More about rule execution order in transactions

GeneXus™ 16

More about order of execution in transaction rules — GeneXus

Sample transactions

Name	Type
Flight	Flight
FlightId	Id
FlightDepartureAirportId	Id
FlightDepartureAirportName	Name
FlightDepartureCountryId	Id
FlightDepartureCountryName	Name
FlightDepartureCityId	Id
FlightDepartureCityName	Name
FlightArrivalAirportId	Id
FlightArrivalAirportName	
FlightArrivalCountryId	
FlightArrivalCountryName	
FlightArrivalCityId	
FlightArrivalCityName	
FlightPrice	
FlightDiscountPercentage	
AirlineId	
AirlineName	
AirlineDiscountPercentage	
FlightFinalPrice	
FlightCapacity	
FlightAvailableSeats	
Seat	
FlightSeatId	
FlightSeatChar	
FlightSeatLocation	

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

```
Error( "The seat quantity mustn't be less than eight")
    if FlightCapacity < 8
    on AfterLevel
    Level FlightSeatChar;

Default( FlightAvailableSeats, FlightCapacity );
```

Invoice — Structure | Web Form | Rules | Events | Variables | Patterns

Name	Type	Formula	Nullable
Invoice	Invoice		
InvoiceId	Numeric(4.0)		No
InvoiceDate	Date		No
CustomerId	Numeric(4.0)		No
CustomerName	Character(20)		
CustomerTotalPurchases	Price		
InvoiceTotalAmount	Price	sum(InvoiceFlightAmount);	
Flight	Flight		
FlightId	Id		No
FlightAvailableSeats	Numeric(4.0)		
InvoiceFlightSeatQty	Numeric(4.0)		No
FlightFinalPrice	Price	FlightPrice*(1-AirlineDiscountPercentage...	
InvoiceFlightAmount	Price	InvoiceFlightSeatQty*FlightFinalPrice	

Next, we will talk in more detail about the times available to condition the triggering of rules, especially in transactions that have more than one level.

To explain this topic, we will use an invoice transaction (Invoice) that has a second level (Flight), to represent the flights included in the invoice.

Note that in the Flight transaction we have added the **FlightAvailableSeats** attribute, which will be used to record the seats available in each flight. The number of seats available will diminish every time that an invoice is issued to a customer who has purchased a number of places (seats) in the flight.

Note: In the Customer transaction we've also added the CustomerTotalPurchases attribute to record the total amount spent by the customer in flight ticket purchases.

Note that the InvoiceTotalAmount, InvoiceFlightPrice and InvoiceFlightAmount attributes of the Invoice structure are formulas. Next, we will talk about the rules stated for Invoice to set its behavior.

Rules of the transaction

Name	Type	Formula
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

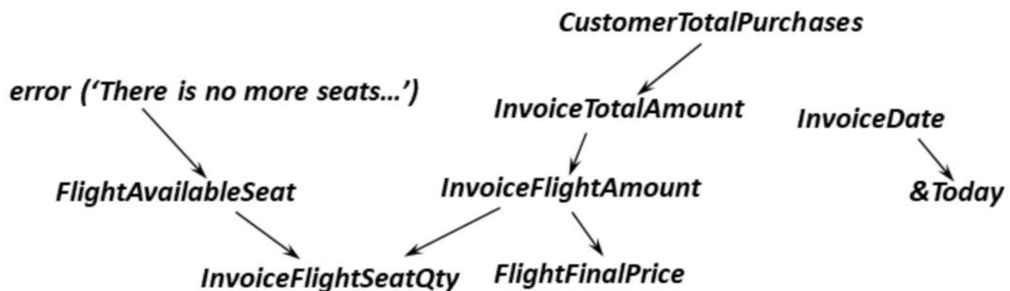
```
1 Default( InvoiceDate, &Today );
2
3 Subtract( InvoiceFlightSeatQty, FlightAvailableSeats );
4
5 Error( 'There is no more seats for sale' )
6     if FlightAvailableSeats < 0;
7
8 Add( InvoiceTotalAmount, CustomerTotalPurchases );
```

More about order of execution in transaction rules

GeneXus

Evaluation tree of rules and formulas

(R) Default(InvoiceDate, &Today);
(R) Add(InvoiceTotalAmount, CustomerTotalPurchases);
(F) InvoiceTotalAmount = **Sum**(InvoiceFlightAmount)
(F) InvoiceFlightAmount = FlightFinalPrice*InvoiceFlightSeatQty
(F) FlightFinalPrice = FlightPrice*(1-AirlineDiscountPercentage…)
(R) Subtract(InvoiceSeatQty, FlightAvailableSeats);
(R) Error("There is no more seats for sale") if FlightAvailableSeats < 0



The order in which rules and formulas are triggered by GeneXus is based on the definition of rules and formulas shown on screen.

When generating the program associated with the Invoice transaction, GeneXus will extract the dependencies existing between the rules and formulas created. Also, it will create a dependency tree (or evaluation tree) in a logical manner that will determine the evaluation sequence.

We can imagine that the tree is executed in a bottom-up manner. That is to say, every time an attribute value is updated, all the rules and formulas that depend on this attribute (and that are located upwards in the tree) are executed.

For example, if the number of seats in an invoice line (InvoiceFlightSeatQty) is updated, since this attribute is part of the formula that calculates the cost of the flight (InvoiceFlightAmount), this formula will be triggered again. The same would happen after changing the final price of the flight, FlightFinalPrice, which is also included in the formula.

If the price for a flight is changed, the formula corresponding to the invoice total (InvoiceTotal) also has to be triggered again because it depends on the value of each flight included in the invoice. Lastly, changing the total also implies having to trigger the rule Add(InvoiceTotal, CustomerTotalPurchases) because the customer's total purchases have to be updated.

In addition to triggering all the formulas and rules included in the right branch of the tree from the attribute InvoiceFlightSeatQty, the formulas and rules included in the left branch will also be triggered. That is to say, when the value of the InvoiceFlightSeatQty attribute is changed, the rule Subtract(InvoiceFlightSeatQty, FlightAvailableSeats) that updates the number of seats available on the flight (FlightAvailableSeats) is also triggered again.

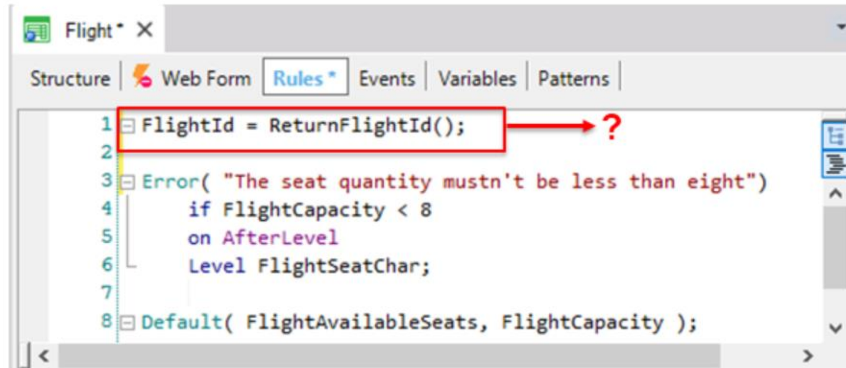
Therefore, since this rule changes the value of the FlightAvailableSeats attribute, it will be evaluated whether to trigger the rule Error('There are no more seats...') if FlightAvailableSeats< 0;

In sum, the rules and formulas stated in a transaction are usually interrelated and GeneXus determines the dependencies existing between them, as well as the order in which they are evaluated.

Review triggering events

- The rules we define are usually executed at the expected time.
- But in some cases we need to modify the moment when a rule is triggered.

Example:



In the Flight transaction, the FlightId flight identifier is set as autonumbered.

If in reality the flight identifier is made up by letters that identify the airline and by numbers, a numeric, autonumbered FlightId attribute cannot be used in this case.

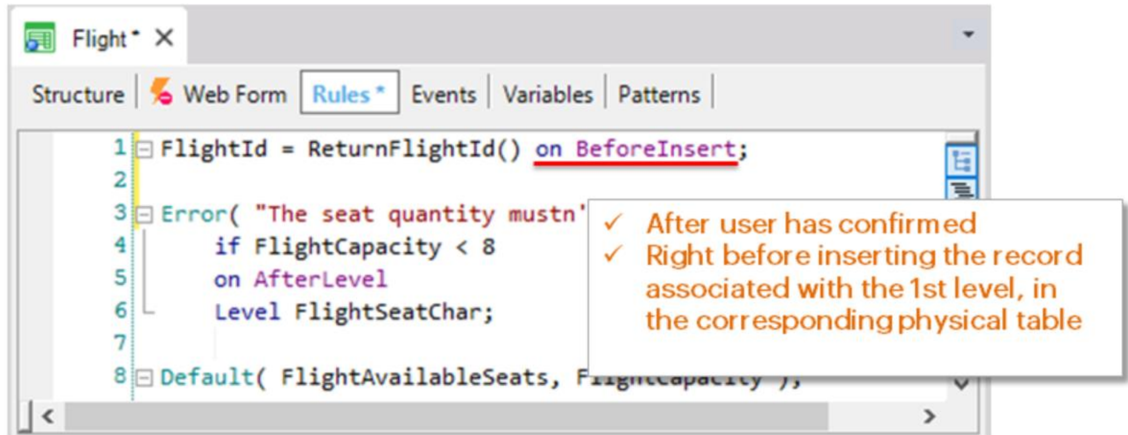
Suppose that we have a procedure, ReturnFlightId, which returns the identifier to be assigned to a new flight. (In a real-life situation, this procedure should choose the number of flight depending on the airline, departure and destination, but we will simplify it).

If we type the rule shown above, when will it be triggered? In the Flight header, regardless of the operation performed. That is to say, if any detail related to the flight is changed, the ReturnFlightId procedure is invoked again to assign a new flight identifier, but in fact we want this to happen only if we're inserting a new flight.

In addition, even when we're inserting a flight, we should obtain a new FlightId only if the transaction is confirmed, because we may cancel the addition or it may be automatically canceled by an error. In these cases, a new flight identifying number would be "taken" but it would never be actually used.

Review triggering events

- By adding a triggering event we change the moment when the rule is executed by default



To make sure that only one flight ID number is used if the transaction is confirmed, and that a number is obtained only when a flight is inserted, we add the on BeforeInsert triggering moment to the rule that assigns the FlightId.

This triggering moment will occur after the transaction has been confirmed; also, it will be executed at the server level.

In addition, "before insert" means that it will be triggered after the flight header data has been validated and right before saving the header data in the database.

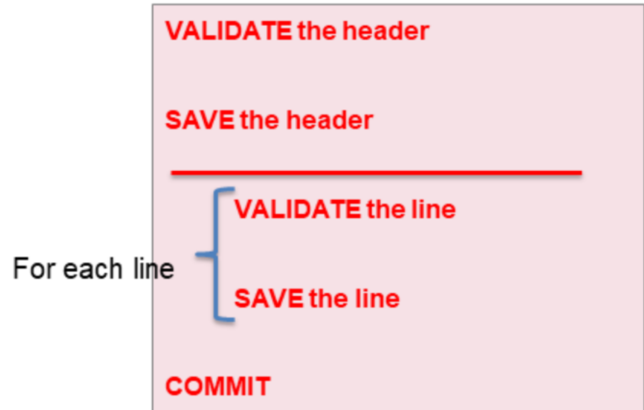
Next, we will see that there are some key moments during the server's operation in relation to data processing and the database.

Operations on data on the web server

(After pressing Confirm)

In single-level transactions:

In two-level transactions:



After pressing Confirm, data travels from the web client (browser) to the web server.

On the server, operations are performed on data, such as:
- Validate data,
- Save data in the database, and
- Commit to the database.

If the transaction has two levels, after saving the header, for each line:
- data will be Validated and then
- the line will be Saved.

Lastly, the Commit operation will be performed to have the header details and all the transaction lines consolidated in the database.

More about order of execution in transaction rules

GeneXus

Moments for triggering rules

In the server, after pressing Confirm

In single-level transactions:

On BeforeValidate
VALIDATE
On AfterValidate
On BeforeInsert/BeforeUpdate/ BeforeDelete
SAVE
On AfterInsert/AfterUpdate/ AfterDelete
On BeforeComplete
COMMIT
On AfterComplete

In 2-level transactions:

On BeforeValidate
VALIDATE the header
On AfterValidate
On BeforeInsert/BeforeUpdate/BeforeDelete
SAVE the header
On AfterInsert/AfterUpdate/AfterDelete
For each line:
On BeforeValidate
VALIDATE the line
On AfterValidate
OnBeforeInsert/BeforeUpdate/BeforeDelete
SAVE the line
On AfterInsert/AfterUpdate/AfterDelete
On AferLevel Level Line attribute
On BeforeComplete
COMMIT
On AfterComplete

Once the transaction data reaches the web server, the rules and formulas will be triggered again at the server level.

During this execution, several moments are available relatet to the Validation, Save and Commit of data on the server. These triggering moments can be assigned to rules, so as to control when they should be triggered.

This allows us to customize the triggering moment of a rule, so that it isn't triggered at the time chosen by GeneXus according to its evaluation tree, but when we want.

Let's start by looking at the moments related to Validation: BeforeValidate and AfterValidate.

Triggering event: **BeforeValidate**
This triggering event occurs right before the information of the instance you are working with (header or line x) is validated (or confirmed). That is, it will occur right before the “header validation” action or “line validation” action, as applicable. Note that here too all the rules that are not conditioned to any triggering event will have been triggered according to the evaluation tree.

Triggering event: **AfterValidate**

The triggering event AfterValidate allows you to specify that a rule be executed immediately before physically saving each instance of the level which the rule is associated to, in the corresponding physical table, and after the data of that instance have been validated.

In other words, if an **AfterValidate** triggering event is added to a rule, the rule will be executed for each instance of the level it is associated to, **immediately before the instance is physically saved (whether it is inserted, modified or deleted) as a record in the physical table associated to the level.**

Rules with triggering moments

```
Flight*
1 FlightId = ReturnFlightId() on BeforeInsert;
```

✓After the user has confirmed

✓on BeforeInsert: Right before inserting the record associated with the 1st level, in the corresponding physical table.

On BeforeValidate
VALIDATE
On AfterValidate
On BeforeInsert/BeforeUpdate/ BeforeDelete
SAVE
On AfterInsert/AfterUpdate/ AfterDelete

On BeforeComplete
COMMIT
On AfterComplete

Sometimes the Autonumber property is not available to automatically and correlatively number the attributes that are simple primary key. That function is provided by the database managers (DBMSs) and GeneXus takes advantage of it and enables its use; however, when we're not working with a database manager, such feature is not available.

In the example that we've used, each flight identifier must be generated in a specific way. Also, the autonumbered feature isn't useful, so a procedure has been coded to implement this numbering.

As we've explained, we need to have the rule executed immediately after the user has confirmed it, and only if a new flight is being created.

These two possible definitions are correct to implement what we need:

FlightId = ReturnFlightId() if Insert on AfterValidate;

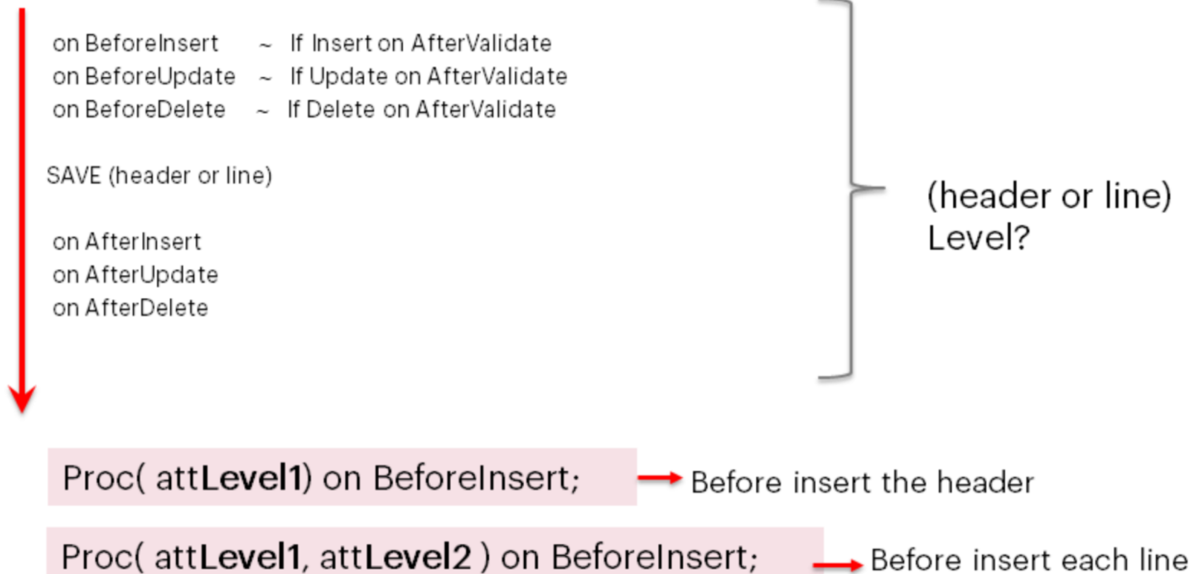
or

FlightId = ReturnFlightId() on BeforeInsert;

In the first definition we're indicating that the procedure must be executed only if an addition is being made (in the triggering condition: if Insert), right after validating the first level data (because there's only one attribute involved in the rule, and it belongs to the transaction's first level) and right before physically saving the record (in the triggering event: on AfterValidate).

There are three triggering events that occur at the same time as AfterValidate, but which already intrinsically contain the mode. These are: BeforeInsert, BeforeUpdate, and BeforeDelete.

In the second proposition, it would be redundant to condition the rule to "If Insert", because BeforeInsert intrinsically indicates that it is an insertion.



Therefore, the following equivalences are valid:

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

If we define a rule in which we also include the triggering event **on AfterValidate**, or **on BeforeInsert, BeforeDelete, BeforeUpdate**, but where, unlike in the examples we just saw, we reference at least one attribute of the second level of the transaction in which we are defining the rule, the rule will be associated to the second level.[1] Therefore, the rule will be executed **immediately before physically saving** each instance corresponding to the second level of the transaction.

Triggering events: **AfterInsert, AfterUpdate, AfterDelete**
Just as there is a triggering event that allows you to define that certain rules be executed immediately before the physical saving of each instance of a level (AfterValidate, BeforeInsert, BeforeUpdate, BeforeDelete), there are also triggering events to define that certain rules be executed **immediately after** a level's **instances are physically inserted, updated or deleted**. These events are AfterInsert, AfterUpdate and AfterDelete.

The triggering event AfterInsert allows you to define that a rule be executed immediately after physically inserting each instance of the level which the rule is associated to; AfterUpdate, after the instance is physically updated, and AfterDelete, after it is deleted.

Triggering events: examples of good and bad programming

Case: Print customer's details

```
PrintCustomer(CustomerId) on AfterValidate;
```

It's incorrect because it is invoked BEFORE saving, so the table will not show the changes made to the customer 's information

```
PrintCustomer(CustomerId) on AfterInsert, AfterUpdate;
```

It's correct!

```
PrintCustomer(CustomerId) on AfterDelete;
```

It's incorrect because it is invoked AFTER deleting, so the customer will not be found on the table.

Let's suppose that we want to call a report in the "Customer" transaction that gives a printout of the data of each customer worked with through the transaction. At what point should we call the report from the transaction?

Proposal 1: PrintCustomer.call(*CustomerId*) **on AfterValidate**;

This triggering event should not be added to the report calling rule, because if it were, the report would be called **immediately before the physical saving** of each customer. Consequently, the report would not find the customer with the customer's data on the CUSTOMER table (if a customer was being inserted through the transaction), or it would find the customer with data that was not updated (if a customer was being updated through the transaction). If instead, a customer was being deleted through the transaction, the report would find the data of the customer in the CUSTOMER table and would list them **precisely before the physical update (deletion).**

If this is what we want, that is, if we want to issue a list with the data of each customer that is deleted, we would have to define the following rule:

PrintCustomer.call(*CustomerId*) **on BeforeDelete**;
or its equivalent:
PrintCustomer.call(*CustomerId*) if delete **on AfterValidate**;

to limit the triggering of the rule to customer deletions only, because only then would it be correct to use the AfterValidate triggering event (as we need to issue the report precisely before deletion).

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

The triggering event **AfterInsert** occurs immediately after each instance associated to a certain level of the transaction is physically inserted (in this case, since the only attribute involved in the rule is *CustomerId,* the rule is associated to the first and only level of the "Customer" transaction).

The triggering event **AfterInsert** occurs immediately after each instance associated to a certain level of

the transaction is physically inserted (in this case, since the only attribute involved in the rule is *CustomerId,* the rule is associated to the first and only level of the "Customer" transaction).

As is evident from its name, the **AfterInsert** triggering event only occurs when a new instance is **inserted** (precisely after being inserted as a physical record). This is why when the **on AfterInsert** triggering event is added to a rule, there is no need to add the **if insert** triggering condition.

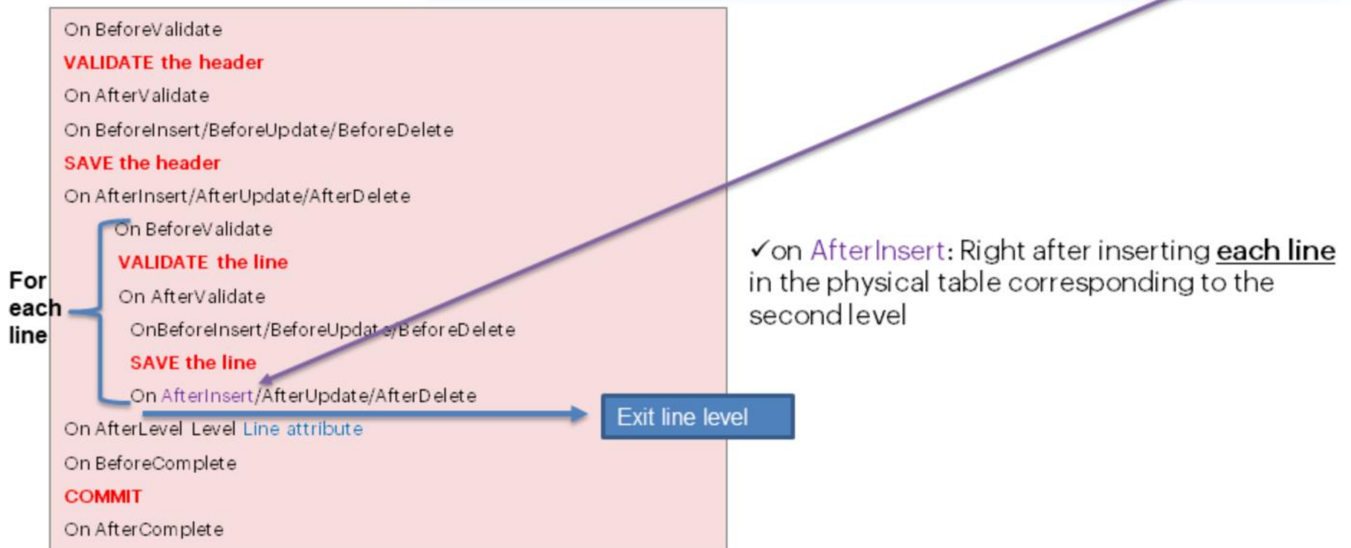
The AfterUpdate triggering event only takes place when a record is changed. That's why the triggering condition "If Update" doesn't need to be added to it.

Adding this triggering event to the report calling rule is correct, because the procedure would be called **immediately after physically inserting or updating** each customer. So the report would find the customer with the customer's data updated in the CUSTOMER table, and would print the data. Now, take into account that the customers is not commited!

Proposal 3: PrintCustomer(CustomerId) on AfterDelete;

The triggering event **AfterDelete** occurs immediately after each instance associated to a certain level of the transaction is physically deleted (in this case, since the only attribute involved in the rule is *CustomerId,* the rule is associated to the first and only level of the "Customer" transaction).

This triggering event <u>should not</u> be added to the report calling rule, because the report would be called **immediately after the physical deletion** of each customer. Consequently, the report would not find the customer with the customer's data in the CUSTOMER table.



If we define a rule in which we also include the triggering event **on AfterInsert**, but unlike in the examples we just saw, we reference at least one attribute of the second level of the transaction in which we are defining the rule, the rule will be associated to the second level. Therefore, the rule will be executed **immediately after physically inserting** each instance corresponding to the second level of the transaction.

Something similar happens with on **AfterUpdate** and on **AfterDelete**.

Let's now expand our earlier scheme of the actions surrounding the triggering events seen so far:

DATA VALIDATION
AfterValidate – BeforeInsert – BeforeUpdate – BeforeDelete
RECORD SAVING (insert, update, delete, as appropriate)
AfterInsert – AfterUpdate – AfterDelete

This scheme is repeated for each of the level's instances. For example, let's apply this to entering lines corresponding to seats of a flight. This scheme will occur for each line, so that we can imagine a loop that is repeated until the last line of the grid is saved as physical record.

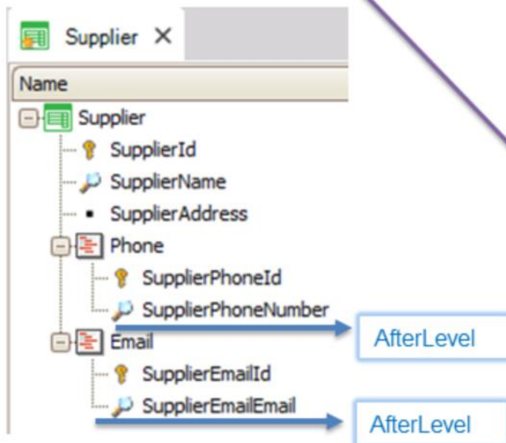
The action that comes after the last line is saved would be to exit this level (in this case, the flight seat level). And after that action, unless there's another level, which would take us back to the previous scheme, the last action in the execution (i.e. commit) will occur.

Between the exit level action and the commit action, we'll have an event (BeforeComplete) and another event for after the commit action (AfterComplete).

More about order of execution in transaction rules GeneXus

AfterLevel and BeforeComplete triggering moments

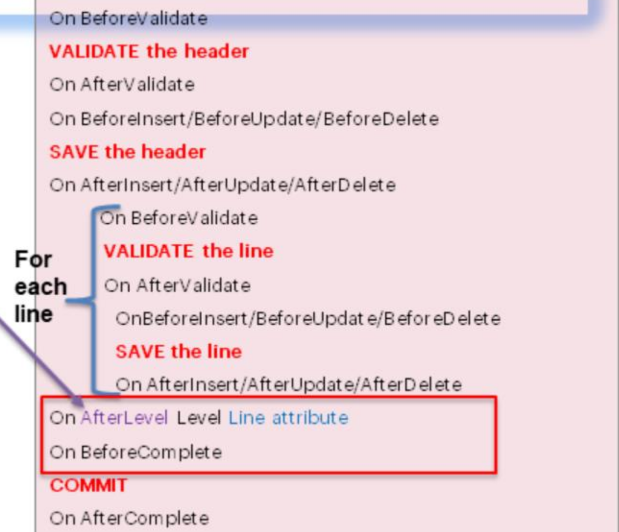
```
Error('The seat quantity should be equal or greather than 8') if FlightCapacity<8
    on AfterLevel
    Level FlightSeatChar;
```



AfterLevel

AfterLevel

For each line



The **AfterLevel** triggering event allows you to define a rule so that it is executed **immediately after** the iteration of a certain level is finished.

SYNTAX: *rule* [**if** *triggering condition*] [**on AfterLevel Level** *attribute*]**;**

WHERE:
rule: is a rule of the kind admitted in transactions
triggering condition: is a Boolean expression that admits attributes, variables, constants, and functions, as well as Or, And, and Not operators.
attribute: is an attribute belonging to the level after whose iteration you want the rule to be executed.

If the attribute specified after the **AfterLevel** triggering event belongs to the second level of the transaction, the rule will be executed when the iteration of all the lines in the second level is finished.

And if the attribute specified after the **AfterLevel** triggering event belongs to the first level —following the same line of reasoning— the rule will be executed after the iteration through all the headers is finished. Note that this occurs at the very end, that is, after all the headers and their lines have been entered and after closing the transaction (at that point all the headers will have been iterated). Therefore, if the attribute specified belongs to the first level, the rule will be triggered only once before the Exit Event (this is an event that is executed only once when the transaction is closed in runtime, as we will see later).

This triggering moment is used in the rule stated to validate that 8 seats or more are entered for each flight displayed on screen.

The event called **BeforeComplete**, in this case, is the same as the **AfterLevel** event. If we look at the scheme shown above, we can see that the moment between exiting the last level and performing the commit action is the moment in which these events occur. Both triggering moments occur at the same time as long as the level exited is the last one.

To help understand this, suppose that there's a transaction with two parallel levels; for example, a

Supplier transaction that has a nested level for telephone numbers and another level for email addresses:

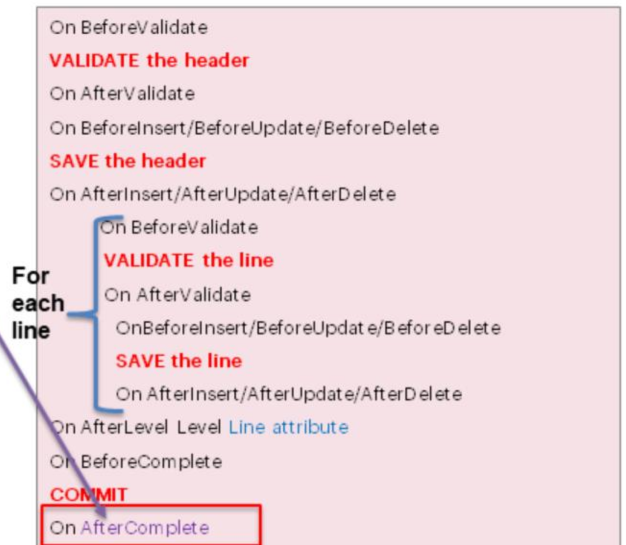
The triggering moment of a rule conditioned to: **On AfterLevel Level *SupplierPhoneNumber*** WILL NOT COINCIDE with the triggering moment of a rule conditioned to **on BeforeComplete.** The rules conditioned to on **AfterLevel Level *SupplierEMailEMail*** will match because it is the last subordinate level of the transaction.

While the first rule will be triggered when the telephone level is exited, and before validating all the e-mails, the second rule will be triggered after exiting this last level.
In this case, the **BeforeComplete** event will coincide with the **AfterLevel Level *SupplierEMailEMail*** event.

AfterComplete Triggering moment

```
PrintFlight(FlightId) on AfterComplete;
```

✓on AfterComplete:
Right after performing **Commit** to the database



This event corresponds to the instant immediately after the commit.

If three flights are entered in the Flight transaction (information regarding the 1st level + its respective seats) and the transaction is closed, there will be 3 commits occurred and after each commit the rules will be executed with **on AfterComplete** triggering event.

The header's attributes still have value in memory when the rules are executed with AfterComplete triggering event. Even when the commit has already been executed, it is typical to invoke a procedure that lists the information saved and committed, passing by parameter the primary key to the object called.

We will talk more about this event when we study **transactional integrity**.

Execution in 2-level transaction

Interactively and before pressing Confirm:

Airline Name	TAM
Airline Discount Percentage	10
Final Price	750
Capacity	6

Seat

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

[New row]

CONFIRM CANCEL DELETE

STAND-ALONE RULES

- THEY ARE TRIGGERED AS SOON AS THE TRN IS EXECUTED
- THERE ARE NO CONDITIONS SET FOR THEIR EXECUTION, AND THEY DON'T HAVE TO WAIT FOR DATA IN ORDER TO BE EXECUTED

RULES AND FORMULAS TO THE EXTENT THAT ASSOCIATED DATA FROM THE 1ST LEVEL IS AVAILABLE

RULES AND FORMULAS TO THE EXTENT THAT ASSOCIATED DATA FROM THE 2ND LEVEL IS AVAILABLE. } **FOR EACH LINE**

As the user enters data in the transaction and before pressing Confirm, the rules and formulas will be triggered according to the evaluation tree to the extent that the various attributes are involved. Stand-alone rules will be the first rules to be triggered.

They are the rules that:

1. May be executed with the information provided by the parameters received.
2. Do not depend on anything to be executed.

- Examples of stand alone rules (that may be executed with the information provided by parameters):

  &A = parameter2;

  Msg('...') if *parameter1* = 7;

- Examples of stand alone rules (that do not depend on anything to be executed):

  msg('You are in the flight transaction');

  &A = 7;

Therefore, they are the first rules executed.

Following the execution of the stand alone rules the rules and formulas associated with the 1st level in the transaction **with no triggering event defined** (that is, with no specification of **on ...**) are executed, provided that the values involved to execute them become available.

And when working on the 2nd level (grid), **for each line**, the rules and formulas associated with the 2nd level in the transaction **with no triggering event defined** (that is, with no specification of **on ...**) are executed provided that the values involved to execute them become available.

More about order of execution in transaction rules GeneXus

Execution in two-level transaction

After confirming the data, executions take place on the server in this order:

Airline Name TAM
Airline Discount Percentage 10
Final Price 750
Capacity 6
Seat

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

[New row]

CONFIRM CANCEL DELETE

STAND-ALONE RULES
RULES AND FORMULAS OF 1ST LEVELATTRIBUTES WITHOUT TRIGGERING MOMENTS
BeforeValidate
VALIDATE THE HEADER
AfterValidate / BeforeInsert - Update - Delete
SAVE THE HEADER
AfterInsert / Update / Delete
RULES AND FORMULAS OF 2ND LEVEL ATTRIBUTES WITHOUT TRIGGERING MOMENTS
BeforeValidate
VALIDATE THE LINE
AfterValidate / BeforeInsert - Udpate - Delete
SAVE THE LINE
AfterInsert/Update/Delete
FOR EACH LINE
END 2ND LEVEL ITERATION
AfterLevel Level 2ndLevelattribute - BeforeComplete
COMMIT
AfterComplete

After the user presses on Confirm, data travels to the web server. The server runs through the form again as if it was a user and the rules and formulas will be triggered again. However, if now there's a rule conditioned to a triggering moment in the server, it will be triggered at its own time and not according to the evaluation tree.

The order of execution will be as follows:

Execution of Stand-alone Rules.

Execution of all rules and formulas associated with the 1st level which have no triggering event defined.

Execution of rules associated with the 1st level with **BeforeValidate** triggering events .

Validation of data entered for the 1st level.

Execution of rules associated with the 1st level with **AfterValidate** triggering events.

• if the saving corresponded to an insertion: the rules associated to the first level of the transaction with **AfterInsert** triggering event will be executed.
• if the saving corresponded to an update: the rules associated to the first level of the transaction with **AfterUpdate** triggering event will be executed.
• if the saving corresponds to a deletion: the rules associated with 1st level in the transaction with **BeforeDelete** triggering event will be executed.

The SAVING action is executed. This means that the instance corresponding to the 1st level in the transaction will be physically saved as physical record in the corresponding table (in this example, in the FLIGHT table).

Immediately after that saving:

· if the saving was an insertion: the rules associated with the 1st level in the transaction, with **AfterInsert** triggering event, are executed.
· if the saving was an update: the rules associated with the 1st level in the transaction, with **AfterUpdate** triggering event, are executed.
· if the saving was a deletion: the rules associated with the 1st level in the transaction, with **AfterDelete** triggering event, are executed.

After running all the operations explained so far, a COMMIT operation will be made, and all the rules with AfterComplete triggering event will be executed.

It must be kept in mind that all the operations explained will be executed in the order in which they have been described, for each flight accessed through the Flight transaction (whether to enter, edit or delete them).

Learning the order in which the rules in a transaction are executed, the triggering events which are available to assign to them, the exact moment when they are triggered, and what actions take place before and after each triggering event is very important, because this knowledge is essential to be able to program the transactions' behavior correctly.

Exercises

When will these rules be triggered?

- *Something(FlightId) on BeforeInsert;*

  An instant before inserting the record corresponding to the 1st level.

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

  An instant before inserting each record corresponding to the 2nd level.

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

  Same as the previous case. Note that **Level FlightSeatChar** specifies that reference is made to the BeforeInsert of lines and not of the header.

Exercises

Some of these rules are no programmed correctly. Which ones?

- *FlightPrice = 2000* on AfterInsert;

Incorrect: The last moment to assign value to a header attribute is immediately prior to saving it (BeforeInsert).

- *Something(FlightPrice) on AfterInsert;*

Correct: here the value of a header attribute is passed. This value is available in memory. Last moment possible for using it AfterComplete.

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

Incorrect: the rule with no triggering event is associated with the 2nd level, meaning that it would be triggered for each line. But the triggering event conditions it to be executed upon exiting the lines… and we no longer have 2nd level's attribute values.

Rules with the same triggering event

Are triggered in the same order in which they were defined

Example 1

```
xxx() On AfterComplete;
yyy() On AfterComplete;
```

Example 2

Option 1

```
Something(FlightId, &flag) On AfterComplete;
error('     ') if &flag = 'N' On AfterComplete;
```

Option 2

```
&flag = Something(FlightId) On AfterComplete;
error('     ') if &flag = 'N' On AfterComplete;
```

When two or more rules are defined in a transaction with the same triggering event and there is no dependence between them, they will be executed according to the order of definition.

Examples:

Example 1: Since the two rules defined are conditioned with the same triggering event and there is no dependence between them, they will be executed in the same order in which they were written.

Example 2: In a transaction it is necessary to invoke a procedure that performs a specific validation and returns a 'Y' or an 'N' value; and if the value returned is 'N', then an error message must be informed.

To solve this we will evaluate the two option shown above.

In the first alternative we defined a rule **that invokes, as program,** an object and an **error** rule. Both rules have the same triggering event, and *apparently there would be dependence between them* since the **error** rule is conditioned to the value of variable *&flag,* and the *&flag* variable is passed by parameter in the invocation.

However, even when such dependence may seem evident because in the procedure we will be programming the output variable *&flag,* in the rules section of the transaction —where the rules we are considering are located—, the specifier of GeneXus cannot tell whether the parameters passed in an **invocation as program** are input, output, or input-output. Therefore, the specifier will not find interdependence between the **call and error** rules because the *&flag* variable could be passed as input variable to the procedure, and in that case for example, there would be no dependence, by virtue of which the **invocation as program** is to be executed first, and the **error** rule afterwards.

So, in conclusion, *no dependences are detected between the rules of option 1, thus they will be triggered in the order in which they were written.* It is important to see here that if the rules were written in reverse order (that is: first the **error** rule and then the **invocation as program**), then the behavior will not be the

one expected in many cases.

Regarding the second alternative, we should note that it consists of a rule **that invokes as function the object *Something* and an error rule**. Both rules have the same triggering event, and *in this case there is dependence between them*, because the **error** rule is conditioned to the value of the *&flag* variable, and since the **invocation to the procedure is done as function**, to GeneXus it is clear that the *&flag* variable returns modified from the procedure. So, GeneXus understands that the **invocation to the procedure as function** must be triggered first, followed by the **error** rule, because the *&flag* variable is loaded through the invocation to the procedure and after the variable has a value there will have to be an evaluation as to whether the **error** rule is to be triggered or not.

In case Option 2 then, regardless of the order in which the two rules were defined, **the invocation to the procedure as function** will be triggered first, and the **error** rule will be triggered afterwards (if the triggering condition has been complied with, of course).

This is why invocation as function instead of as program is recommended for defining validations of this type.

GeneXus™

The power of doing.

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