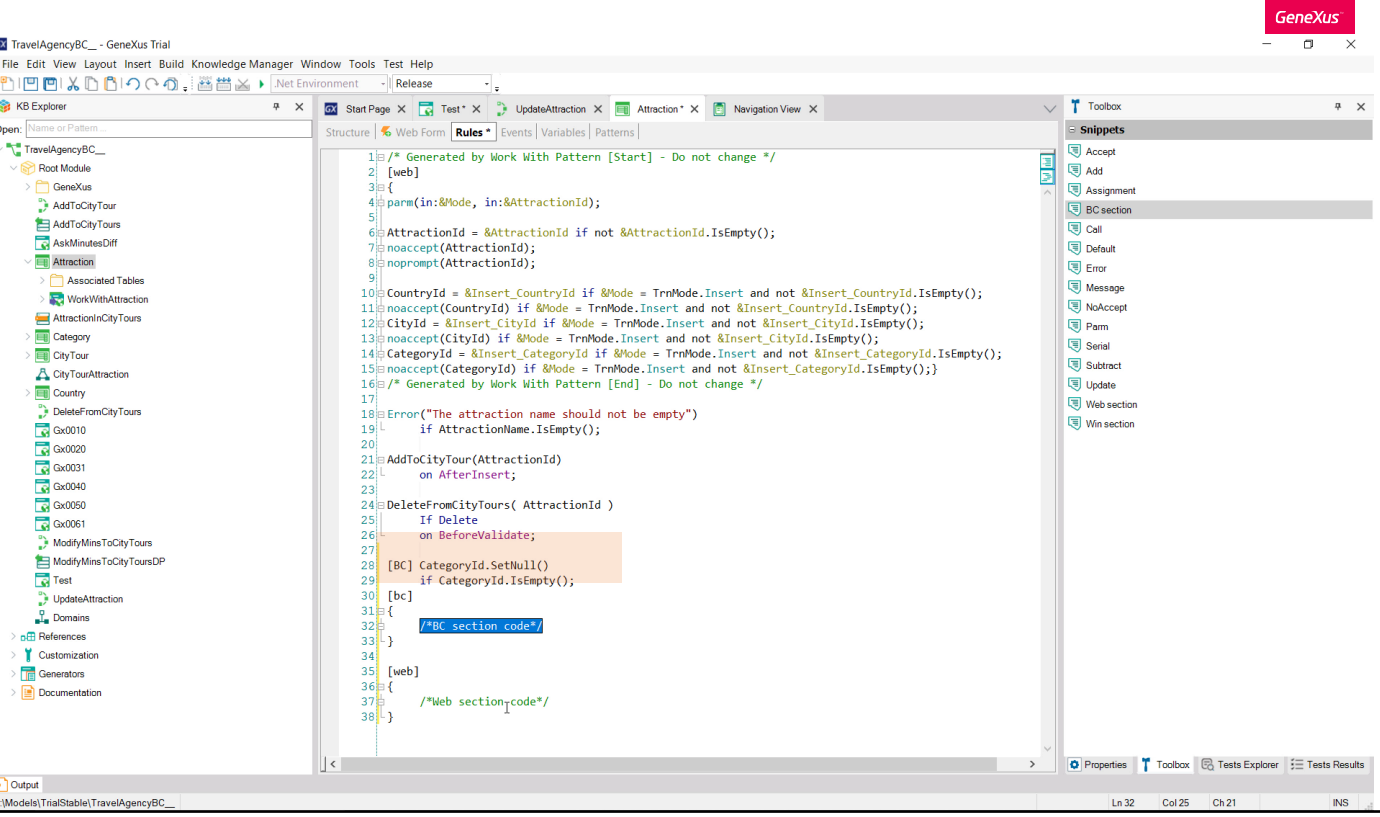
Tenemos dos formas de hacer esto. O lo hacemos a nivel local, solo aquí, o hacemos el mismo mapeo que consigue la propiedad del atributo, pero esta vez válido también en el Business Component, donde quiera que se use.

Para lo primero, alcanzará con especificar que si está vacía la variable CategoryId, al elemento del Business Component le configure el nulo. El método SetNull aplica a atributos que acepten nulos como a sus correspondientes en las variables Business Component, como es este caso.

Si ahora probamos, vemos que lo conseguimos.

La desventaja de esta alternativa es que si en otro lado queremos dejar nula esta clave foránea a través de una variable Business Component tendremos que hacer esto mismo.

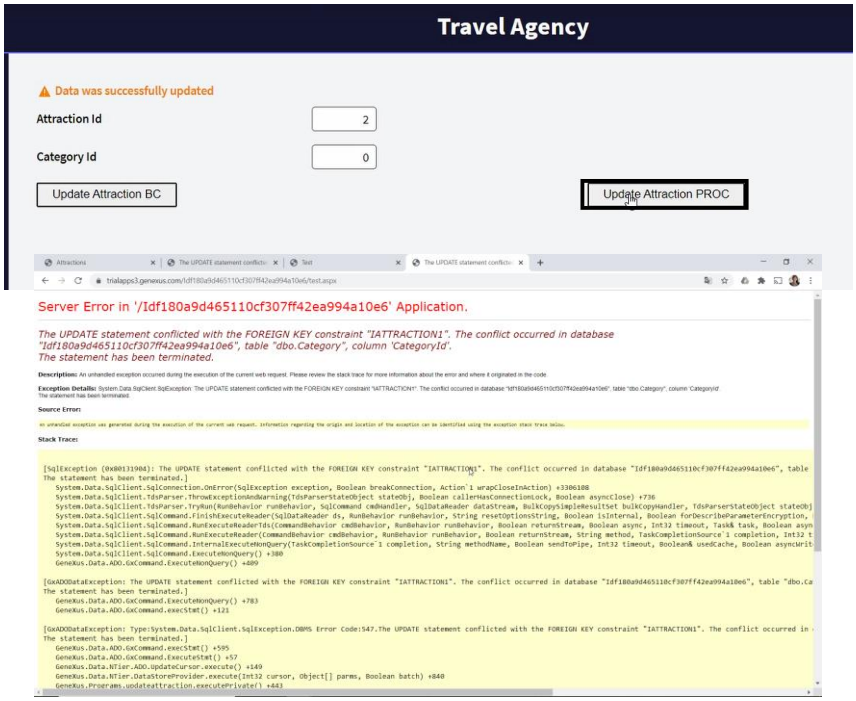
La otra opción, más general, decíamos, es incorporar este SetNull como regla en la transacción. Aquí lo quitamos entonces...



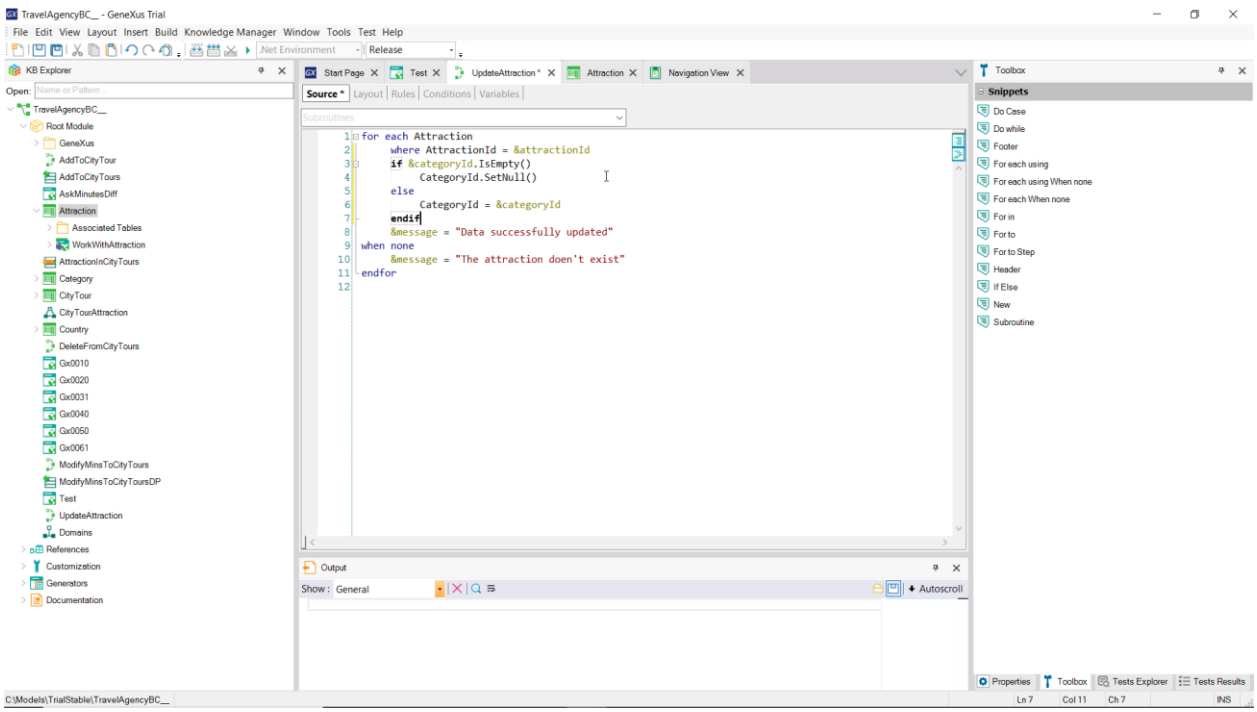
Y vamos a la transacción y especificamos que le asigne nulo al atributo cuando su valor se deje vacío. Esto mismo ya es lo que por defecto se realiza cuando se utiliza la pantalla de la transacción, gracias a esta propiedad, por lo que, estrictamente hablando, podemos pedir que esta regla solo se ejecute para el Business Component.

Siempre podemos indicar que una regla solo se ejecute cuando se trata de la transacción Web, o cuando se trata del Business Component, colocando esta marca.
De hecho, si queremos escribir un conjunto de reglas que solo apliquen a la transacción Web, o que solo apliquen al Business Component, lo hacemos así.

Probemos. Vemos que efectivamente es otra solución.



Por último, ¿qué pasa ahora con la actualización a través del procedimiento? Tenemos exactamente el mismo problema.



La solución aquí solo puede ser la local. Probemos.

Y vemos que lo conseguimos.

GeneXus™

training.genexus.com
wiki.genexus.com