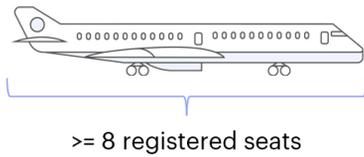


# Eventos de disparo de reglas en transacciones

Continuación

*GeneXus™*



```

Flight X
Structure | Web Form | Rules | Events | Variables | Patterns
1 Error("The seat quantity mustn't be less than eight")
2   if FlightCapacity < 8;

```

### Flight

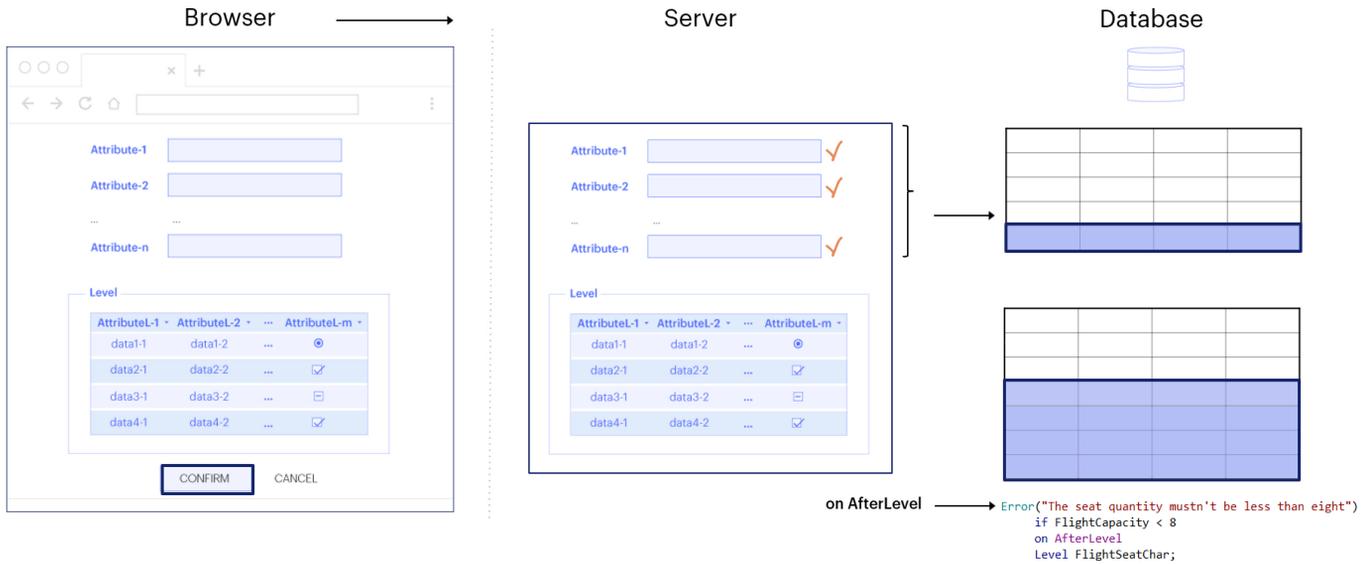
Navigation: << < > >> SELECT

Id	0	The seat quantity mustn't be less than eight
Departure Airport Id		
Departure Airport Name		
Departure Country Id	0	
Departure Country Name		
Departure City Id	0	
Departure City Name		
Arrival Airport Id	0	

An orange arrow points upwards from the 'Departure Airport Id' field towards the error message.

En el video anterior vimos que hay casos en los que el momento elegido por GeneXus para ejecutar una regla no es el que necesitamos, por lo que debemos poder indicarle a la regla cuál es el momento adecuado. Estudiamos el caso en el que necesitábamos controlar que cada vuelo tuviera al menos 8 asientos registrados; como GeneXus ejecutaba la regla antes de dar tiempo al usuario a que registrara los asientos,

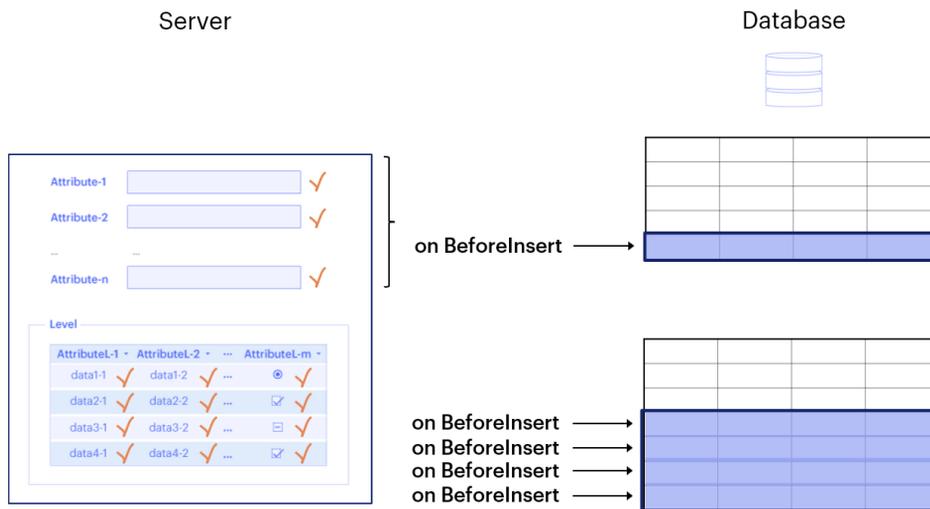
## AfterLevel event



debimos retrasar el momento que había elegido inicialmente para dispararla, utilizando el momento de disparo *on AfterLevel* con un atributo de las líneas (*FlightSeatChar*).

Con esto logramos que la regla se disparara después de recorrer el nivel asociado a los asientos.

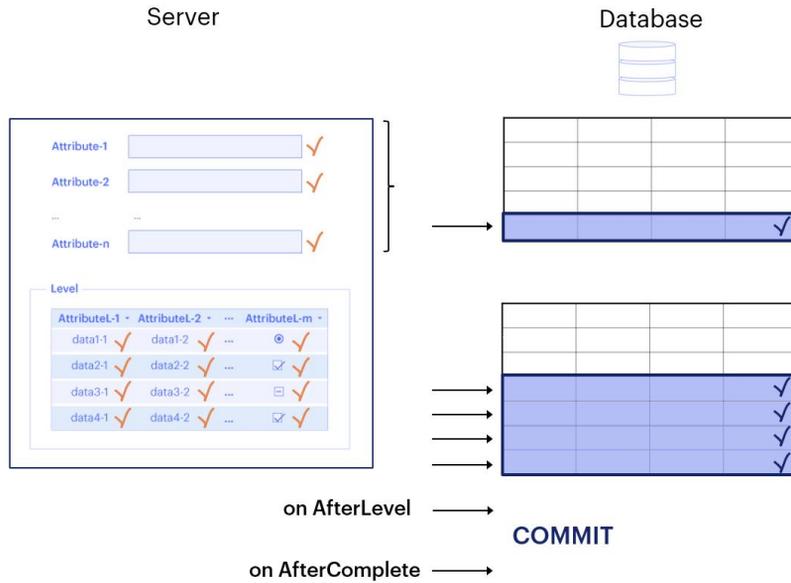
## BeforeInsert event



Mencionamos también que disponemos de otros momentos de disparo, como "on BeforeInsert" si quisiéramos hacer o evaluar algo inmediatamente antes de que los datos del cabezal o de cada línea sean insertados en la base de datos,

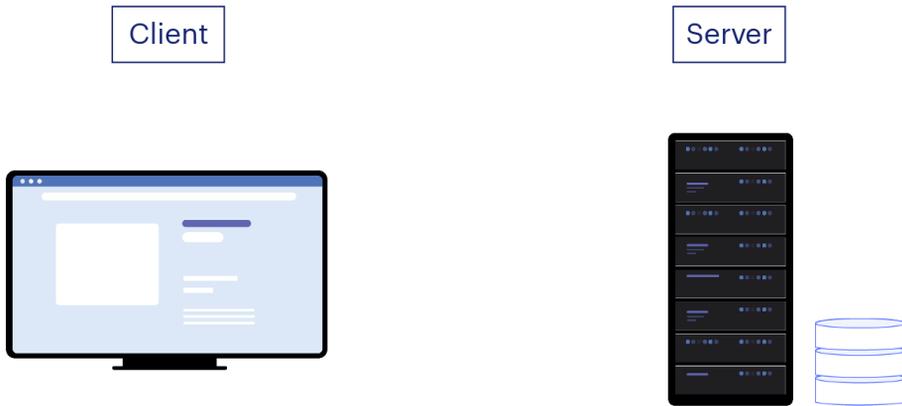


## AfterComplete event



y el momento "on AfterComplete", que corresponde al instante de tiempo inmediatamente posterior al Commit, comando cuyo objetivo es dar por buenos los datos insertados, modificados o eliminados.

## Execution of rules



En este video, analizaremos con más profundidad estos momentos de disparo. Pero antes, recordemos que hay reglas que son validadas tanto en el cliente (browser) como en el servidor, y otras que lo son exclusivamente en el servidor, porque tienen que ver con la base de datos.

```
Error("It is not possible to leave a flight without a price.")  
if FlightPrice.IsEmpty();
```

Arrival Country Id	2
Arrival Country Name	France
Arrival City Id	1
Arrival City Name	Paris
Price	<input type="text" value="0.00"/> <span style="background-color: yellow;">It is not possible to leave a flight without a price.</span>
Discount Percentage	<input type="text" value="0"/>
Airline Id	<input type="text" value="0"/> <input type="button" value="↑"/>
Airline Name	
Airline Discount Percentage	0
Final Price	0.00



Por ejemplo: si tuviéramos una regla Error que impidiera ingresar o modificar un vuelo sin precio, apenas saliéramos de ese campo aparecería el mensaje de error.

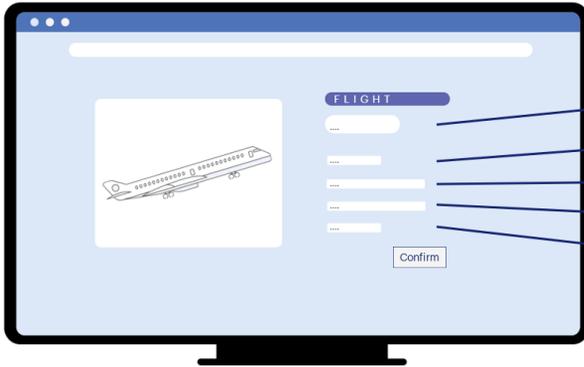
CLIENT

```
Error("It is not possible to leave a flight without a price.")
if FlightPrice.IsEmpty();
```

Price  It is not possible to leave a flight without a price.

SERVER

```
Error("It is not possible to leave a flight without a price.")
if FlightPrice.IsEmpty();
```



### Client-Side validation

Esto ocurre porque el cliente realiza la validación para brindar una buena experiencia al usuario, dándole respuestas de forma ágil para que sienta que su interacción con el sistema es fluida. A esta validación realizada del lado del cliente, se le llama Client-Side Validation.

Pero esta regla volverá a ejecutarse posteriormente en el servidor, ya que es él quien se encarga de que no haya violaciones de seguridad, y es el único que tiene permitido operar sobre la base de datos, por lo que debe asegurarse de que toda la lógica sea consistente. Ejecutará todas las reglas de vuelta, teniendo toda la información. Eso sucede luego de que el usuario presione el botón Confirm.

Allí los datos viajan desde el cliente web (browser) al servidor web, y éste vuelve a recorrer el form de arriba a abajo, como si fuera un usuario, por lo que todas las reglas y fórmulas volverán a dispararse en el orden que corresponda a los atributos del form a los que están asociadas.

Pero además, todas las reglas que tengan un evento de disparo asociado (es decir, que cuenten con el prefijo **on** al final de su declaración), serán ejecutadas en el momento correspondiente a ese evento. Esos eventos solo ocurren en el servidor y capturan el momento anterior o posterior a un hito, que en general tiene que ver con la base de datos.

## Milestones when inserting a flight

- Validate the header record to be inserted
- Insert the header record in the table

Then the same for each line:

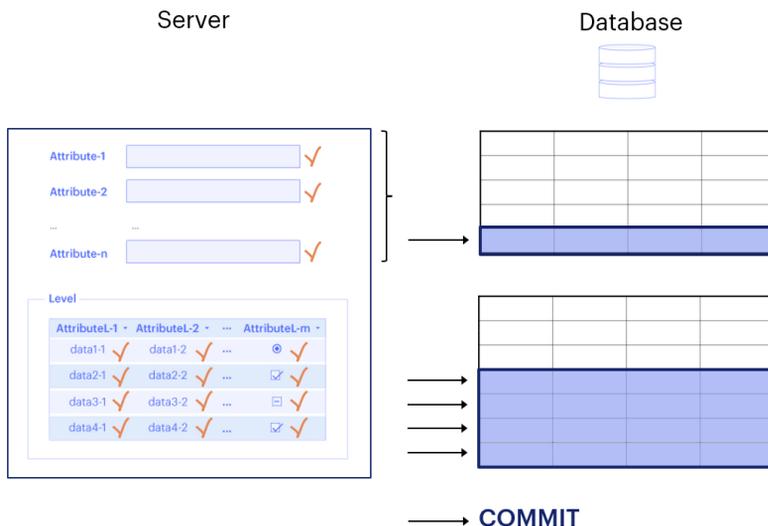
- Validate the line record to be inserted
- Insert the line record in the table

The next milestone is when:

- You finish working with the level

And the next one is:

- The Commit action



¿Cuáles son esos hitos?

Pensemos, por simplicidad, que estamos insertando un vuelo. En orden:

- Dar por válido el registro del cabezal que se va a insertar (asegura que los controles de integridad referencial y de duplicados no fallaron; ocurre después de haber pasado por cada campo del cabezal y de haber ido disparando cada regla asociada, al final de todo eso)
- Insertar el registro del cabezal en la tabla correspondiente

Y luego lo mismo para cada línea:

- Dar por válido el registro de la línea que se va a insertar (asegura que los controles de integridad referencial y de duplicados no fallaron)
- Insertar el registro de la línea en la tabla correspondiente

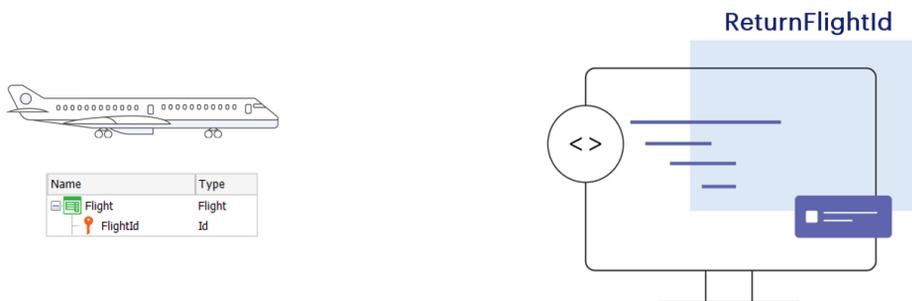
Al finalizar este proceso, donde tanto el registro correspondiente al cabezal como los correspondientes a todas las líneas fueron insertados en la base de datos, el hito siguiente es cuando:

- Se termina de trabajar con el nivel

Y el siguiente es:

- La acción de Commit, que da por buenas todas esas operaciones sobre la base de datos, las deja permanentes.

Entonces, tenemos eventos que capturan el momento anterior o posterior a estos hitos, de manera tal de poder ejecutar reglas en esos momentos precisos.



```

Flight * X
Structure | Web Form | Rules * | Events | Variables | Patterns
1 FlightId = ReturnFlightId();
2
3 Error("The seat quantity mustn't be less than eight")
4   if FlightCapacity < 8
5     on AfterLevel
6     Level FlightSeatChar;
7

```

Actualmente, en la transacción Flight, tenemos definido al identificador del vuelo (FlightId) como autonumerado.

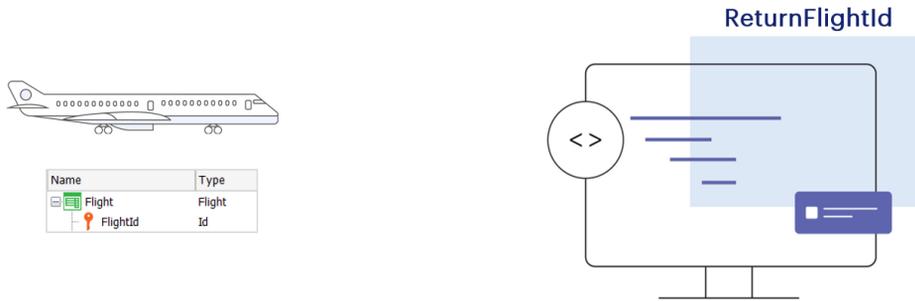
Si en la realidad, como suele ser el caso, el identificador del vuelo está formado por letras y números en función de la aerolínea, el origen y el destino, no nos sirve el atributo FlightId numérico, ni autonumerado.

Supongamos entonces que tenemos un procedimiento llamado ReturnFlightId, al que podemos invocar para que se ejecute y nos devuelva el identificador a ser asignado a un nuevo vuelo.

(Estudiaremos el objeto procedimiento más adelante, por lo cual usted no podrá, por el momento, reproducir lo que aquí mostraremos).

Si escribimos la regla que vemos arriba, donde lo estamos llamando y asignando su resultado al atributo FlightId, ¿en qué momento se disparará?

Ni bien se abra la transacción o se quiera editar un flight preexistente. No condicionamos el disparo de la regla al modo, por lo que se ejecutará ya sea que el usuario esté accediendo a la transacción Flight para insertar, o para modificar, o para eliminar un vuelo, cuando en realidad lo que queremos es que el procedimiento sea invocado solamente si estamos insertando un vuelo nuevo.



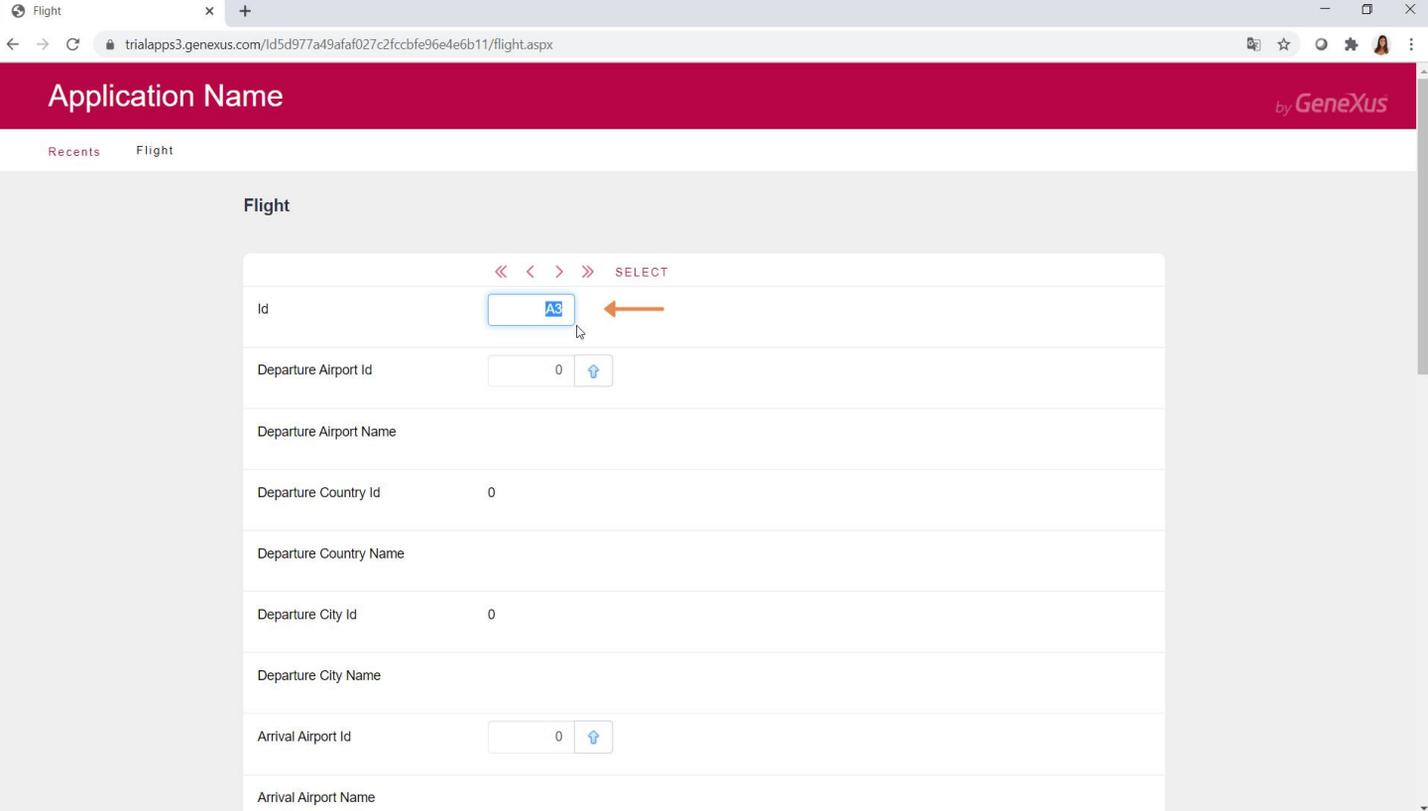
```

Flight * X
Structure Web Form Rules * Events Variables Patterns
1 FlightId = ReturnFlightId() if insert;
2
3 Error("The seat quantity mustn't be less than eight")
4   if FlightCapacity < 8
5     on AfterLevel
6     Level FlightSeatChar;
7

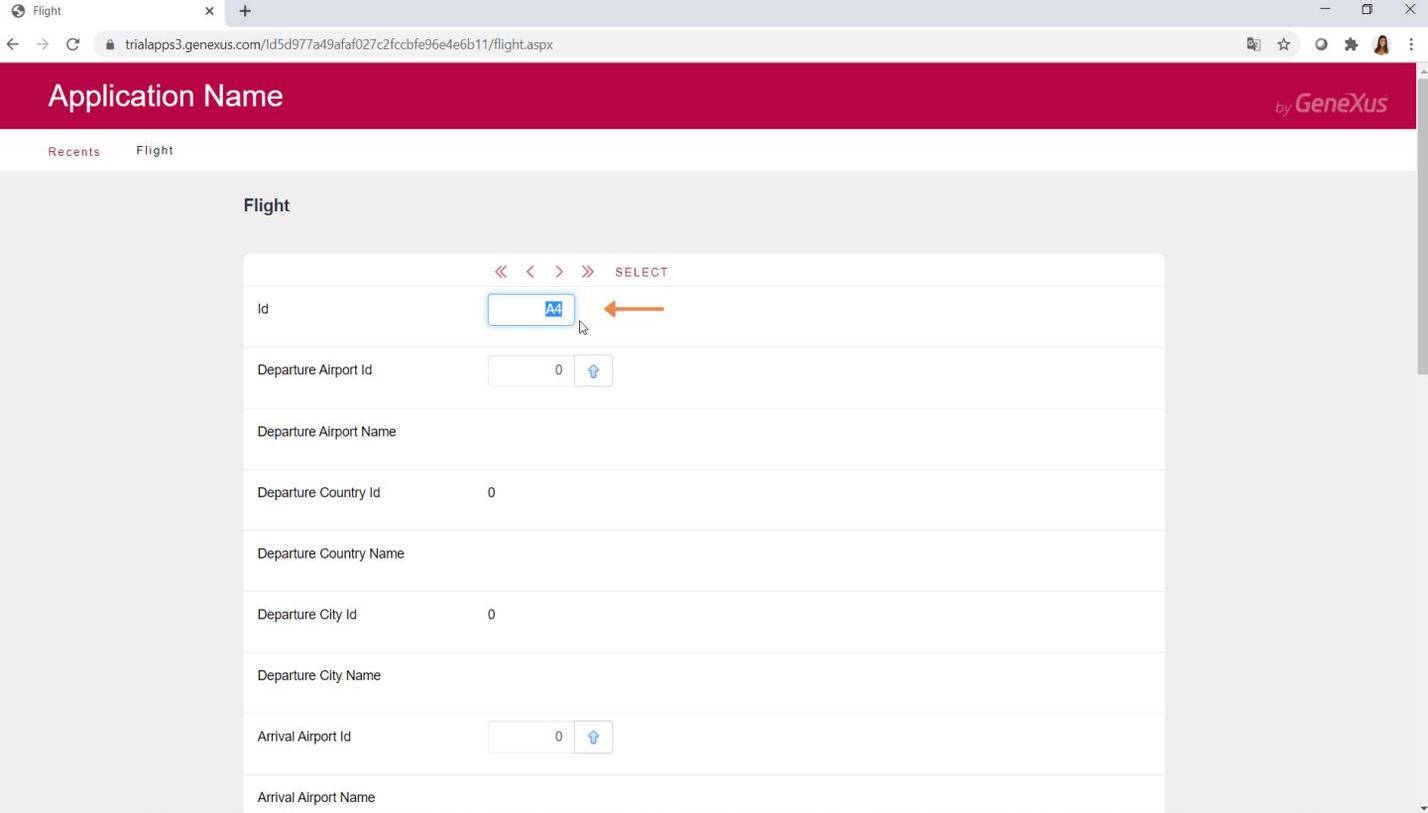
```

Vale aclarar además, que si estuviéramos en la transacción Flight en modo Update, (es decir, si el vuelo ya existiera y estuviéramos sólo modificándolo), no podríamos modificar el valor del identificador del vuelo, ya que las claves primarias no pueden modificarse. Si necesitamos modificar una clave primaria no tendremos más remedio que crear un registro nuevo con el nuevo valor de clave, y borrar el anterior.

Así que condicionamos la regla a que se ejecute solo cuando se está queriendo insertar.



Vemos que como la transacción se abre en ese modo, ya se está disparando la regla que le está asignando valor al atributo, antes de haber hecho nada.  
Pero nos podría pasar que habiendo completado datos de cabecal y alguna línea, tuviéramos que cancelar el ingreso por algún motivo, para hacerlo luego.

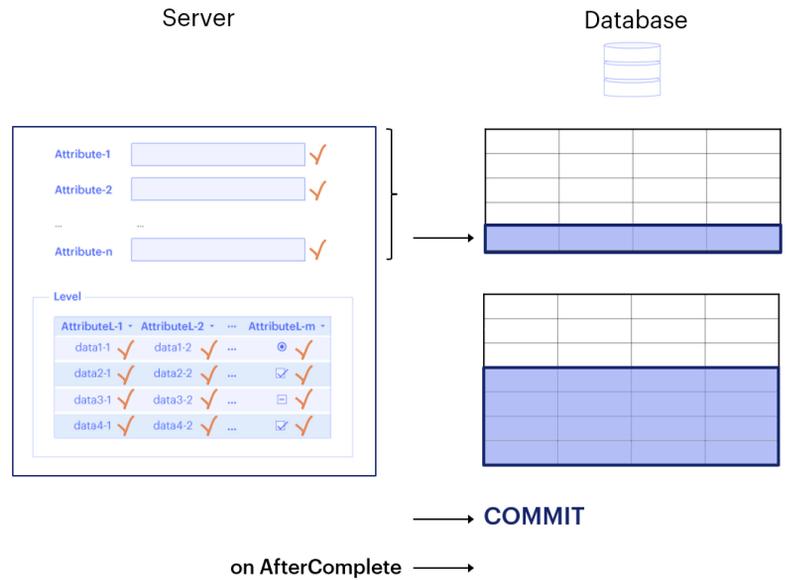


Cuando volvemos a intentarlo vemos que nos dará otro número de vuelo. Es que le habíamos pedido un número al procedimiento que al final no usamos, y ese número quedó perdido.

```

Flight* X
Structure | Web Form | Rules* | Events | Variables | Patterns
1 FlightId = ReturnFlightId() if insert on AfterComplete;
2
3 Error("The seat quantity mustn't be less than eight")
4   if FlightCapacity < 8
5     on AfterLevel
6     Level FlightSeatChar;

```



Para asegurarnos de no gastar un número que no usaremos, ¿qué tal si llamamos al procedimiento que nos da el número después del Commit, porque allí estamos completamente seguros de que todo quedará en la base de datos?

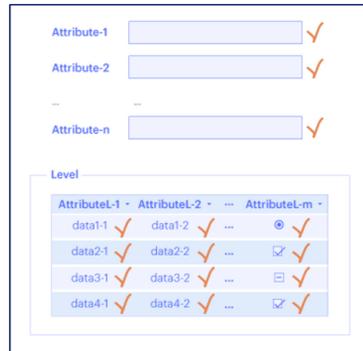
```

Flight * X
Structure | Web Form | Rules * | Events | Variables | Patterns
1 FlightId = ReturnFlightId() on BeforeInsert;
2
3 Error("The seat quantity mustn't be less than eight")
4   if FlightCapacity < 8
5     on AfterLevel
6     Level FlightSeatChar;

```

Server

Database



on BeforeInsert

COMMIT

Hmmm, no, ¡es muy tarde! ¿Por qué? Porque FlightId es un atributo del cabezal. El último momento para darle valor es el inmediatamente anterior a que se inserte el registro en la base de datos, y ese momento es el BeforeInsert. Ya no necesitamos condicionar a If Insert, pues ya está contemplado en el evento de disparo, que solo ocurrirá si se está queriendo insertar:

Flight

Application Name by Genexus

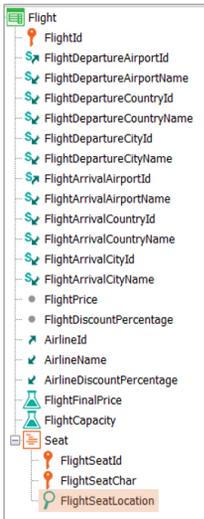
Recents Flight

### Flight

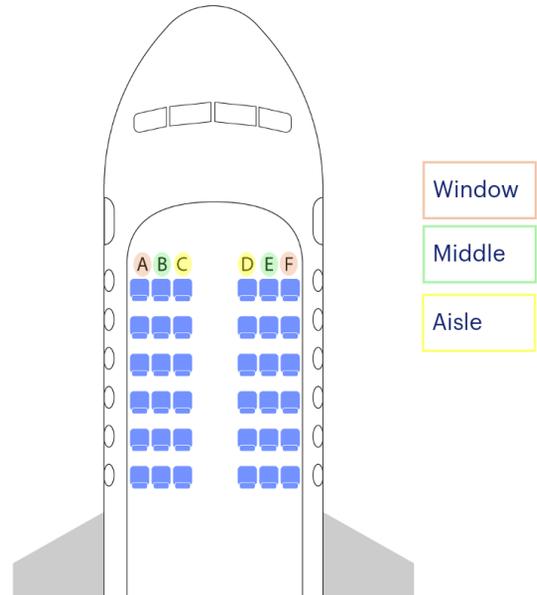
		SELECT
Id	<input type="text"/>	←
Departure Airport Id	<input type="text" value="0"/> <input type="button" value="↑"/>	
Departure Airport Name		
Departure Country Id	<input type="text" value="0"/>	
Departure Country Name		
Departure City Id	<input type="text" value="0"/>	
Departure City Name		
Arrival Airport Id	<input type="text" value="0"/> <input type="button" value="↑"/>	
Arrival Airport Name		

Si ejecutamos, vemos que ya no le asigna de entrada el número al Id. Ni siquiera lo hace cuando pasamos a ingresar líneas.





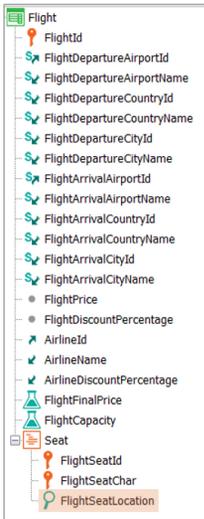
Seat			
	Seat Id	Seat Char	Seat Location
x	1	A	Window
	0	A	Window
			[New row]



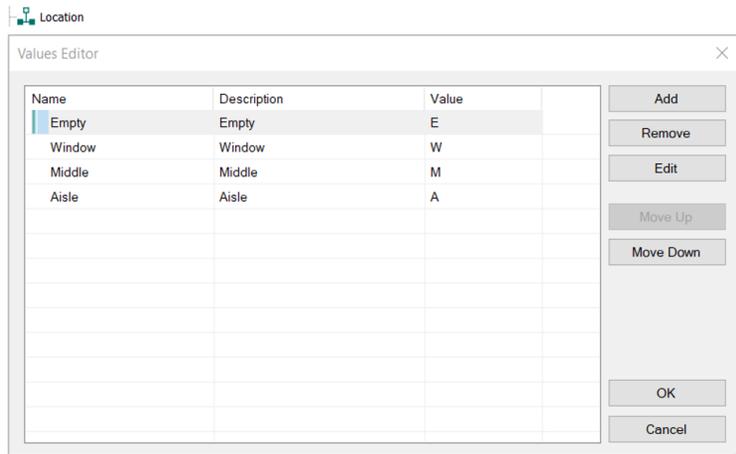
Veamos un ejemplo de BeforeInsert de las líneas:

Supongamos que queremos que el valor para el atributo FlightSeatLocation, en lugar de que sea elegido por el usuario que está trabajando con la transacción, sea asignado por una regla, que coloque el valor "Ventana" cuando el valor del atributo FlightSeatChar es A o F, "Medio" cuando el valor de FlightSeatChar es B o E, y "Pasillo" cuando su valor es C o D.

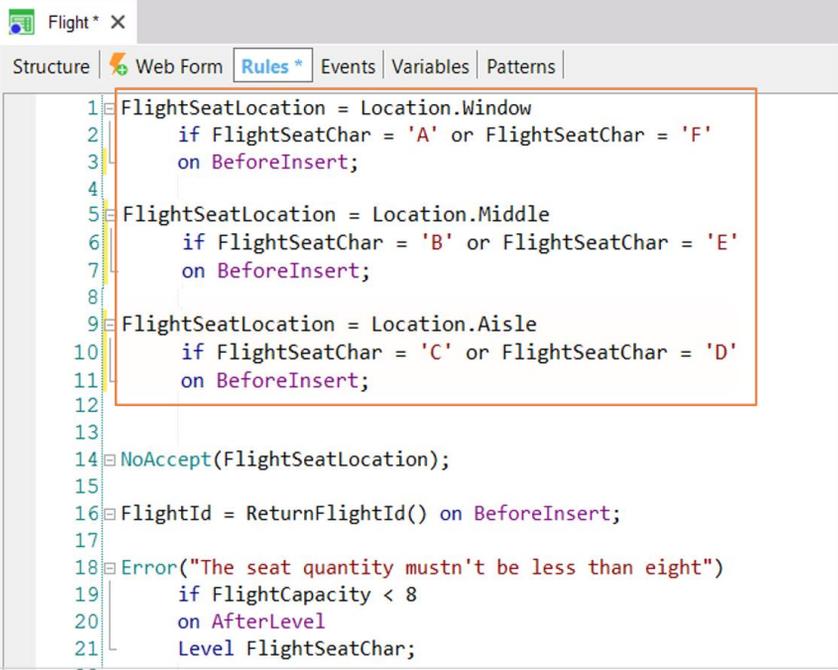
Aclaremos que esto podríamos resolverlo sin necesidad de utilizar eventos de disparo, y hasta sería mejor, porque el usuario vería inmediatamente en la pantalla el valor del SeatLocation, puesto que la regla se ejecutaría inmediatamente en el cliente, pero, solo a los efectos de entender el momento de disparo, imaginemos que nos interesara hacer esta asignación únicamente si estamos seguros de que el asiento se va a ingresar al vuelo, es decir, inmediatamente antes de insertar la línea.



```
NoAccept(FlightSeatLocation);
```



Usaremos la regla NoAccept para impedir que el usuario escoja la ubicación, agregaremos el valor "Empty" al dominio Location,



```
1 FlightSeatLocation = Location.Window
2   if FlightSeatChar = 'A' or FlightSeatChar = 'F'
3     on BeforeInsert;
4
5 FlightSeatLocation = Location.Middle
6   if FlightSeatChar = 'B' or FlightSeatChar = 'E'
7     on BeforeInsert;
8
9 FlightSeatLocation = Location.Aisle
10  if FlightSeatChar = 'C' or FlightSeatChar = 'D'
11    on BeforeInsert;
12
13
14 NoAccept(FlightSeatLocation);
15
16 FlightId = ReturnFlightId() on BeforeInsert;
17
18 Error("The seat quantity mustn't be less than eight")
19   if FlightCapacity < 8
20     on AfterLevel
21     Level FlightSeatChar;
```

y crearemos las siguientes reglas.

Flight

trialapps3.genexus.com/Id5d977a49afaf027c2fccbfe96e4e6b11/flight.aspx

Discount Percentage: 0

Airline Id: 1

Airline Name: TAM

Airline Discount Percentage: 0

Final Price: 4000.00

Capacity: 0

Server

Database

on BeforeInsert

**Seat**

Seat Id	Seat Char	Seat Location
1	A	Empty
1	B	Empty
1	C	Empty
1	D	Empty
1	E	Empty
1	F	Empty
2	A	Empty
2	B	Empty

```

FlightSeatLocation = Location.Window
if FlightSeatChar = 'A' or FlightSeatChar = 'F'
on BeforeInsert;

FlightSeatLocation = Location.Middle
if FlightSeatChar = 'B' or FlightSeatChar = 'E'
on BeforeInsert;

FlightSeatLocation = Location.Aisle
if FlightSeatChar = 'C' or FlightSeatChar = 'D'
on BeforeInsert;

```

[New row]

Insertemos un nuevo vuelo desde el aeropuerto de Guarulhos hasta el aeropuerto Charles de Gaulle, de precio 4000, sin descuento y aerolínea TAM. Pasemos ahora a registrar sus asientos: Completaremos la primera fila, y dos asientos más de la segunda.

Antes de la inserción física de cada línea, será ejecutada la regla correspondiente, según el FlightSeatChar que hayamos escogido.

Así, la transacción en el servidor validará los datos de la primera línea, donde SeatLocation quedará empty y paso siguiente ejecutará la primera regla, pues SeatChar es A, y entonces cambiará el SeatLocation por Window, e inmediatamente grabará la línea en la tabla.

Flight

trialapps3.genexus.com/Id5d977a49afaf027c2fccbfe96e4e6b11/flight.aspx

Airline Name	TAM
Airline Discount Percentage	0
Final Price	4000.00
Capacity	8

**Seat**

Seat Id	Seat Char	Seat Location
×	1 A	Window
×	1 B	Middle
×	1 C	Aisle
×	1 D	Aisle
×	1 E	Middle
×	1 F	Window
×	2 A	Window
×	2 B	Middle

```

FlightSeatLocation = Location.Window
if FlightSeatChar = 'A' or FlightSeatChar = 'F'
on BeforeInsert;

FlightSeatLocation = Location.Middle
if FlightSeatChar = 'B' or FlightSeatChar = 'E'
on BeforeInsert;

FlightSeatLocation = Location.Aisle
if FlightSeatChar = 'C' or FlightSeatChar = 'D'
on BeforeInsert;

```

on AfterInsert?

0	A	Empty
0	A	Empty
0	A	Empty

Luego pasará a hacer lo mismo con la segunda línea: validará todos los campos, SeatChar estará con valor Empty, y luego ejecutará las reglas BeforeInsert que correspondan, que en este caso será la segunda, cambiando el valor de FlightSeatLocation por Middle, e inmediatamente grabará la segunda línea en la tabla FlightSeat. Y así seguirá con las demás líneas.

Vemos que la ubicación quedó correctamente asignada.

Pero... ¿podríamos haber condicionado estas reglas al momento on AfterInsert?

Hagamos el cambio en GeneXus, y veamos qué sucede en ese caso:

Flight

trialapps3.genexus.com/Id5d977a49afaf027c2fccbfe96e4e6b11/flight.aspx

### Seat

Seat Id	Seat Char	Seat Location
1 A	Window	
1 B	Middle	
1 C	Aisle	
1 D	Aisle	
1 E	Middle	
1 F	Window	
2 A	Window	
2 B	Middle	
3 A	Empty	

0 A ▾ Empty

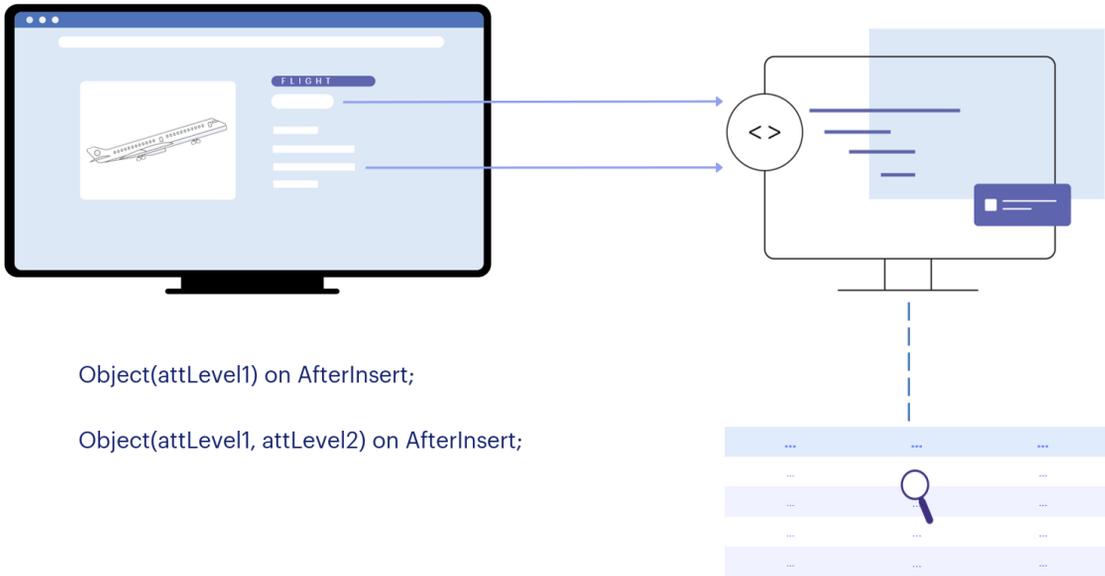
[New row]

CONFIRM CANCEL DELETE

Vayamos a modificar el vuelo que acabamos de crear, y agreguémosle un nuevo asiento:  
3 A.

Veamos ahora cómo quedó el vuelo: ¡el asiento que acabamos de registrar quedó con ubicación vacía! ¿Por qué?

Porque el evento de disparo `on AfterInsert`, se ejecuta después de que el registro fue grabado en la tabla correspondiente, por lo cual ya es demasiado tarde para asignar un valor a un atributo. Como ya hemos mencionado, el último momento disponible para asignar valores, es el `BeforeInsert`, antes de que el cabezal o cada línea se haya grabado.



Object(attLevel1) on AfterInsert;

Object(attLevel1, attLevel2) on AfterInsert;

Por lo tanto, podríamos usar On AfterInsert si necesitáramos por ejemplo llamar a otro objeto de la KB pasándole solo el identificador del cabezal o de la línea y allí lo encuentre (al cabezal o a la línea) y extraiga de allí toda la otra información que necesite del registro para hacer lo que sea que tenga que hacer con eso. Para ello, debemos asegurarnos de llamar a ese objeto luego de haber insertado el cabezal o línea, según corresponda, por tanto, on AfterInsert.



PrintFlight(FlightId) on AfterInsert;

***	***	***
***	***	***
***	***	***
***	***	***
***	***	***

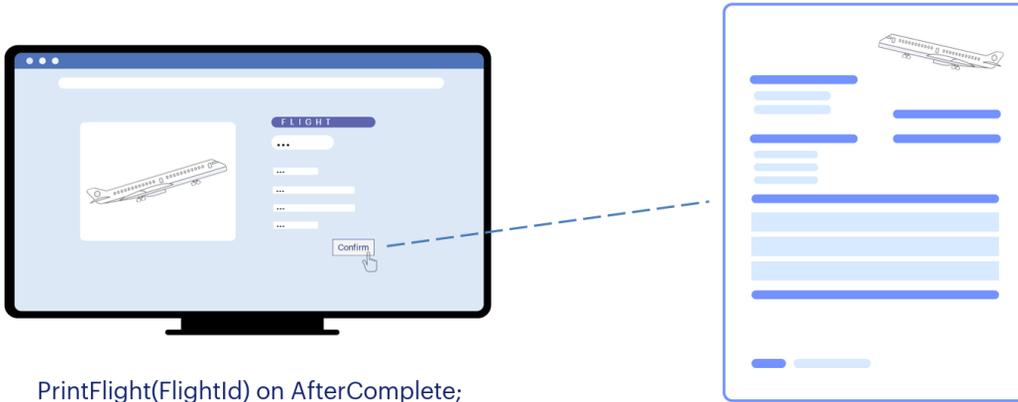


→ COMMIT

Entonces, si quisiéramos imprimir un listado con los datos de un vuelo cada vez que se inserta uno nuevo, ¿usaríamos on AfterInsert?

Podríamos, pero aquí el procedimiento solo podrá imprimir la información del cabezal, porque las líneas no se manipularon en absoluto y mucho menos se insertaron. No existen aún en la tabla de asientos. Si eso no nos importara, porque el procedimiento PrintFlight solamente va a imprimir datos del cabezal del vuelo, igual sería mala idea invocarlo on AfterInsert, ya que debemos tener en cuenta que en ese momento los datos todavía no están seguros en la tabla, porque no se ha ejecutado el Commit.

Esto quiere decir que si se produjera un corte de luz, o un error en la validación de alguno de los campos que siguen, es decir, los de las líneas, la grabación se desharía, por lo que el listado se habría generado con información que en realidad ya no existe.



PrintFlight(FlightId) on AfterComplete;



→ **COMMIT**  
on AfterComplete

Para asegurarnos de que estamos trabajando con información que ya está segura en la base de datos, disponemos del evento on AfterComplete, que sucede después del Commit.

En este caso se irá a buscar a la tabla Flight el registro correspondiente al FlightId enviado y a la tabla FlightSeat todos los registros de ese FlightId, que son todos sus asientos.



PrintFlight(FlightId) on AfterLevel Level FlightSeatChar;

***	***	***
***	***	***
***	***	***
***	***	***

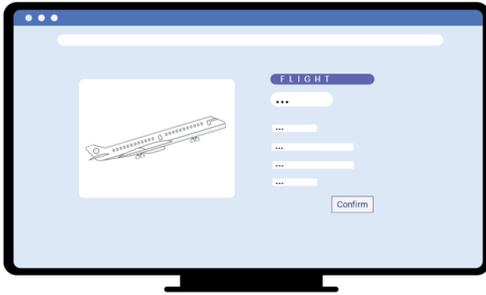
on AfterLevel

→ **COMMIT**

on AfterComplete

¿Qué pasaría si invocaríamos on AfterLevel de un atributo de las líneas?

Eso será inmediatamente antes del Commit, por lo que los registros estarán grabados en las tablas pero aún no se habrán dado por buenos, por lo que si cuando se termina de imprimir la información del vuelo hay un corte de energía, no se llegó a realizar el Commit, y cabezal y líneas serán eliminados de la base de datos.



PrintFlight(FlightId, FlightSeatId, FlightSeatChar) on AfterComplete;

...	...	...
...	...	...
...	...	...
...	...	...

→ **COMMIT**  
on AfterComplete



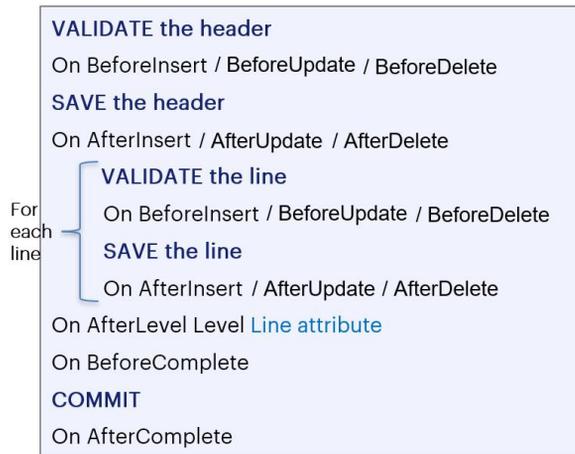
Only the header attributes are available in the on AfterComplete triggering event.

Ahora supongamos que fue declarada la siguiente regla:

¿Qué valores de FlightSeatId y de FlightSeatChar enviaría GeneXus, si en el segundo nivel hay N asientos? Esta regla no tiene sentido, es funcionalmente incorrecta.

En el momento AfterComplete, los atributos del cabezal aún se encuentran con su valor en memoria, a diferencia de los atributos del segundo nivel, cuyo valor se perdió porque ya salimos de él.

## Rule Triggering Events



En este video mostramos ejemplos de los momentos BeforeInsert y AfterInsert, que se dispararán únicamente si estamos insertando registros, pero también disponemos de BeforeUpdate y AfterUpdate para el caso en que estemos modificándolos, y BeforeDelete y AfterDelete, para el caso en que estemos eliminándolos.

Existen otros momentos de disparo que no estudiaremos en este curso. Si está interesado, puede profundizar en este tema en el curso del siguiente nivel.

# GeneXus™

[training.genexus.com](http://training.genexus.com)  
[wiki.genexus.com](http://wiki.genexus.com)  
[training.genexus.com/certifications](http://training.genexus.com/certifications)