Actualización de la base de datos

Update con Business component. Detrás de escena

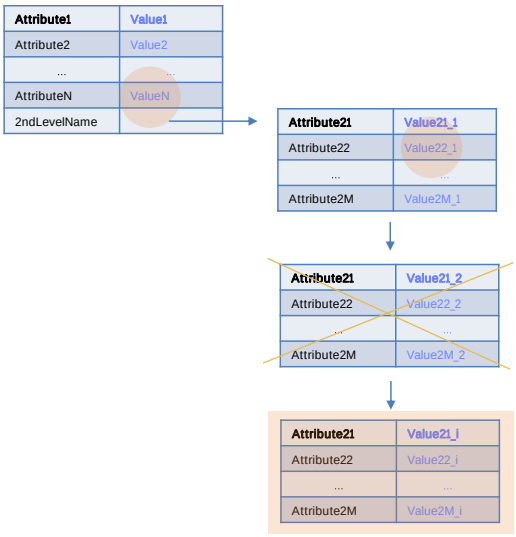
GeneXus™

Transaction

Attribute1*
Attribute2
...
AttributeN

2ndLevelName
{
Attribute21*
Attribute22
...
Attribute2M
}

&BC



&BC.Update()

&BC.**Mode**()

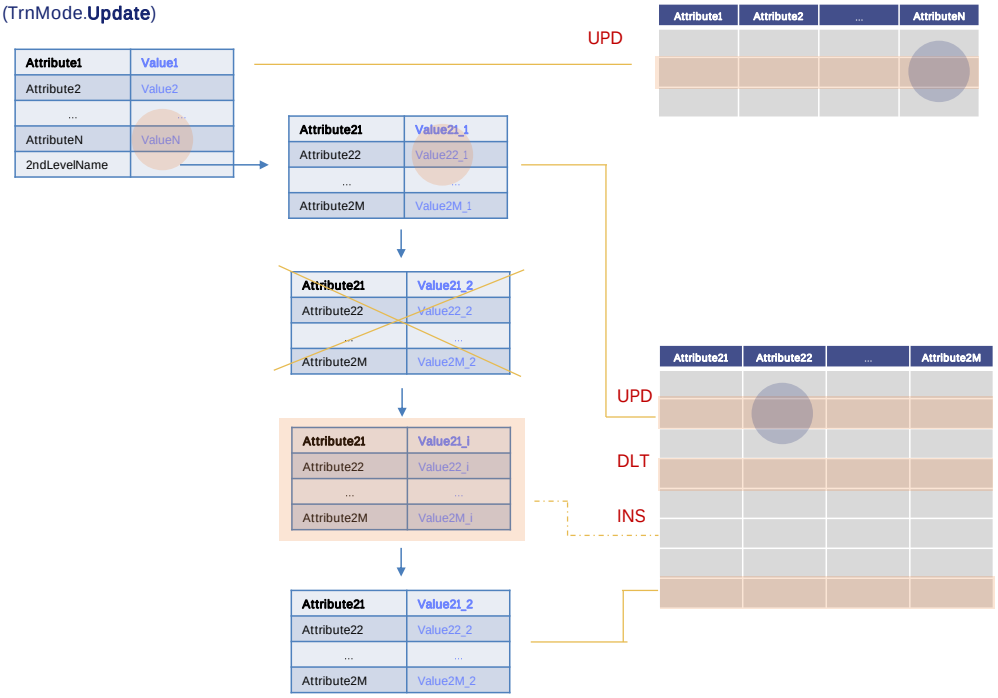
TrnMode.Update

Supongamos que tenemos una transacción de dos niveles, con la propiedad Business Component prendida, y una variable de ese tipo de datos.

Habíamos visto previamente que dependiendo del modo de la variable lo que se hará internamente al ejecutarse el método Update.

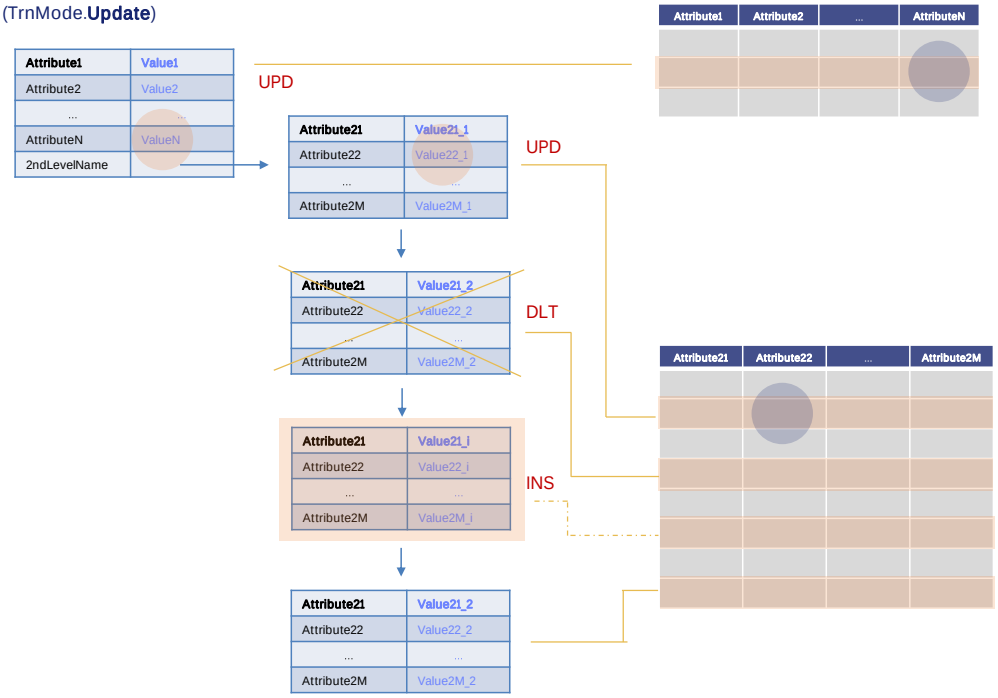
Si la variable se encontraba en modo Update, entonces se actualizaba su cabezal (independientemente de que se hubiera o no modificado alguna de sus propiedades –es decir, elementos, es decir, atributos-) y si se había realizado algo sobre la colección de líneas se reflejaba eso en los registros de la base de datos. Entraremos aquí en un poco más de detalle.

&BC (TrnMode.**Update**)



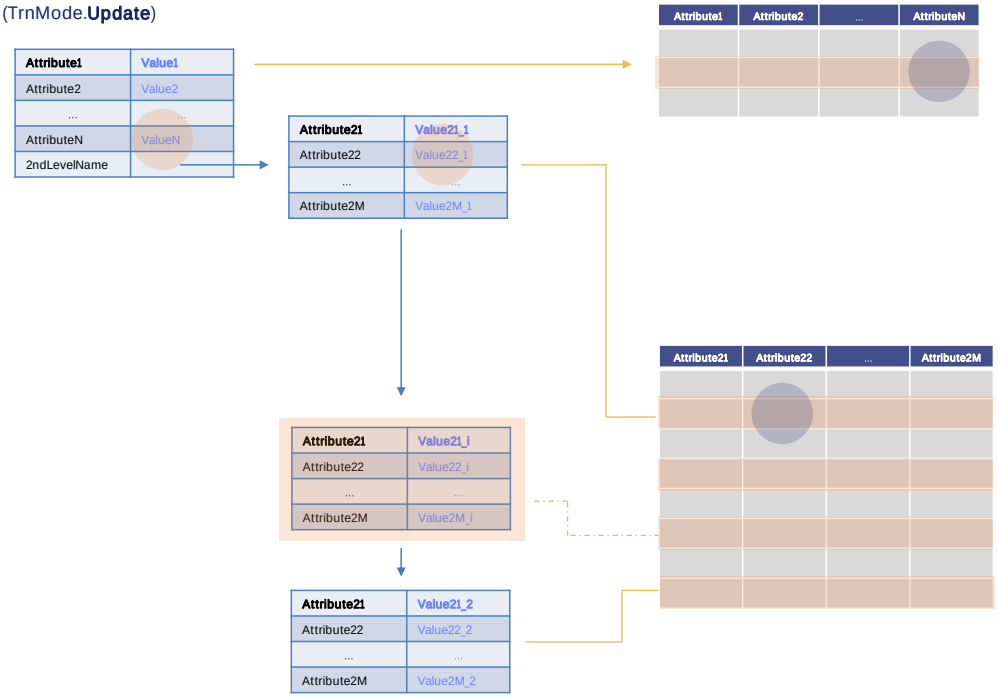
¿Cómo se opera en verdad por detrás? ¿Se va a la base de datos en busca de las diferencias entre los registros existentes y los datos que contiene el BC al momento del Update?

&BC (TrnMode.**Update**)



¿O se mantiene en la memoria del BC todo lo que se realizó sobre él como para hacer repercutir esas operaciones ahora sobre la base de datos, sin chequear exactamente la consistencia entre memoria y base de datos?

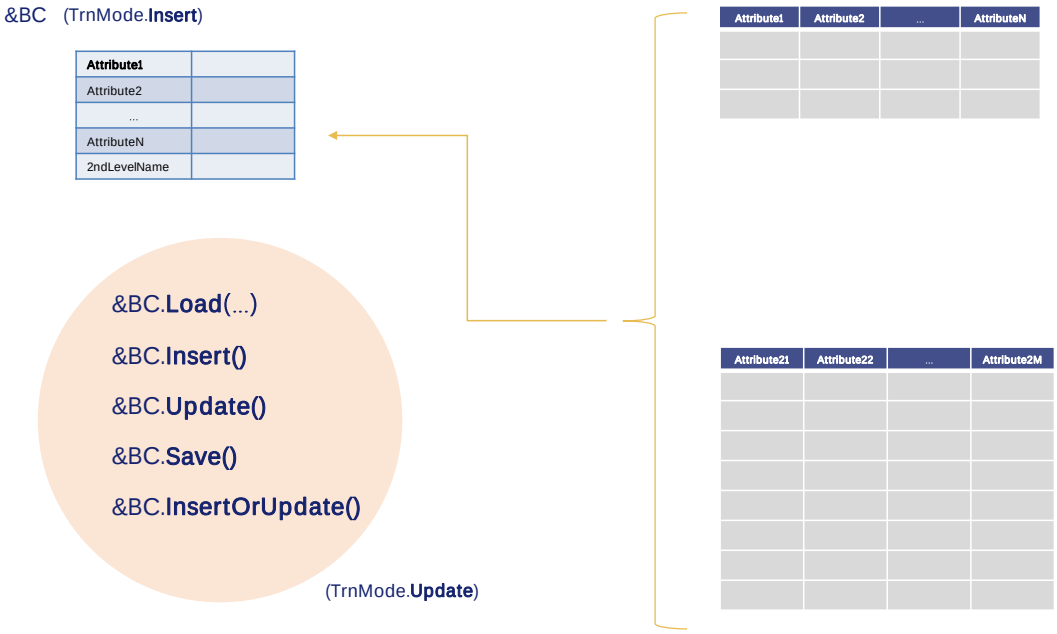
&BC (TrnMode.**Update**)



Si la respuesta fuera la primera, entonces tendría que accederse al registro en la tabla del cabezal y comparar todos los campos, para ver si alguno cambió y así actualizar en el registro el campo particular o los campos particulares que hayan cambiado. O directamente sobrescribir el registro para ahorrarse la comparación.

Y luego acceder a todos los registros correspondientes a las líneas, y si alguno no está en la colección del BC, entonces borrarlo de la base de datos. Y luego recorrer la colección y para cada ítem ver si el registro existe, y si es así ver si algo cambió como para modificarlo o sobrescribirlo directamente. Si en éste nada, entonces podríamos dejarlo como está. Y luego nos quedaría este ítem que no tiene un registro en la base de datos, así que se inserta.

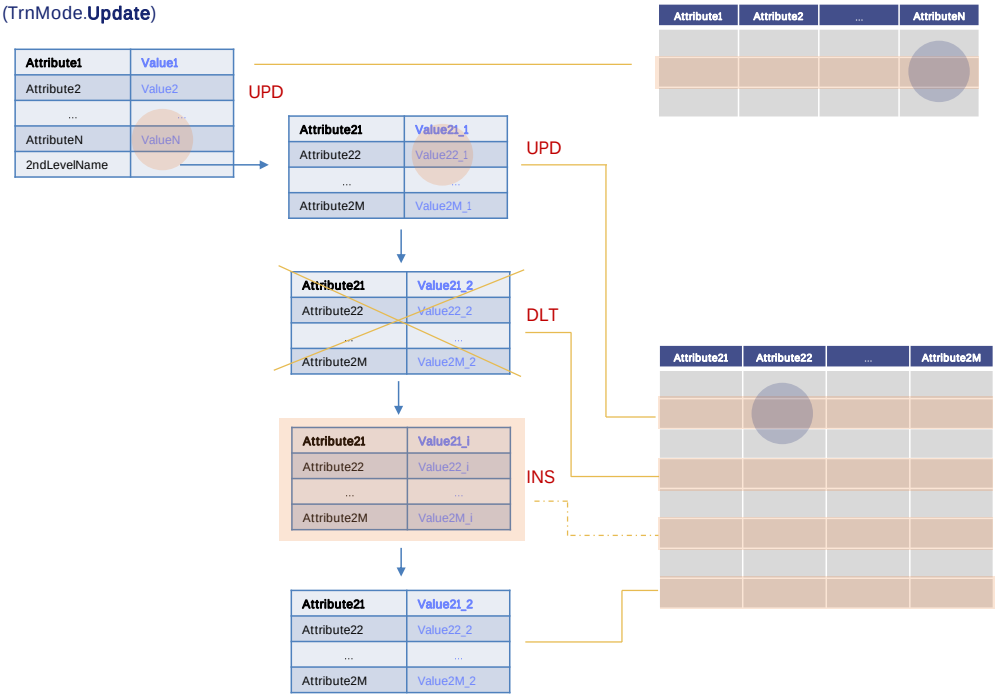
Hacer toda esta comparación sería muy costoso. No es lo que se hará.



Lo que en verdad hace GeneXus es lo segundo. Al comienzo de la ejecución del objeto donde se encuentra toda variable BC comienza en modo Insert porque cuando se la declara ya se reserva espacio de memoria vacío para ella.

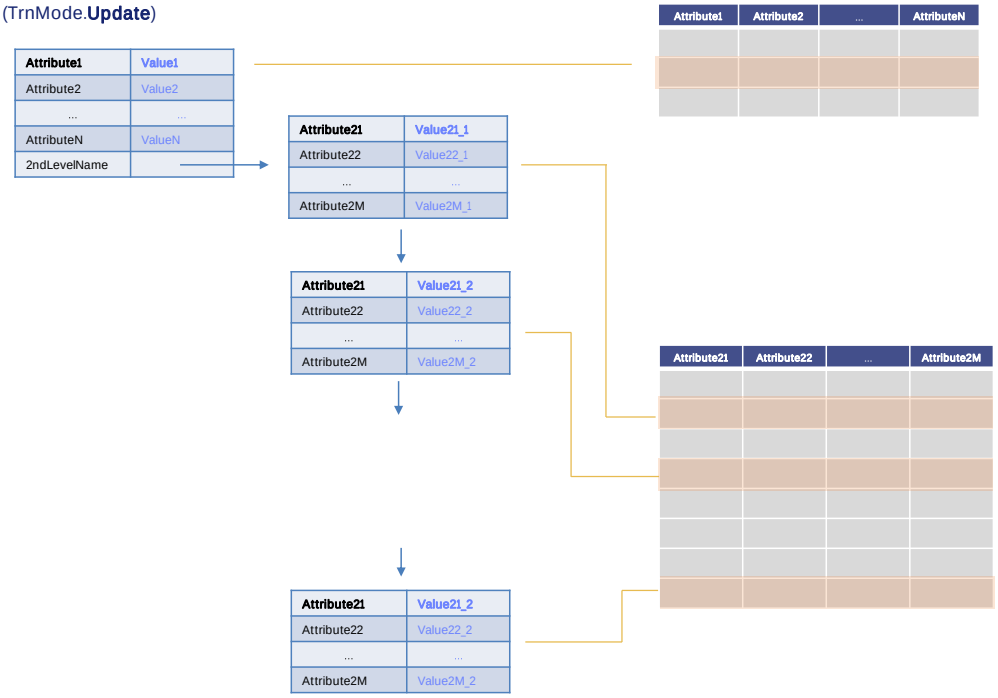
Para que una variable BC quede en modo Update tiene que habérsela cargado de la base de datos, ya sea con Load, ya sea con una operación previa (exitosa) de Insert, Update, Save, o InsertOrUpdate. Recién allí queda en modo Update.

&BC (TrnMode.**Update**)

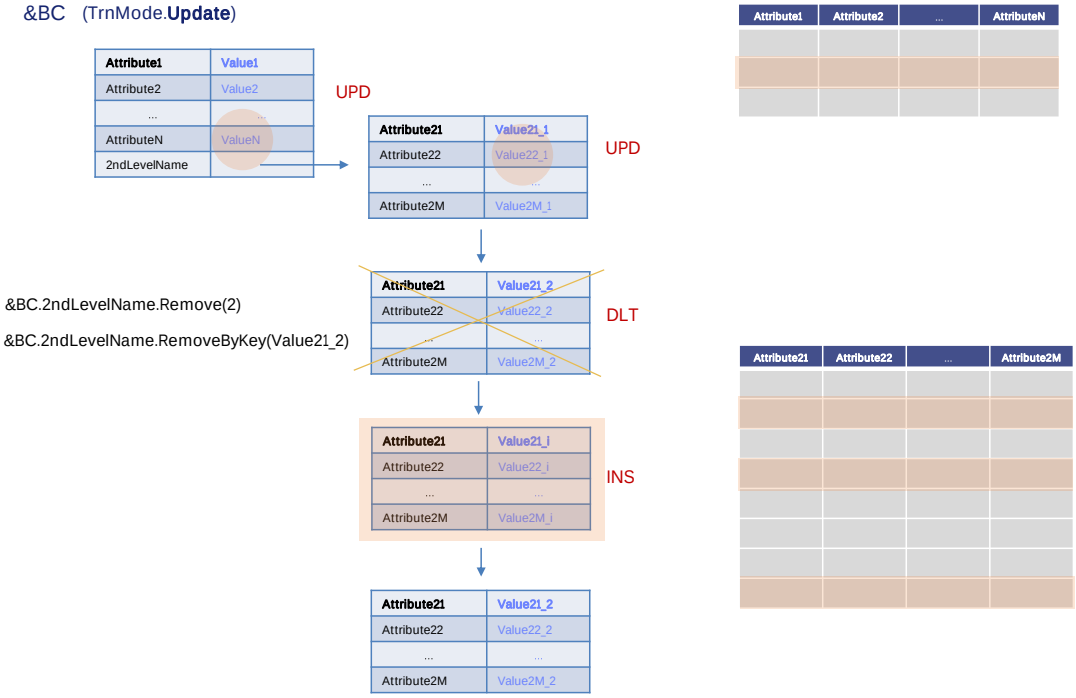


Por tanto desde la última operación que dejó cargada a la variable, al manipularla por código se va a registrar dentro de su información oculta qué se está haciendo con cabezal y con los ítems: si actualizarlos, marcarlos para ser eliminados, o insertarlos. Para luego ejecutar esas operaciones.

&BC (TrnMode.**Update**)



Vayamos más despacio. Supongamos que se hizo un Load de la variable &BC que la dejó en modo Update y con esta información (que coincide con la de la base de datos).

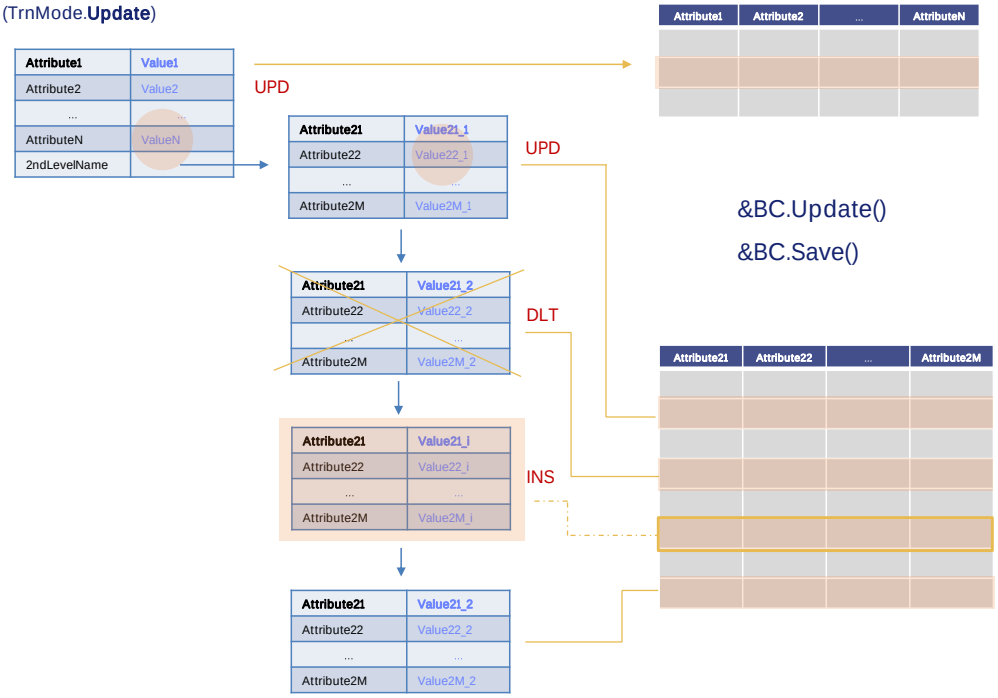


Luego manipulamos por código la variable, haciendo, por ejemplo, lo siguiente:

- modificando este valor del cabezal,
- este de este ítem,
- eliminado este otro ítem (ya sea con el método Remove de la colección, pidiendo que elimine el segundo ítem, o el RemoveByKey, al que debemos pasarle el identificador). Al hacer esto el ítem no quedará, en verdad, eliminado de la colección, sino **marcado** para ser eliminado, pero seguirá estando allí, es decir, será una eliminación lógica),
- agregando este otro,
- y dejando el último tal cual estaba.

Entonces quedan así marcados los elementos del BC: cabezal y primer ítem para ser actualizados, segundo ítem marcado para ser eliminado, tercer ítem marcado para ser insertado y el último en principio no necesitaría tener ninguna marca, porque no se hizo nada con él. Volveremos sobre esto luego.

&BC (TrnMode.**Update**)

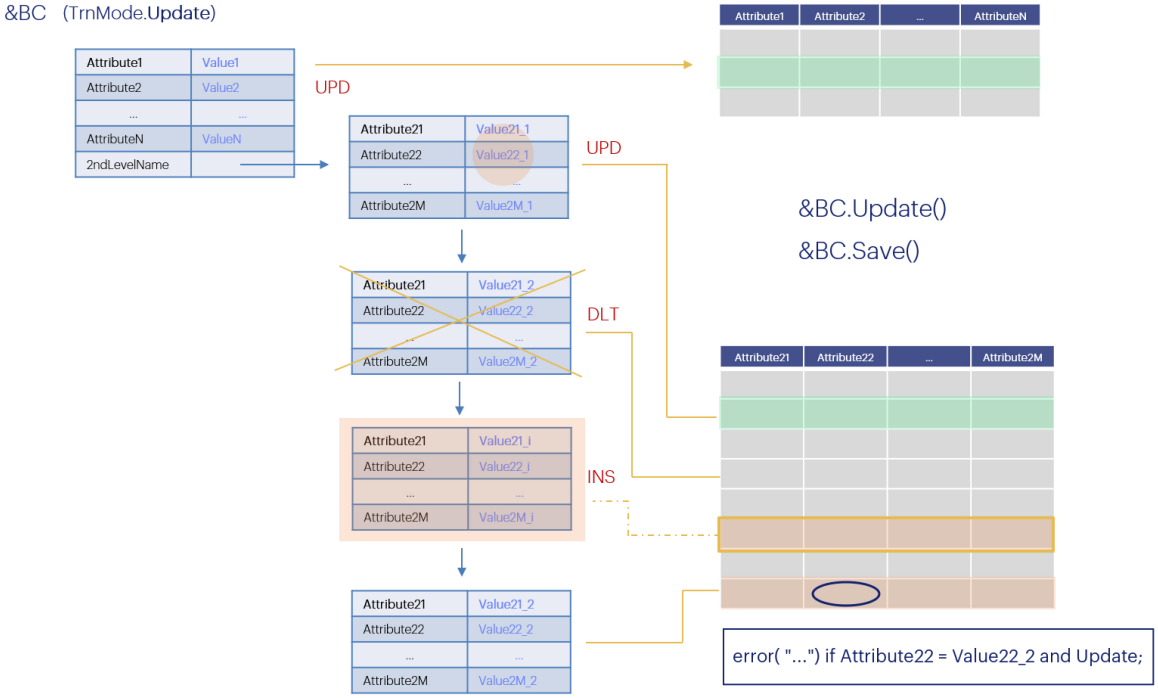


Damos la orden de Update (o Save, que en este caso es lo mismo porque la variable está en modo Update).

Va a intentar sobrescribir el cabezal, haya o no cambiado algo en él (para lo cual, claramente primero deberá chequear que exista un registro en la tabla con esa clave primaria, de lo contrario ya sabemos que fallará por PrimaryKeyNotFound). Esto es lo mismo que hace la transacción cuando se presiona Confirm en modo Update. Va a actualizar el cabezal en la base de datos así el usuario no haya modificado ninguno de sus campos.

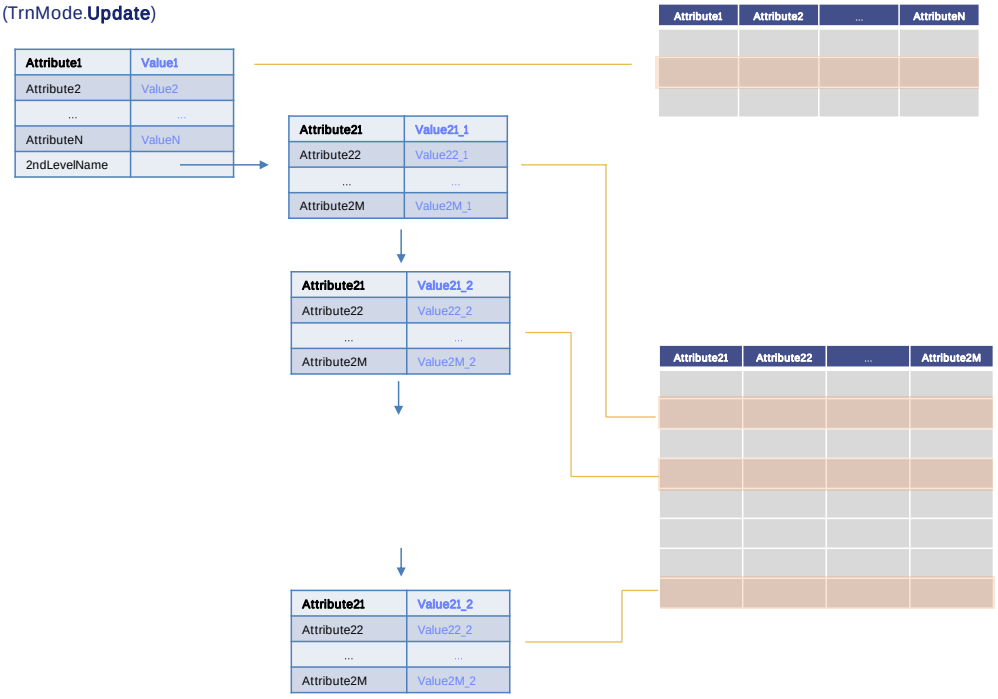
Y luego, para cada ítem de la colección:

- si está marcado como update, se sobrescribe el registro entero,
- si está marcado como eliminado entonces se busca y elimina el registro de la tabla,
- si está marcado como nuevo en la colección, entonces se inserta en la tabla,
- Y si no se hizo nada con él... bueno, aquí hay una diferencia con la transacción: igual se lo sobrescribe. Es como si quedara en modo Update.



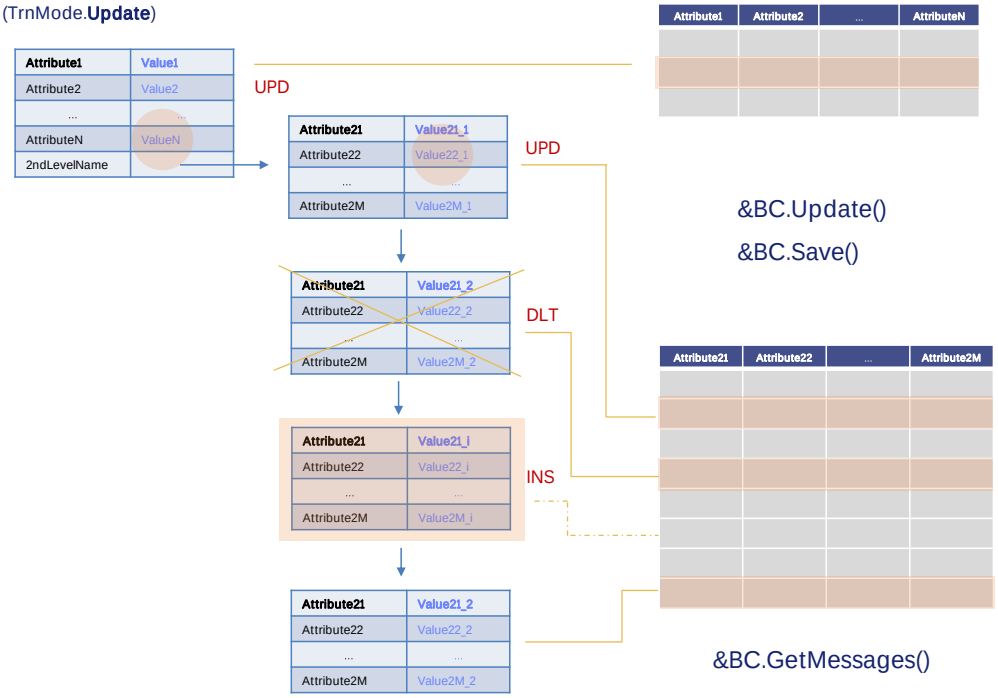
Una advertencia: en el momento en el que este video fue filmado se estaba revisando si no modificar este comportamiento para que el business component funcione igual que la transacción. En la transacción, si se está en modo Update y a una línea no se la hace nada, esa línea no se procesa en absoluto. Por tanto ni siquiera se evalúan reglas para ella. En cambio, para el BC, como sí se procesa la línea, puesto que se la actualiza de todos modos, se evaluarán y dispararán sus reglas. Es importante estar advertido de esto.

&BC (TrnMode.**Update**)



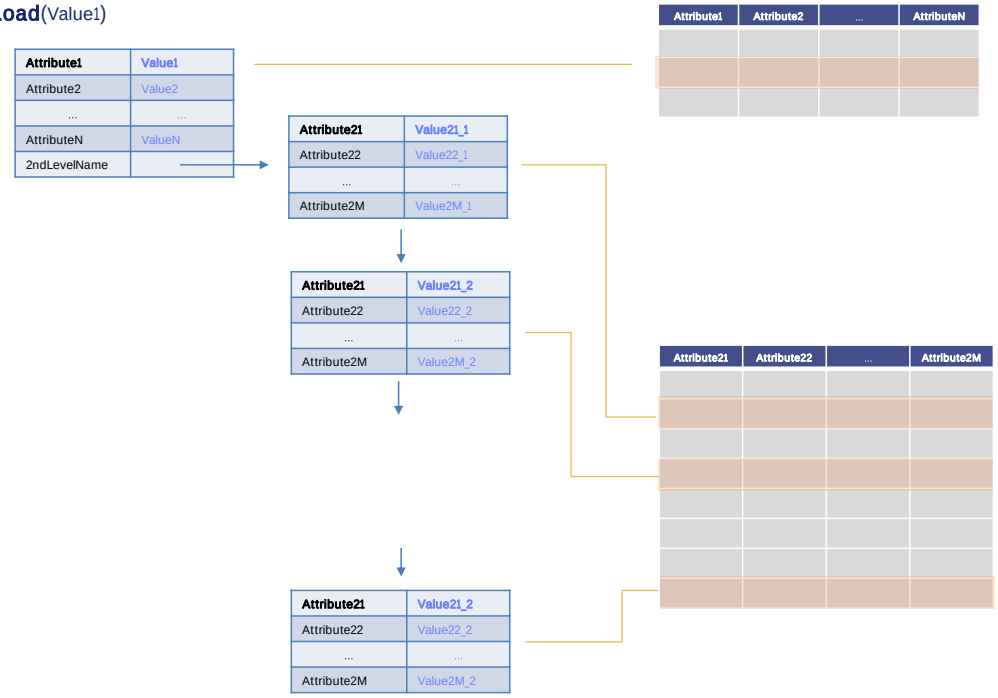
Volvamos a lo nuestro. Por supuesto esta solución reposa sobre una hipótesis: que el contenido del BC cuando fue cargado...

&BC (TrnMode.**Update**)

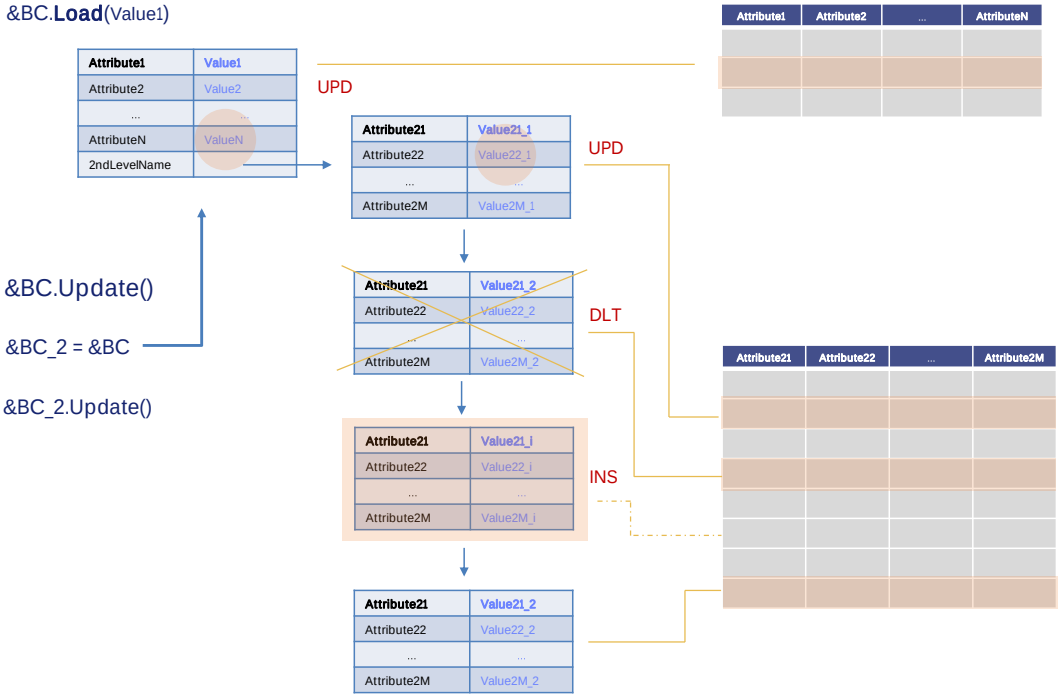


...antes de hacerle los cambios por código, coincide con el estado de la base de datos inmediatamente antes de ejecutar la operación de Update o Save. Hay que tener esto presente, puesto que si alguien cambió el estado de la base de datos para estos registros entre medio, el resultado dependerá de tales cambios, por lo que conviene hacer una revisión de los mensajes producidos tras el Update o Save.

&BC.**Load**(Value1)



¿Y qué pasaría si por ejemplo cargáramos primero la variable &BC de la base de datos...

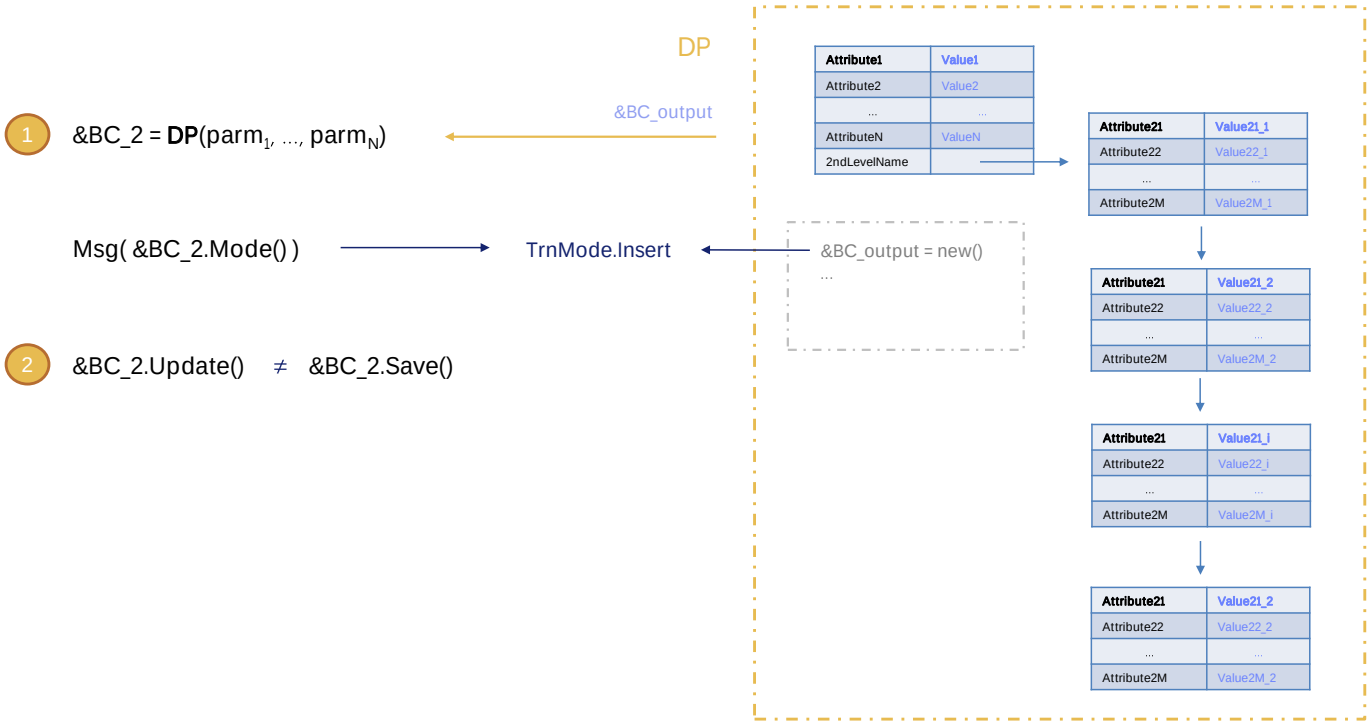


... la manipuláramos como hicimos aquí, y luego en lugar de hacer un Update o Save, se la asignáramos a otra variable BC, por ejemplo a una a la que le hubiéramos aplicado un new antes de la asignación? Y luego le aplicáramos el Update a esta nueva variable.

Podríamos creer que aquí la cosa se complejiza, porque la variable &BC_2 estará en modo Insert y no Update, por el new. Y también podríamos vernos llevados a creer que se habrá perdido el registro de las operaciones que se efectuaron sobre los ítems. Pero nada de esto será cierto.

Si bien la variable &BC_2 quedó en modo Insert luego de esta asignación, esta otra echará por tierra aquello. Es que de hecho ya no apuntará más al nuevo espacio de memoria, sino que pasará a apuntar al de la variable &BC. Es que la asignación no hace una copia de lo asignado, sino que apunta a exactamente ese mismo lugar, como un puntero. Por este motivo asumirá tanto el modo de la variable &BC, como el registro de las operaciones. Y cuando llegue el pedido de Update, será lo mismo que haber hecho el Update de la variable &BC.

Si el modo de &BC_2 en este punto es Insert, en este otro será el modo de &BC, que en nuestro ejemplo era Update.



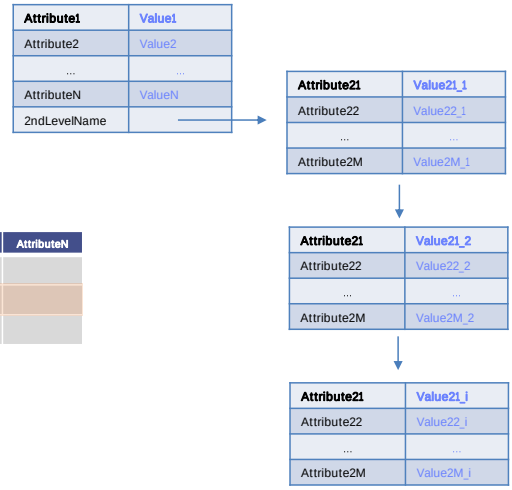
¿Por qué esto no funciona así cuando a una variable BC le asignamos el resultado de un Data Provider?

Si aquí preguntamos el modo de la variable &BC_2 nos sorprenderá encontrar que es Insert. Es que el Data Provider lo primero que hace internamente es un new, por lo que la variable que va devolver va a estar, indefectiblemente, en modo Insert.

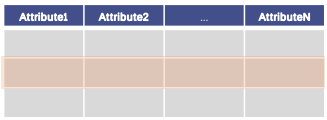
Por tanto este caso es equivalente al que no analizamos aún aquí pero sí en videos del curso Advanced: qué pasa cuando la variable está en modo Insert y se pide una operación de Update. Aquí no da lo mismo que la operación que se pida sea la de Update y no la de Save.

1 &BC = new() → TrnMode.Insert

2 &BC



3 &BC.Update() ≠ &BC.Save()



Es que, pensemos la de Save. Supongamos que primero pedimos memoria, por lo que la variable está en Insert, luego le asignamos valores a las propiedades del cabezal y agregamos 3 líneas, y luego ejecutamos el Save. Intentará insertar el BC en la base de datos, lo que solo podrá ser exitoso si los valores indicados en el BC para la clave primaria no corresponden a un registro existente en la tabla de la base de datos para el cabezal. Recordemos que el método Save siempre va a intentar realizar la operación que corresponda al modo en el que se encuentra la variable. En este caso si el registro existía, fallará el Save().

En cambio, ¿qué pasará con el Update cuando el modo es Insert? En este caso, por estar la variable en modo Insert puede suponerse que no fue cargada de la base de datos. Dicho de otro modo, es ciega al registro de la base de datos que el desarrollador está suponiendo que existe (supone que existe, claramente, porque de lo contrario no pediría explícitamente un Update).

En definitiva, el desarrollador cargó la variable con valores que no tienen por qué haberse extraído de la base de datos. Esto significa, lógicamente, que no realizó la operación de Load, porque en ese caso la variable estaría en Update si ese Load hubiera sido exitoso.
Entonces es de suponer que para la clave primaria se asignaron valores que el desarrollador supone corresponden a un registro existente, puesto que si no es así, fallará la operación de Update.

¿Cómo funciona aquí el Update?

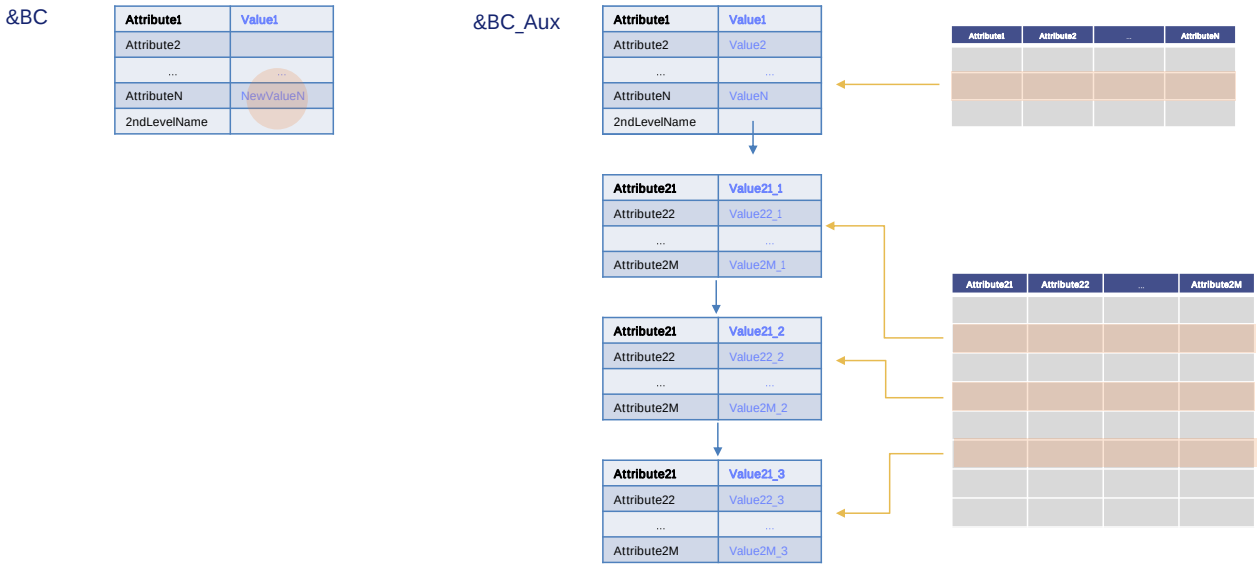
&BC

Attribute1	
Attribute2	
...	...
AttributeN	
2ndLevelName	

Attribute1	Attribute2	...	AttributeN

Attribute21	Attribute22	...	Attribute2M

Supongamos que esta es la variable sobre la que hicimos un new o que utilizamos por primera vez dentro del código, por lo que estará en modo Insert.



&BC.Update()

Luego solamente asignamos valor a la propiedad que corresponde a la clave primaria, para poder identificar al registro de la base de datos que corresponderá al cabezal. Y le damos valor a la propiedad que estamos queriendo modificar, la que corresponde a este atributo. Todas las demás propiedades las dejamos sin tocar, es decir, quedan vacías.

Luego manipulamos la colección de líneas, ahora veremos qué podemos hacer, y luego ejecutamos el método Update.

Internamente el método detecta que la variable está en Insert, por lo que creará una variable auxiliar sobre la que aplicará un Load con los valores de la clave primaria de la variable del desarrollador. Por tanto, si el registro existe se cargará en la variable auxiliar exactamente la información de la base de datos para ese identificador.

¿Qué hace luego? En el ejemplo donde solamente se modificó una propiedad del cabezal...

&BC

Attribute1	Value1
Attribute2	
...	...
AttributeN	NewValueN
2ndLevelName	

&BC_Aux

Attribute1	Value1
Attribute2	Value2
...	...
AttributeN	NewValueN
2ndLevelName	

Attribute1	Attribute2	...	AttributeN

Attribute21	Value21_1
Attribute22	Value22_1
...	...
Attribute2M	Value2M_1

Attribute21	Value21_2
Attribute22	Value22_2
...	...
Attribute2M	Value2M_2

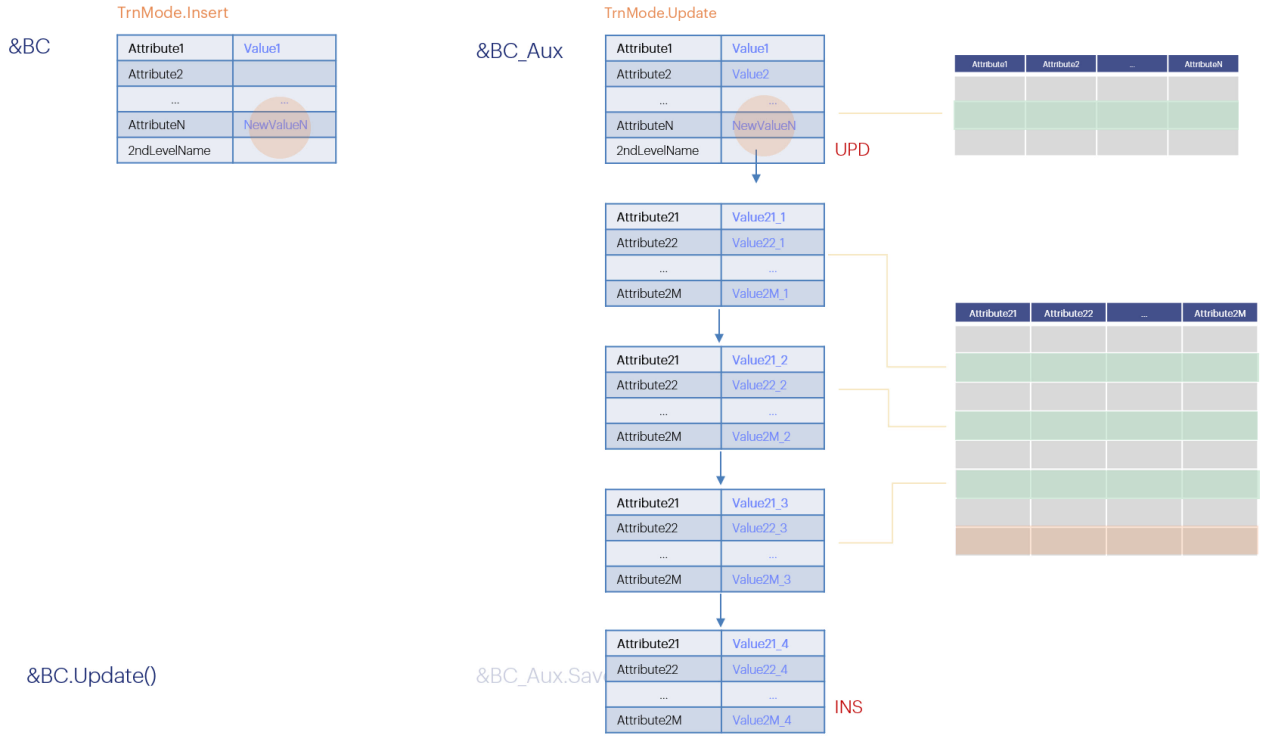
Attribute21	Value21_3
Attribute22	Value22_3
...	...
Attribute2M	Value2M_3

Attribute21	Attribute22	...	Attribute2M

&BC.Update()

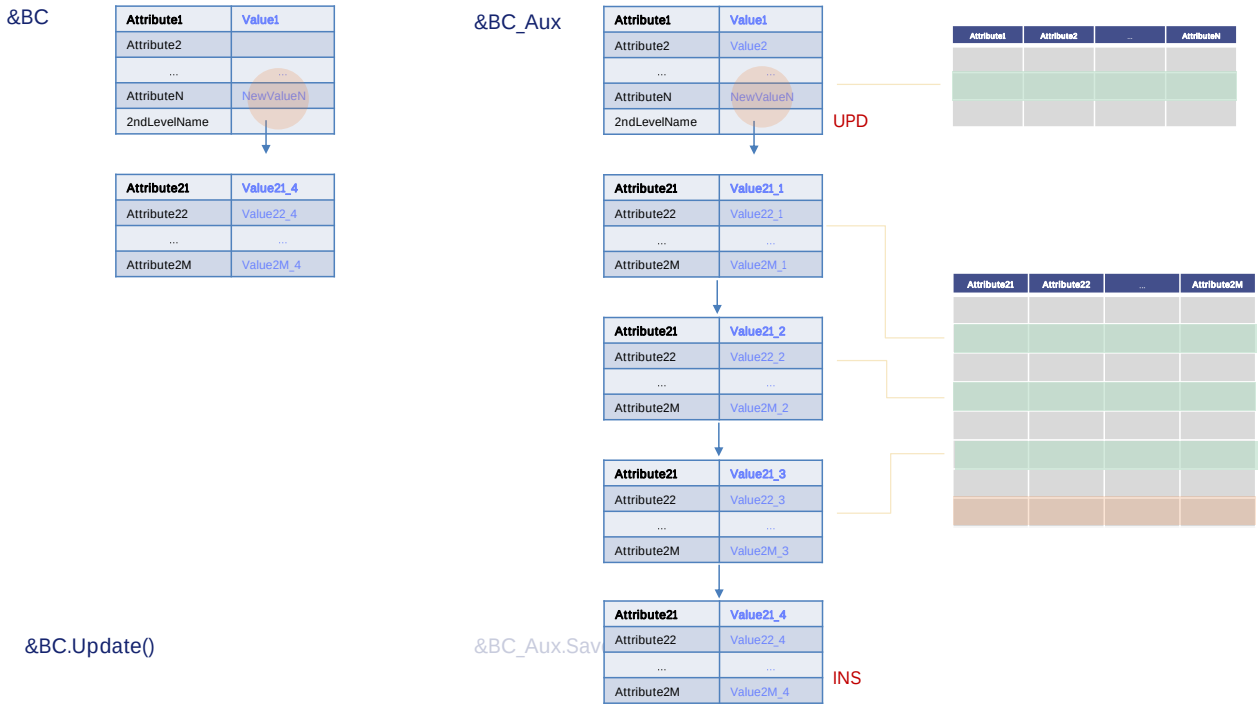
&BC_Aux.Save()

...copia ese valor para la misma propiedad de la variable auxiliar, sobrescribiéndola. Y sobre la variable BC auxiliar ejecuta un Save, que es lo mismo que analizamos al principio del video, dado que la variable auxiliar sí está en modo Update.

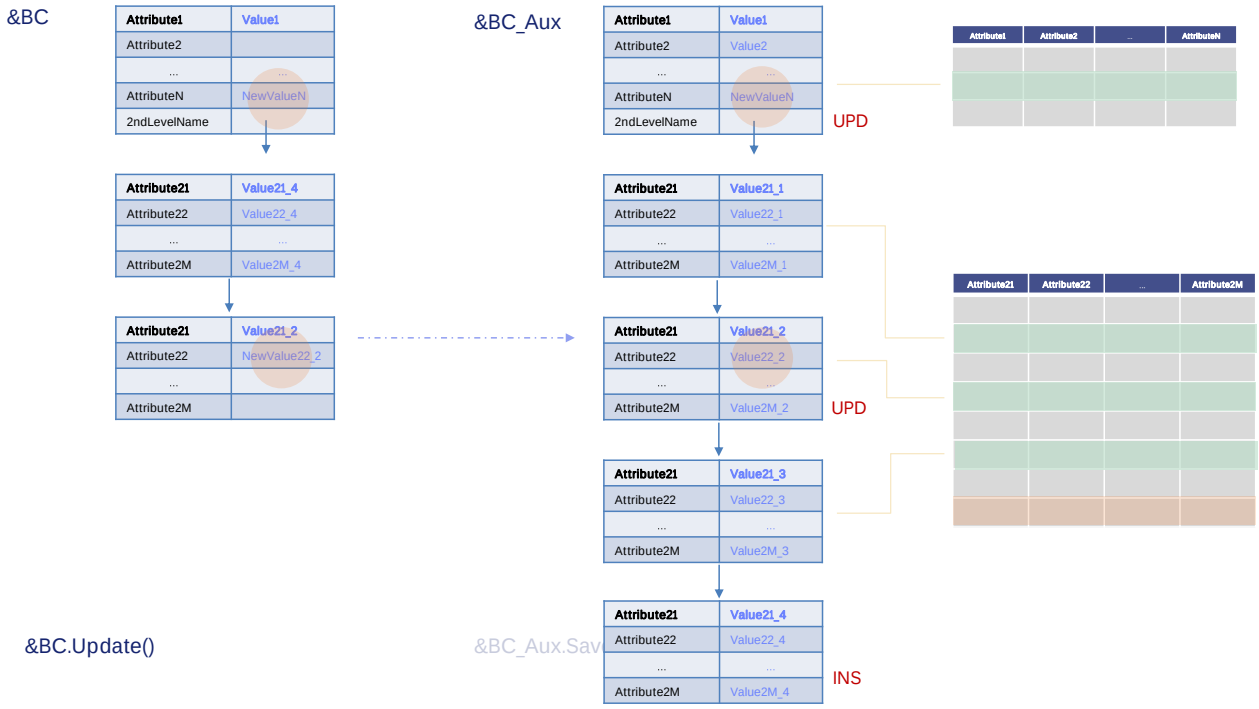


Ahora pensemos el asunto de las líneas y hagamos la explicación completa.

Supongamos que lo que queríamos era, además de modificar este atributo, agregar un registro nuevo a la tabla del segundo nivel. Para conseguir que se haga esto sobre la variable auxiliar que repercutirá en el Insert de este registro en la base de datos...

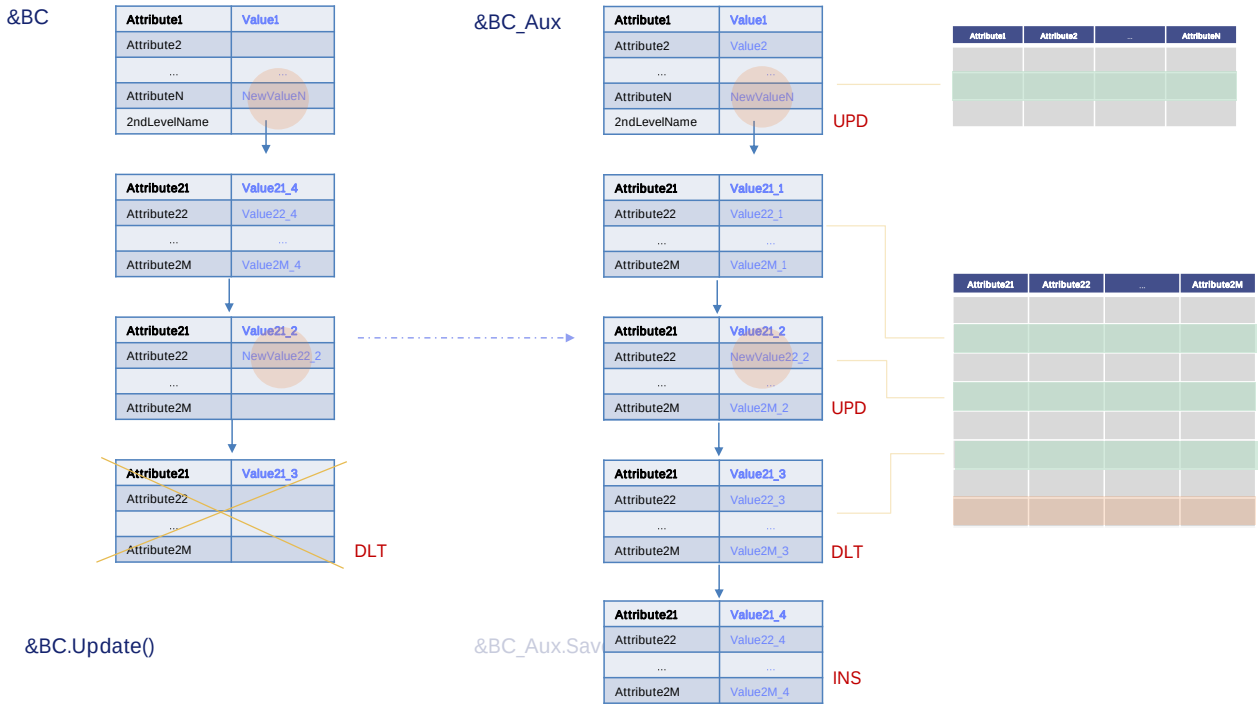


...agregamos un ítem a la colección de nuestra variable, y le asignamos todos sus valores. Lo que sucederá por detrás es que cuando se vaya a evaluar este ítem para ver qué se hace con él sobre la variable auxiliar, se buscará con el método GetByKey sobre la colección de líneas si existe o no una con este valor. Si no se encuentra, entonces se agrega.



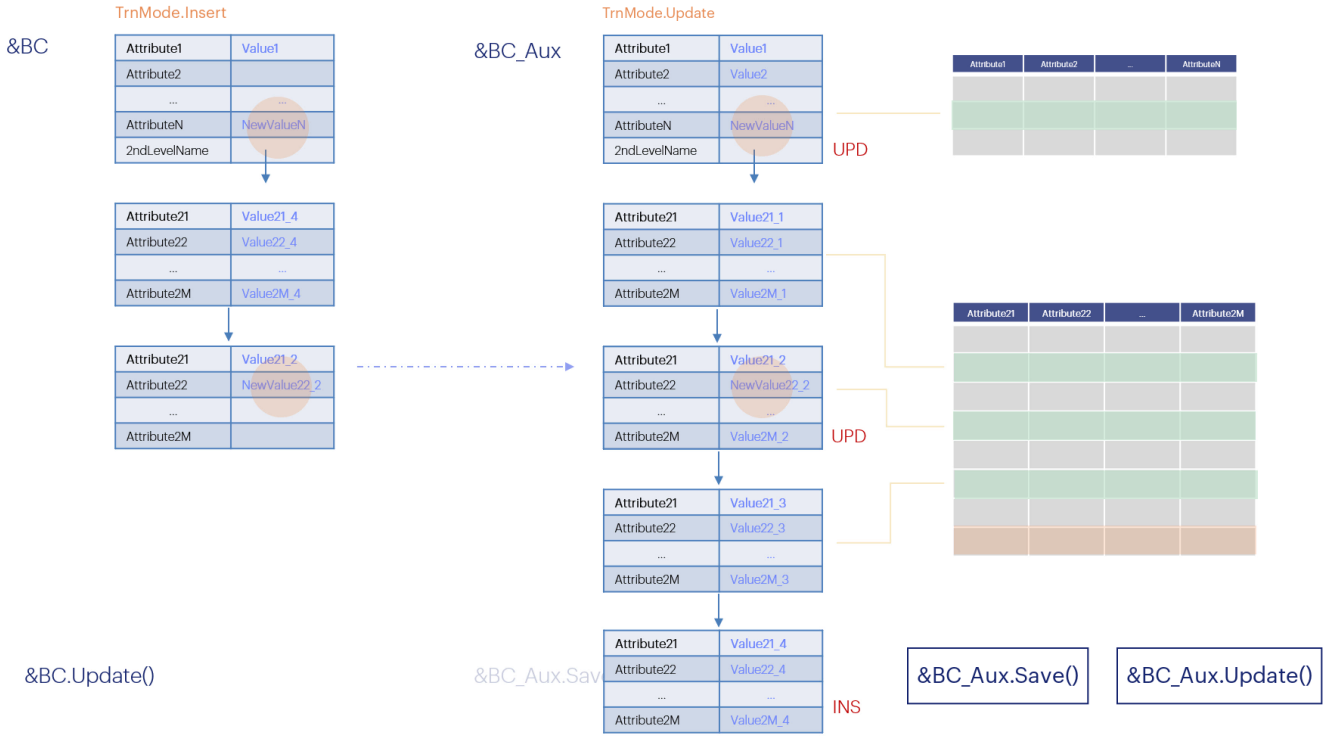
Luego supongamos que queremos actualizar un registro existente, por ejemplo este valor, por lo que agregamos otro ítem pero para el que solo asignamos valor al identificador y a la propiedad que corresponde al atributo que queremos modificar. Solamente indicamos el valor nuevo. Todas las demás propiedades las dejamos sin tocar, es decir, vacías.

Cuando se analiza qué hacer con este ítem sobre la variable auxiliar, se hace un GetByKey, y como se encuentra ítem, se copia únicamente lo que se tocó, es decir, esta propiedad.



Y por último supongamos que queremos eliminar un registro, por lo que podríamos suponer que nos alcanza con agregar otro ítem, al que solo le asignamos valor a la propiedad que lo identifica, y luego ejecutamos un Remove o RemoveByKey de **ese ítem**, de manera que quede marcado para ser eliminado.

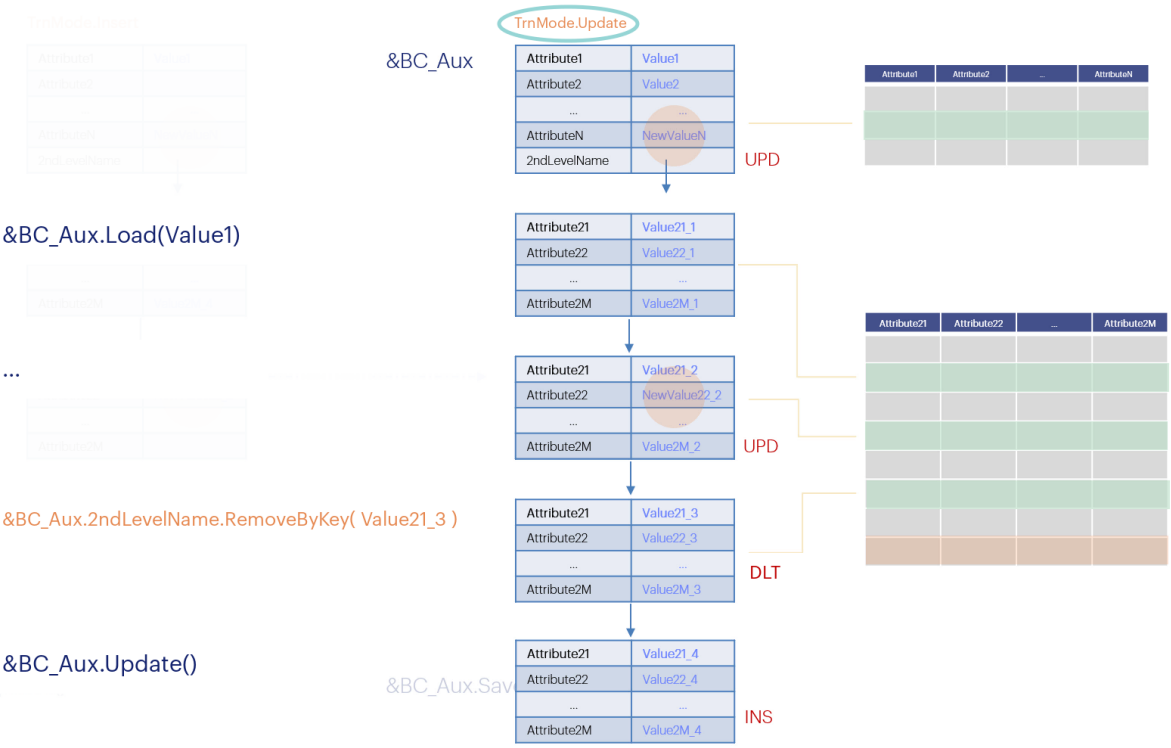
Sin embargo esto no funcionará así. Si la variable BC está en modo Insert, cuando se elimina el ítem de la colección no se lo deja marcado para ser eliminado, sino que directamente se lo elimina. Esto significa que **no** podemos **eliminar líneas** con variables BC en **modo Insert**. Necesariamente deberán estar en modo Update para poder hacer esto, porque solo en ese modo sí deja marcado el ítem para ser eliminado.



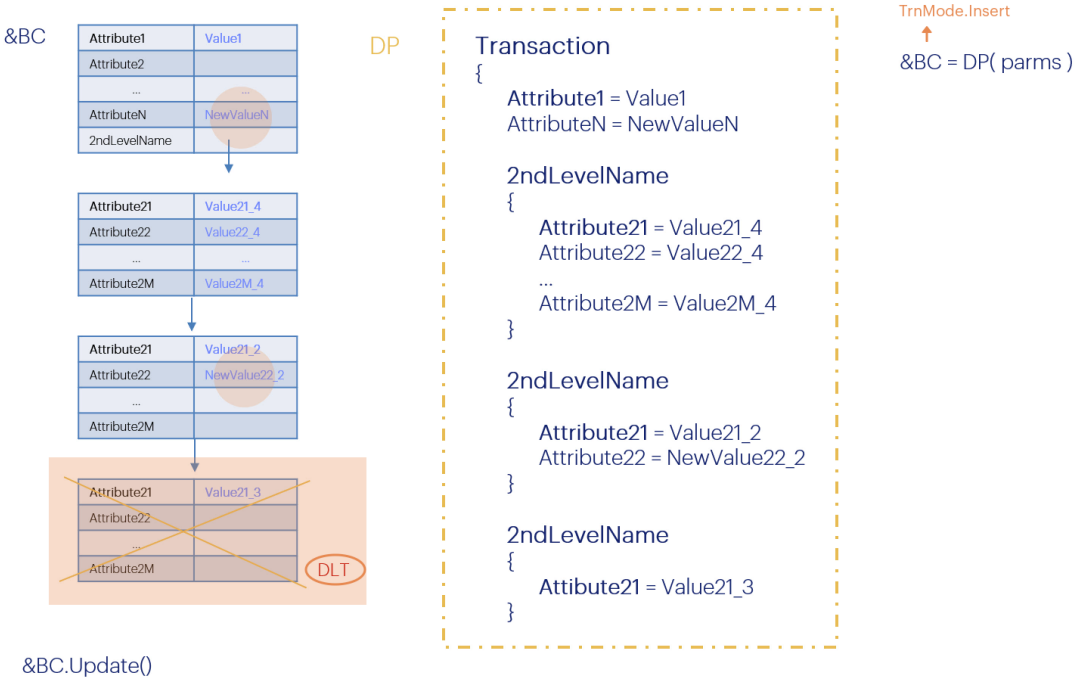
En definitiva, entonces, se utiliza una variable en Insert solo cuando se quiere modificar y/o agregar líneas, pero no eliminar.

Y lo que el desarrollador hace sobre esta variable nueva es indicar únicamente los valores que quiere modificar del cabezal, y para cada línea que quiere modificar, su identificador y valor o valores a modificar; y para una línea nueva brinda todos sus valores.

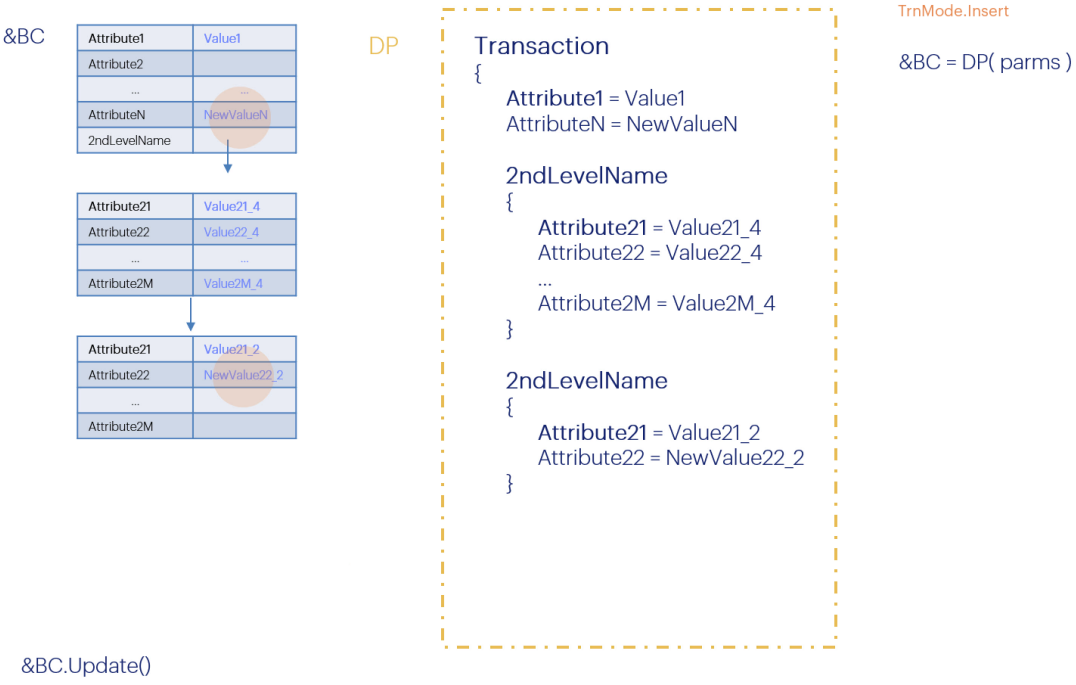
Esto tendrá las consecuencias que acabamos de analizar sobre la variable auxiliar, sobre la que luego se hará el Save(), que en ella sí coincide con el Update.



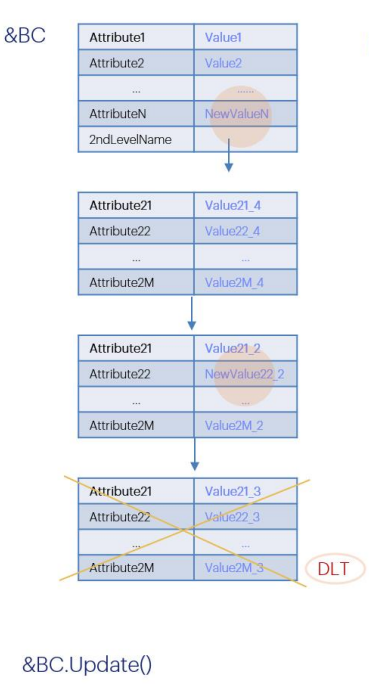
Por lo que vimos, entonces, si quisiéramos no solo modificar datos sino también eliminar líneas nos convendrá hacer un Load explícito para que la variable quede cargada de la base de datos y allí hacer todas las operaciones, incluida la eliminación de ítems.



Sin embargo, si vamos a utilizar un Data Provider para cargar la variable &BC, como en ese caso sabemos que estará indefectiblemente en modo Insert, ¿cómo logramos eliminar esta línea?



Para obtener lo que buscamos, en este caso no deberíamos agregar el ítem a la variable, claramente, sino que solamente deberíamos realizar los cambios concernientes a modificaciones y nuevos ítems...



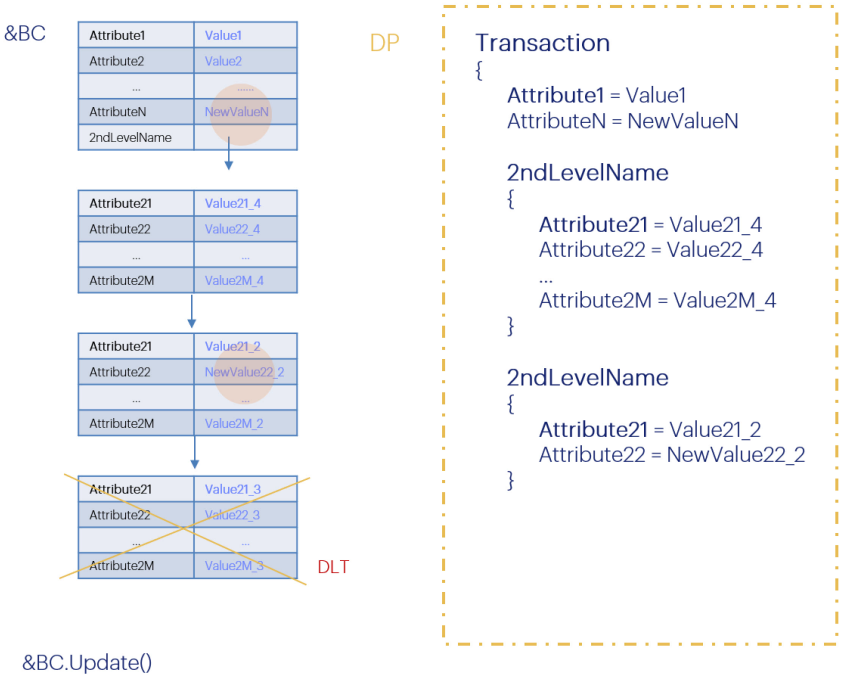
DP

```
Transaction
{
    Attribute1 = Value1
    AttributeN = NewValueN

    2ndLevelName
    {
        Attribute21 = Value21_4
        Attribute22 = Value22_4
        ...
        Attribute2M = Value2M_4
    }

    2ndLevelName
    {
        Attribute21 = Value21_2
        Attribute22 = NewValue22_2
    }
```

```
&BC = DP( parms )

If &BC.Update()

    &BC.2ndLevelName.RemoveByKey(Value21_3)

    If &BC.Update()

        Commit

    endif

endif
```

&BC.Update()

...ejecutar el Update y luego, de ser exitoso, como la variable quedará en modo Update, allí sí podemos agregar el Remove o RemoveByKey porque allí tras el Update sabemos que la variable quedará cargada con toda la información de la base de datos, incluyendo la línea que queremos eliminar. Entonces el RemoveByKey sí la marcara para ser eliminada en el próximo Update, que es lo que deberemos ejecutar para que esta acción se realice. Luego podremos commitear.



Sin embargo, en la versión 17 de GeneXus hay un bug por el cual si la variable &BC estaba en modo Insert al hacer el Update, aunque éste sea exitoso, no actualiza el contenido de la variable dejándola tal cual la base de datos y por tanto con la línea que queremos eliminar cargada. Para la versión 18 estará solucionado, pero por ahora, entonces, como workaround podríamos explicitar la carga.

Con esto concluimos el análisis teórico exhaustivo sobre cómo se actualiza a través del Business Component.

En el siguiente video veremos todo esto con un ejemplo.

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