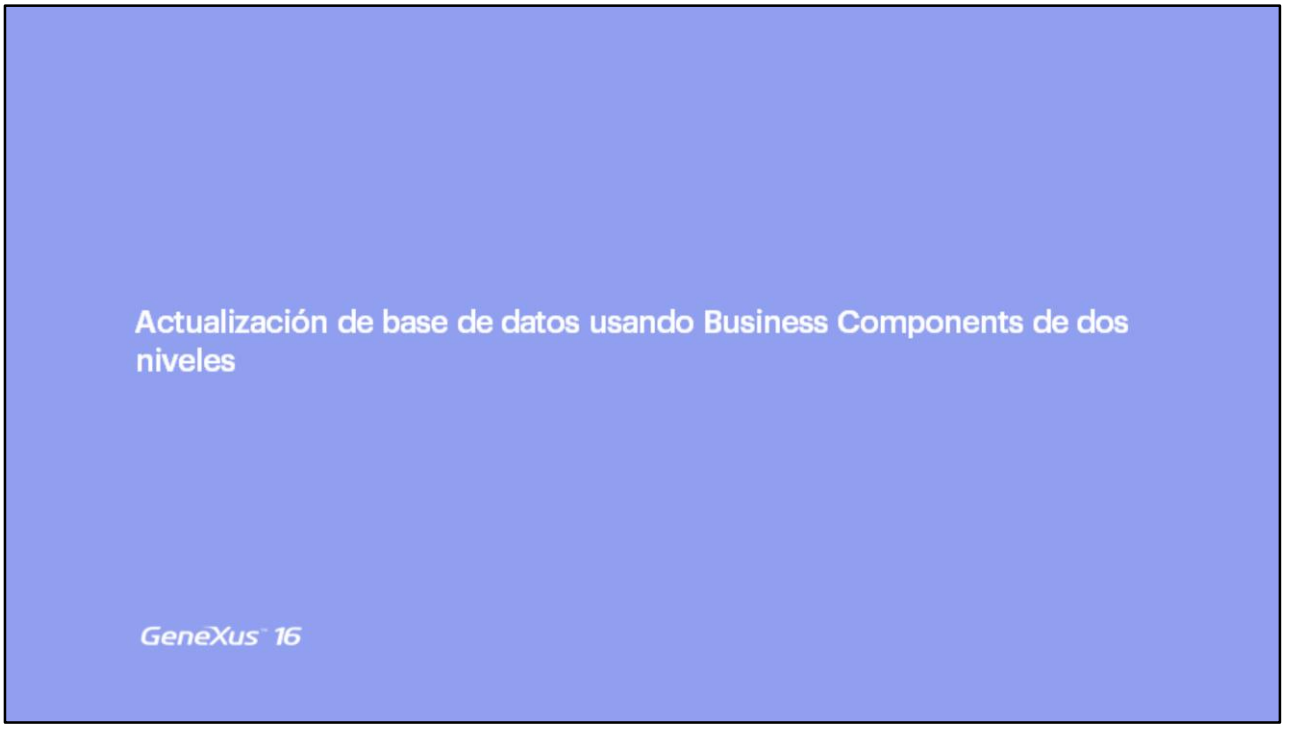
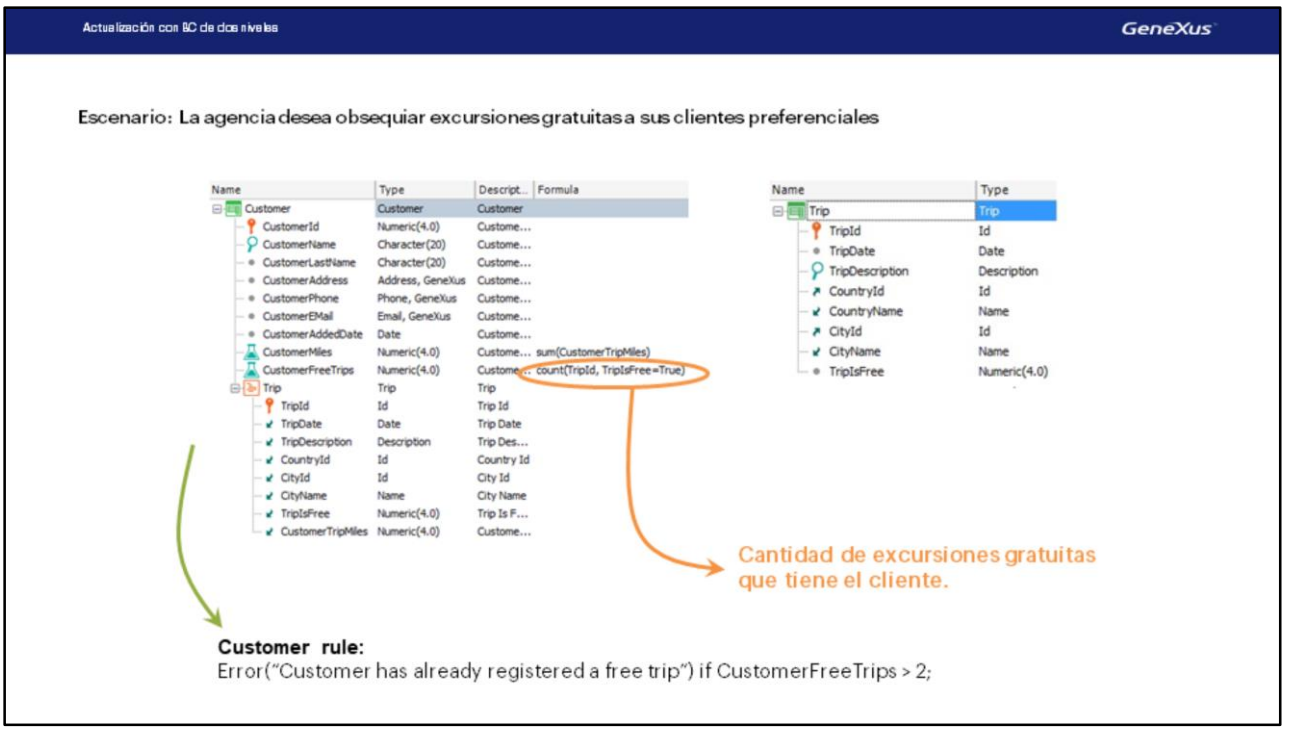


Ya hemos visto anteriormente el concepto general de Business Component y su aplicación en una transacción de un solo nivel.

Veamos ahora qué sucede cuando para una transacción de dos niveles creamos su Business Component.



La Agencia ofrece excursiones a distintas ciudades, para visitar sus atracciones turísticas. Cada cierto tiempo la Agencia obsequia una excursión gratuita a sus clientes preferenciales.

De cada cliente el sistema necesita registrar las excursiones que ha contratado, así como las gratuitas ofrecidas por la agencia. La agencia obsequia hasta 2 excursiones por cliente.

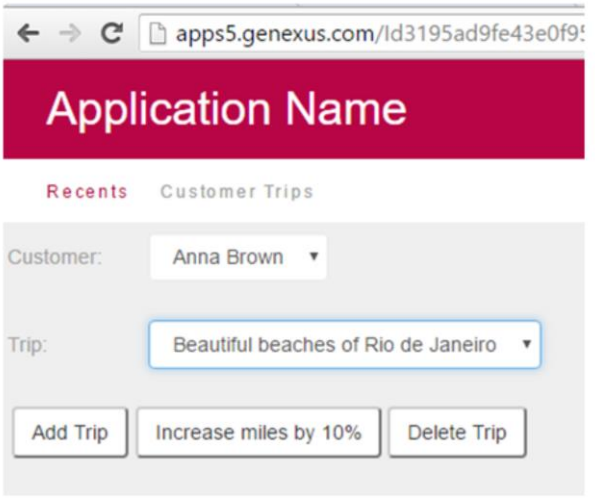
Para registrar las excursiones de la agencia utilizaremos una transacción Trip que entre su información registra a través del atributo booleano TripIsFree, si la excursión es gratuita o no. Para registrar las excursiones de un cliente, agregamos un segundo nivel a la transacción Customer con TripId como identificador, los atributos inferidos de la transacción Trip, y CustomerTripMiles como único atributo secundario, para registrar las millas que obtiene el cliente por realizar esa excursión.

Para controlar que no se le obsequien más de dos excursiones gratuitas a cada cliente, hemos declarado el atributo fórmula CustomerFreeTrips y la regla:

Error("Customer already has registered a free trip") if CustomerFreeTrips > 2;

Escenario: Requerimiento

Un panel como este:



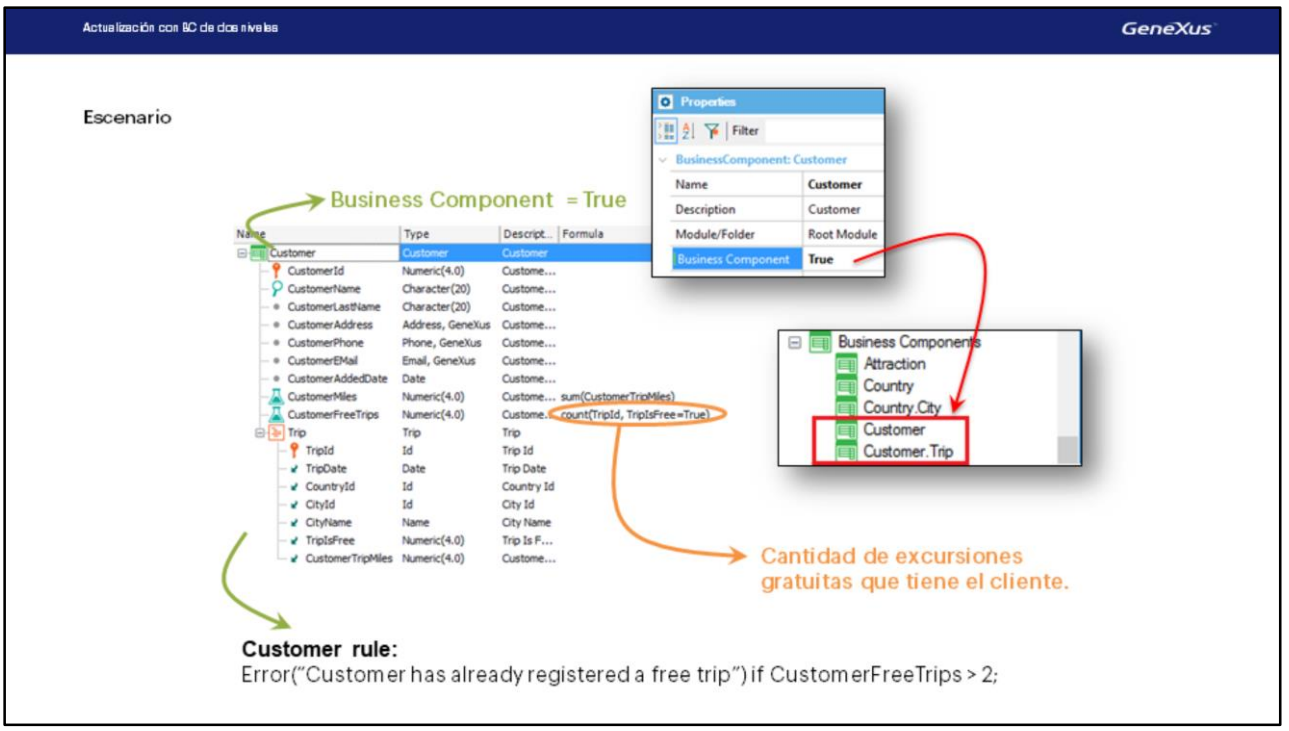
Se ofrecerá al usuario un panel como el que vemos arriba, que le permita elegir mediante un combo dinámico un cliente y mediante otro combo dinámico una excursión.

Luego de elegir el cliente y la excursión, tendremos tres operaciones posibles:

- Add Trip: intentará agregar la excursión seleccionada al cliente elegido. Si la excursión es gratuita y el cliente ya tiene 2 excursiones gratuitas asignadas, no deberá permitirse la inserción y un mensaje en pantalla deberá informarnos sobre el error.
- Increase miles by 10%: si el cliente tiene registrada la excursión seleccionada, intentará aumentar un 10% su cantidad de millas. En caso contrario, un mensaje en pantalla nos indicará que no fue posible la operación.
- Delete Trip: intentará eliminar en el registro del cliente la excursión seleccionada. Si esa operación no puede realizarse (por ejemplo por no encontrarse esa excursión para ese cliente) un mensaje deberá informar al usuario del problema.

La pantalla que mostramos se implementará con un web panel, objeto que estudiaremos más adelante, por lo que sus detalles de funcionamiento no se explicarán aquí.

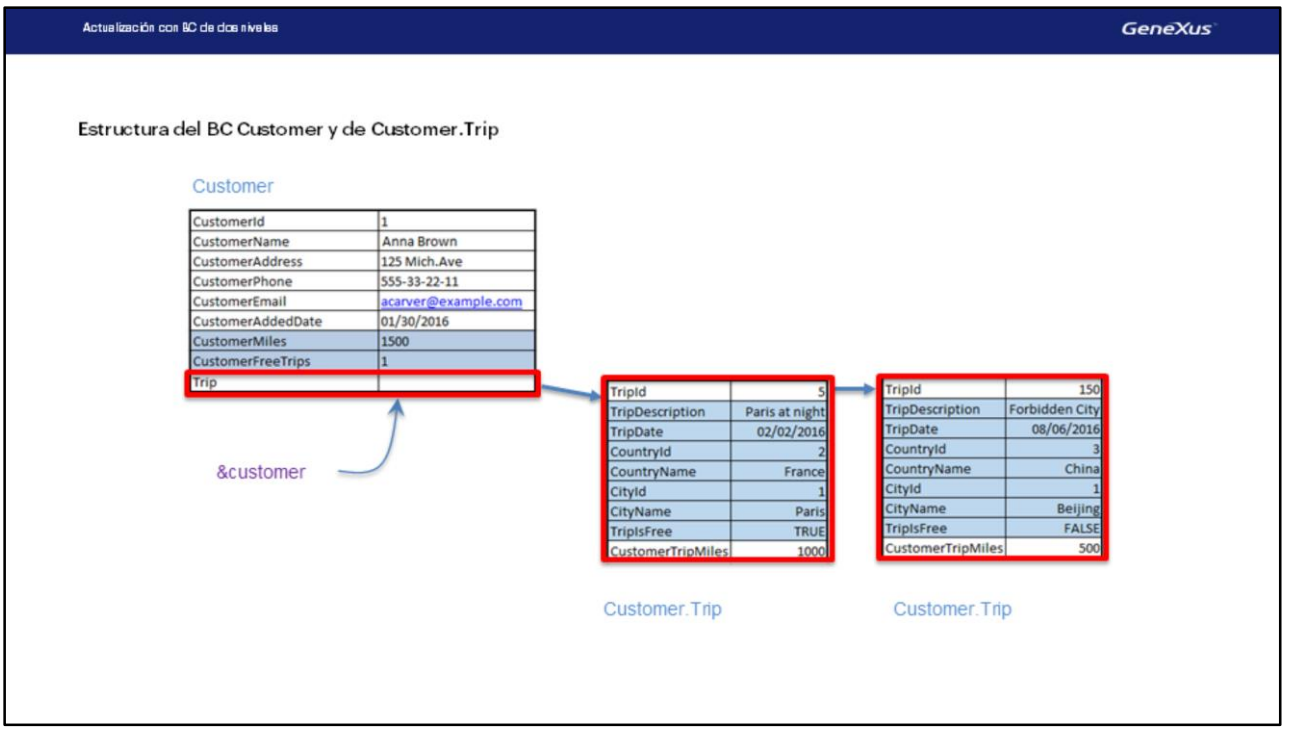
Nos concentraremos en ver cómo programamos el código correspondiente a cada botón, es decir, cómo realizar esa inserción, modificación y eliminación de lo que correspondería a una línea de la transacción Customer, pero sin usar la transacción.



Crearemos el Business Component asociado a la transacción. Para eso, declaramos su propiedad Business Component con el valor True.

Al grabar, sabemos que GeneXus crea el tipo de datos Customer asociado a la transacción. Pero como se trata de una transacción de dos niveles, crea también el tipo de datos Customer.Trip asociado específicamente a las líneas del segundo nivel, que en nuestro ejemplo corresponden a las excursiones del cliente.

Lo encontraremos en la KB como un tipo de datos business component más, pero, a diferencia del tipo de datos Business Component correspondiente a la transacción (en nuestro caso, Customer), que nos permite realizar operaciones como si de la transacción se tratase, este segundo solamente nos permite definir la estructura de una línea de la transacción, pero no tiene métodos específicos asociados, como Insert, Update, Delete, Save, etc.



Aquí vemos la estructura del Business Component Customer, y la del Customer.Trip.
Si observamos bien, una variable &customer de tipo Customer, tendrá un elemento por cada atributo del cabezal de la transacción y tendrá un elemento para la colección de líneas. Así, tendremos:

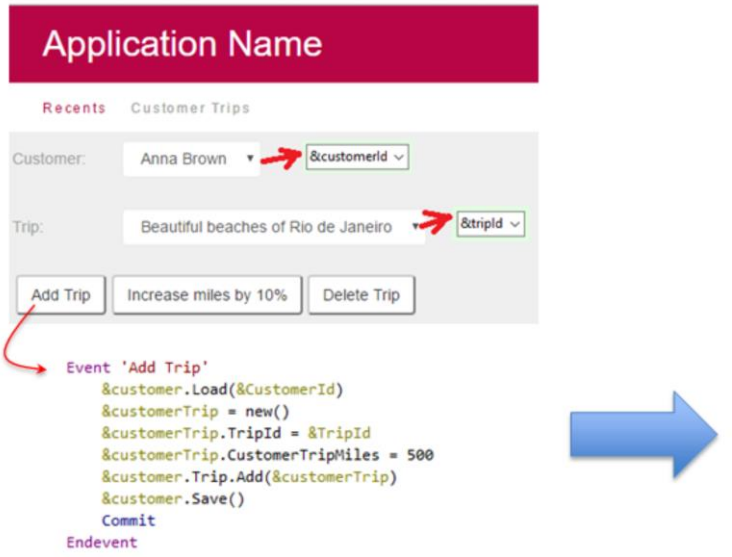
&customer.CustomerId basado en el atributo CustomerId, de tipo Id (Numeric)
&customer.CustomerName basado en el atributo CustomerName de tipo Name (Character)
...
&customer.CustomerAddedDate basado en atributo CustomerAddedDate de tipo Date
&customer.CustomerMiles basado en atributo CustomerMiles de tipo Numeric
&customer.CustomerFreeTrips basado en el atributo CustomerFreeTrips de tipo Numeric
&customer.Trip de tipo colección de Customer.Trip

Observemos que los atributos CustomerMiles y CustomerFreeTrips que en la transacción son atributos fórmula, sólo se utilizan a la hora de consultar la información de un cliente ya cargado en la variable &customer (como en el ejemplo que mostramos arriba).

El tipo de datos Customer.Trip corresponde a una estructura con los elementos correspondientes a los atributos de nivel Trip de la transacción. Otra vez, los atributos que en ese segundo nivel de la transacción son inferidos, serán atributos de consulta únicamente.

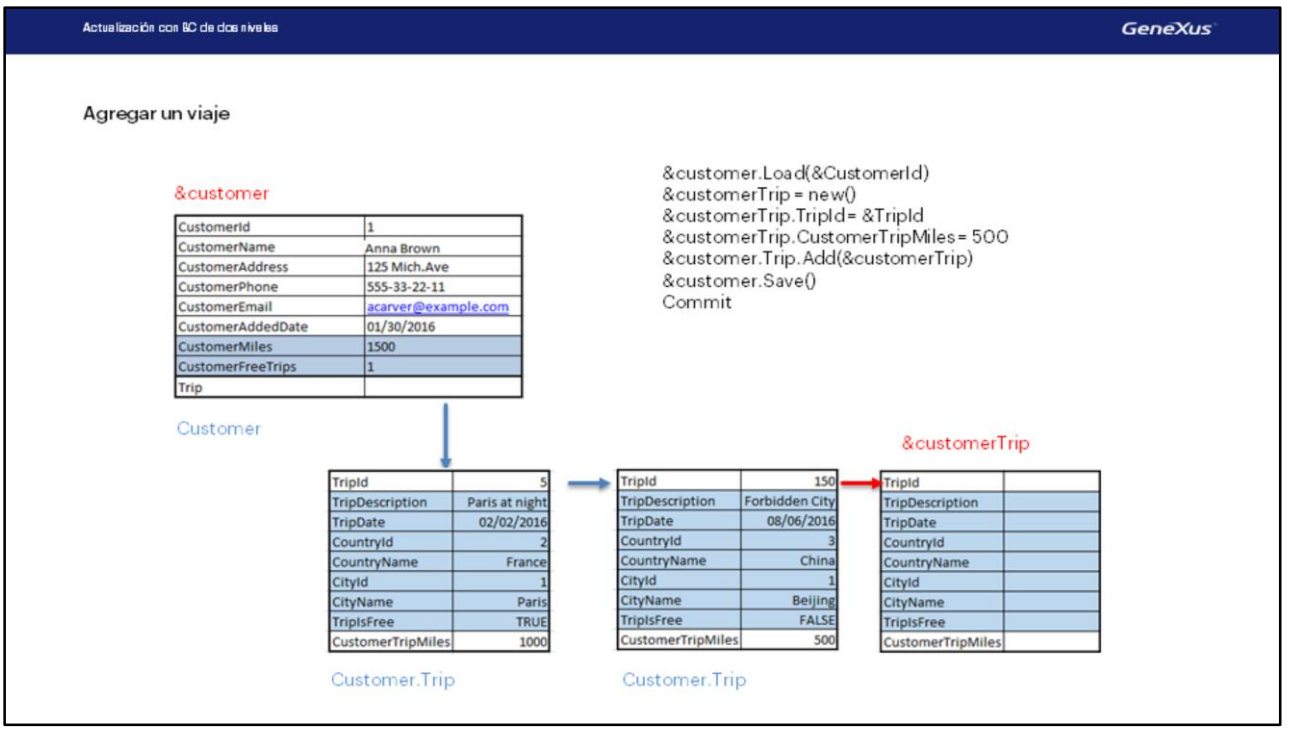
Como veremos en lo que sigue, para insertar una nueva “línea” o para modificar una ya existente, necesitaremos hacerlo utilizando una variable de este tipo de datos: Customer.Trip.

Agregar un viaje



Sin entrar en los detalles del web panel, el combo que pide al usuario el cliente será una variable &CustomerId basada en el tipo de datos del atributo CustomerId (que mostrará los datos del atributo CustomerFullName), y el que pide las excursiones será otra variable, &TripId, basada en el tipo de datos del atributo TripId (que mostrará en ejecución los datos del atributo TripDescription).

Empecemos por la acción asociada al botón “Add Trip”. Este será el código. Lo explicamos en lo que sigue.



Si estuviéramos trabajando con la transacción, lo primero que deberíamos hacer es ubicar el cliente con el que queremos trabajar. Para eso cargamos (a través del método Load) la variable &customer (del tipo el Business Component Customer) con el valor elegido por el usuario en la variable &CustomerId en pantalla del web panel. En el ejemplo, el customer de Id 1.

Ahora, si estuviéramos trabajando en la transacción de Customer para agregar una excursión deberíamos agregar una línea. Esto lo hacemos mediante el método new, que nos permite crear la variable &customerTrip del tipo de datos de los ítems de la línea: Customer.Trip. La usaremos para insertar la nueva excursión para ese cliente. Completamos sus valores: TripId con la excursión elegida por el usuario en el panel, y guardada en la variable &TripId, y le asignamos el valor de las millas de la excursión, en este caso, 500.
A diferencia del trabajo con la transacción, donde tenemos las líneas en pantalla disponibles, aquí tenemos que agregar explícitamente este ítem a la colección de líneas del cliente. Para ello hacemos uso del método Add (de las colecciones). Recordemos que &customer.Trip es una colección de ítems del tipo Customer.Trip.
Por último, tal como haríamos en la transacción, debemos confirmar la operación.
Para ello agregamos el Save del business component Customer y el Commit.

Nota: aquí podríamos haber utilizado, en lugar del método Save, el método Update, ya que si bien estamos insertando una línea nueva, a nivel del cliente lo estamos actualizando. Por eso el método Insert no sería correcto, pues estos métodos se están aplicando a la variable correspondiente al cabezal.

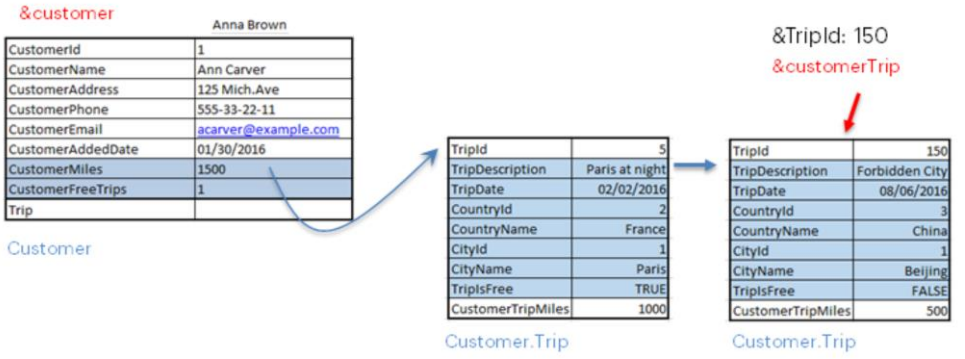
Si la operación no se pudiera completar, luego veremos cómo manejar los errores.

Modificar un viaje: Incrementar el total de las millas un 10%


```
Event 'Increase miles by 10%'
    &customer.Load(&CustomerId)
    &customerTrip = &customer.Trip.GetByKey(&TripId)
    &customerTrip.CustomerTripMiles = &customerTrip.CustomerTripMiles *1.10
    &customer.Save()
    Commit
Endevent
```

Veamos ahora la acción asociada al botón “Increase miles by 10%”. Aquí presentamos el código y lo explicamos en la página siguiente.

Modificar un viaje: Incrementar el total de las millas un 10%



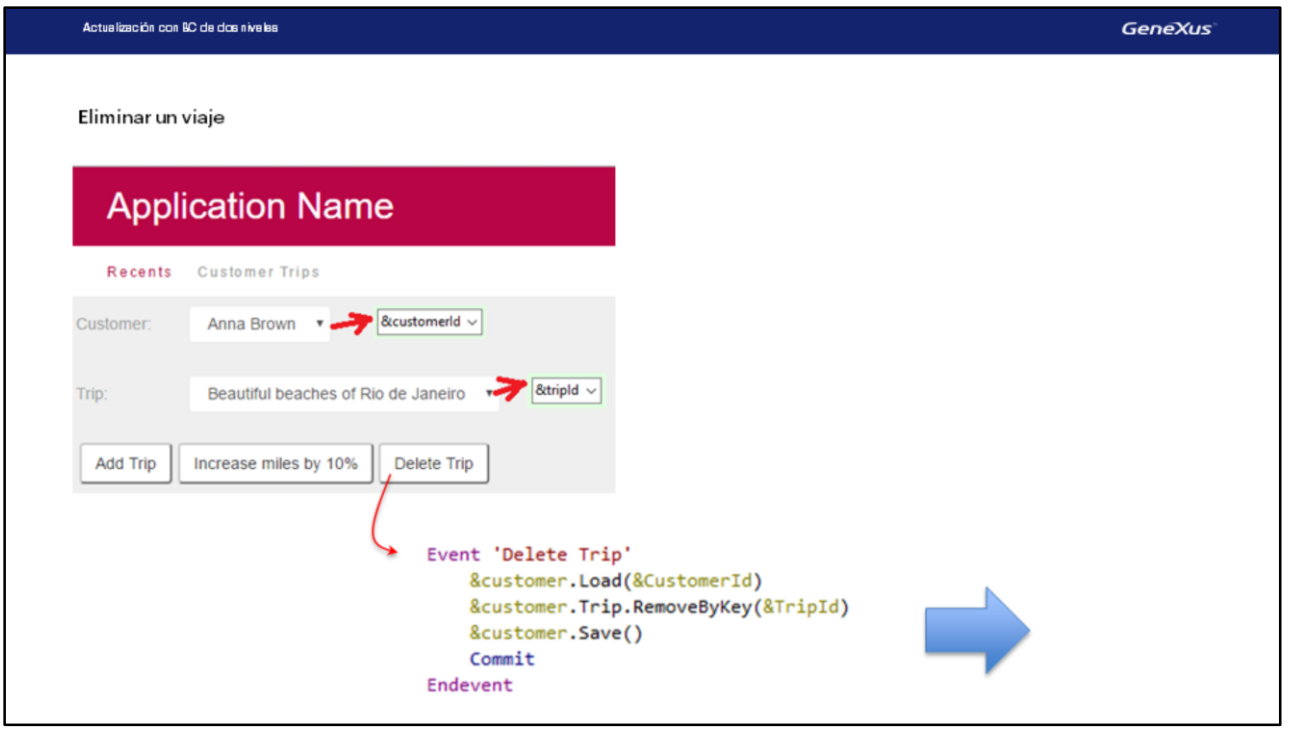

```
&customer.Load(&CustomerId)
&customerTrip = &customer.Trip.GetByKey(&TripId)
&customerTrip.CustomerTripMiles = &customerTrip.CustomerTripMiles *1.10
&customer.Save()
Commit
```

1. Tal como haríamos en la transacción, debemos cargar el cliente al que le queremos modificar las millas de una de sus excursiones: Load.
2. Debemos acceder a la excursión que queremos modificar: para ello utilizamos un método de la colección de líneas llamado GetByKey, al que le pasamos el identificador de la línea, y nos devuelve esa línea en la variable &customerTrip. No se trata de una copia de la línea, sino de la propia línea.
3. Modificamos el valor que nos interesa, en este caso las millas de la excursión
4. Hacemos Save y
5. Commit

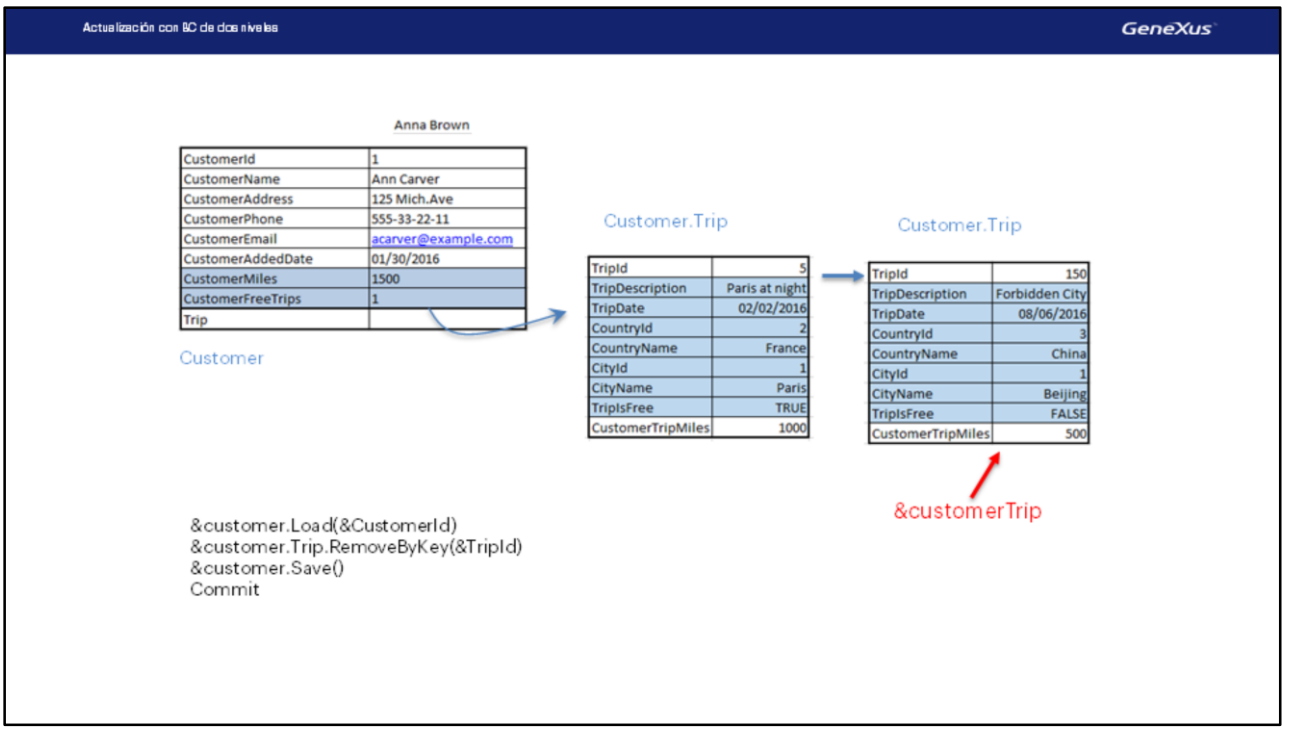
Nota: si tuviéramos algún otro atributo secundario y no le asignamos valor en forma explícita, quedará con el valor que tenía. Dicho de otro modo, solamente se asignan valores a los elementos que se deseen cambiar.

Valen las mismas consideraciones que hicimos antes respecto a los métodos Insert y Update en lugar del Save.

Si la operación no se pudiera completar, luego veremos cómo manejar los errores.

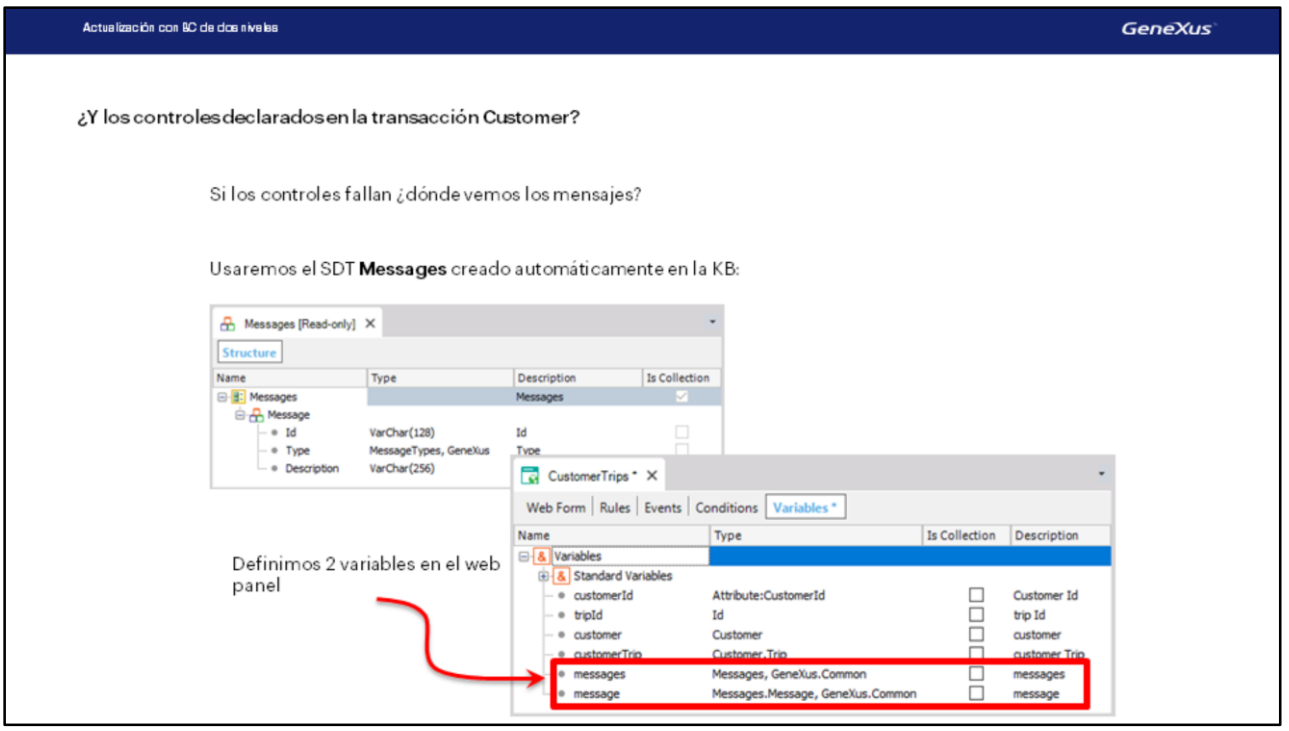


Veamos ahora la acción asociada al botón “Delete Trip”. Aquí presentamos el código y lo analizamos en la página siguiente.



Para eliminar una línea de la trnsacción nos posicionamos sobre la línea y la marcamos para eliminar. Esto en el business component se realiza a través del método RemoveByKey de la colección de las líneas. A este método hay que pasarle por parámetro el identificador de la línea a ser eliminada.
En la tranasacción para efectivamente eliminar la línea, presionamos Confirm. Aquí, ejecutamos el método Save. Otra vez, como vimos antes, podríamos reemplazar el Save por Update, pero no por Delete, dado que estos métodos aplican al cabezal. Y el cabezal ya existe.

Si la operación no se pudiera completar a continuación veremos cómo manejar los errores.



En la acción de agregar una excursión, ¿qué sucede si la excursión que estamos queriendo insertar:
1) Ya estaba asignada al cliente?
2) Es gratuita y el cliente ya tenía 2 excursiones de estas características?

En ambos casos la inserción fallará, en 1) porque el BC chequeará la unicidad de la clave compuesta del registro que estamos queriendo insertar y en 2) el BC disparará la regla error que la transacción tenía especificada para controlar la cantidad de excursiones gratuitas.

Si bien este control se dispara automáticamente al aplicar el método Save() a la variable del Business Component, para que el usuario final se entere, debemos obtener y mostrar explícitamente los mensajes que se emitieron.

¿A qué tipo de mensajes nos referimos? A los propios que emite GeneXus por controles de consistencia de los datos, unicidad de valores de llaves primarias, y a los mensajes asociados a nuestras propias reglas Error y Msg.

¿Cómo hacemos entonces para recuperar los mensajes que ocurrieron?

Utilizamos el tipo de datos estructurado Messages que automáticamente define GeneXus al crear una nueva KB. La finalidad de este tipo de datos **colección** es permitirnos acceder a los mensajes que se emiten en la ejecución de un Business Component.

De modo que definimos dos variables:
&messages (del tipo de datos Messages, colección) y
&message (del tipo de datos Messages.Message, que representa un item de la colección).

¿Y los controles declarados en la transacción Customer?

Recuperación de mensajes y errores:

```
&customerTrip.TripId  = &TripId
&customerTrip.CustomerTripMiles  = 500
&customer.Trip.Add(&customerTrip)
&customer.Save()
If &customer.Success()
    Commit
Else
    &messages = &customer.GetMessages()
    for &message in &messages
        msg(&message.Description)
    Endfor
Endif
```

Recuperamos la colección de mensajes emitidos y recorremos la colección mostrando cada descripción

Aplicamos el método GetMessages a la variable &Customer y obtenemos la colección de mensajes que ocurrieron.

Por último, utilizando la estructura “For item in colección”, recorremos cada mensaje de la variable colección de mensajes y desplegamos su descripción.

El mismo código coloreado se agregaría a los ejemplos anteriores al editar o eliminar una excursión.

Más de Business Components...

- http://wiki.genexus.com/commwiki/servlet/wiki?5846,Toc%3ABusiness+Component,

GeneXus™

The power of doing.

More videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications