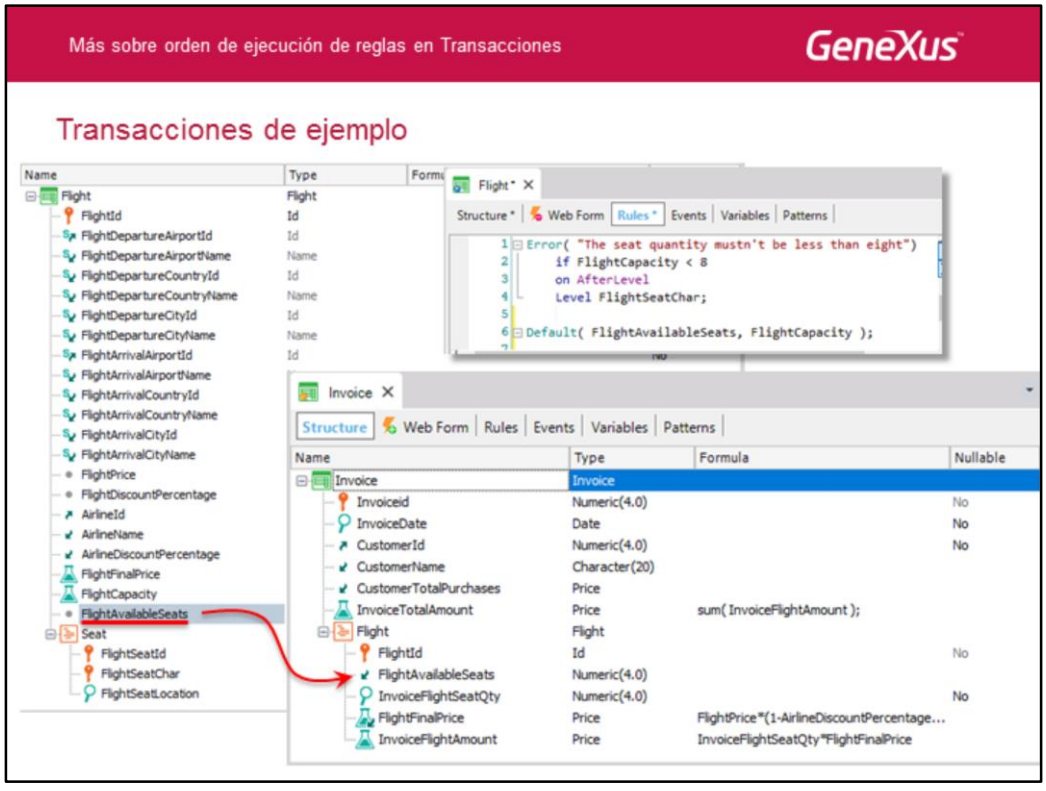
Más sobre orden de ejecución de reglas en Transacciones

GeneXus 15



A continuación abordaremos más en profundidad los momentos que tenemos disponibles para condicionar el disparo de reglas, en particular en transacciones de más de un nivel.

Para entender el tema, nos basaremos en la transacción de Facturas (Invoice) que cuenta con un segundo nivel (Flight), que representa los vuelos incluidos en la factura.

Observemos que hemos agregado a la transacción Flight el atributo **FlightAvailableSeats**, el que se usará para registrar los asientos disponibles de cada vuelo, que se irán decrementando cada vez que se realiza una factura para un cliente que compra una cantidad de lugares (asientos) en el vuelo.

Nota: También hemos agregado a la transacción Customer el atributo CustomerTotalPurchases para registrar el total comprado por el cliente por concepto de pasajes de vuelos.

Observemos que los atributos InvoiceTotalAmount, InvoiceFlightPrice e InvoiceFlightAmount de la estructura de Invoice son fórmulas. A continuación veamos las reglas que definimos para Invoice, para especificar su comportamiento.

GeneXus

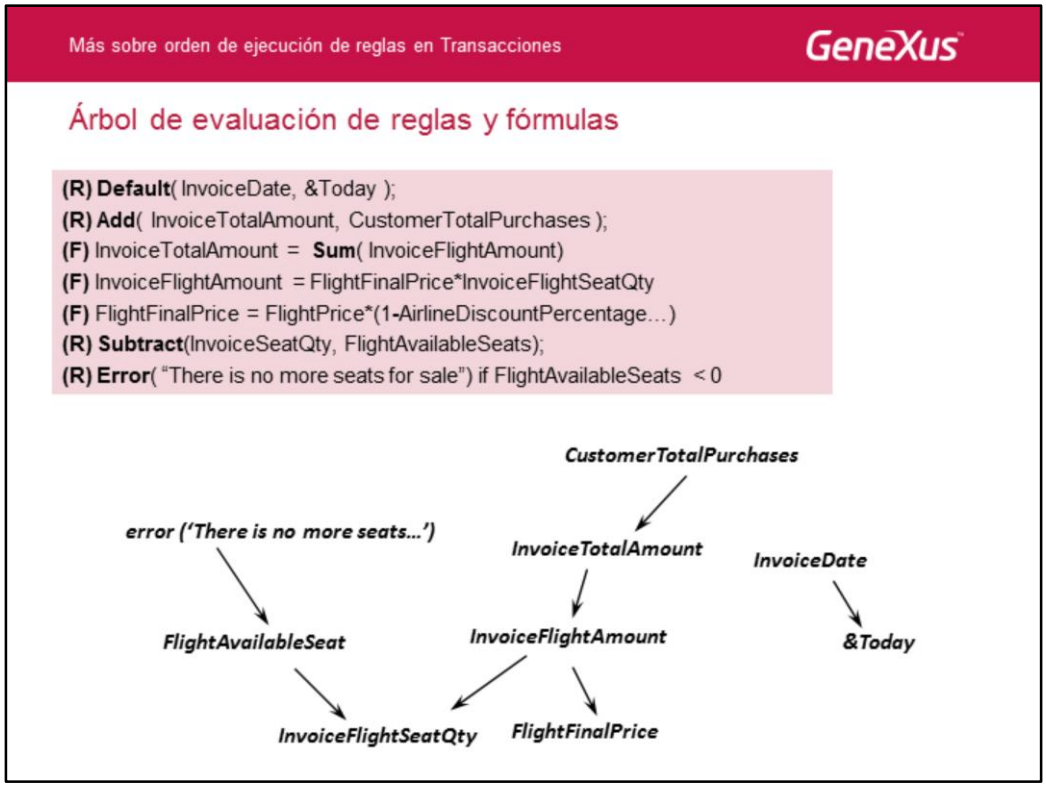
Reglas de la transacción

Invoice	Invoice	
Invoiceid	Numeric(4.0)	
InvoiceDate	Date	
CustomerId	Numeric(4.0)	
CustomerName	Character(20)	
CustomerTotalPurchases	Price	
InvoiceTotalAmount	Price	sum(InvoiceFlightAmount);
Flight	Flight	
FlightId	Id	
FlightAvailableSeats	Numeric(4.0)	
InvoiceFlightSeatQty	Numeric(4.0)	
FlightFinalPrice	Price	FlightPrice*(1-AirlineDiscountPercentage...
InvoiceFlightAmount	Price	InvoiceFlightSeatQty*FlightFinalPrice

Invoice * ×

Structure | Web Form | Rules * | Events | Variables | Patterns

```
1  Default( InvoiceDate, &Today );
2
3  Subtract( InvoiceFlightSeatQty, FlightAvailableSeats );
4
5  Error( 'There is no more seats for sale' )
6      if FlightAvailableSeats < 0;
7
8  Add( InvoiceTotalAmount, CustomerTotalPurchases );
```



Para considerar el orden que GeneXus elige para disparar las reglas y fórmulas, consideremos las definiciones de reglas y fórmulas que se muestran en pantalla.

Al momento de generar el programa asociado a la transacción Invoice, GeneXus extraerá las dependencias existentes entre las reglas y fórmulas definidas y construirá lógicamente un árbol de dependencias (o árbol de evaluación) que determinará la secuencia de evaluación.

Podemos imaginar que el árbol se ejecuta de abajo hacia arriba, es decir que cada vez que cambia el valor de un atributo, se ejecutan todas las reglas y fórmulas que dependen de ese atributo (y que en el árbol se encuentran hacia arriba).

Por ejemplo, si cambia la cantidad de asientos en una línea de una factura (InvoiceFlightSeatQty), como este atributo interviene en la fórmula que calcula el importe del vuelo (InvoiceFlightAmount), dicha fórmula se redisparará. Lo mismo sucedería si cambiara el precio final del vuelo FlightFinalPrice que también interviene en la fórmula.

Al cambiar el importe de un vuelo, deberá redispararse también la fórmula correspondiente al total de la factura (InvoiceTotal) ya que depende del valor de cada vuelo de la factura. Por último, por cambiar el total también se tendrá que disparar la regla Add(InvoiceTotal, CustomerTotalPurchases) porque deberá actualizarse el total de compras del cliente.

Además de dispararse todas las fórmulas y reglas involucradas en la rama derecha del árbol desde el atributo InvoiceFlightSeatQty, también se dispararán las fórmulas y reglas involucradas en la rama izquierda. Es decir, que al cambiar el valor del atributo InvoiceFlightSeatQty, se redisparará también la regla Subtract(InvoiceFlightSeatQty, FlightAvailableSeats) que actualiza la cantidad de

asientos disponibles en el vuelo (FlightAvailableSeats).

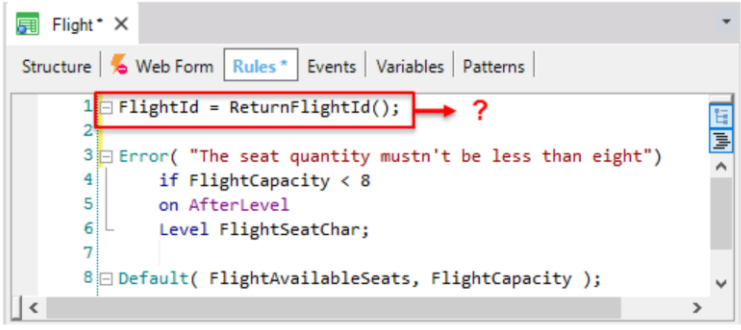
Y en consecuencia, por modificar esta regla el valor del atributo FlightAvailableSeats, se evaluará si habrá que disparar la regla Error('There is no more seats…') if FlightAvailableSeats< 0;

Concluyendo, las reglas y fórmulas que se definen en una transacción suelen estar interrelacionadas y GeneXus determina las dependencias entre ellas así como su orden de evaluación.

Repaso eventos de disparo

- Generalmente las reglas que definimos se ejecutan en el momento que pretendemos.
- Sin embargo en algunos casos notamos la necesidad de modificar el momento de disparo de una regla.

Ejemplo:



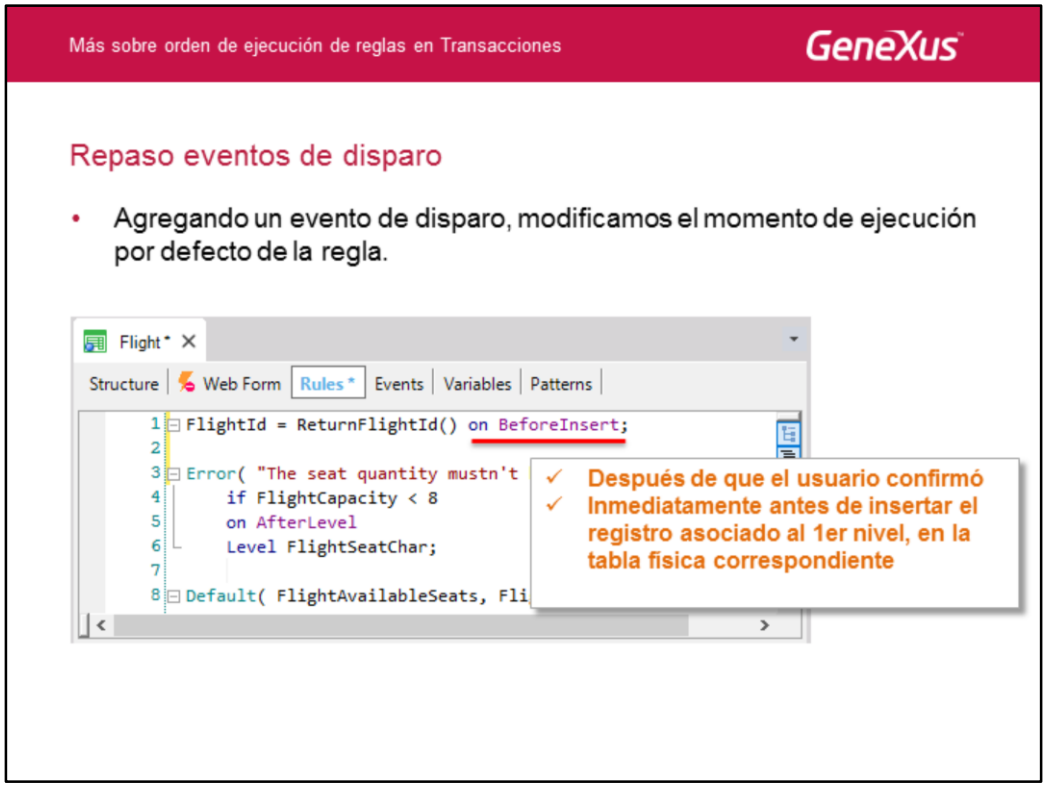
En la transacción Flight, al identificador de vuelo FlightId lo tenemos definido como autonumerado.

Si en la realidad el identificador de vuelo está formado por letras que identifican a la compañía y números, no nos sirve el atributo FlightId numérico, ni autonumerado.

Supongamos que tenemos un procedimiento, ReturnFlightId, que es el que nos devuelve el identificador a ser asignado a un nuevo vuelo. (En la realidad este procedimiento debería elegir el número de vuelo en función de la aerolínea, el origen y el destino, pero por simplificación lo obviaremos).

Si escribimos la regla que vemos arriba, ¿en qué momento se disparará? En el cabezal de Flight y cualquiera sea la operación que se realice. Es decir que si se modifica algo del vuelo, se vuelve a invocar al procedimiento ReturnFlightId para que asigne un nuevo identificador del vuelo, cuando en realidad queremos que pase esto solamente si estamos insertando un vuelo nuevo.

Además, aun cuando estemos insertando un vuelo, deberíamos obtener un nuevo FlightId solamente si confirmamos la transacción, ya que podría pasar que nosotros canceláramos el ingreso o que hubiera un error y se produjera una cancelación automática. En estos casos, se estaría “gastando” un nuevo número para el identificador de vuelo, que no llegaría a usarse.

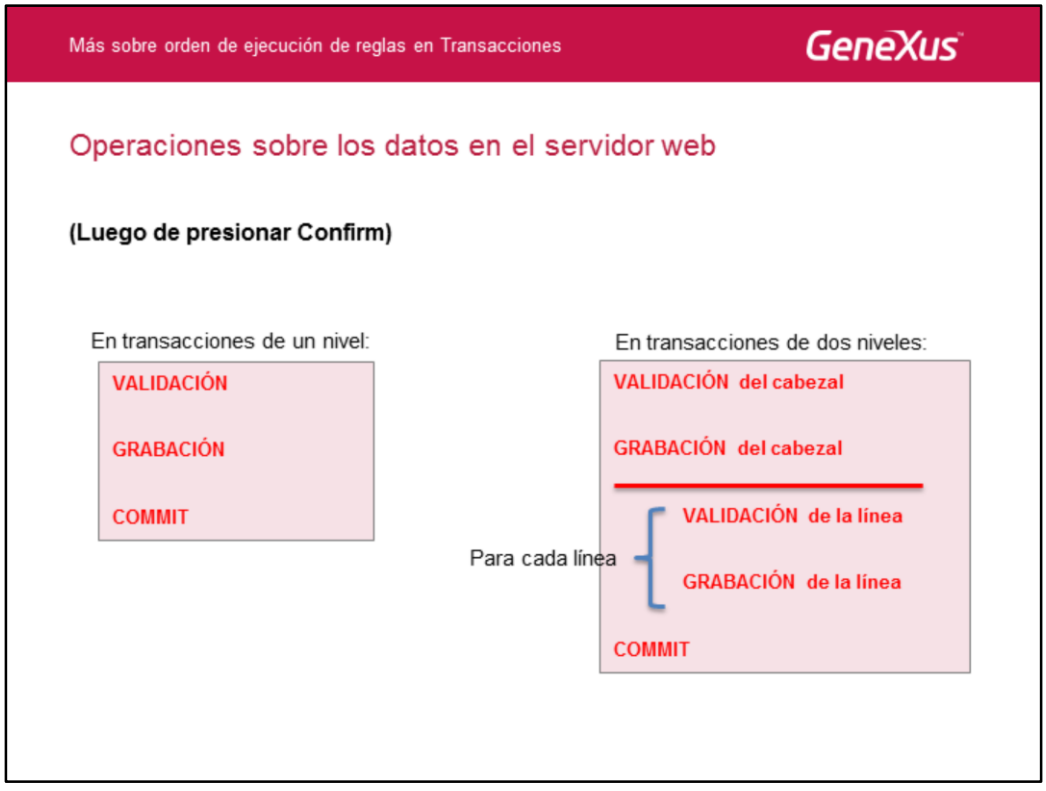


Para asegurarnos que solamente se use un número de id del vuelo si se confirmó la transacción y además que se obtenga uno solamente cuando estamos insertando un vuelo, agregamos a la regla que asigna el FlightId, el momento de disparo on BeforeInsert.

Este momento de disparo se disparará después de que confirmemos la transacción y se ejecutará a nivel del servidor.

Además before insert significa que se disparará luego de que se validaron los datos del cabezal del vuelo e inmediatamente antes que se graben los datos del cabezal en la base de datos.

Veamos a continuación que existen algunos momentos claves en la operación del servidor, con relación al proceso de los datos y la base de datos.

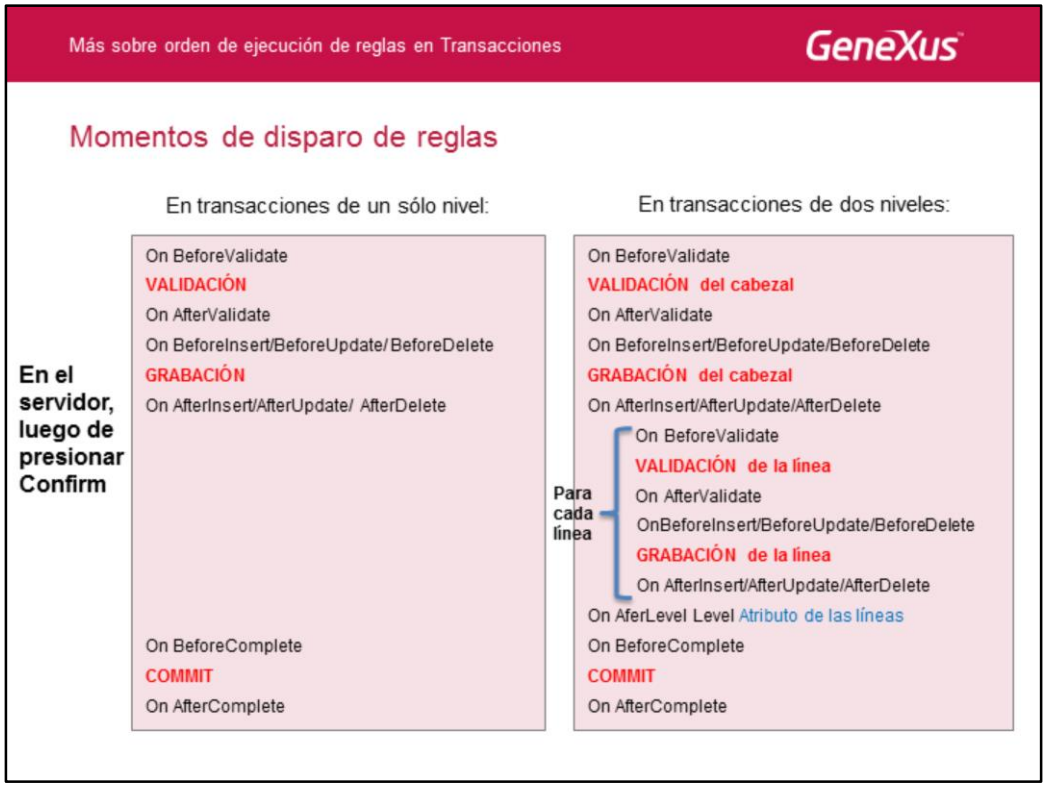


Después de que se presiona Confirm, la información viaja del cliente web (browser) al servidor web.

En el servidor hay operaciones que se realizan sobre los datos, que son:
- la Validación de los datos,
- la Grabación de los datos en la base de datos y
- el Commit a la base de datos.

Si la transacción es de dos niveles, entonces luego de la grabación del cabezal, se producirá para cada línea:
- la Validación y luego
- la Grabación de la línea.

Por último se producirá el Commit, que se encargará de consolidar en la base de datos los datos del cabezal y todas las líneas de la transacción.



Una vez que la información de la transacción llega al servidor web, las reglas y fórmulas volverán a dispararse a nivel del server.

En esta ejecución, tenemos disponibles varios momentos relacionados a las operaciones de Validación, Grabación y Commit de los datos en el servidor. Estos momentos especiales se los podemos asignar a las reglas, de forma de controlar cuándo deben dispararse.

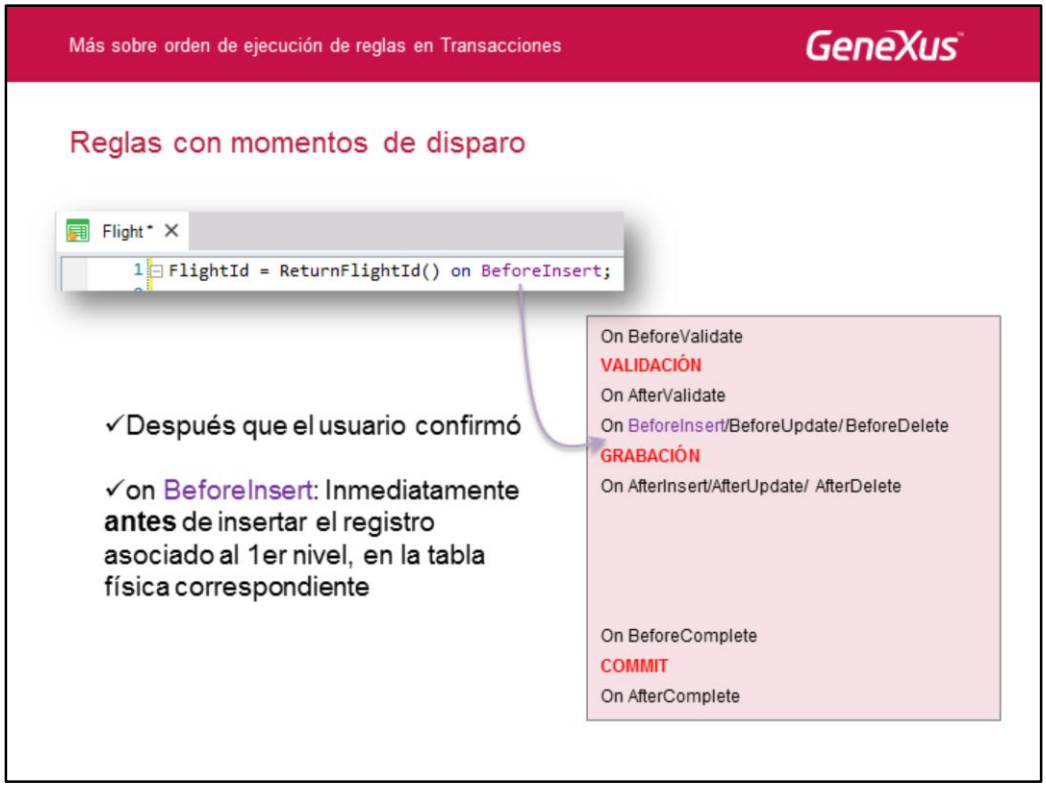
Esto nos permite personalizar el momento de disparo de una regla, para que la misma no sea disparada en el momento elegido por GeneXus según su árbol de evaluación, sino cuando nosotros queremos.

Comencemos por los momentos relacionados a la Validación: BeforeValidate y AfterValidate.

Evento de disparo: **BeforeValidate**
Este evento de disparo ocurre un instante de tiempo antes de que la información de la instancia con la que se está trabajando (cabezal o línea x) sea validada. Es decir, ocurrirá un instante de tiempo antes de la acción de “validación del cabezal” o “validación de la línea”, según corresponda. Observar que aquí también se habrán disparado ya todas las reglas que no estén condicionadas a evento de disparo alguno.

Evento de disparo: **AfterValidate**
El evento de disparo AfterValidate permite especificar que una regla se ejecute inmediatamente antes de que se grabe físicamente cada instancia del nivel al cual está asociada la regla, en la tabla física correspondiente, y después de que se hayan validado los datos de esa instancia.

En otras palabras, si se le agrega el evento de disparo AfterValidate a una regla, la misma se ejecutará para cada instancia del nivel al cual esté asociada, inmediatamente antes de que la instancia se grabe físicamente (ya sea que se inserte, modifique o elimine) como registro en la tabla física asociada al nivel.



A veces no contamos con la posibilidad de utilizar la propiedad Autonumber para numerar de forma automática y correlativa los atributos que son clave primaria simple. Tal funcionalidad es provista por los manejadores de base de datos (DBMSs) y GeneXus la aprovecha y permite usarla; sin embargo cuando no trabajamos con un DBMS, esa opción no está disponible.

En el ejemplo que hemos planteado, el identificador de cada vuelo se debe generar de una forma particular y no nos sirve el autonumerado, por lo que se ha codificado un procedimiento para resolver esa numeración.

Tal como hemos explicado, necesitamos que la regla se ejecute inmediatamente después de que el usuario haya confirmado y sólo si estaba insertando un vuelo nuevo.

Estas dos posibles definiciones son correctas para definir lo que necesitamos:

```
FlightId = ReturnFlightId() if Insert on AfterValidate;
```

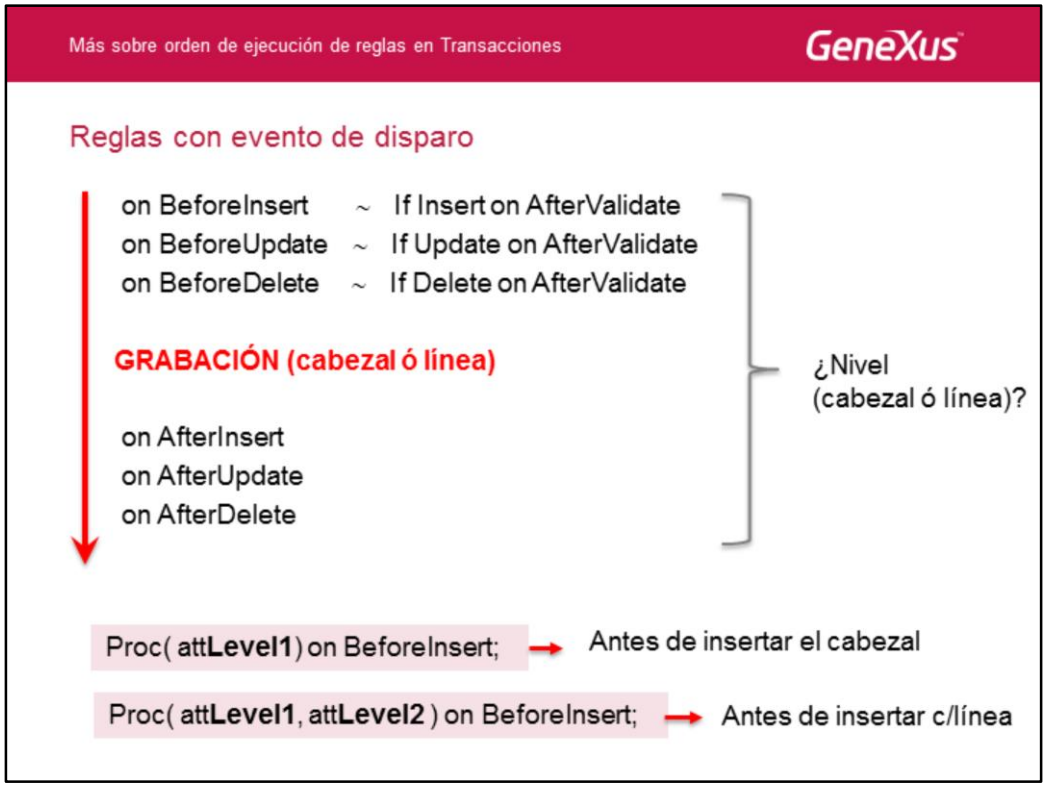
o bien

```
FlightId = ReturnFlightId() on BeforeInsert;
```

En la primera definición estamos especificando que el procedimiento se ejecute únicamente si se trata de una inserción (por la condición de disparo: if Insert), inmediatamente después de validados los datos del primer nivel (porque hay solamente un atributo involucrado en la regla, que pertenece al primer nivel de la transacción) y justo antes de que se grabe físicamente el registro (por el evento de disparo: on AfterValidate).

Existen tres eventos de disparo que ocurren en el mismo momento que el evento AfterValidate, pero que ya contienen intrínseco el modo. Ellos son: BeforeInsert, BeforeUpdate y BeforeDelete.

Por este motivo son equivalentes las reglas presentadas arriba y en la 2da propuesta, sería redundante condicionar la regla a “If Insert”, ya que el BeforeInsert tiene intrínseco que se trata de una inserción.



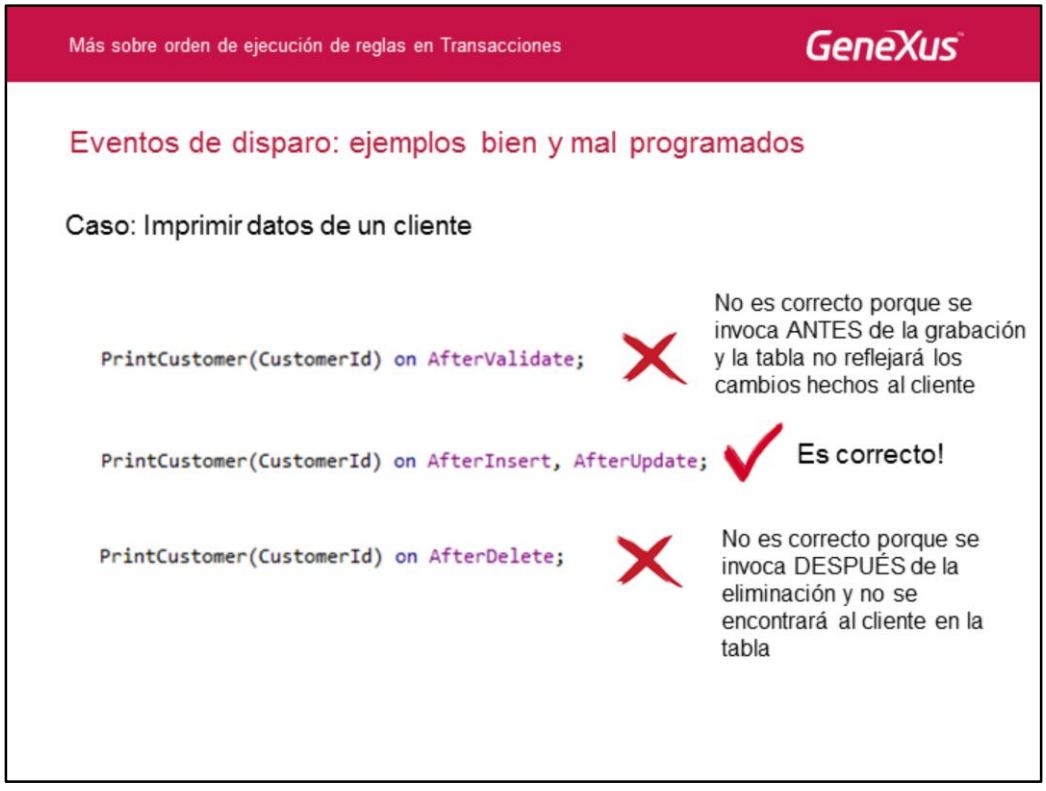
De modo que valen las siguientes equivalencias:

on BeforeInsert ~ If Insert on AfterValidate
on BeforeUpdate ~ If Update on AfterValidate
on BeforeDelete ~ If Delete on AfterValidate

Si definimos una regla a la cual le incluimos también el evento de disparo on AfterValidate, u on BeforeInsert, BeforeDelete, BeforeUdate pero a diferencia del ejemplo recién visto, se referencia en la regla al menos un atributo del segundo nivel de la transacción, la misma estará asociada al segundo nivel. Por lo tanto, la regla se ejecutará inmediatamente antes de que se grabe físicamente cada instancia correspondiente al segundo nivel de la transacción.

Eventos de disparo: **AfterInsert**, **AfterUpdate**, **AfterDelete**
Así como existe un evento de disparo que permite definir que determinadas reglas se ejecuten inmediatamente antes de que se produzca la grabación física de cada instancia de un nivel (AfterValidate, BeforeInsert, BeforeUpdate, BeforeDelete), también existen eventos de disparo para definir que ciertas reglas se ejecuten **inmediantamente después** de que se inserten, modifiquen o eliminen físicamente instancias de un nivel. Estos eventos son AfterInsert, AfterUpdate y AfterDelete.

El evento de disparo AfterInsert permite definir que una regla se ejecute inmediatamente después de que se inserte físicamente cada instancia del nivel al cual está asociada la regla; el AfterUdate luego de que se actualice físicamente la instancia, y el AfterDelete luego de que se elimine.



Supongamos que en la transacción Customer queremos invocar a un procedimiento que realice la impresión de los datos de cada cliente con el cual se trabaje (inserte o modifique) por medio de la transacción.

¿En qué momento debemos realizar la invocación al reporte desde la transacción?

Propuesta 1: PrintCustomer(CustomerId) on AfterValidate;

No es adecuado agregarle este evento de disparo a la regla de invocación porque el procedimiento se invocaría inmediatamente antes de la grabación física de cada cliente. En consecuencia, no se encontraría al cliente con sus datos en la tabla CUSTOMER (si se estaba insertando un cliente por medio de la transacción), o lo encontraría con sus datos desactualizados (si se estaba modificando un cliente por medio de la transacción). Si en cambio se estaba eliminando un cliente por medio de la transacción, se encontrarían los datos del cliente en la tabla CUSTOMER y los listaría justamente antes de la eliminación física.

Si se deseara emitir un listado con los datos de cada cliente que se elimine, sería adecuado definir la siguiente regla:

PrintCustomer(CustomerId) on BeforeDelete;

o su equivalente:

PrintCustomer(CustomerId) if Delete on AfterValidate;

para restringir el disparo de la regla únicamente a cuando se esté eliminando un cliente, porque es el único caso en el sería correcto utilizar el evento de disparo AfterValidate.

Propuesta 2: PrintCustomer(CustomerId) on AfterInsert, AfterUpdate;

El momento de disparo AfterInsert ocurre inmediatamente después de insertada físicamente cada instancia asociada a cierto nivel de la transacción (en este caso, como el único atributo involucrado en la regla es CustomerId, se trata de una regla asociada al primer y único nivel de la transacción Customer).

Como lo indica claramente su nombre, el evento de disparo AfterInsert sólo ocurre al insertar una nueva instancia (precisamente luego de ser insertada como registro físico). Es por ello que cuando se agrega este evento de disparo a una regla, no es necesario agregarle la condición de disparo if insert.

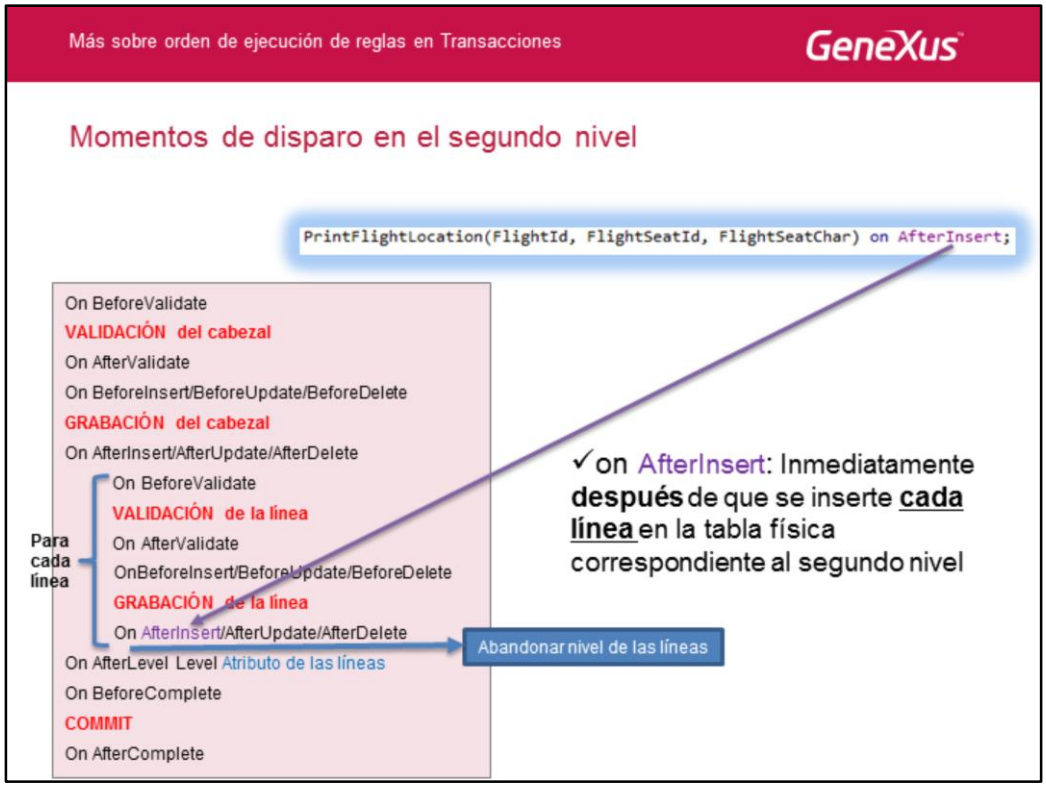
El momento de disparo AfterUpdate ocurre inmediatamente después de modificada físicamente cada instancia asociada a cierto nivel de la transacción (en este caso se trata de una regla asociada al primer y único nivel de la transacción Customer). El evento de disparo AfterUpdate sólo ocurre al modificar un registro. Es por ello que no es necesario agregarle la condición de disparo if update.

La propuesta es correcta, ya que agregarle estos momentos de disparo a la regla de invocación al procedimiento hará que éste se ejecute inmediatamente después de que se inserte o modifique cada cliente, así que el procedimiento encontrará al cliente con sus datos correctos en la tabla CUSTOMER y los imprimirá. Ahora, tener en cuenta que ¡el cliente no estará commiteado!

Propuesta 3: PrintCustomer(CustomerId) on AfterDelete;

El evento de disparo AfterDelete ocurre inmediatamente después de eliminada físicamente cada instancia asociada a cierto nivel de la transacción (en este caso, como el único atributo involucrado en la regla es CustomerId, se trata de una regla asociada al primer y único nivel de la transacción Customer).

No es correcto agregarle este evento de disparo a la regla de invocación al procedimiento porque éste se invocaría inmediatamente después de la eliminación física de cada cliente. En consecuencia, el procedimiento no encontrará al cliente con sus datos en la tabla CUSTOMER.



Si definimos una regla a la cual le incluimos el evento de disparo on AfterInsert, pero a diferencia de los ejemplos recién vistos, se referencia en la regla al menos un atributo del segundo nivel de la transacción en la cual se está definiendo la regla, la misma estará asociada al segundo nivel. Por lo tanto, la regla se ejecutará **inmediatamente después** de que se **inserte** físicamente cada instancia correspondiente al segundo nivel de la transacción.

Análogo es el caso de on AfterUpdate y on AfterDelete.

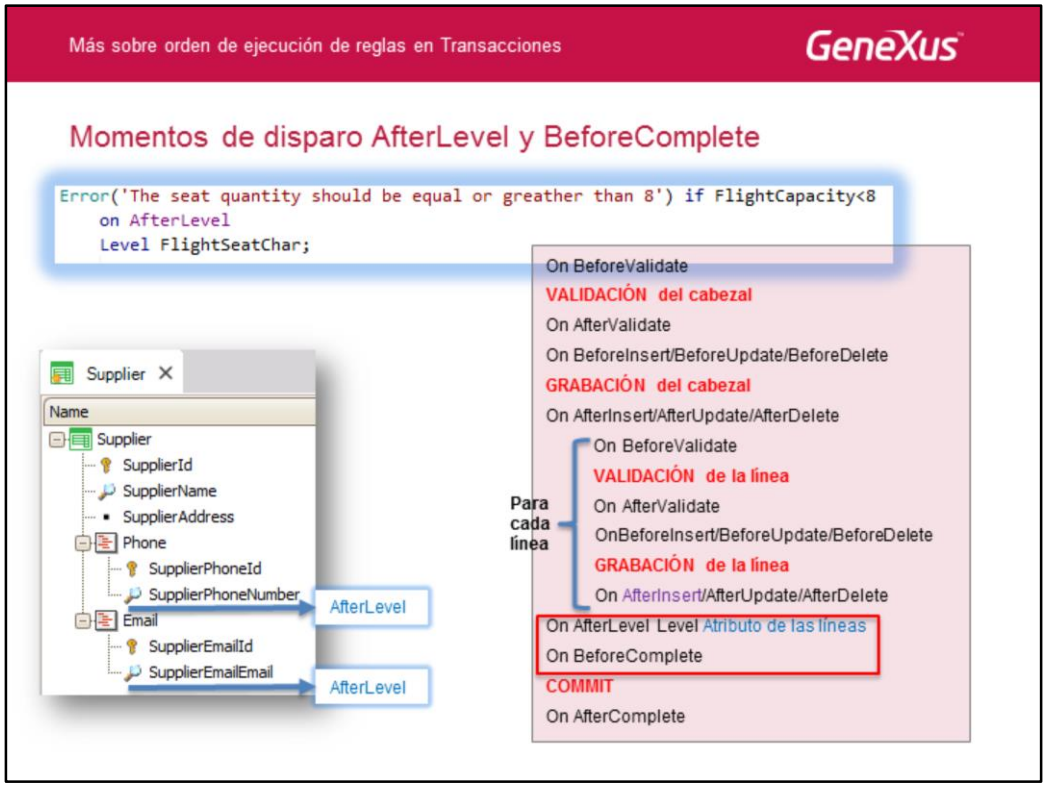
Ampliamos el esquema que habíamos efectuado antes, de las acciones que rodean a los eventos de disparo vistos hasta ahora:

VALIDACIÓN DE LOS DATOS
AfterValidate – BeforeInsert – BeforeUpdate – BeforeDelete
GRABACIÓN DEL REGISTRO (insert, update, delete según corresponda)
AfterInsert – AfterUpdate – AfterDelete

Este esquema se repite para cada instancia del nivel. Por ejemplo, pensemos en el ingreso de los asientos de un vuelo. Para cada línea con un asiento, ocurrirá este esquema, por lo que podemos pensar en un loop que se repite hasta que se termina de grabar la última línea del grid como registro físico.

La acción que sucede a la grabación de la última línea sería el abandonar ese nivel (en este caso, el nivel de los asientos de los vuelos) Y luego de esa acción, a menos que venga otro nivel con el que se volvería a ingresar en el esquema anterior, ocurrirá la última acción en la ejecución, que es el commit.

Entre la acción de abandonar el nivel, y el commit tendremos un evento (BeforeComplete) y otro para luego del commit (AfterComplete).



El evento de disparo **AfterLevel** permite definir que una regla se ejecute **inmediatamente después** de terminar de iterar determinado nivel.

SINTAXIS: *regla* [if *condición de disparo*] [**on AfterLevel Level** *atributo*];

DONDE:
regla: es una regla de las permitidas en transacciones
condición de disparo: es una expresión booleana que permite involucrar atributos, variables, constantes y funciones, así como los operadores Or, And, Not.
atributo: es un atributo perteneciente al nivel para el cual se desea que luego de ser iterado, se ejecute la regla.

Si el atributo que se especifica a continuación del evento de disparo **AfterLevel** pertenece al segundo nivel de la transacción, la regla se ejecutará cuando se hayan terminado de iterar todas las líneas del segundo nivel.

Y si el atributo que se especifica a continuación del evento de disparo **AfterLevel** pertenece al primer nivel —siguiendo el mismo concepto— la regla se ejecutará cuando se haya terminado de iterar por todos los cabezales. Observar que esto se da al final de todo, es decir, una vez que se hayan ingresado todos los cabezales y sus líneas y se cierre la transacción (en ese momento se habrán iterado todos los cabezales). Por lo tanto, si el atributo especificado pertenece al primer nivel, la regla se disparará una vez sola antes del Evento Exit (es un evento que se ejecuta una única vez cuando se cierra una transacción en tiempo de ejecución, como veremos).

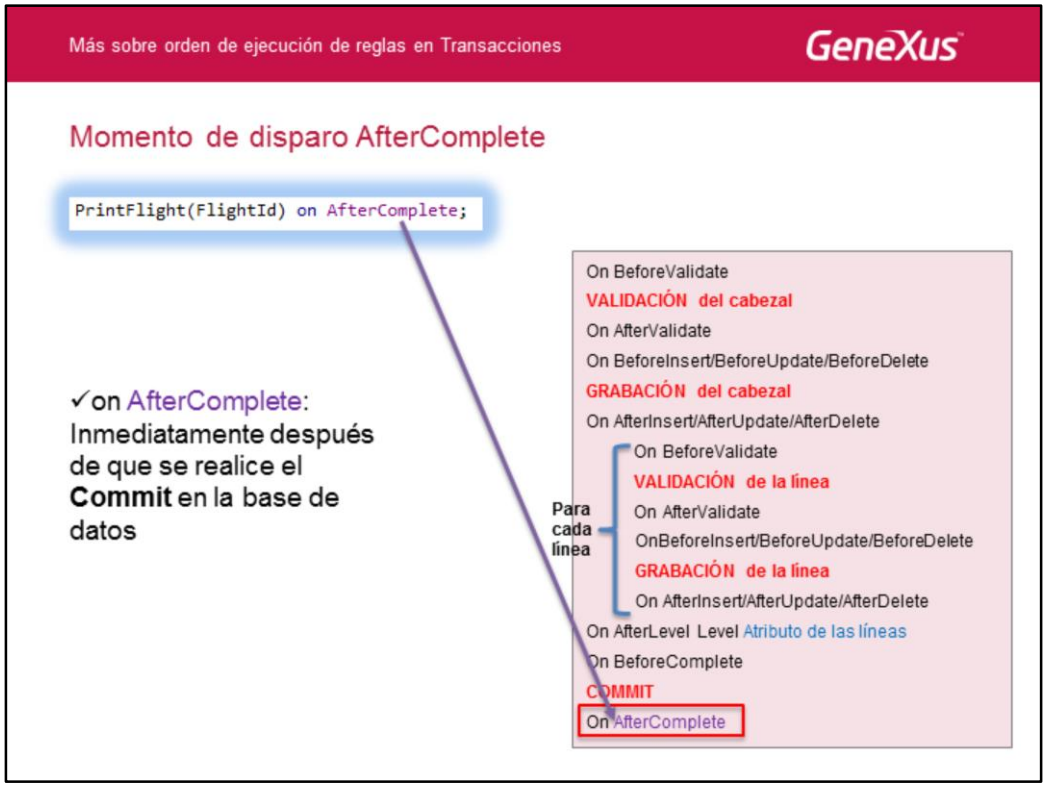
Este momento de disparo lo utilizamos en la regla que definimos para validar que ingresen 8 asientos o más para cada vuelo que vemos en pantalla.

El evento de nombre **BeforeComplete**, en este caso, coincide con el **AfterLevel**. Si observamos el esquema presentado en la página anterior, podemos ver que el instante de tiempo que hay entre que se abandona el último nivel y se realiza el commit es el instante en que ocurren estos eventos. Ambos momentos de disparo coinciden en el tiempo siempre y cuando el nivel abandonado sea el último.

Para entender mejor esto, supóngase que tenemos una transacción con dos niveles paralelos, por ejemplo una transacción Supplier donde tenemos un nivel anidado para los números telefónicos y otro para sus direcciones de mail.

El momento en que deberá dispararse una regla condicionada a: **on AfterLevel Level *SupplierPhoneNumber*** NO COINCIDIRÁ con el de una regla condicionada a on BeforeComplete, pero sí las reglas condicionadas a **on AfterLevel Level *SupplierEMailEMail*** por ser el último nivel subordinado de la transacción.

Mientras que la primera se disparará cuando se abandona el nivel de los teléfonos, y antes de entrar a validar todos los emails, la segunda se disparará después de abandonar este último nivel.
En este caso el evento **BeforeComplete** coincidirá con el **AfterLevel Level *SupplierEMailEMail***.

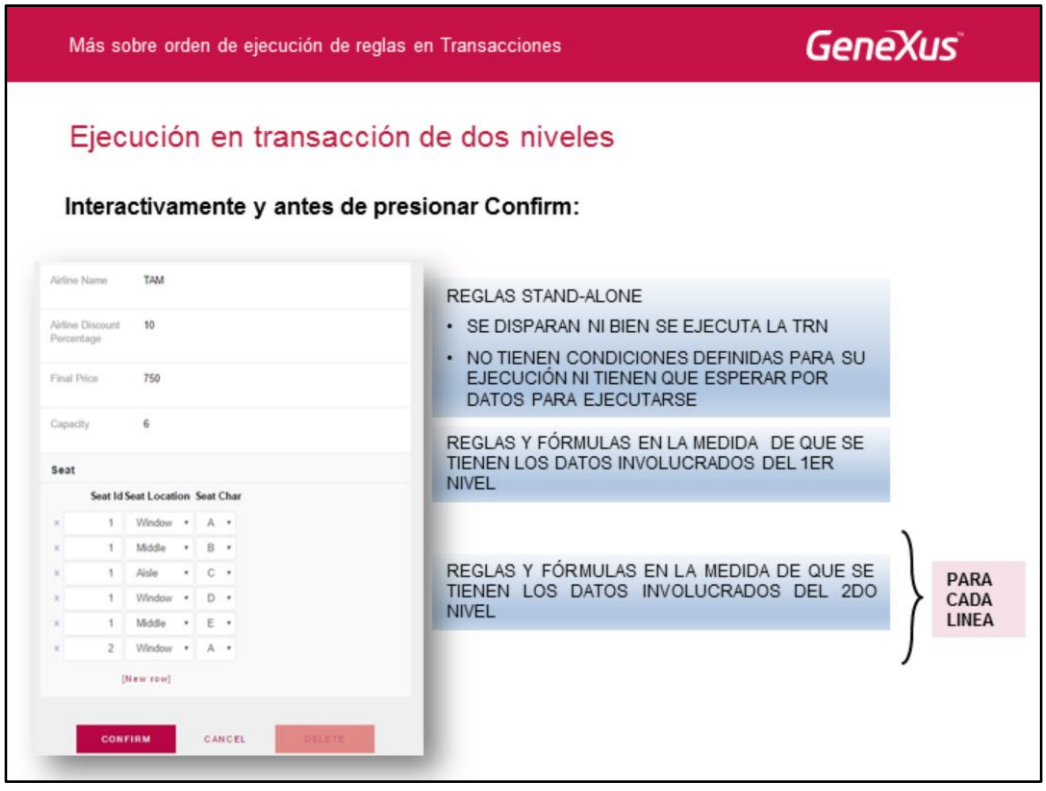


Este evento corresponde al instante de tiempo inmediatamente posterior al commit.

Si en la transacción Flight se ingresan 3 vuelos (información del 1er nivel + sus respectivos asientos) y se cierra la transacción, habrán ocurrido 3 commits y a continuación de cada commit se habrán ejecutado las reglas con evento de disparo **on AfterComplete**.

Los atributos del cabezal aún se encuentran con valor en memoria cuando se ejecutan las reglas con evento de disparo AfterComplete. Si bien ya se ejecutó el commit, es muy típico invocar a un procedimiento que lista la información grabada y commiteada, pasándole por parámetro la clave primaria al objeto llamado.

Hablaremos más de este evento cuando estudiemos la **integridad transaccional**.



A medida de que el usuario va ingresando datos en la transacción y antes de presionar Confirm, se irán disparando las reglas y fórmulas de acuerdo al árbol de evaluación, en la medida de que los distintos atributos estén involucrados. Las primeras reglas en dispararse serán las reglas stand-alone, que son aquellas reglas que:

1. Pueden ejecutarse con la información provista por los parámetros recibidos.
2. No dependen de nada para ejecutarse.

Ejemplos de reglas stand-alone (por poder ejecutarse con la información provista por los parámetros):

```
&A = parámetro2;
Msg( '...' ) if parámetro1 = 7;
```

Ejemplos de reglas stand alone (por no depender de nada para ejecutarse):

```
msg( 'You are in the flight transaction');
&A = 7;
```

Por lo tanto, son las primeras reglas que se ejecutan.

Luego de la ejecución de las reglas stand alone, se ejecutan las reglas y fórmulas asociadas al primer nivel de la transacción que no tengan evento de disparo definido (es decir que no tengan especificado **on** …) en la medida de que se vaya disponiendo de los valores involucrados para ejecutarlas.

Y al trabajar en el 2do nivel (grid), para cada línea se ejecutan las reglas y fórmulas asociadas al 2do nivel de la transacción que no tengan evento de disparo definido (es decir que no tengan especificado **on**…) en la medida de que se vaya disponiendo de los valores involucrados para ejecutarlas.

Luego de que el usuario presiona Confirm, los datos viajan al servidor web. El servidor vuelve a recorrer el form como si fuera un usuario y las reglas y fórmulas volverán a dispararse, pero ahora si hay alguna regla condicionada a un momento de disparo en el servidor, se disparará en su momento específico y no de acuerdo al árbol de evaluación.

El orden de ejecución será el siguiente:

Se ejecutan las reglas stand-alone.

Se ejecutan todas las reglas y fórmulas asociadas al 1er nivel que no tienen definido evento de disparo.

Se ejecutan las reglas asociadas al 1er nivel (cabezal) con eventos de disparo **BeforeValidate**.

Se ejecuta la VALIDACIÓN de los datos ingresados para el 1er nivel (cabezal).

Se ejecutan las reglas asociadas al 1er nivel (cabezal) con eventos de disparo **AfterValidate**.

Seguidamente a la ejecución de las reglas asociadas al primer nivel con alguno de estos eventos de disparo se ejecuta la acción de **grabación**; es decir, *se grabará físicamente la instancia correspondiente al primer nivel de la transacción como registro físico en la tabla correspondiente (en este ejemplo, en la tabla FLIGHT)*.

Inmediatamente después de haberse grabado esa instancia:

- si la grabación correspondió a una inserción: se ejecutarán las reglas asociadas

al primer nivel de la transacción con evento de disparo **AfterInsert**.
- si la grabación correspondió a una actualización: se ejecutarán las reglas asociadas al primer nivel de la transacción con evento de disparo **AfterUpdate**.
- si la grabación correspondió a una eliminación: se ejecutarán las reglas asociadas al primer nivel de la transacción con evento de disparo **AfterDelete**.

Luego de haberse ejecutado todas las operaciones explicadas hasta el momento, se efectuará un COMMIT y a continuación se ejecutarán las reglas con evento de disparo AfterComplete.

Es de fundamental importancia que quede claro que todas las operaciones explicadas se ejecutarán en el orden en el que se han descrito para cada vuelo con el cual se trabaje por medio de la transacción Flight (ya sea que se ingrese, modifique o elimine).

Es importante asimilar el orden en el que se ejecutan las reglas en una transacción, cuáles son los eventos de disparo disponibles para asignarles, cuándo se disparan exactamente, y qué acciones ocurren antes y después de cada evento de disparo, ya que solamente conociéndolos bien se podrá programar el comportamiento de las transacciones adecuadamente.

Ejercicios

¿Cuándo se dispararán las siguientes reglas?

- *Something(FlightId) on BeforeInsert;*

  Un instante antes de insertar el registro correspondiente al 1er nivel.

- *Something(FlightId, FlightSeatId, FlightSeatChar) on BeforeInsert;*

  Un instante antes de insertar cada registro correspondiente al 2do nivel.

- *Something(FlightId) on BeforeInsert Level FlightSeatChar;*

  Ídem que el anterior. Observar que **Level FlightSeatChar** especifica que se está hablando del BeforeInsert de las líneas y no del cabezal.

Ejercicios

Algunas reglas están mal programadas ¿Cuáles?

- *FlightPrice = 2000 on AfterInsert;*

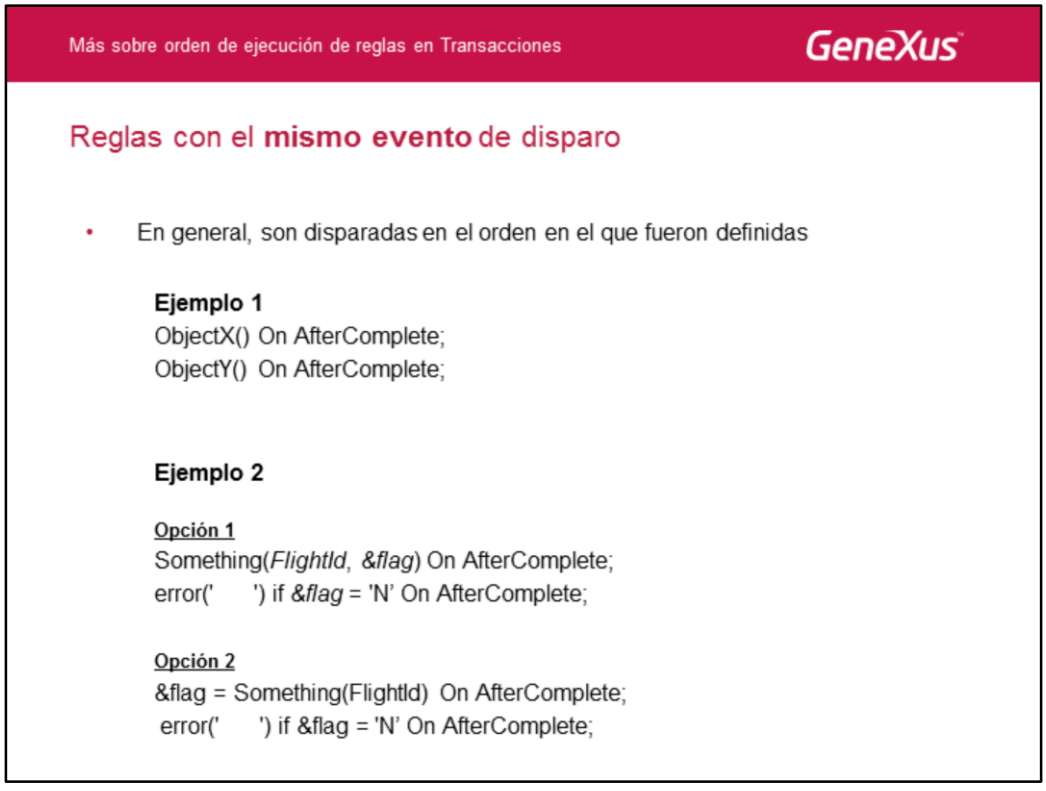
  Incorrecto: El último momento para asignar valor a un atributo del cabezal es inmediatamente antes de su grabación (BeforeInsert).

- *Something(FlightPrice) on AfterInsert;*

  Correcto: aquí se está pasando el valor de un atributo del cabezal. Se dispone de ese valor en memoria. Último momento posible para utilizarlo AfterComplete.

- *Something(FlightId,FlightSeatId,FlightSeatChar) on AfterLevel Level FlightSeatChar;*

  Incorrecto: la regla sin el evento de disparo está asociada al 2do nivel, es decir, se dispararía por cada línea. Pero el evento de disparo la condiciona a ejecutarse al salir de las líneas… y ya no tenemos valores de atributos del 2do nivel.



Cuando en una transacción se definen dos o más reglas con el mismo evento de disparo y no existe ninguna dependencia entre ellas, las mismas se ejecutarán respetando el orden de definición.

Ejemplos:

Ejemplo 1: Como las dos reglas definidas están condicionadas con el mismo evento de disparo y no existe ninguna dependencia entre ellas, las mismas se ejecutarán en el mismo orden en el cual se han escrito.

Ejemplo 2: En una transacción se necesita invocar a un procedimiento que realiza determinada validación y retorna un valor ‘S’ o ‘N’; si el valor devuelto es ‘N’, se debe dar un mensaje de error.

Para resolver esto, evaluaremos las dos opciones que se muestran arriba.

En la primera alternativa se ha definido una regla que invoca como programa a un objeto y una regla error. Ambas reglas tienen el mismo evento de disparo, y aparentemente existiría dependencia entre ellas, ya que la regla de error está condicionada al valor de la variable &flag, y la variable &flag se pasa por parámetro en la invocación.

Sin embargo, si bien la dependencia nos puede parecer evidente porque en el procedimiento programaremos la variable &flag como de salida en la sección de reglas de la transacción —que es donde se encuentran las reglas que estamos viendo—, el especificador de GeneXus no puede saber si los parámetros pasados en una invocación como programa son de entrada, de salida, o de entrada-salida; en consecuencia el especificador no encontrará interdependencia entre las reglas call y error, ya que la variable &flag podría ser pasada como variable de entrada al

procedimiento, y en ese caso por ejemplo, no habría una dependencia por la cual primero se deba ejecutar la invocación como programa y luego la regla error.

Así que concluyendo, no se detectan dependencias entre las reglas de la opción 1, por lo que las mismas se dispararán entonces en el orden en el que estén escritas. Es importante ver que si las reglas estuvieran escritas en orden inverso (es decir, primero la regla error y después la invocación como programa), el comportamiento no será el esperado en muchos casos.

Con respecto a la segunda alternativa, observemos que la misma consiste en una regla que invoca como función al objeto *Something* y una regla error. Ambas reglas tienen el mismo evento de disparo, y en este caso sí existe dependencia entre ellas, ya que la regla error está condicionada al valor de la variable &flag, y como la invocación al procedimiento se realiza como función, para GeneXus queda claro que la variable &flag vuelve modificada del procedimiento; por lo tanto GeneXus entiende que primero se debe disparar la invocación al procedimiento como función y luego la regla error, porque la variable &flag se carga mediante la invocación al procedimiento y luego de que dicha variable tenga valor, es que habrá que evaluar si disparar la regla error, o no.

En el caso Opción 2 entonces, independientemente del orden de definición de ambas reglas, la invocación al procedimiento como función se disparará primero, y luego de ello, se disparará la regla error (en caso de que se cumpla la condición de disparo, claro está).

Por esta razón se recomienda que siempre que se quieran definir validaciones de este tipo, invocar como función en vez de como programa.


GeneXus™
Videos
training.genexus.com
Documentación
wiki.genexus.com
Certificaciones
training.genexus.com/certificaciones