Business Components

Update to GeneXus
from Evolution 3 to version 15

GeneXus 15

GeneXus

REVIEW

What do we already know about Business Components?

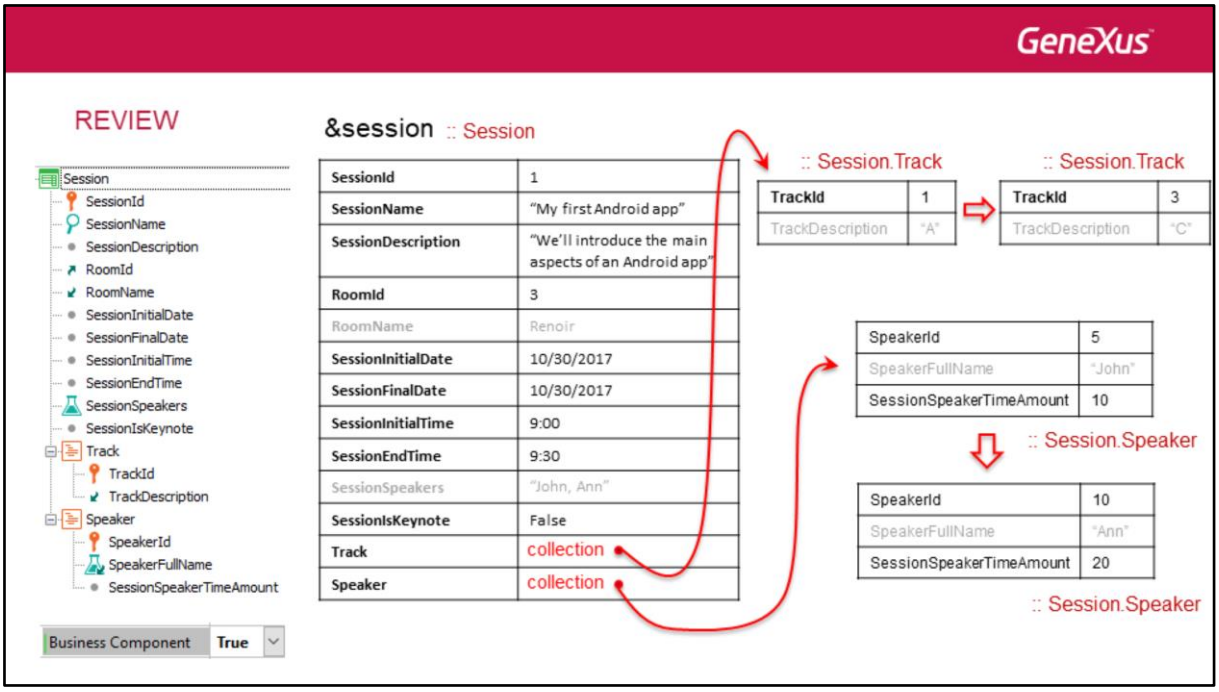
- They are structured data created from transactions, and they keep their logic.
- They are used through variables that have SDT structure (equivalent to that of the transaction).
- Operations to:
  - obtain data values for the transaction identifier,
  - perform an Insert in the database,
  - Update or
  - Delete,

  are made through methods applied to these variables.
- Specific operations exist to work with lines which are collections.

¿Qué es lo que ya sabíamos acerca de los Business Components?

- Que son tipos de datos estructurados creados a partir de transacciones, que conservan su lógica.
- Que se utilizan a través de variables que tienen estructura de SDT (equivalente a la de la transacción).
- Que las operaciones para:
  - obtener los valores de los datos para el identificador de la transacción,
  - realizar Insert en la base de datos,
  - Update o
  - Delete,

  se realizan a través de métodos aplicados a esas variables.
- Que hay operaciones específicas para trabajar con las líneas, que son colecciones.



Aquí tenemos un ejemplo. De la transacción Session que registra las conferencias de un evento se crea (prendiendo su propiedad Business Component) el tipo de datos Session, business component y un tipo de datos por cada subnivel de la transacción. En este caso dos: Session.Track y Session.Speaker.

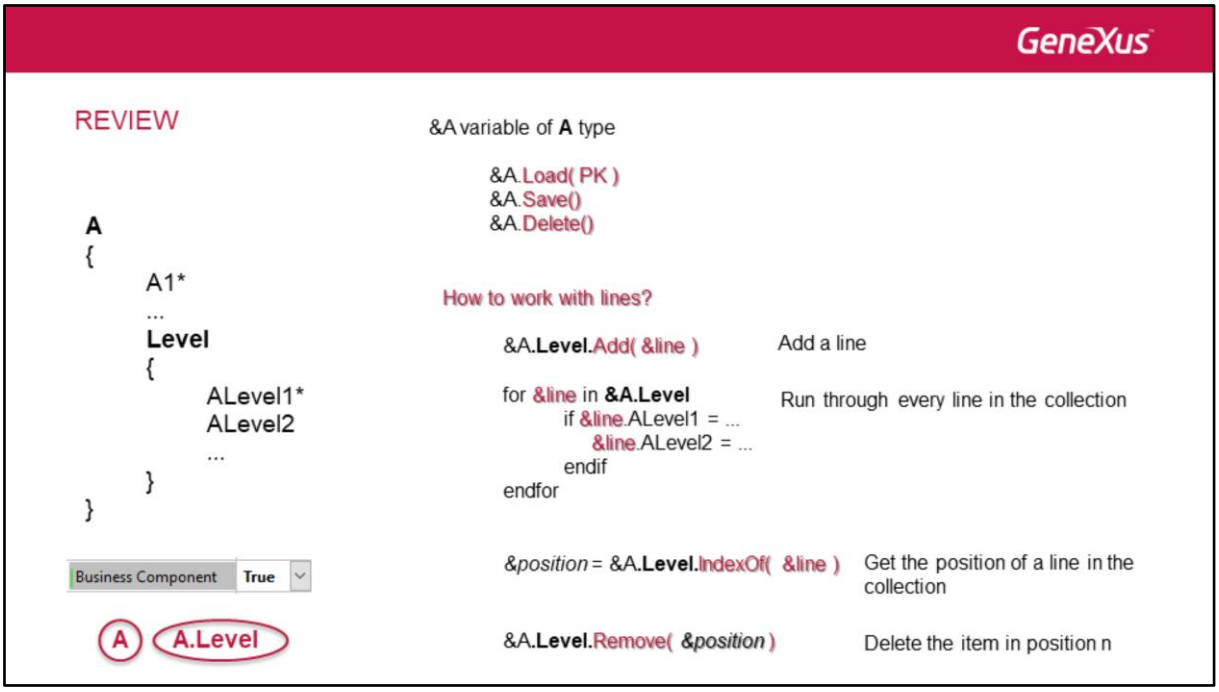
Track corresponde a los tipos de las conferencias. Y Speaker a sus oradores. Existirán transacciones que registran los tracks y los speakers. Aquí se asignan los tracks y los speakers que correspondan, de esos existentes, a cada conferencia. Por eso TrackDescription y SpeakerFullName son atributos inferidos. De la misma manera existe una transacción Room para registrar las salas en las que las conferencias se llevarán a cabo, y por esa razón RoomName será un atributo inferido.

Luego, para trabajar con los datos como si se tratara de la transacción se puede crear una variable &session del tipo de datos Session, que es como un SDT. Observe que los campos Track y Speaker serán colecciones de tipos de datos Session.Track y Session.Speaker respectivamente, que corresponden a cada línea de los subniveles Track y Speaker de la transacción.

Aquí la variable &session tiene para Track una colección de dos tracks. Y para Speaker una colección de dos oradores.

Utilizaremos este ejemplo para mostrar las mejoras en GeneXus 15 para el trabajo con Business Components.

Recordemos que los campos que son inferidos en la transacción aparecen en el Business Component para poder traer a memoria esos valores inferidos, como es el caso de RoomName. Y también que las fórmulas aparecen para lo mismo, como es el caso de SessionSpeakers, que concatena los nombres de los speakers de la session. Los atributos TrackDescription y SpeakerFullName, inferidos en la transacción, también, para las líneas, serán cargados en memoria en el Business Component, cuando corresponda.



Generalizando, aquí tenemos una transacción llamada A que tiene un subnivel llamado Level.

Al prenderle la propiedad Business Component se crean dos tipos de datos: uno con el mismo nombre que la transacción, **A**, y otro **A.Level** que es el tipo de datos de cada línea del nivel **Level** de la transacción.

La forma de cargar una variable &A con los datos de la base de datos para la primary key era utilizar el método **Load** pasándole por parámetro el valor del identificador de la transacción.
El método **Save**, recordemos, permitía grabar cabezal y líneas de la variable &A de acuerdo a su modo (si era Insert tratará de crear un nuevo A, si era Update tratará de actualizar el A que presume ya existente). Y el método Delete permitía eliminar el cabezal y sus líneas de la base de datos.

Estos métodos solamente podían aplicarse al cabezal, pero actuaban sobre cabezal y sobre lo efectuado en las líneas.

La cosa, sin embargo, se complicaba cuando teníamos que trabajar con las líneas de la transacción pues requerían operaciones de más bajo nivel, que son las operaciones para trabajar con colecciones de SDTs (pues las líneas son una colección de business component del tipo A.Level, en nuestro caso).

El agregar una línea era la más fácil de las operaciones porque sólo requería crearse una variable del tipo de datos el Business Component de las líneas (Business Component que en nuestro ejemplo es A.Level). Si le llamamos &line a esa variable, debíamos cargar los valores de los atributos de las líneas: ALevel1, ALevel2, etc., en esa variable &line y luego utilizar el método Add sobre el campo Level de la variable &A, que es la colección de líneas. Y con eso agregábamos, entonces, una nueva línea.

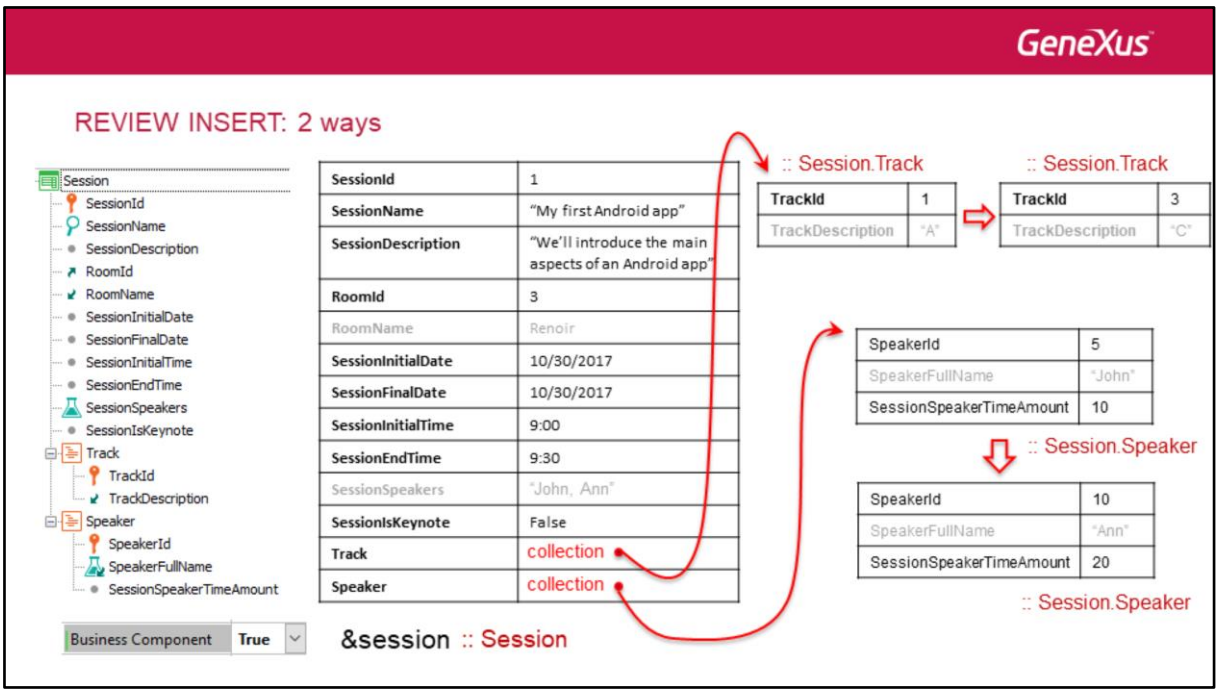
Se complicaba bastante más el actualizar una línea existente o, aún más, eliminarla, porque había que ubicarla primeramente en la colección de líneas. Para ello se solía utilizar el comando for in, que permitía acceder por referencia al ítem concreto, para así modificarlo.

La eliminación era con el método **Remove** de una colección, que necesitaba la posición del ítem a

ser eliminado. Para obtener la posición de un ítem teníamos el método **IndexOf**.

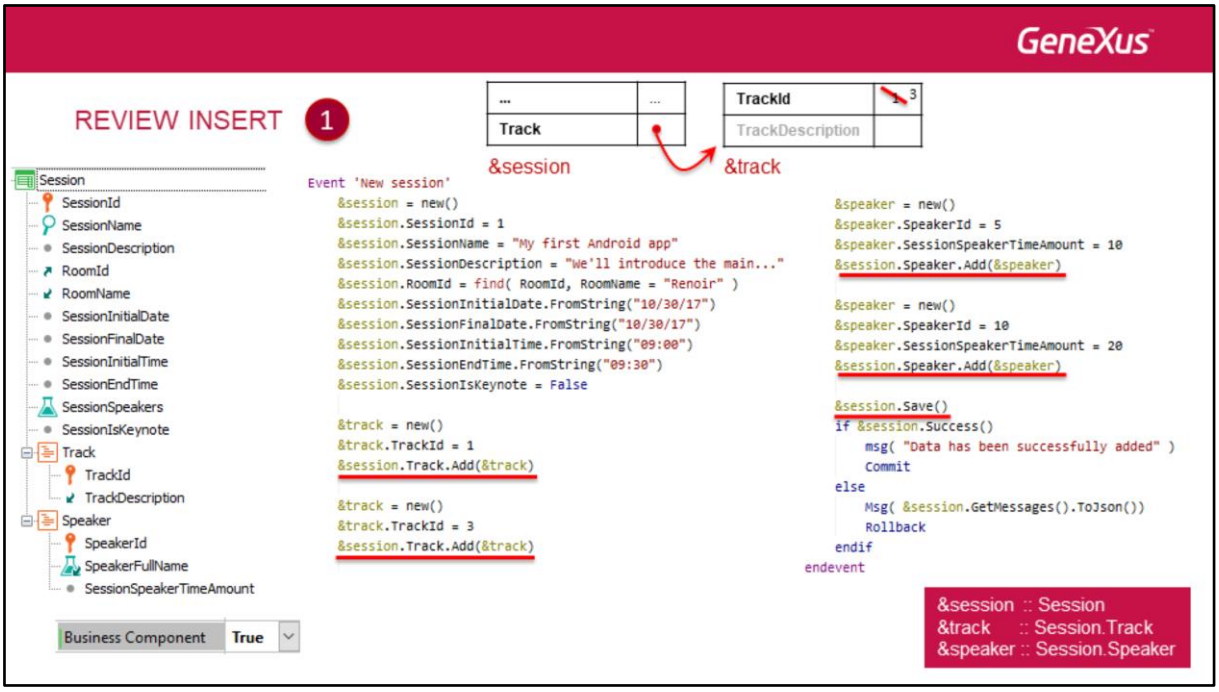
Luego de hacer las operaciones correspondientes con las líneas –ítems de la colección- (como agregar una nueva, modificar el valor de una existente, eliminar otra) había que aplicar el método **Save()** para que pudieran tomar efecto en la base de datos todas esas operaciones, realizándose todos los controles correspondientes (de duplicados, de integridad referencial, disparo de reglas de la transacción).

Veremos cómo las operaciones con las líneas se han simplificado con GeneXus 15, y cómo se han simplificado y especializado los métodos para el Business Component raíz.



Aquí vemos el ejemplo que presentábamos al principio donde queremos cargar una variable &session del tipo el business component Session, con cabezal y dos líneas (ítems) para Session.Track y dos líneas (ítems) para Session.Speaker.

Tenemos dos formas de cargar la variable &session antes de salvar los datos en la base de datos.



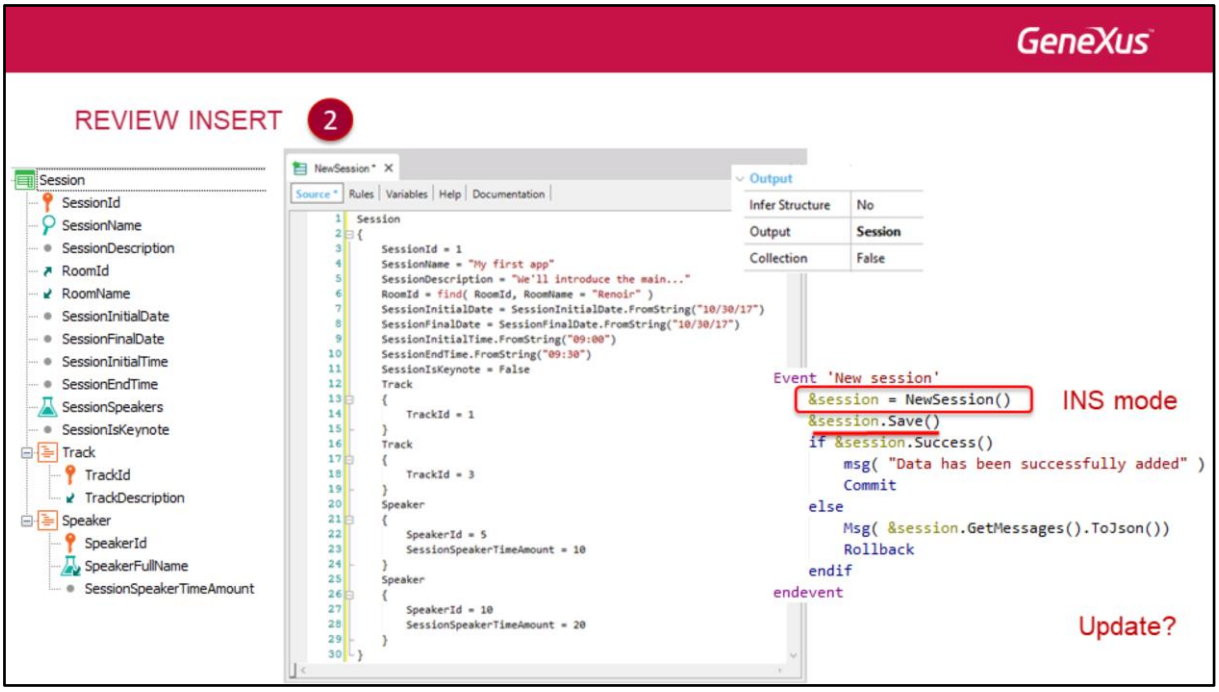
La primera es llenando (por ejemplo dentro de un evento de un web panel) los campos simples del business component &session (todos los que no corresponden a atributos inferidos) , y luego agregando las dos líneas para la colección &session.Track a través del método add, utilizando una variable auxiliar &track del tipo de datos el Business Component de cada línea Track: esto es, Session.Track; y agregando dos líneas para la colección &session.Speaker, también a través del método add, utilizando variable auxiliar &speaker del tipo de datos Session.Speaker, el Business Component correspondiente a cada línea del subnivel Speaker.

Recordemos que debemos pedir memoria cada vez, con el operador new. ¿Por qué?

Teniendo la variable &session, primero tenemos la colección de tracks vacía. La primera vez no necesitamos pedir memoria para la variable &track, porque por estar definida ya se tiene memoria “limpia” para ella. De todas formas es buena práctica hacerlo. Se asigna el valor 1 para TrackId y al utilizar el método Add, se agrega esta posición de memoria a la colección. Si luego no pedimos nuevamente memoria para la misma variable &track lo que sucederá es que al hacer &track.TrackId= 3, ¡se estará modificando ese valor para el primer ítem de la colección! ¡Y ahí estamos en problemas!

Una vez que se cargan los valores de cabezal y líneas en la variable compuesta &session, se realiza la operación de Save, que como la variable &session se inicializó vacía y no se le aplicó ningún Load que la deje en Update, estará en modo Insert, e intentará, entonces, realizar una inserción en la base de datos de cabezal y sus líneas.

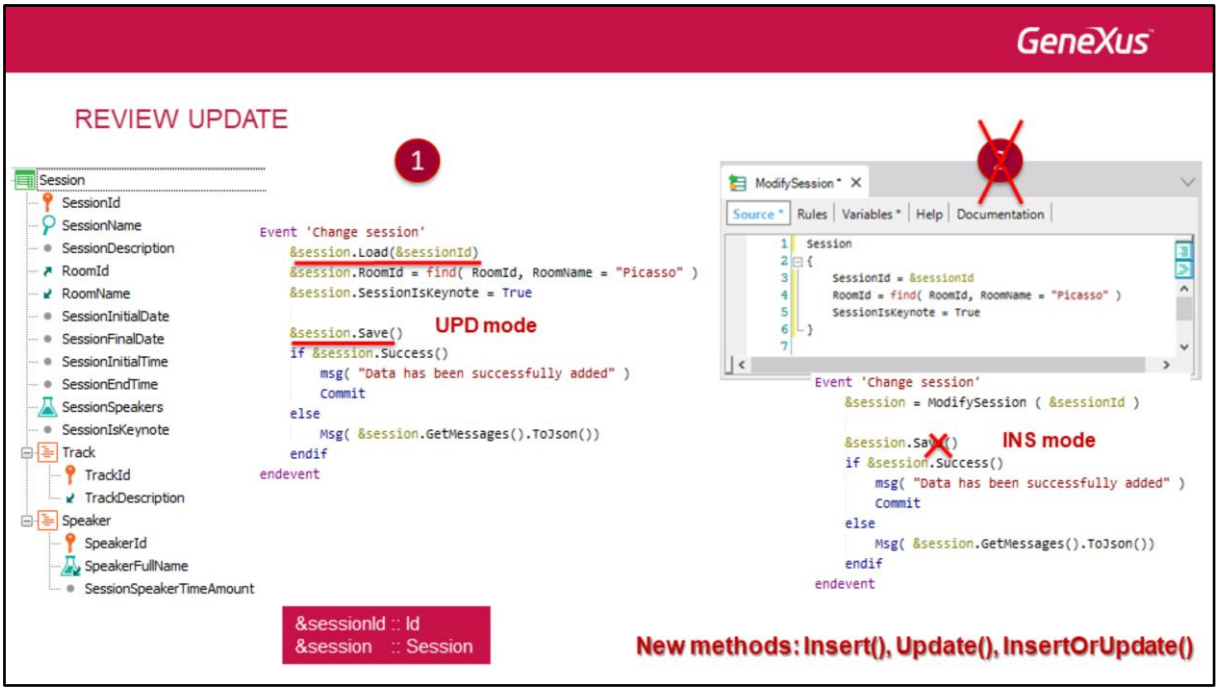
Luego, si la operación es exitosa (método Success devuelve true), se envía mensaje al usuario y se realiza commit. De lo contrario se obtienen los mensajes producidos, a través del método GetMessages que devuelve colección de SDT, Messages, y se lo transforma a Json para mostrarlo rápidamente. La otra opción era recorrer esa colección de mensajes e ir desplegando cada uno.
Y luego se realiza un rollback por si el error dio en alguna línea y por tanto ya se hubieran insertado cabezal y líneas anteriores (suponiendo, claro, que no se quiere dejar en la base de datos cabezal sin líneas, o cabezal y alguna línea, sin todas las demás).



La segunda forma de cargar la variable &session para poder hacer la operación de Save() es utilizar un Data Provider, como se muestra aquí. Que devolverá algo del tipo de datos el Business Component Session. Esto es mucho más fácil que la alternativa anterior.

De hecho, recordemos que alcanza con arrastrar la transacción Session desde el KBExplorer para el Source del Data Provider para que ya se inicialice la estructura de la izquierda, con output Session, y solamente habrá que rellenar los campos y en todo caso agregar grupos Track y Speaker para insertar más líneas, que es lo que hicimos justamente aquí.

Aquí también el Save intentará insertar, puesto que no se hizo un Load previamente sobre &session, por lo que la variable estará en modo Insert. ¿Pero qué pasaría si en verdad lo que quisiéramos fuera hacer un Update? Es decir, modificar algún dato de la session, como por ejemplo la sala, RoomId.

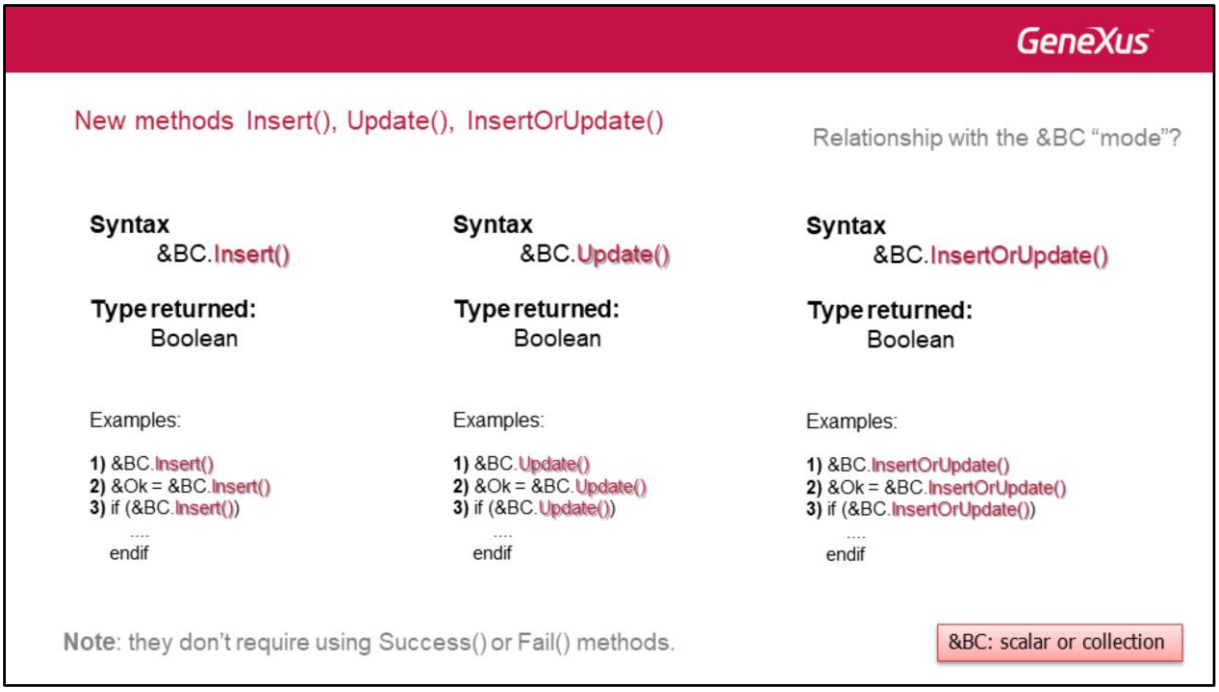


Si quisiéramos cambiarle la sala a una conferencia, y además especificar que además se trata de una Keynote, con GeneXus X Evolution 3 ya no teníamos dos opciones, sino solo una. La actualización “a mano”, por decirle de algún modo.

Una donde cargábamos con Load la variable &session, supongamos a partir de un identificador de sesión que venía por parámetro o era elegido por el usuario en pantalla en una variable &sessionId. Luego modificábamos los dos atributos que nos interesaban en esa variable &session y hacíamos Save, sabiendo que por haber utilizado el método Load la variable estará en modo Update, por lo que se intentará realizar una actualización en la base de datos.

La segunda alternativa, la del Data Provider, no podíamos utilizarla pues al no haber hecho un Load sobre &session la variable estará en modo Insert y el Save intentará insertar una nueva conferencia, en lugar de actualizarla, y eso fallará porque es de suponer que la conferencia de id &sessionId ya existe.

Para poder dejar explícita la operación que se desea y evitarse estos problemas, se han creado los nuevos métodos a nivel del cabezal: Insert(), Update() e InsertOrUpdate(), que ahora estudiaremos.



Aquí presentamos la sintaxis de los tres nuevos métodos aplicables a variables de tipo Business Component de una transacción o colección de Business Components de una transacción. Ya veremos este último caso.

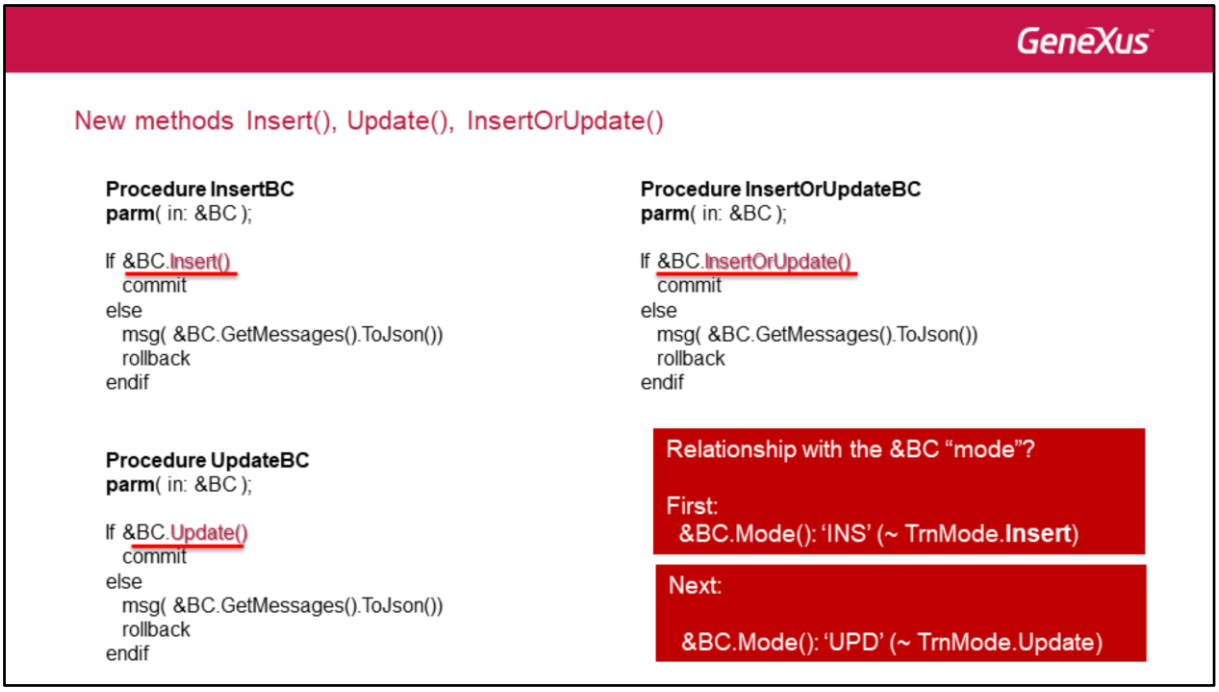
Cuando el método Insert es ejecutado, los datos previamente cargados en memoria en la variable serán insertados solamente si no falla el control de duplicados (tanto por clave primaria como por claves candidatas) y, por supuesto, si la integridad referencial no falla y no se dispara ninguna regla de error de las especificadas en la sección Rules de la transacción.

El caso del Update se resuelve como veremos a continuación, pero claramente intentará actualizar los datos de la variable BC, en la base de datos. Si el registro no existe, fallará. También fallará, lógicamente, si falla alguno de los controles de integridad referencial o alguna regla error de la transacción.

El método InsertOrUpdate intenta primero realizar un Insert pero si ya existe un registro en la base de datos para la clave primaria entonces intenta realizar un Update.

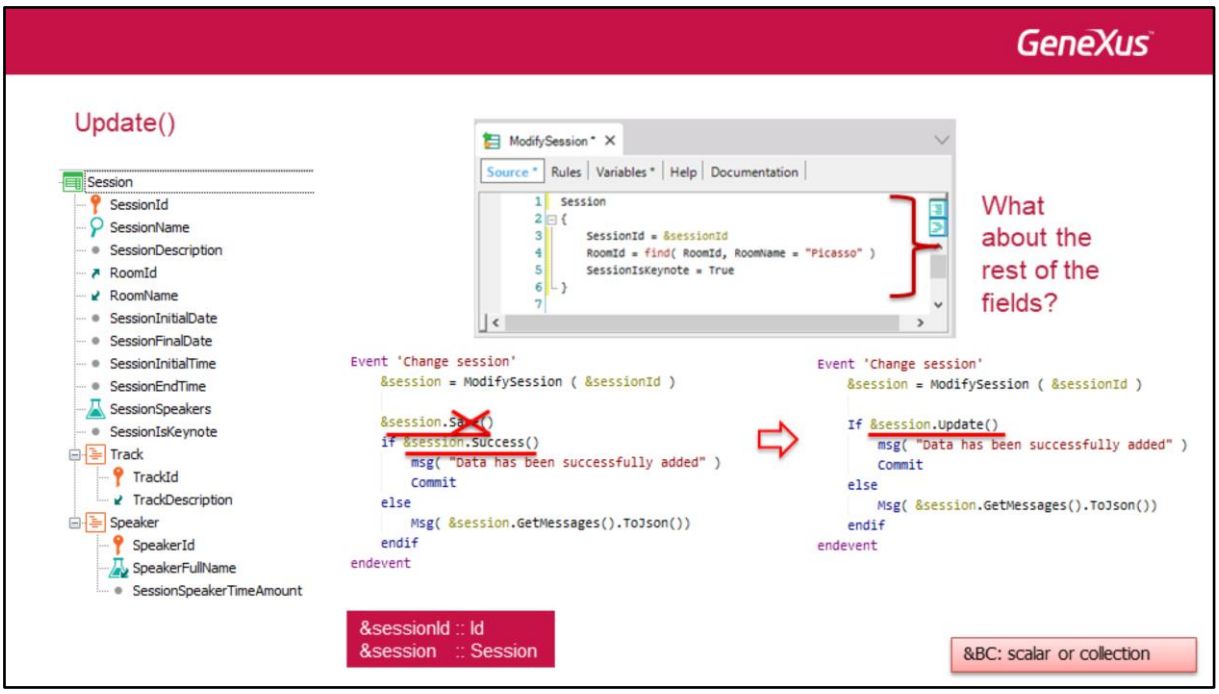
Lo importante aquí es que la operación se intenta realizar en forma independiente al modo que tuviera la variable.

Observemos también que a diferencia del Load(), Save() y Delete() estos métodos devuelven un booleano que indica si la operación fue o no exitosa. Por tanto estos métodos no requieren del método Success() o su contrario, Fail(), para obtener el resultado de la operación. El valor booleano puede ser evaluado por el desarrollador como no. Queda a su criterio.



Esto hará que ahora pueda crearse un procedimiento que reciba por parámetro una variable basada en un BC y que realice la operación que le interese, independientemente de que la variable estará siempre en modo Insert al comenzar a ejecutarse el source del procedimiento, debido a que no se hizo ningún Load sobre ella. Después de las operaciones Insert, Update o InsertOrUpdate sí, evidentemente, quedará en modo **TrnMode.Update**, si no ocurrió ninguna falla.

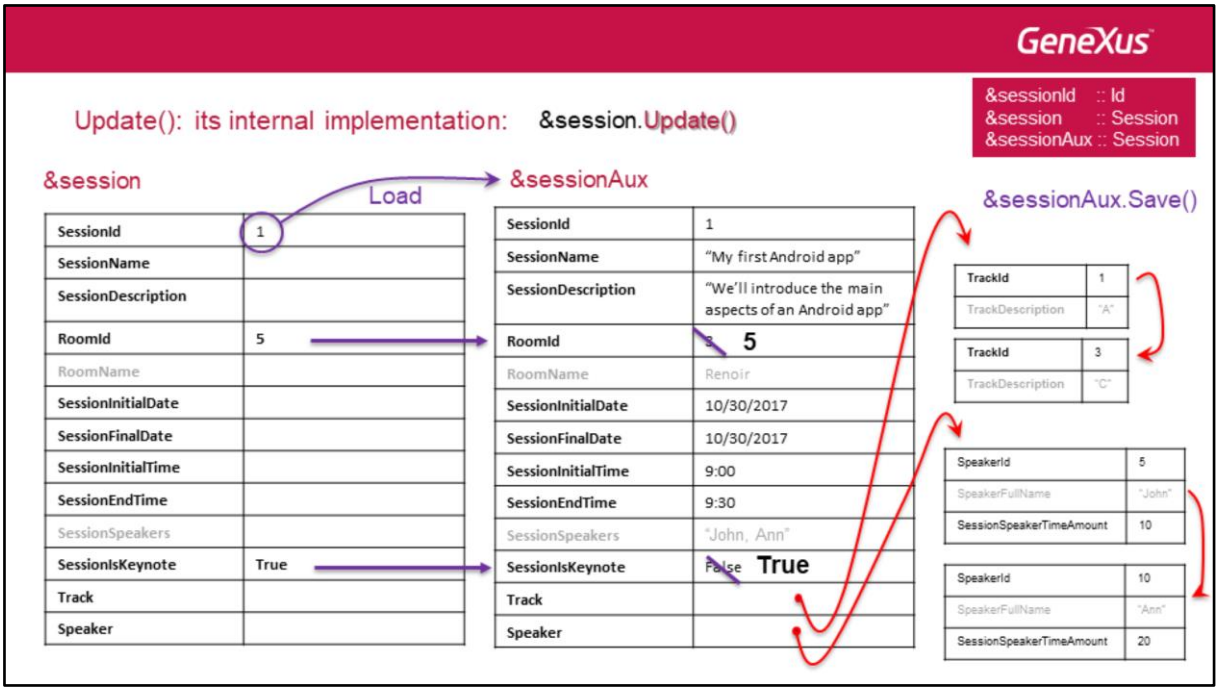
Recordemos que TrnMode es un tipo de dato enumerado, predefinido, para representar los modos Insert, Update, Delete y Display. Los valores que asume son 'INS', 'UPD', 'DLT' y 'DSP', respectivamente, que son los valores que devuelve el método Mode aplicado a una variable business component.



Aquí vemos resuelto el caso que vimos antes de Update que no podíamos resolver directamente con el Save().

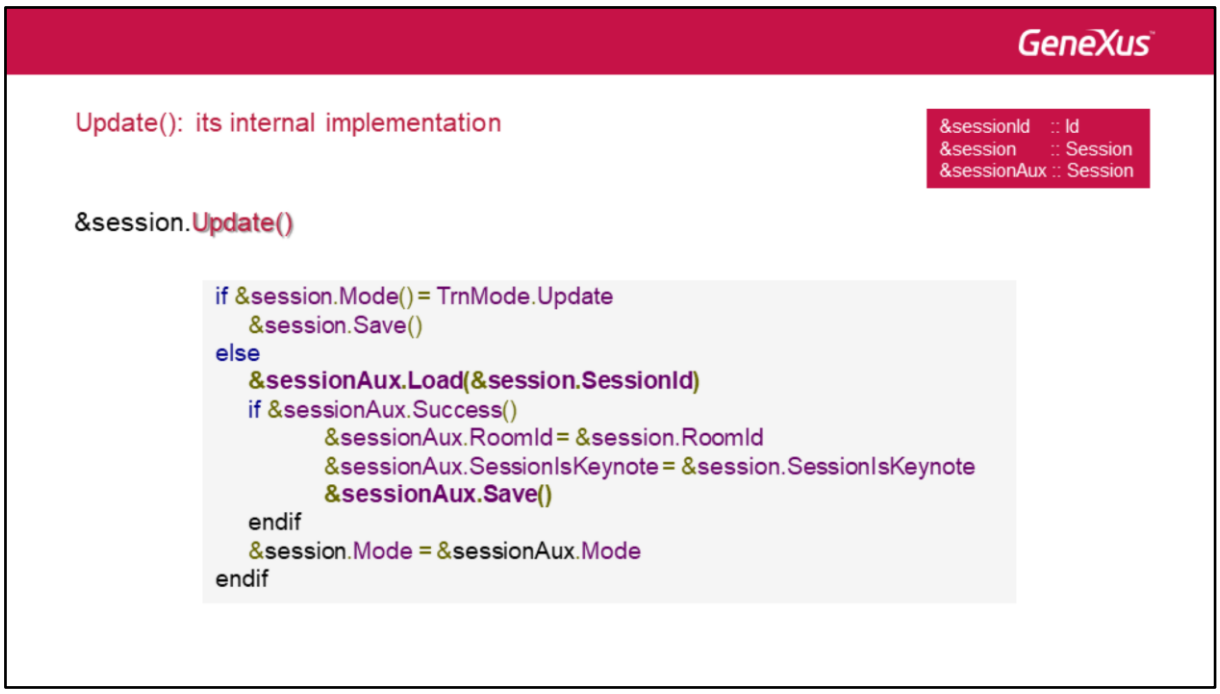
Observemos que aquí estamos cargando la variable &session con lo que nos devuelve un Data Provider que solamente carga el campo del identificador, que es enviado por parámetro desde el evento, y los dos campos que quiere modificar. Por lo que la variable &session tiene vacíos todos los otros campos.

¿Qué hace internamente el método Update?



Como la variable &session está en modo Insert, pues desde que se ha creado no se ha ejecutado un método Load que la deje en Update, entonces el método Update() creará una variable auxiliar –aquí la hemos llamado &sessionAux- en la que hace un Load del SessionId que se especificó en &session. En esta variable auxiliar entonces se traen de la base de datos todos los datos para ese identificador.

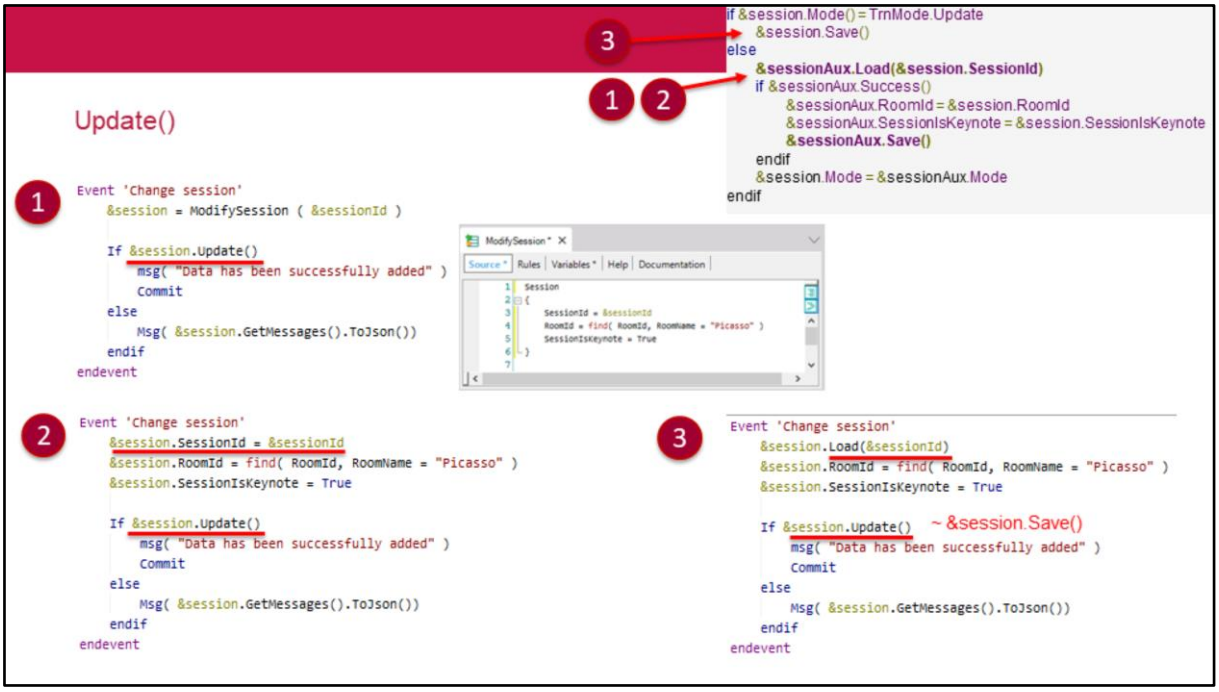
Y luego lo que se hace es modificar en esta variable únicamente los campos que estén asignados en &session, y se hace un Save, dado que esta variable &sessionAux sí estará en modo Update.



Aquí vemos cómo sería el algoritmo interno del Update.

Si la variable sobre la que se hace el Update ya está en ese modo, entonces directamente se hace un Save sobre ella. De lo contrario, se tiene que utilizar una variable auxiliar, tal como vimos. Y al finalizar se le asigna a la variable &session el modo en el que termina la variable &sessionAux, que si todo fue correcto, será Update.

El método Update() hace todo esto por nosotros, sin que nos preocupemos.

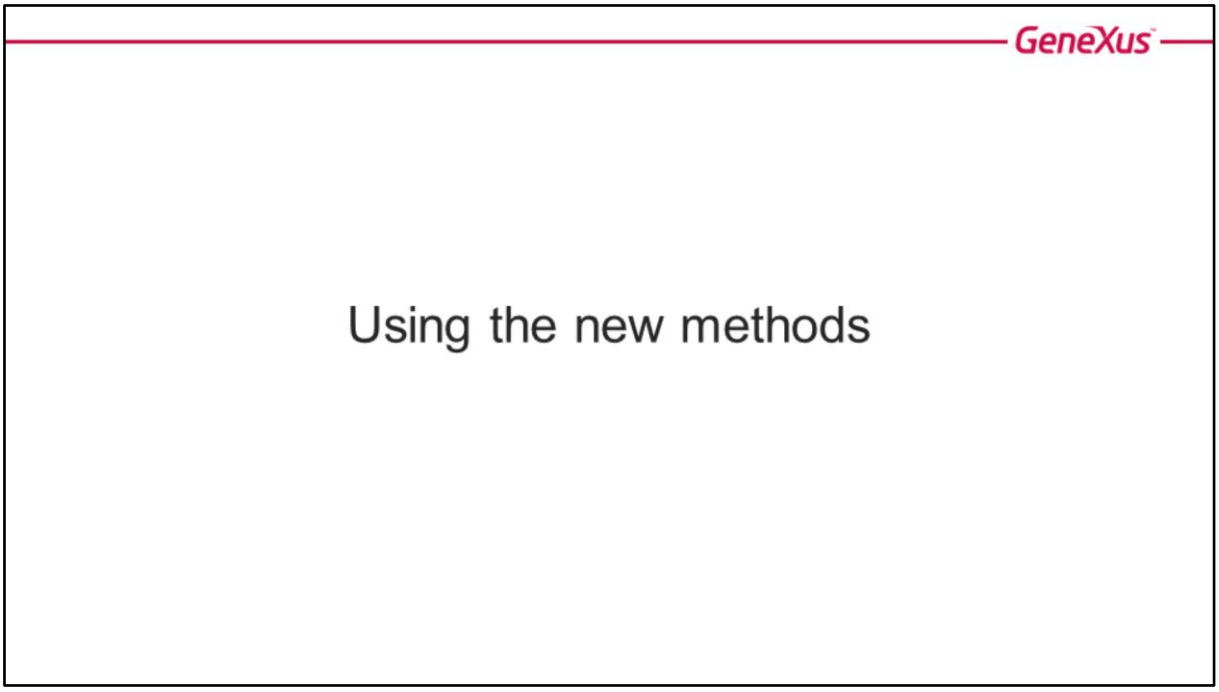


Entonces volviendo al ejemplo, tendríamos tres formas de realizar la actualización deseada.

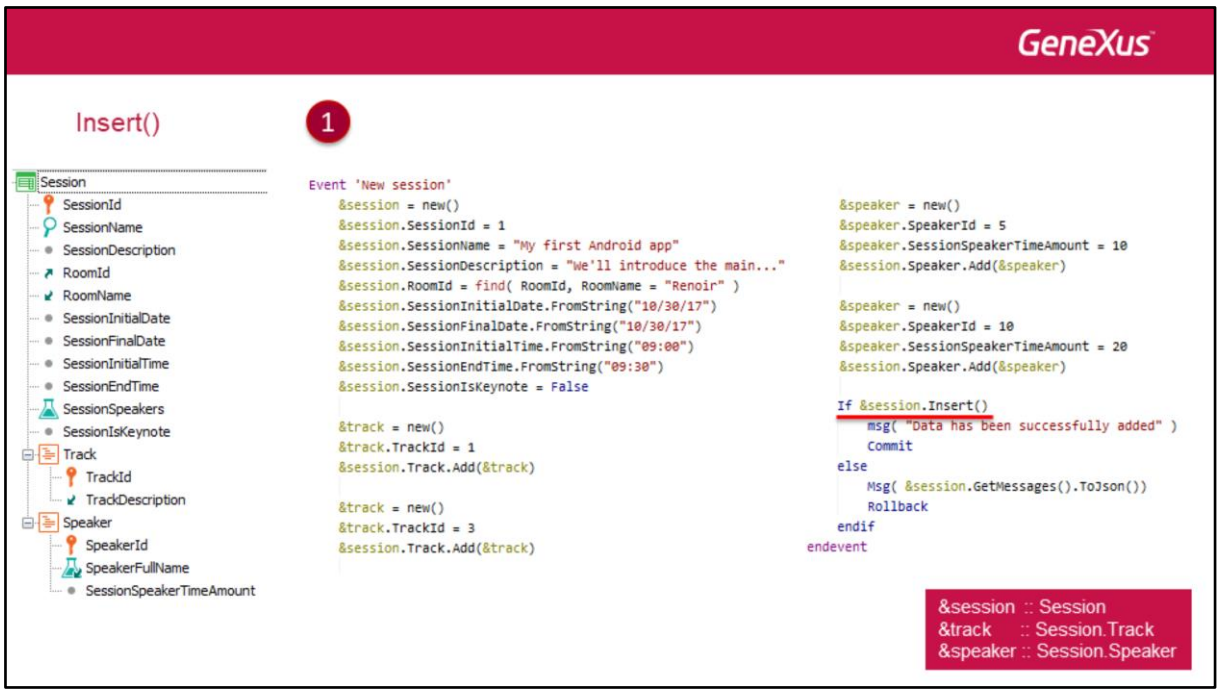
La primera, utilizando el Data Provider para cargar únicamente los datos que cambian. Allí el Update tendrá que crearse variable auxiliar.

La segunda, equivalente, haciendo directamente los cambios en la variable &session pero asignándole valor al atributo clave primaria sin realizar un Load. Aquí también el Update tendrá que crearse variable auxiliar, pues &session estará en modo Insert.

Y la tercera opción es realizando el Load y modificando los campos que deseamos cambiar. Aquí la variable &session estará en modo Update (por haber efectuado el Load), por lo que el método Update no requerirá crear variable auxiliar, y simplemente realizará el Save(). Observemos que esta solución es equivalente por completo a la solución con GeneXus X Evolution 3, con Save().

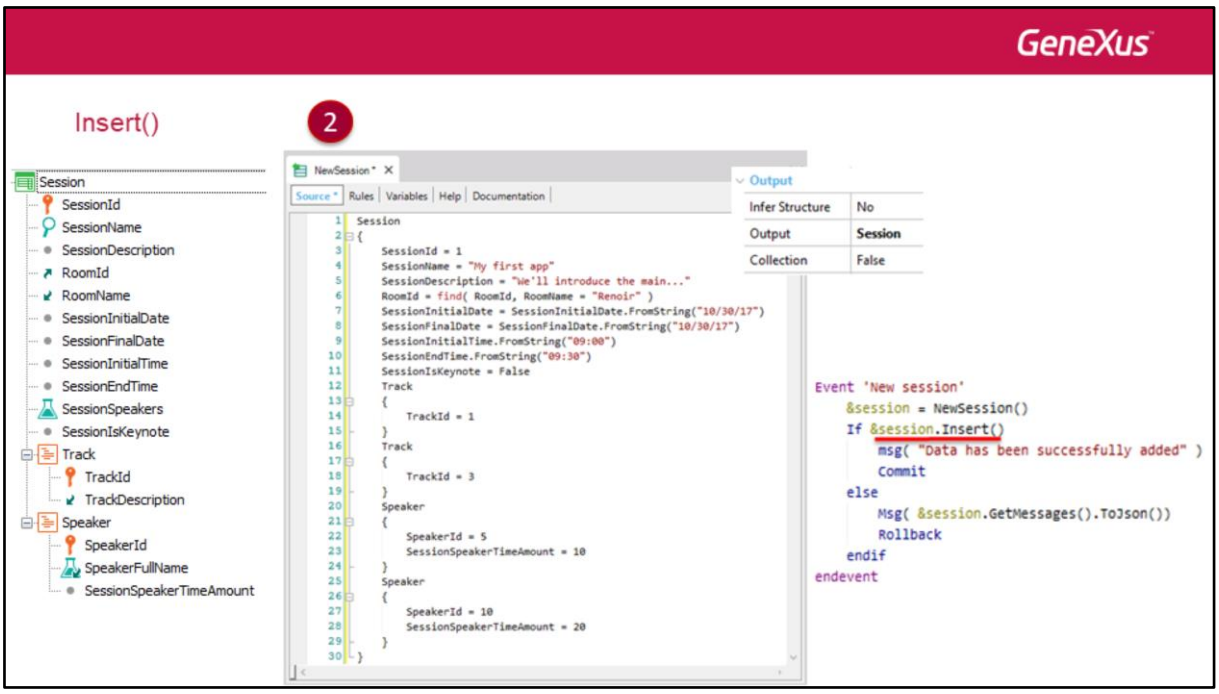


Entonces, utilizando los nuevos métodos...

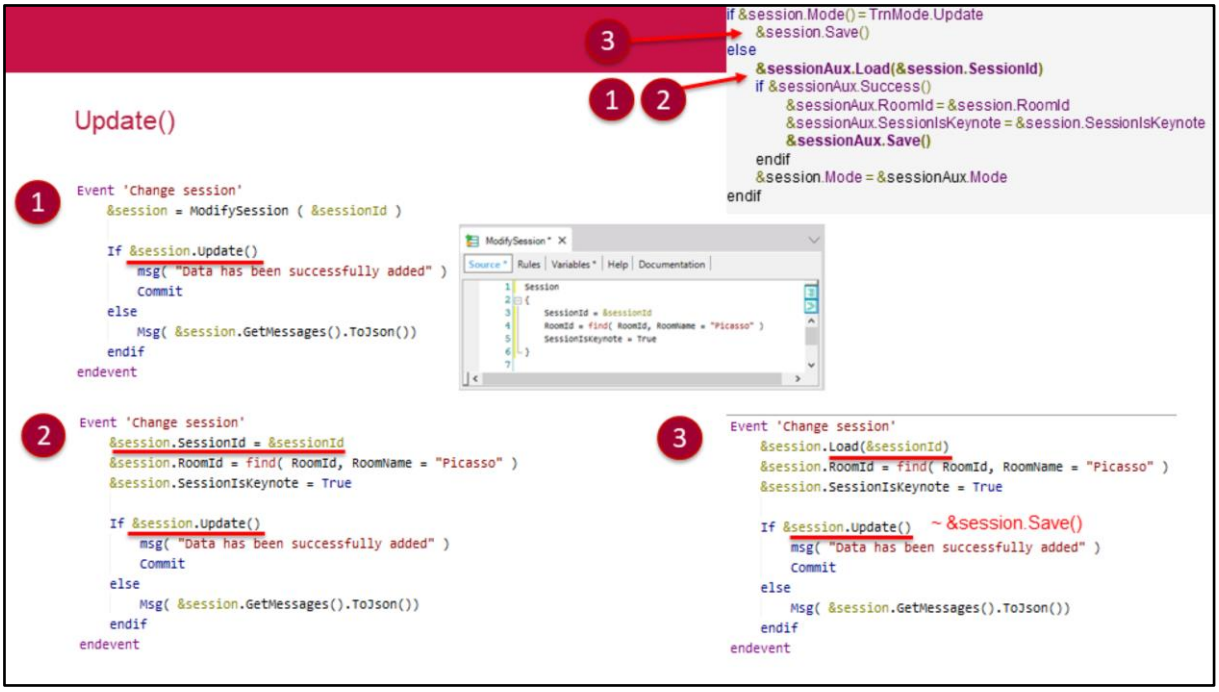


Los ejemplos de Insert nos quedarían así.

Aquí utilizamos el método Insert en lugar del Save(). Y no necesitamos ya preguntar por Success, puesto que el mismo método nos devuelve su resultado.



La solución en la que cargamos la variable &session a partir de un Data Provider es análoga.



El Update ya lo vimos.

GeneXus

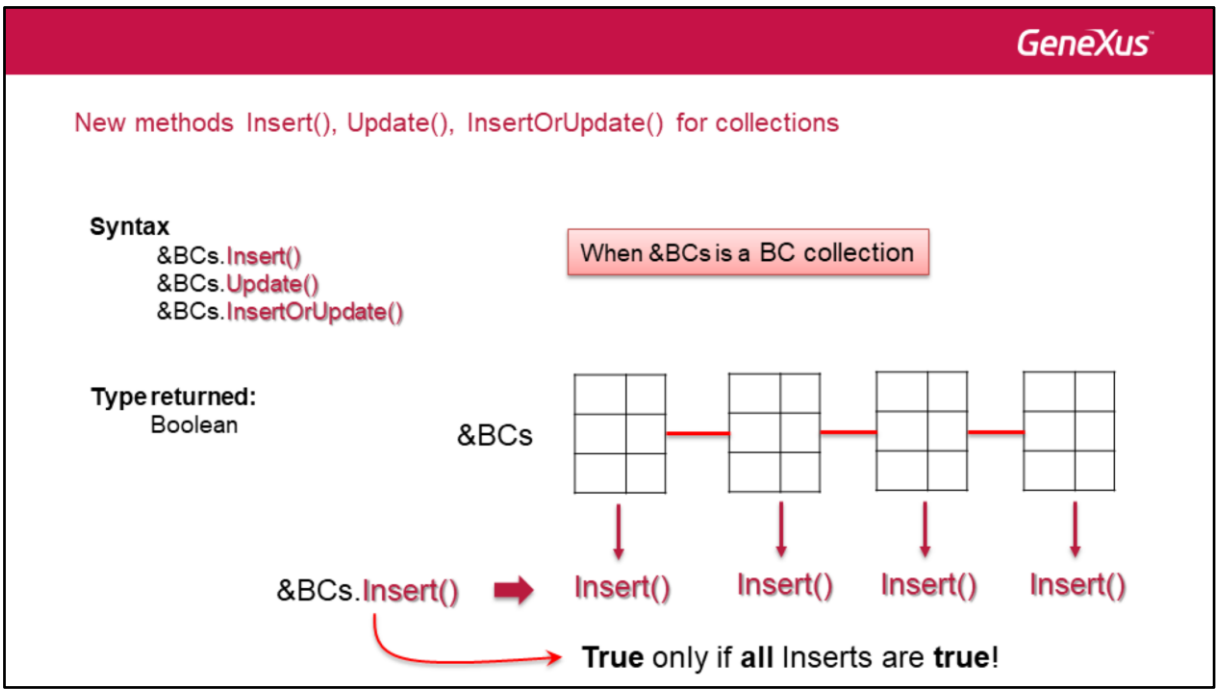
InsertOrUpdate()

It tries to perform an **Insert()**; if the record exists for the value in memory corresponding to the Primary Key or Candidate Key, then it tries to perform an **Update()**.

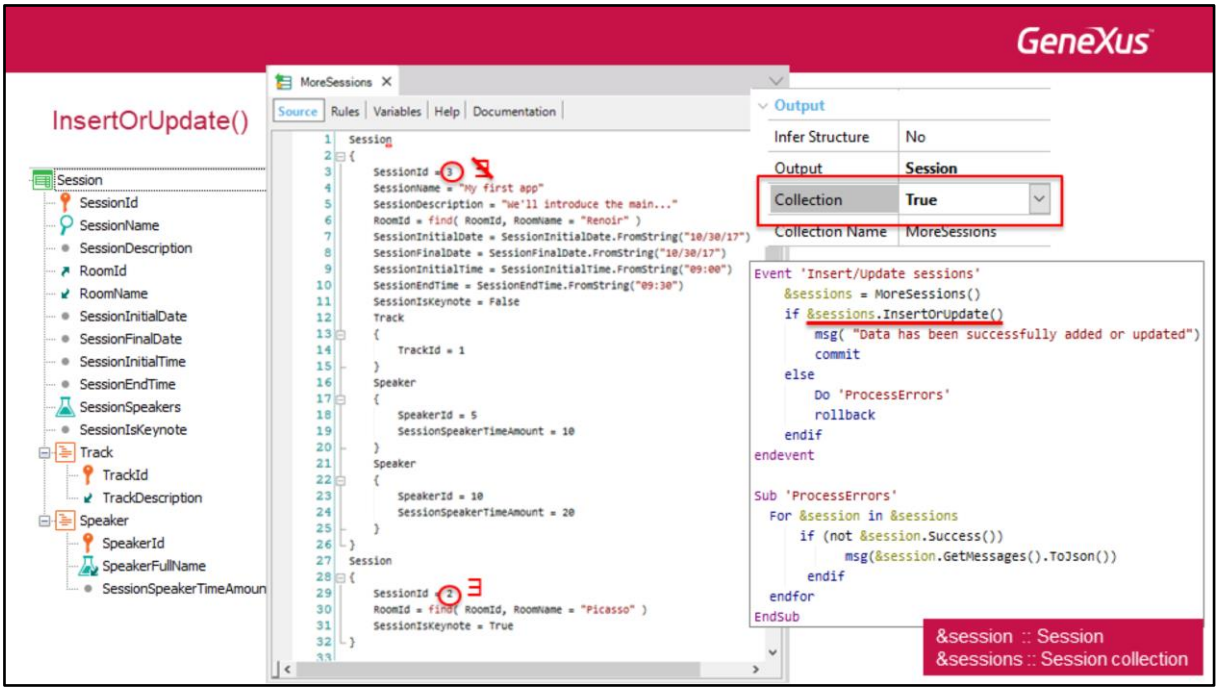
We'll see an example applied to a BC collection.

El método InsertOrUpdate intenta realizar un Insert. Si el registro existe para el valor en memoria correspondiente a la clave primaria o a una clave candidata, entonces intenta realizar un Update.

Veremos un ejemplo de InsertOrUpdate aplicado a una colección de BCs, para lo cual antes veremos qué significa aplicar uno de estos métodos a una colección.



Un método Insert, Update o InsertOrUpdate aplicado a una colección de business components equivale a recorrer la colección uno por uno y aplicarle el método a cada Business Component individual. Pero el resultado sobre la colección será True únicamente si la operación sobre cada business component de ella es True.



Veamos un ejemplo de uso del método InsertOrUpdate, y con una colección de Business Component.

Supongamos que en la base de datos no existe SessionId = 3, pero sí existen SessionId = 1 y SessionId = 2.

El Data Provider esta vez devuelve una colección de Business Component Session, y no uno sólo. Observe que el primer grupo del Data Provider carga cabezal, un track y dos speakers. El segundo carga el SessionId 2 y solamente le asigna valores a RoomId y a SessionIsKeynote, pues quiere modificar esos valores de la session ya existente de Identificador 2.

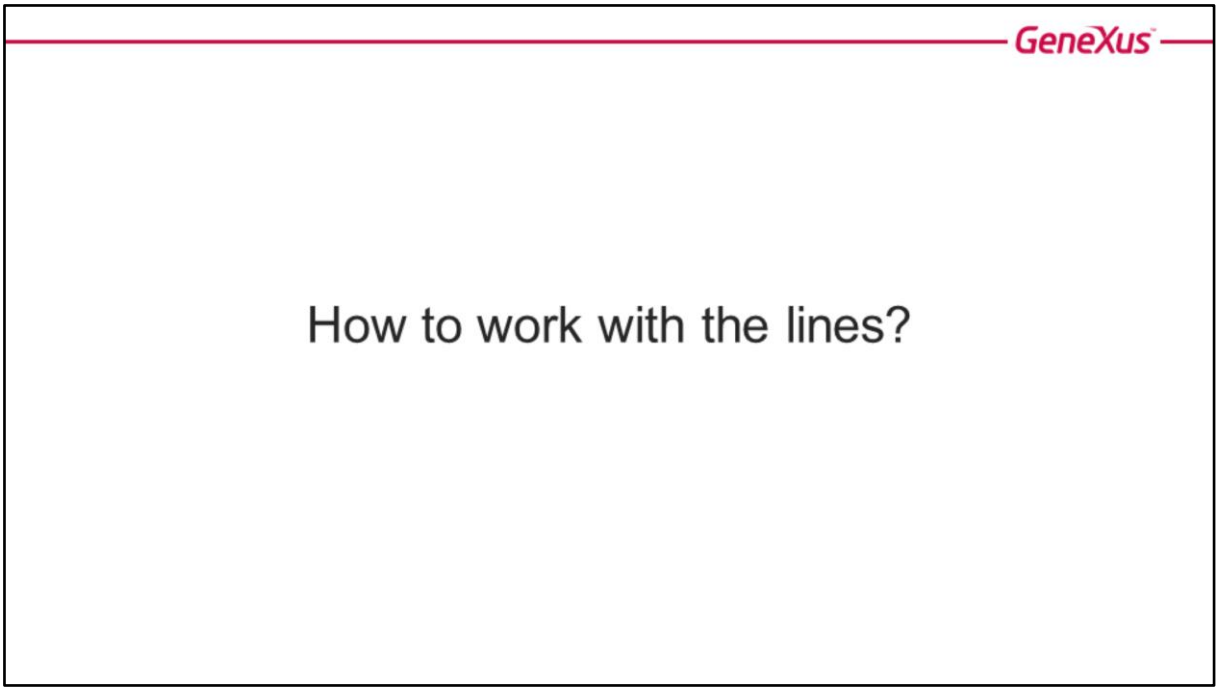
Luego, desde el evento de un web panel se carga en una variable &sessions, colección del tipo de datos el Business Component Session, lo que devuelve ese Data Provider. Luego se invoca al método InsertOrUpdate, que, por lo que dijimos, no es ya una variable Business Component simple, sino una colección. En nuestro caso, una colección con dos elementos (las dos sessions).

El método aplicado a la colección equivale a recorrerla elemento a elemento y aplicar el método a cada uno.

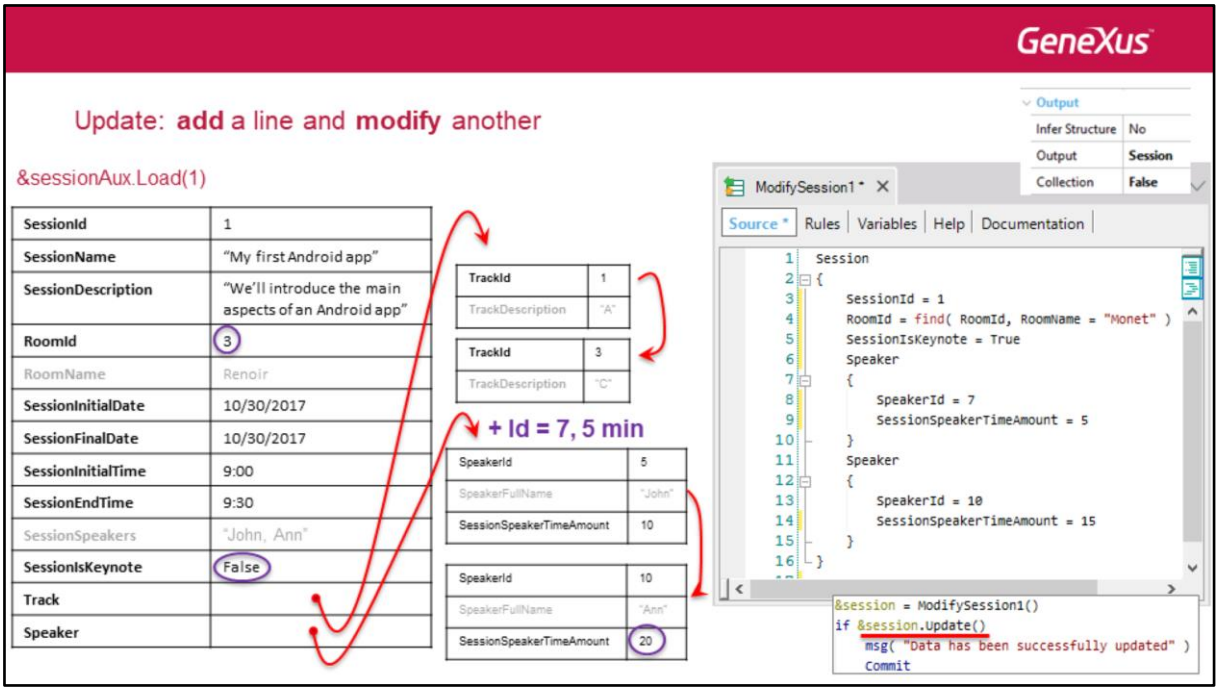
Por tanto, para el primer ítem de la colección, lo insertará sin problema, dado que SessionId = 3 no existía en al base de datos. Para el segundo ítem intentará insertarlo pero encontrará que ya existe un SessionId con valor 2, y entonces realizará el Update.

Si las dos operaciones dan True, entonces se desplegará el mensaje al usuario y se hará commit. De lo contrario se está invocando a una subrutina para procesar los errores, donde se recorre la colección y si no fue exitosa la operación para el ítem actual, entonces se muestran sus mensajes y así.

Y se ejecuta un rollback.



Ahora veremos cómo trabajar con las líneas de un Business Component.



Por ejemplo, cómo agregar una línea y modificar otra

Si hiciérámos un load en una variable auxiliar, &sessionAux, de la session 1 veríamos cargada la estructura como mostramos aquí.

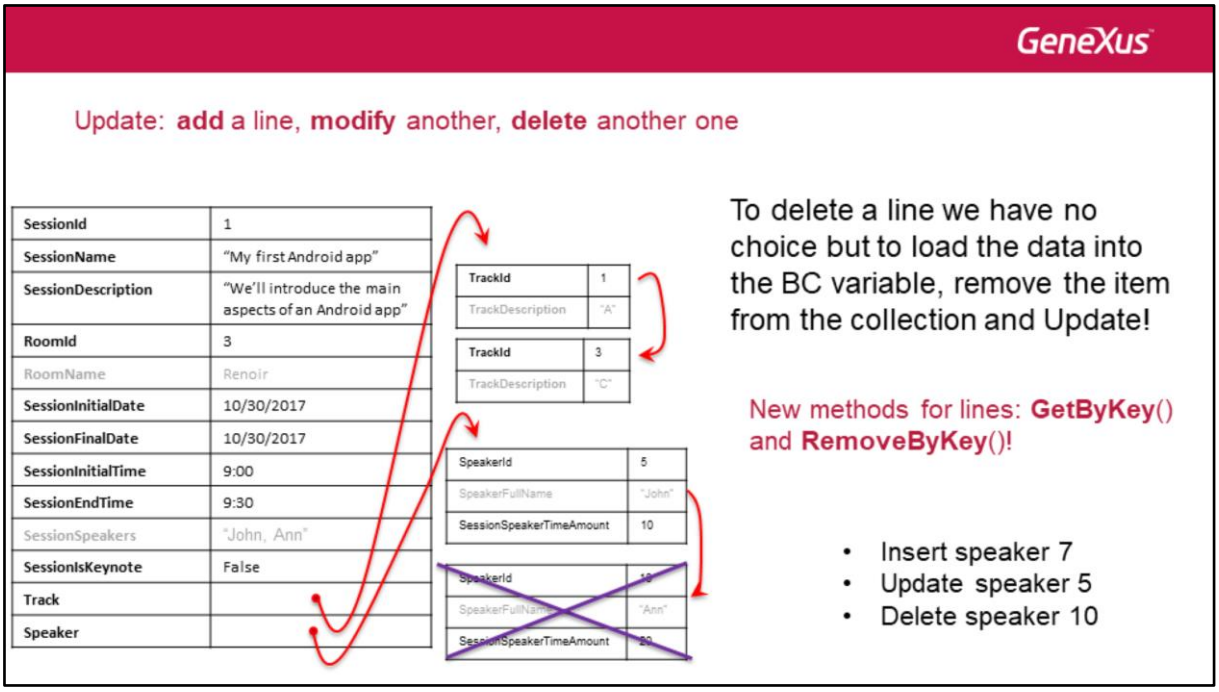
Supongamos que queremos modificar de la session 1 el atributo RoomId, por el RoomId correspondiente a la sala “Monet”, y el SessionIsKeynote que era False, por True. Esas son modificaciones en el cabezal, que ya sabemos cómo realizar. Además queremos agregar el speaker 7, que hablará 5 minutos (esto es agregar una línea) y modificar el tiempo en el que hablará el Speaker 10, para que pase de 20 a 15 minutos (esto significa modificar una línea).

Resumiendo: necesitamos cambiar dos atributos del cabezal, insertar un nuevo item a la colección Speaker, y modificar un item ya existente de esa colección, el del speaker 10.

La manera más fácil de hacerlo es con un Data Provider en el que cargamos los cambios. Observemos que no tocamos en absoluto los tracks ni el speaker 5 que no queremos que sufra ningún cambio.

Luego de cargar la variable &session en un evento con el resultado de este Dara Provider, le realizamos un Update.

Si solamente hubiéramos tenido que realizar los cambios en las líneas (y no hubiéramos tenido que modificar nada del cabezal) también habríamos tenido que utilizar el método Update, puesto que modificar las líneas o insertar líneas a una session ya existente significa actualizar esa session.

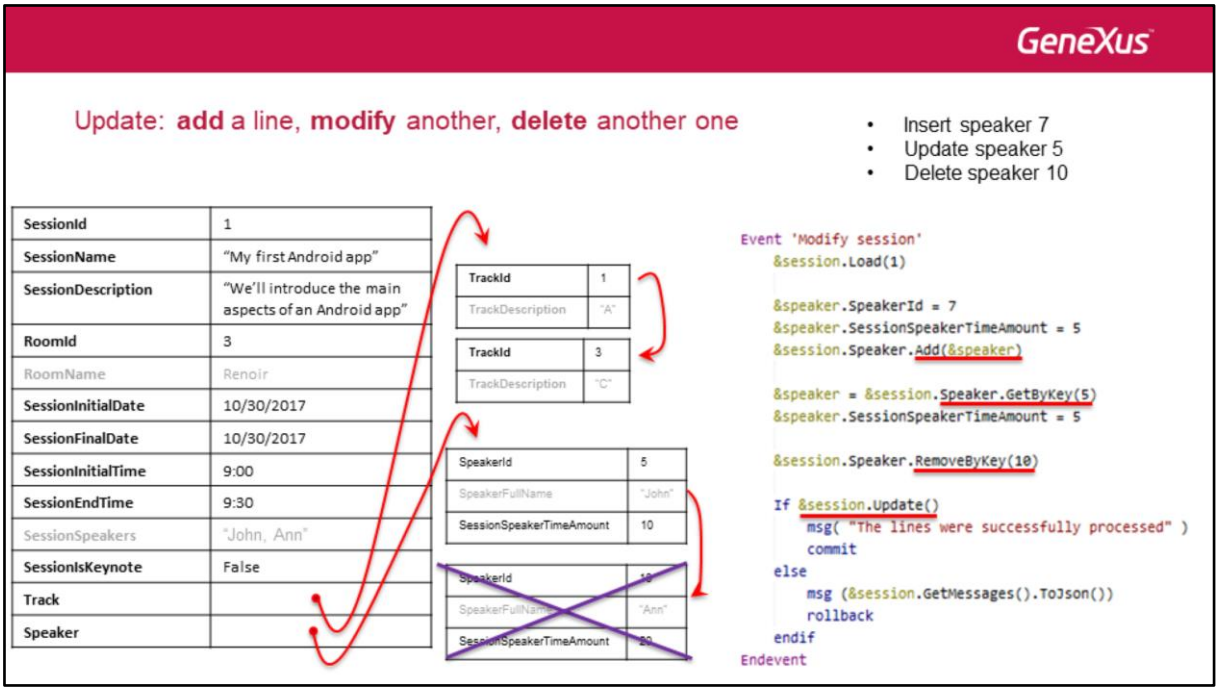


Ahora, ¿qué sucede cuando además tenemos que eliminar una línea?

Cuando es necesario eliminar una línea ya no tendremos más remedio que cargar en una variable Business Component los datos de la base de datos, eliminar el ítem de la colección y salvar (Update o Save). Aquí ya no podremos trabajar con un Data Provider. Tenemos que hacer la eliminación explícitamente.

Trabajar con las líneas en versiones anteriores era muy costoso. Tanto a la hora de actualizar como a la hora de eliminar una línea.
Veremos, pues, los nuevos métodos creados para simplificar la tarea.

Aprovecharemos para hacer manualmente (sin data provider) las tres operaciones sobre las líneas: agregar una nueva, actualizar algún dato de otra, y eliminar otra.



Veamos que primero necesitamos cargar los datos de la session con la que vamos a trabajar en la variable &session. No tenemos otra alternativa que utilizar el método Load pues necesitaremos acceder a los Speakers existentes para poder eliminar el de Id 10.

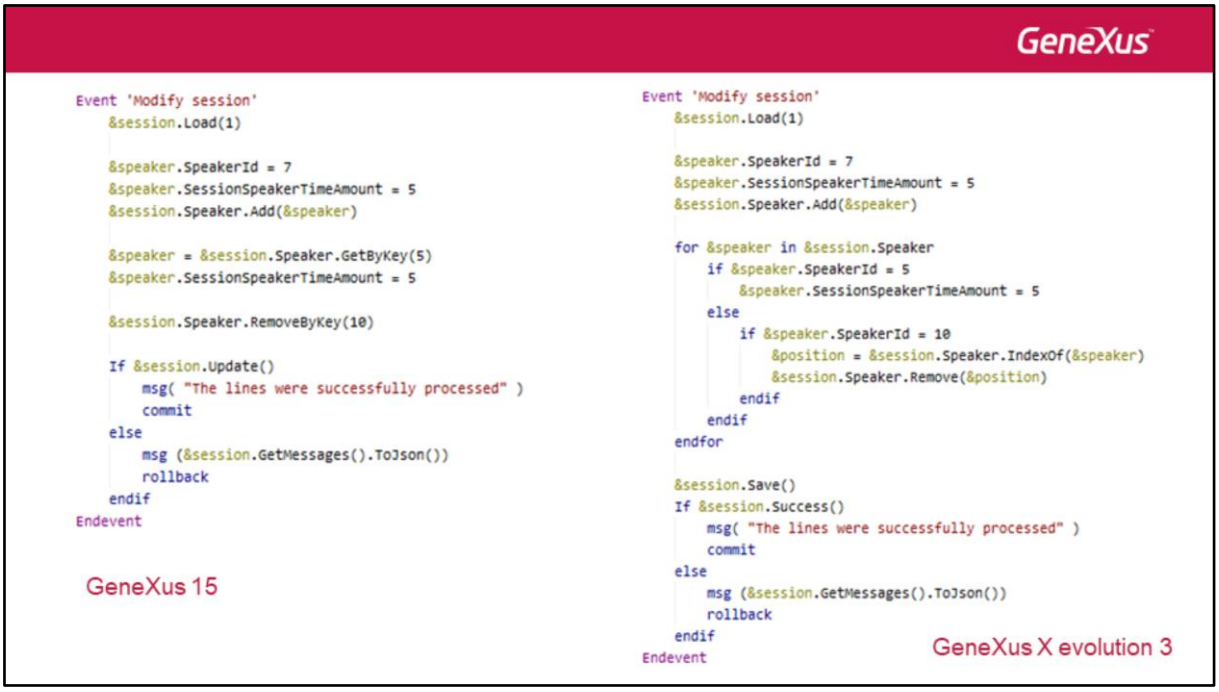
Y luego realizamos las tres operaciones de una forma sumamente sencilla. Y luego hacemos el Update.

La forma de insertar no se modifica respecto a versiones anteriores: sigue siendo a través del método add.

La forma de actualizar y de eliminar sí. Para actualizar solamente necesitamos definir una variable &speaker del tipo de datos el Business Component de la colección de líneas –en nuestro caso Session.Speaker- y aplicar el método **GetByKey** a la colección &session.Speaker. Esto nos devuelve, por referencia, cargado en la variable &speaker ese ítem o línea. Directamente modificamos los campos que nos interesen –en este caso SessionSpeakerTimeAmount-. Como es por referencia, es decir, la variable está apuntando a *esa* posición de memoria, esos cambios afectarán directamente a ese elemento de la colección, por lo que cuando se haga el Update (o Save), esos cambios serán efectuados en la base de datos.

Por último, para eliminar alcanza con utilizar el nuevo método **RemoveByKey** de la colección del Business Component. Elimina busando el Business Component de la colección que tiene esa clave. Esta eliminación es lógica, hasta que no se ejecute la operación correspondiente (Update, InsertOrUpdate o Save) no se realizará físicamente en la base de datos.

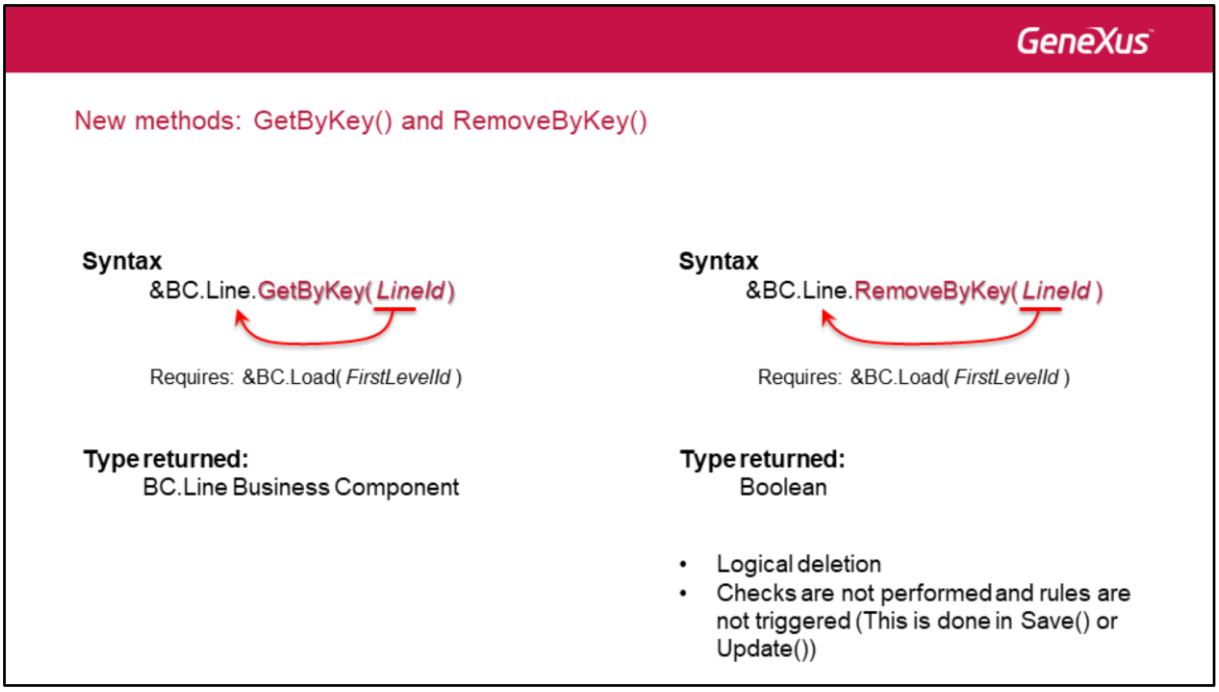
En nuestro ejemplo utilizamos el método Update, y éste será el encargado de efectuar todas las modificaciones, respetando la lógica definida en la transacción y haciendo los chequeos de integridad y control de duplicados consabidos.



Aquí comparamos lo que acabamos de hacer gracias a los nuevos métodos para las líneas ofrecidos por GeneXus 15, con la manera en que habríamos tenido que resolverlo con la versión anterior.

Veamos que para hacer la actualización, que aquí la hacíamos simplemente con el método nuevo GetByKey, teníamos que recorrer la colección de speakers con el comando for in; si el speaker en el que estamos posicionados en ese momento tiene identificador 5, hacemos la actualización del atributo SessionSpeakerTimeAmount; de lo contrario, si tiene el identificador 10, entonces ubicamos la posición de ese speaker en la colección para poder utilizar el método Remove de esa posición para eliminarlo de la colección.
Y luego usábamos el método Save que aplicaba todos los cambios realizados.

Vemos entonces claramente cómo es mucho más sencillo ahora realizar todas esas operaciones con GeneXus 15.

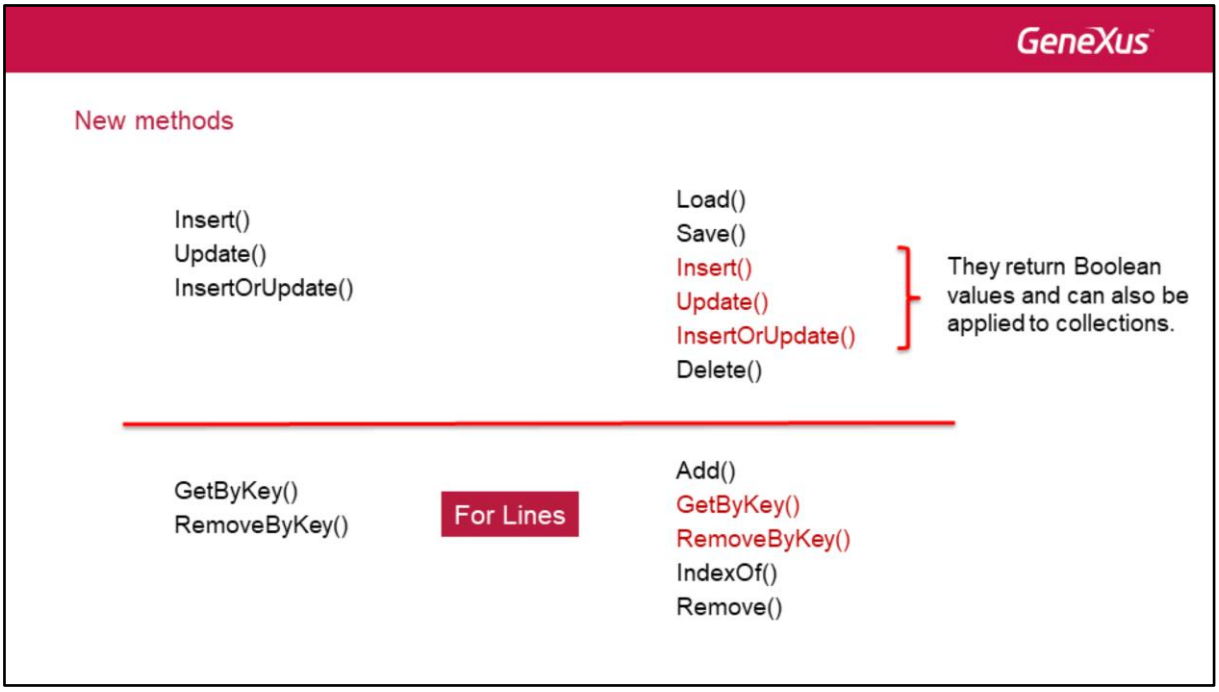


Aquí presentamos la sintaxis de los nuevos métodos.

El **GetByKey()** se aplica a una variable basada en un tipo Business Component de una transacción de dos niveles para obtener el registro que corresponde a la "línea" de acuerdo a su identificador. Antes toda la estructura debe ser cargada en memoria utilizando el método Load, y luego el método GetByKey puede ser utilizado para obtener la línea deseada, especificando el valor del identificador de ese nivel.
Se retorna una referencia al item, del tipo el business component correspondiente, por lo que lo que se modifique a partir de esa referencia, quedará modificado en memoria para la variable &BC.

El **RemoveByKey** es análogo, pero devuelve un booleano, para indicar si se pudo realizar o no la eliminación. Esta eliminación es lógica, solamente marca la línea con el modo Delete, y es luego el siguiente Save() o Update() el que realmente va a realizar el intento de eliminar la línea, aplicando previamente las reglas que correspondan del Business Component, y chequeando la integridad referencial.

En suma



A la izquierda listamos los nuevos métodos. Los de abajo son para líneas de Business Components.

A la derecha los mostramos junto con los demás métodos que siguen vigentes, de la versión X Evolution 3.

Recordemos que los métodos Insert, Update e InsertOrUpdate, a diferencia de Load, Save y Delete devuelven un booleano indicando si la operación fue o no exitosa, por lo que ya no requieren utilizar los métodos Success o Fail para evaluar ese resultado. Y también reparemos en el hecho de que solo ellos pueden ser utilizados para colecciones de business components.

GeneXus™

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications