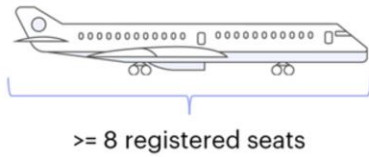


Rule Triggering Events in Transactions

Continued

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

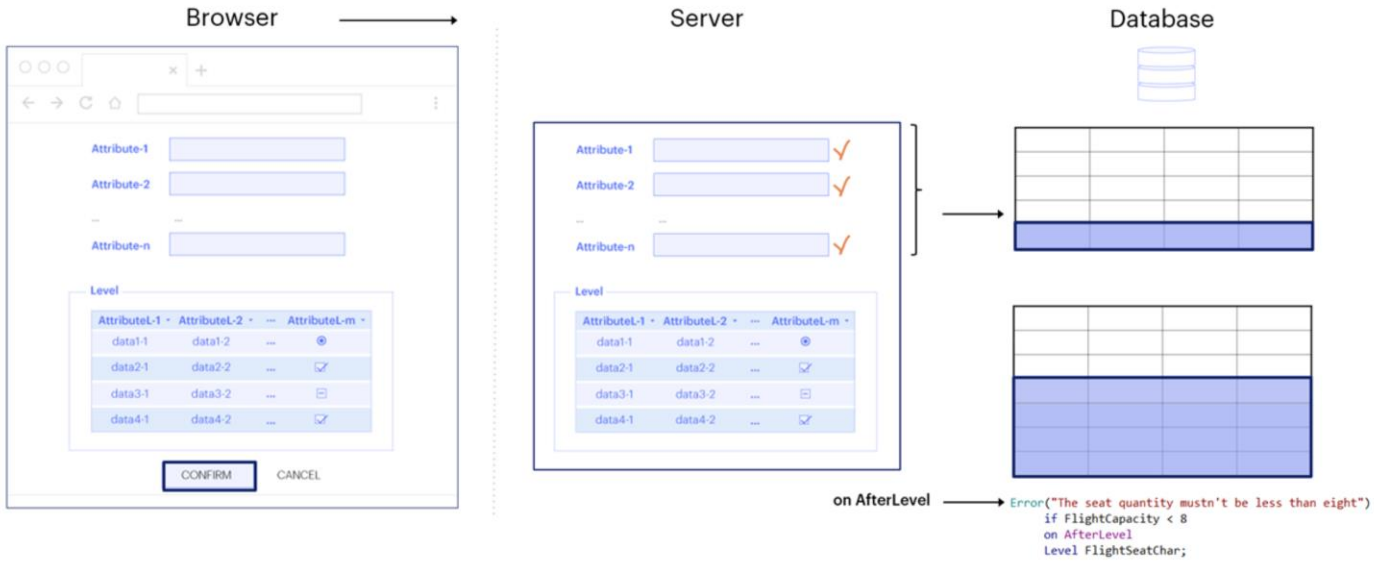
Id	<input type="text" value="0"/>	The seat quantity mustn't be less than eight
Departure Airport Id	<input type="text" value="0"/>	
Departure Airport Name		
Departure Country Id	<input type="text" value="0"/>	
Departure Country Name		
Departure City Id	<input type="text" value="0"/>	
Departure City Name		
Arrival Airport Id	<input type="text" value="0"/>	

An orange arrow points to the 'Departure Airport Id' field.

In the previous video, we saw that there are cases in which the moment chosen by GeneXus to execute a rule is not the one we need, so we must be able to tell the rule which is the right moment.

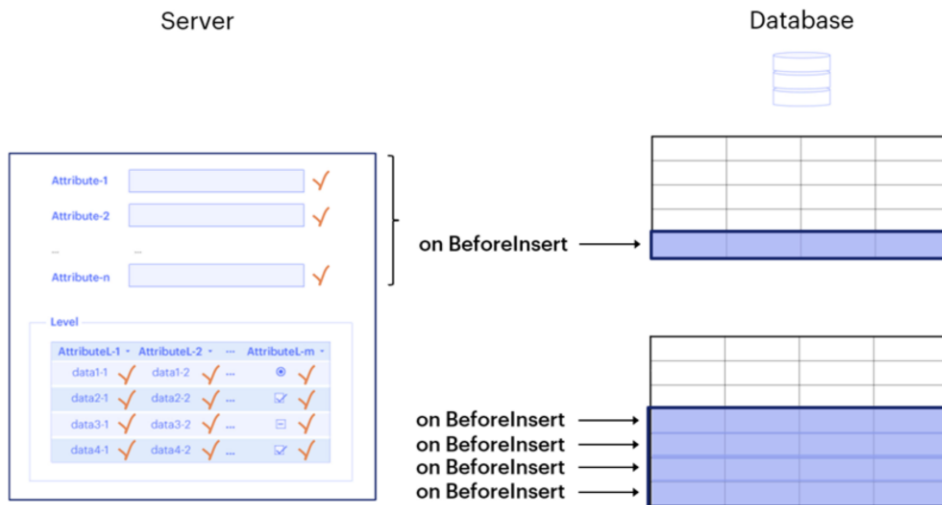
We studied the case in which we needed to control that each flight had at least 8 registered seats. Since GeneXus executed the rule before giving the user time to register the seats,

AfterLevel event



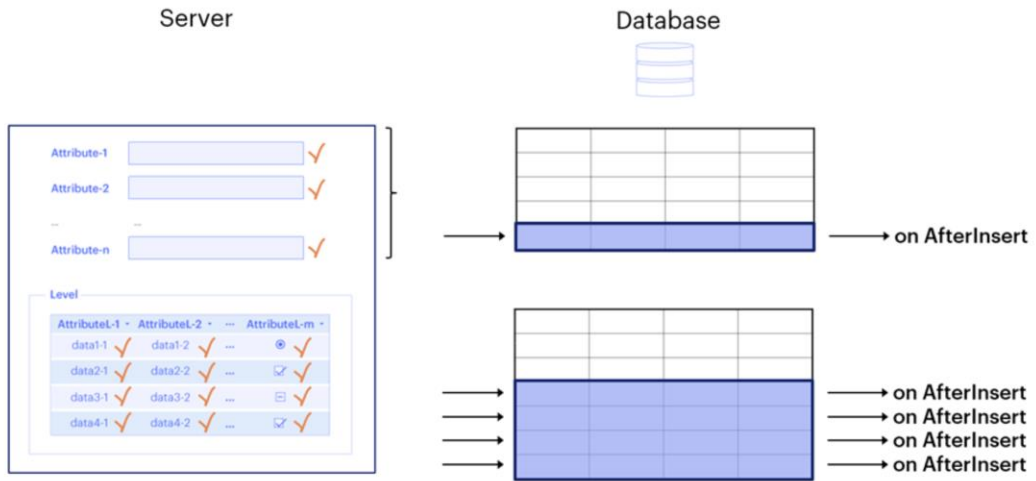
we had to delay the moment it had initially chosen to trigger it, using the *on AfterLevel* triggering moment with a line attribute (*FlightSeatChar*). In this way, we managed to have the rule triggered after running through the level associated with the seats.

BeforeInsert event



We also mentioned that there were other triggering moments, such as “on BeforeInsert,” if we wanted to do or evaluate something immediately before the data from the header or each line is inserted in the database,

AfterInsert event



the “on AfterInsert” moment to indicate that the rule be triggered immediately after the insertion of each header or line,

AfterComplete event




and the "on AfterComplete" moment, which corresponds to the moment immediately following the Commit, a command whose purpose is to validate the data inserted, modified or deleted.


Execution of rules



In this video, we'll take a closer look at these triggering moments. But first, let's remember that some rules are validated both on the client (browser) and on the server, and others are validated only on the server, because they have to do with the database.

```
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"/> It is not possible to leave a flight without a price.
Discount Percentage	<input type="text" value="0"/>
Airline Id	<input type="text" value="0"/> 
Airline Name	
Airline Discount Percentage	0
Final Price	0.00



For example: if we had an Error rule that prevented us from entering or modifying a flight without a price, as soon as we left that field the error message would appear.

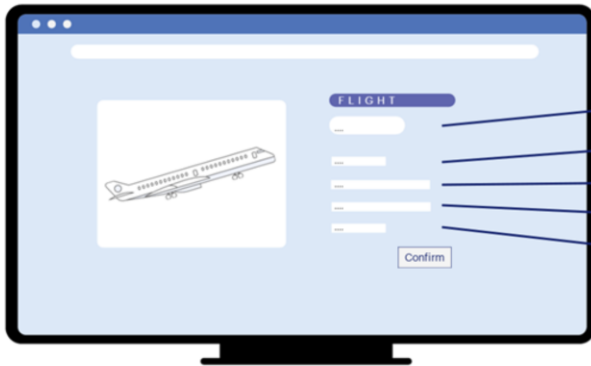
CLIENT

```
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

This happens because the client performs the validation to provide a good user experience, responding to them in an agile way so that they feel their interaction with the system is seamless. This validation performed on the client side is called Client-Side Validation.

But this rule will be executed again later on the server, since it is the server that ensures that there are no security violations, and it is the only one allowed to operate on the database, so it must ensure that all the logic is consistent. It will execute all the rules again, when it has all the information. This happens after the user presses the Confirm button.

There the data travels from the web client (browser) to the web server, and the server runs through the form again from top to bottom, as if it were a user, so all the rules and formulas will be triggered again in the order corresponding to the attributes of the form with which they are associated.

In addition, all the rules that have an associated triggering event (i.e. that have the *on* prefix at the end of their statement), will be executed at the moment corresponding to that event. These events only take place on the server and capture the moment before or after a milestone, which is usually related to the database.

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:

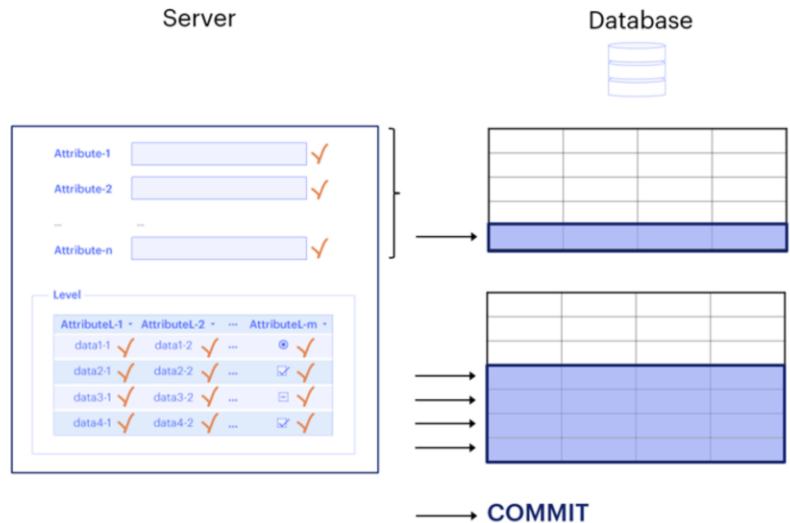
- Validate the line record to be inserted
- Insert the line record in the table

The next milestone is when:

- You finish working with the level

And the next one is:

- The Commit action



Which are those milestones?

Let's think, for the sake of simplicity, that we're inserting a flight. In this order:

- Validate the header record to be inserted (to make sure that the referential integrity and duplicate controls did not fail; it happens after having gone through each field of the header and after having triggered each associated rule, at the end of it all).
- Insert the header record in the corresponding table

Then we do the same for each line:

- Validate the record of the line to be inserted (it makes sure that the referential integrity and duplicate controls did not fail).
- Insert the line record in the corresponding table

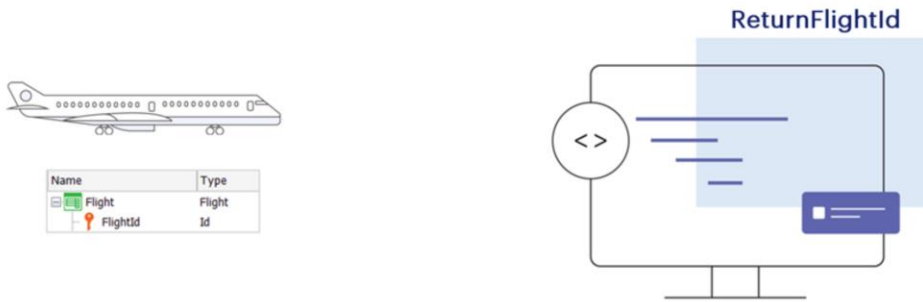
At the end of this process, where both the record corresponding to the header and those corresponding to all the lines have been inserted in the database, the next milestone is when:

- You finish working with the level

And the next one is:

- The Commit action, which validates all these operations on the database, making them permanent.

So, we have events that capture the moment before or after these milestones, so that we can execute rules at those precise moments.



```

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

```

At present, in the Flight transaction, we have defined the flight identifier (FlightId) as autonumbered.

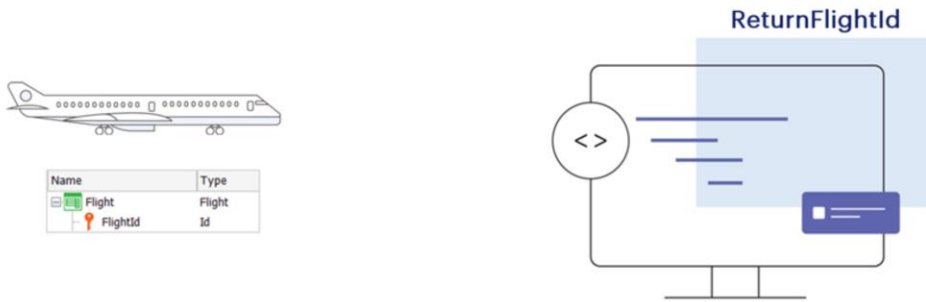
If in real life, as is usually the case, the flight identifier is made up of letters and numbers depending on the airline, departure and destination, the numeric and autonumbered FlightId attribute is of no use to us.

So let's suppose we have a procedure called ReturnFlightId, which we can invoke to be run and return the identifier to be assigned to a new flight.

(We will study the procedure object later, so you cannot, at the moment, reproduce what we will show here).

If we write the rule shown above, where we are calling and assigning its result to the FlightId attribute, when will it be triggered?

As soon as you open the transaction or want to edit a pre-existing flight. We do not condition the triggering of the rule to the mode, so it will be executed regardless if the user is accessing the Flight transaction to insert, modify or delete a flight, when in fact we want the procedure to be invoked only if we are inserting a new flight.



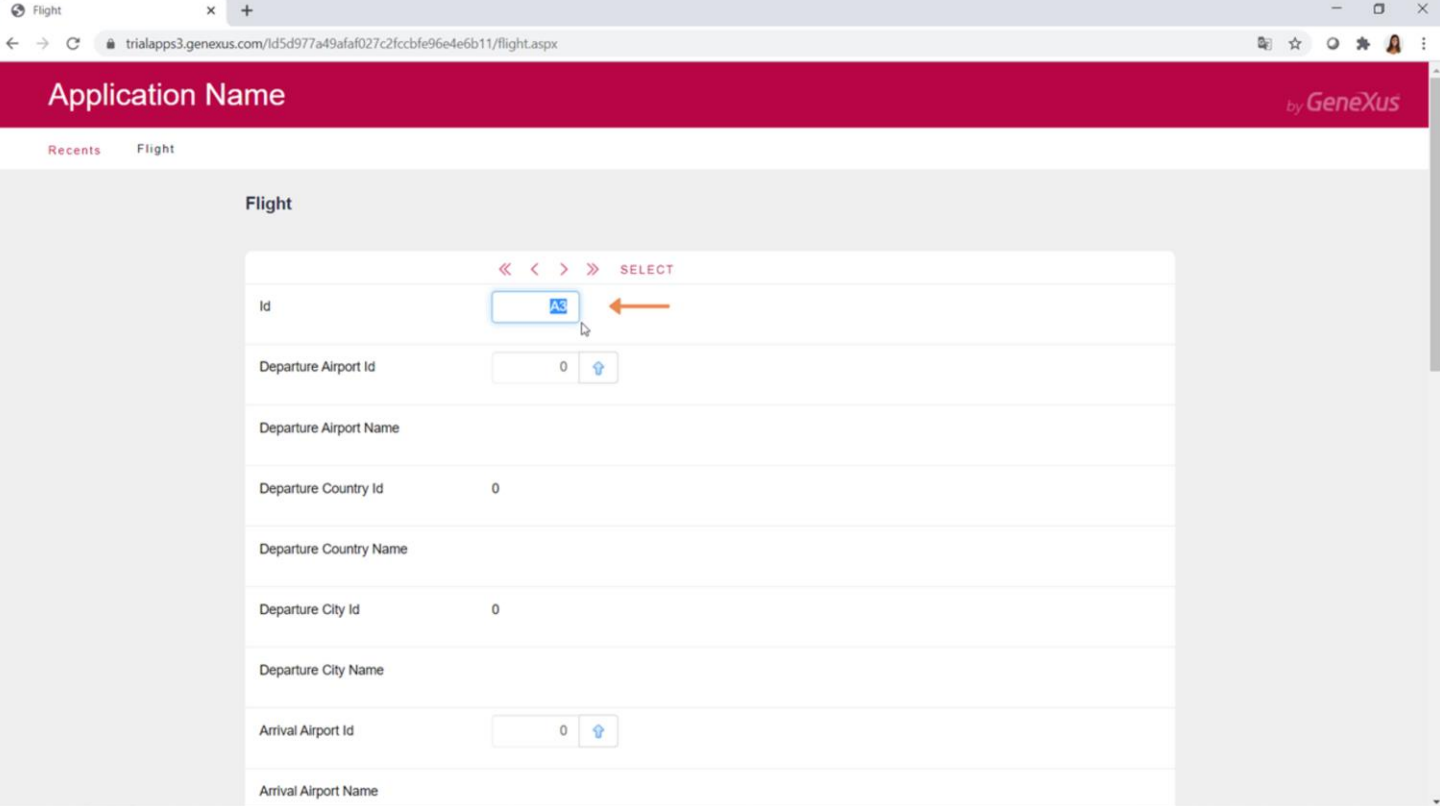
```

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

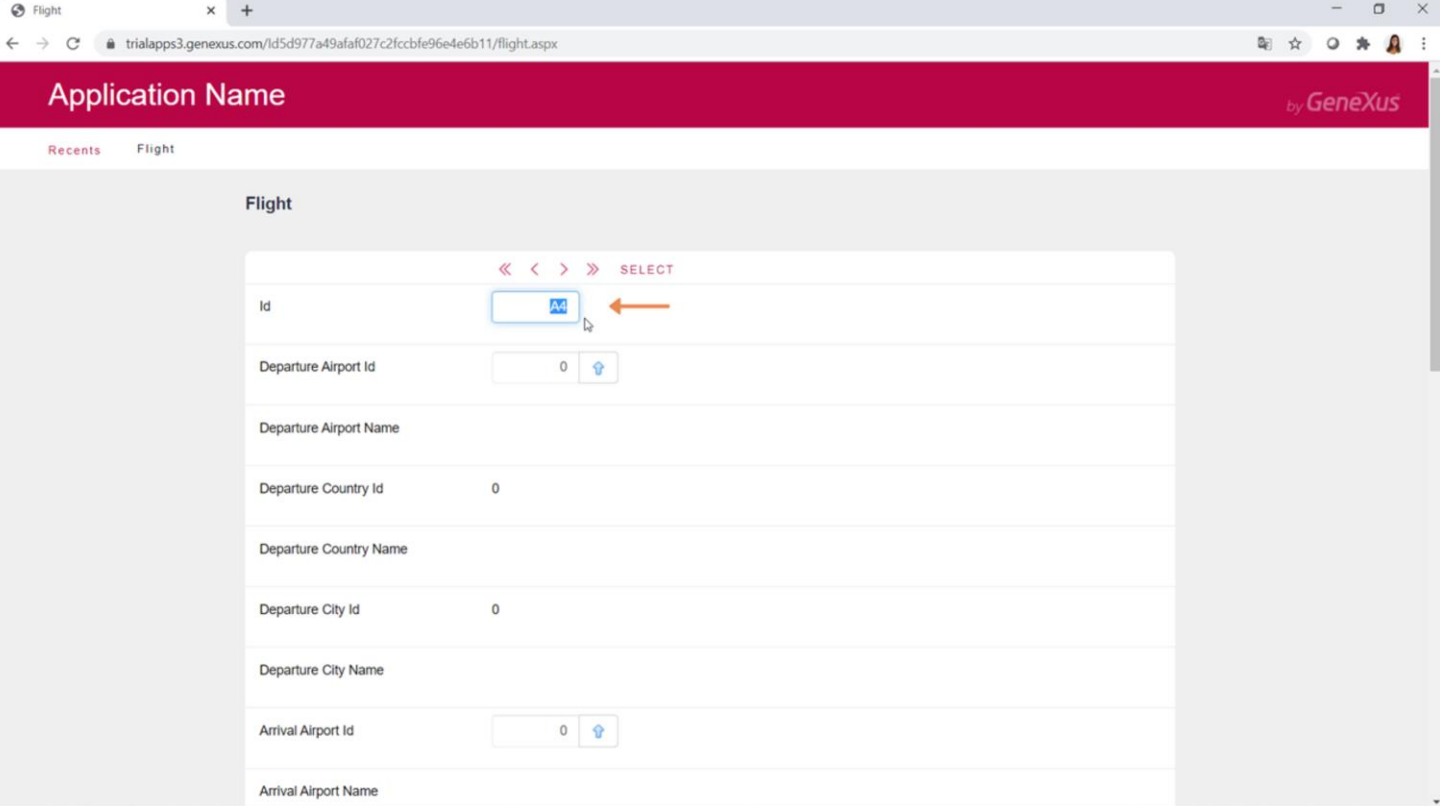
```

It is also worth pointing out that if we were in the Flight transaction in Update mode (i.e. if the flight already existed and we were only modifying it), we would not be able to change the value of the flight identifier, as the primary keys cannot be modified. If we need to change a primary key we have no choice but to create a new record with the new key value, and delete the old one.

So we condition the rule to be executed only when you want to insert data.



Since the transaction opens in that mode, the rule that is assigning a value to the attribute is already being triggered, before anything has been done. But it could happen that after completing the header data and some lines, we have to cancel the entry for some reason, to do it later.

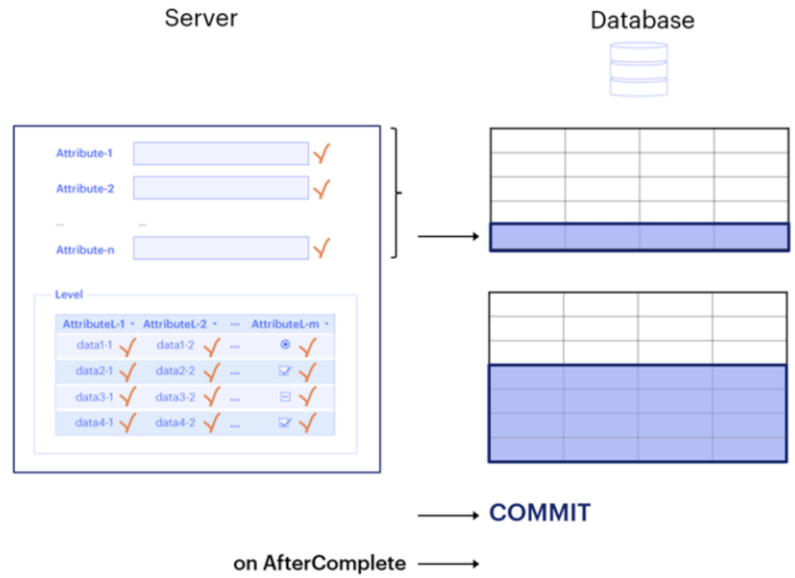


When we try it again, we see that it will give us another flight number. The reason is that we had asked for a number from the procedure that we didn't use in the end, and that number was lost.

```

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;

```

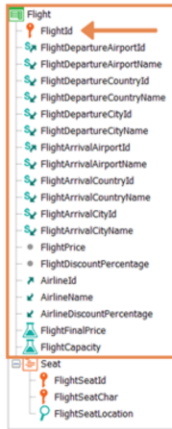


To make sure we don't waste a number that we won't use, how about calling the procedure that gives the number after the Commit, because at that moment we are completely sure that everything will remain in the database?

```

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;

```



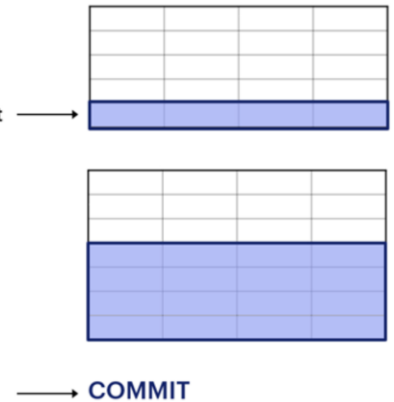
Server



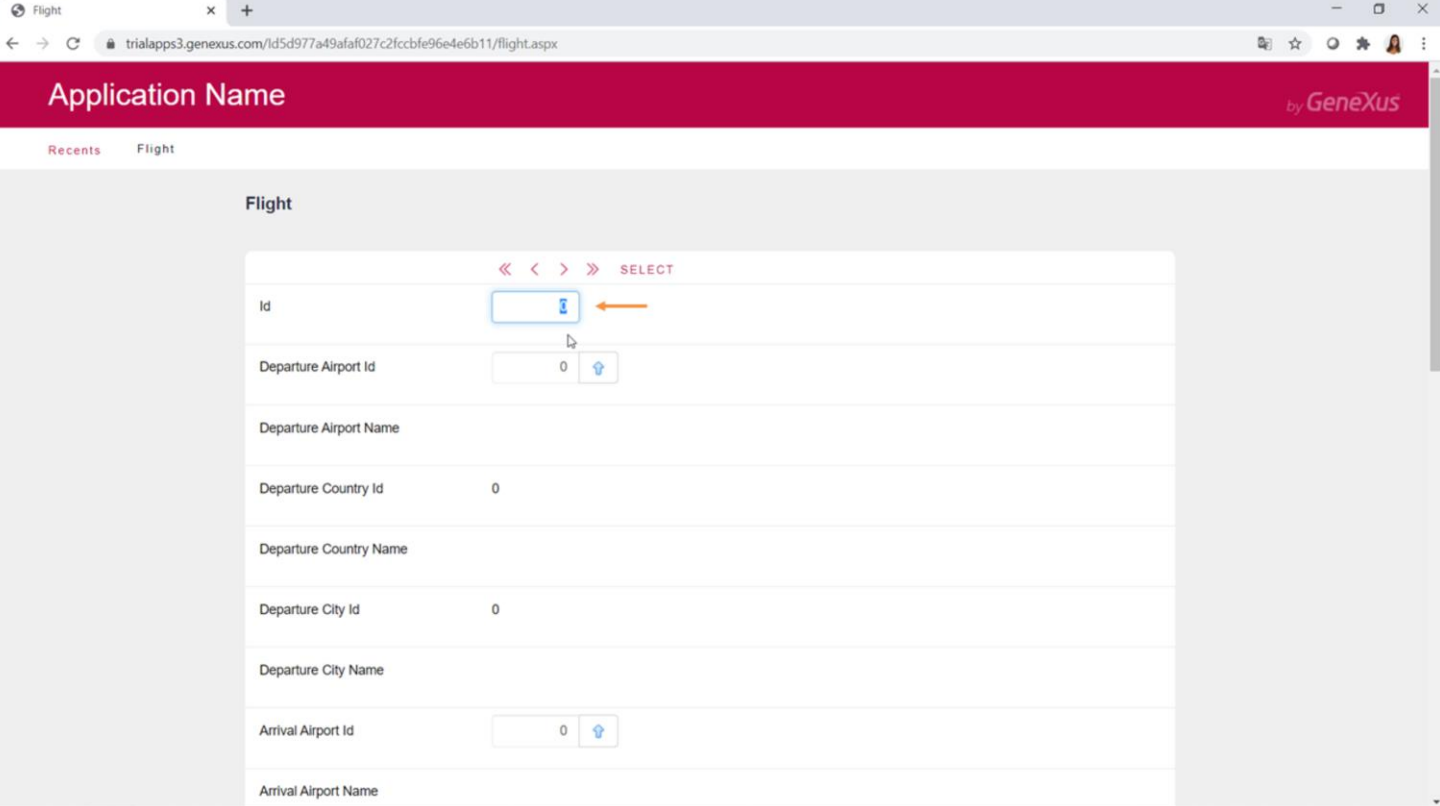
Database



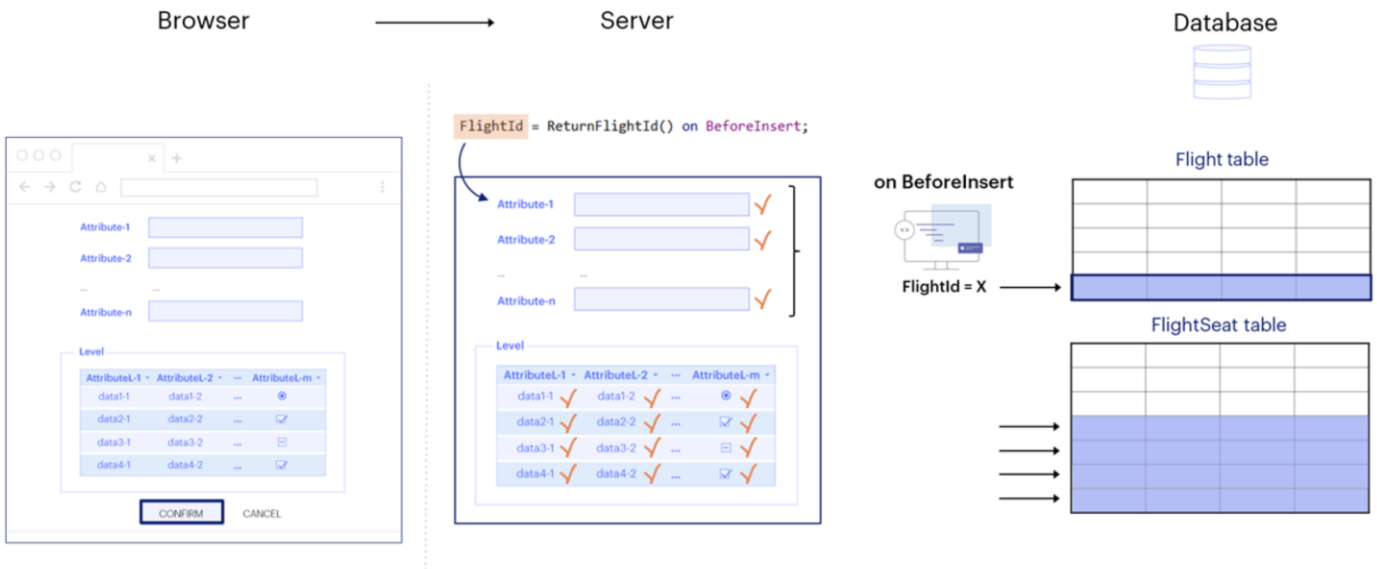
on BeforeInsert



Hmmm, no, that's too late! Why? Because FlightId is an attribute of the header. The last moment to assign it a value is the one immediately before the record is inserted in the database, and that moment is BeforeInsert. We no longer need to condition to If Insert, because the triggering event already takes into account that it will only happen if we want to insert data:



If we run it, we see that it no longer assigns a number to the ID right away. It doesn't even do that when we enter lines.

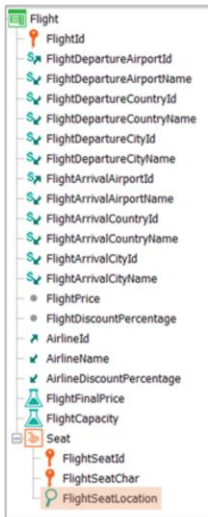


Only after we confirm, all this information goes to the server and it starts executing all the referential integrity controls and rules as it validates each field. When it finishes with those of the header, after having validated everything, the BeforeInsert event occurs. This is where it assigns the number, and immediately afterwards the record is inserted in the Flight table. Then it goes on to validate each field in line 1, and inserts it; next, those in line 2 and inserts it and so on.

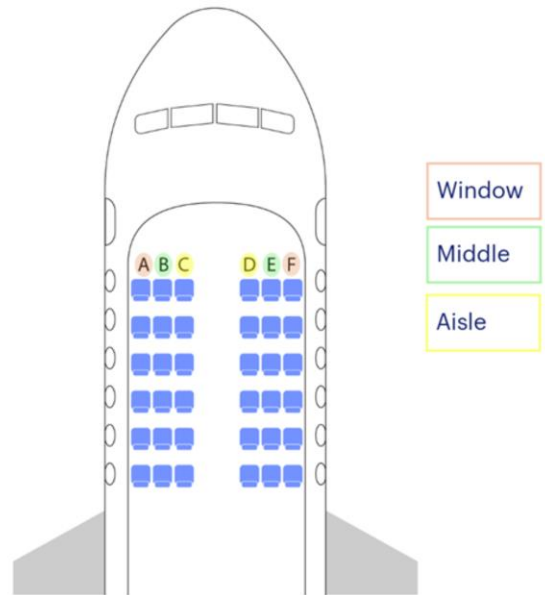
Note that the rule will not be triggered before every line is recorded. This rule will only be triggered in the BeforeInsert of the header! Why? Because in the rule we defined there is an attribute that belongs to the header.

If we define a rule to which we also add the **on BeforeInsert** triggering event, but unlike the previous example, the rule contains a reference to at least one attribute of the second level of the transaction, it will be associated with the second level. Therefore, it will be executed immediately before each instance corresponding to the second level of the transaction is physically saved.

So, we could say that even though the name of the event, BeforeInsert, is the same, they are actually two different ones: either it is the BeforeInsert of the header or the one that applies to the lines.



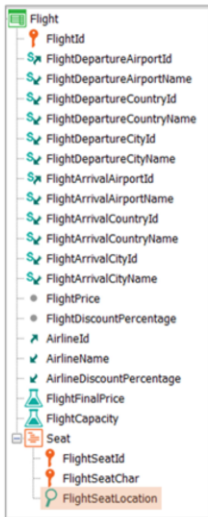
Seat			
	Seat Id	Seat Char	Seat Location
x	1	A	Window
	0	A	Window
	0	A	Window
	0	A	Window
	0	A	Window
	[New row]		



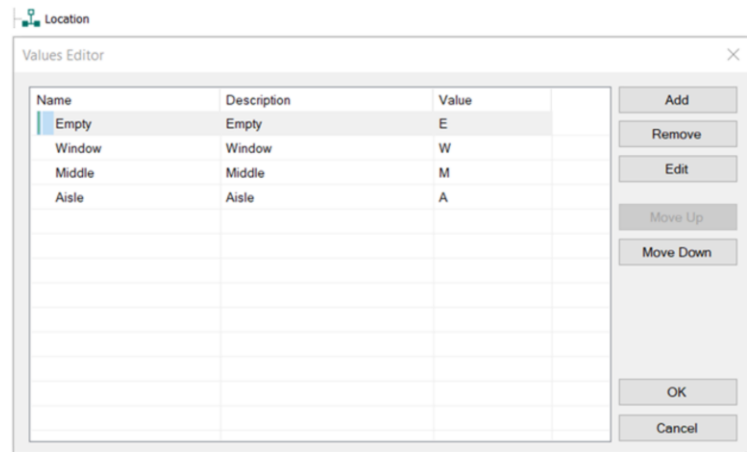
Let's see an example of BeforeInsert for the lines:

Suppose we want the value for the FlightSeatLocation attribute, rather than being chosen by the user working with the transaction, to be assigned by a rule, which sets it to "Window" when the value of the FlightSeatChar attribute is A or F, "Medium" when the value of the FlightSeatChar attribute is B or E, and "Aisle" when its value is C or D.

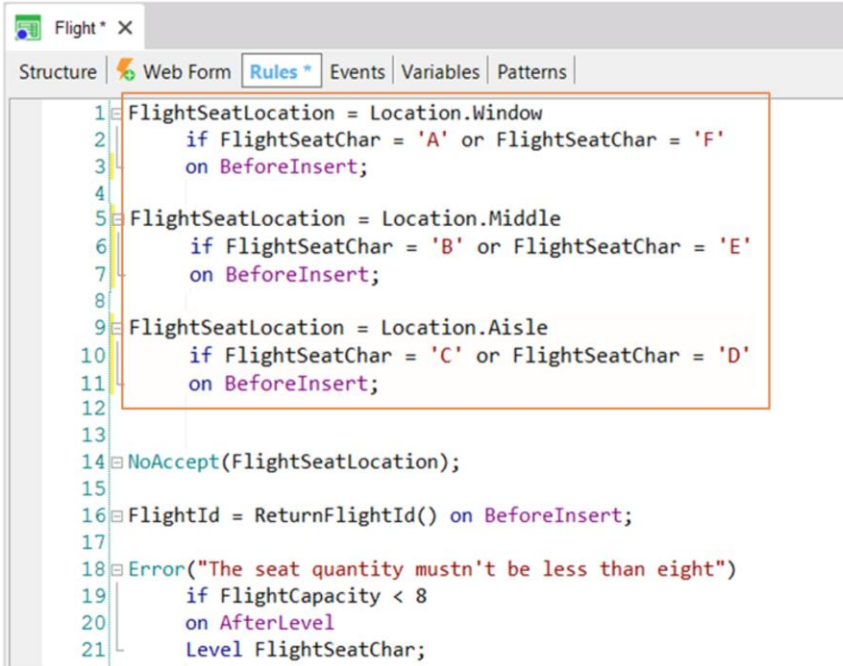
Let's make it clear that we could solve this without using triggering events, and it would even be better, because the user would immediately see on the screen the value of the SeatLocation, as the rule would be executed immediately on the client. However, just to understand the triggering moment, imagine that we would be interested in making this assignment only if we were sure that the seat was going to be entered on the flight; that is, immediately before inserting the line.



```
NoAccept(FlightSeatLocation);
```



We'll use the NoAccept rule to prevent the user from choosing the location, add the value "Empty" to the Location domain,



```
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;
```

and create the following rules.

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

Attribute 1
Attribute 2
Attribute n

Level

Attribute 1	Attribute 2	Attribute n
data1	data2	...
data1	data2	...
data1	data2	...

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)

Let's insert a new flight from Guarulhos airport to Charles de Gaulle airport, with a price of 4000, no discount and TAM airline. Let's go register the seats now:

We'll complete the first row, and two more seats in the second row.

Before the physical insertion of each line, the corresponding rule will be executed, according to the FlightSeatChar we have chosen.

So, the transaction on the server will validate the data of the first line, where SeatLocation will be empty and the next step will execute the first rule, because SeatChar is A, and then it will change the SeatLocation to Window, and immediately save the line in the table.

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?

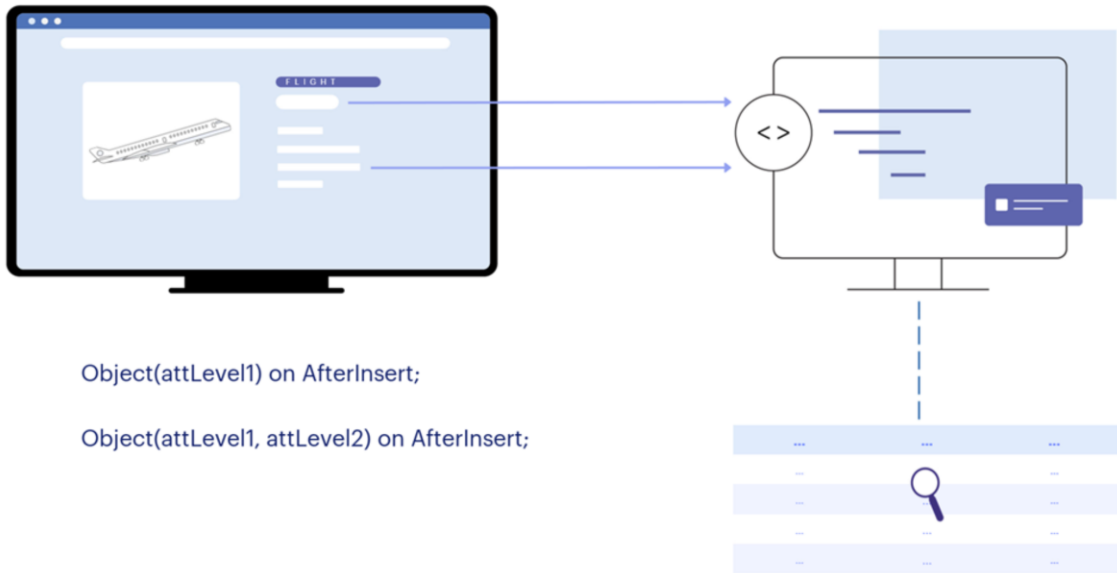
0	A	▼ Empty
0	A	▼ Empty
0	A	▼ Empty

Then it will do the same with the second line: it will validate all the fields – SeatChar will have an Empty value–, and then it will execute the corresponding BeforeInsert rules, which in this case will be the second one, changing the value of FlightSeatLocation to Middle, and will immediately save the second line in the FlightSeat table. And so it will continue with the other lines.

We can see that the location was correctly assigned.

But... could we have conditioned these rules to on AfterInsert?

Let's make this change in GeneXus, and see what happens in this case:



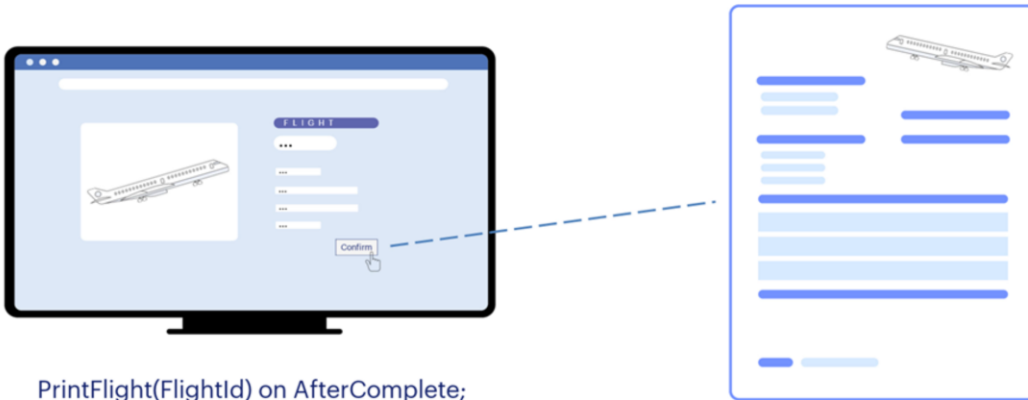
So, we could use on AfterInsert if we needed, for example, to call another object from the KB only by sending it the header or line identifier for that object to go to the corresponding table (that of the header or the line) and there find it (the header or the line) and extract from there all the other information it needs from the record to do whatever it has to do with it. To do so, we must be sure to call that object after inserting the header or line, as appropriate, on AfterInsert.



So, if we wanted to print a list with the details of a flight every time a new one is inserted, would we use on AfterInsert?

We could, but here the procedure will only be able to print the header data, because the lines have not been manipulated at all, let alone inserted. They don't exist yet in the seating table. If that didn't matter to us, because the PrintFlight procedure would only print data from the flight header, it would still be a bad idea to invoke it on AfterInsert, because we must take into account that at that moment the data is still not secure in the table, because the Commit has not been executed.

This means that if there was a power failure, or an error in the validation of any of the following fields, i.e. those of the lines, the recording would be undone, so the list would have been generated with information that actually no longer exists.



PrintFlight(FlightId) on AfterComplete;



→ **COMMIT**
on AfterComplete

To make sure we are working with information that is already secure in the database, we have the on AfterComplete event, which takes place after the Commit.

In this case, it will go to the Flight table to find the record corresponding to the sent FlightId, and to the FlightSeat table to find all the records of that FlightId, which are all its entries.



PrintFlight(FlightId) on AfterLevel Level FlightSeatChar;

***	***	***
---	---	---
---	---	---
---	---	---

on AfterLevel

→ **COMMIT**

on AfterComplete

What would happen if we invoked on AfterLevel of an attribute of the lines?

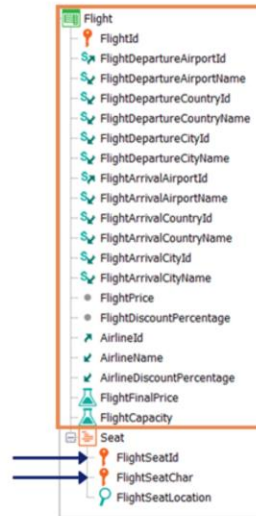
This will happen immediately before the Commit, so the records will be saved in the tables but not yet validated. So, if when you finish printing the flight information there is a power outage and the Commit has not been performed, the header and lines will be removed from the database.



PrintFlight(FlightId, FlightSeatId, FlightSeatChar) on AfterComplete;

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

→ **COMMIT**
on AfterComplete



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

Now suppose the following rule was declared:

What FlightSeatId and FlightSeatChar values would GeneXus send, if there were N seats on the second level? This rule doesn't make sense, it is functionally incorrect.

At the AfterComplete moment, the values of the header attributes are still in memory, unlike the attributes of the second level, whose values were lost because we already left it.

Rule Triggering Events



This video shows examples of BeforeInsert and AfterInsert moments, which will be triggered only if we are inserting records, but we also have BeforeUpdate and AfterUpdate in case we are modifying them, and BeforeDelete and AfterDelete, in case we are deleting them.

There are other triggering moments that will not be studied in this course. If you are interested, you can learn more about this topic in the next level course.

GeneXus™

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