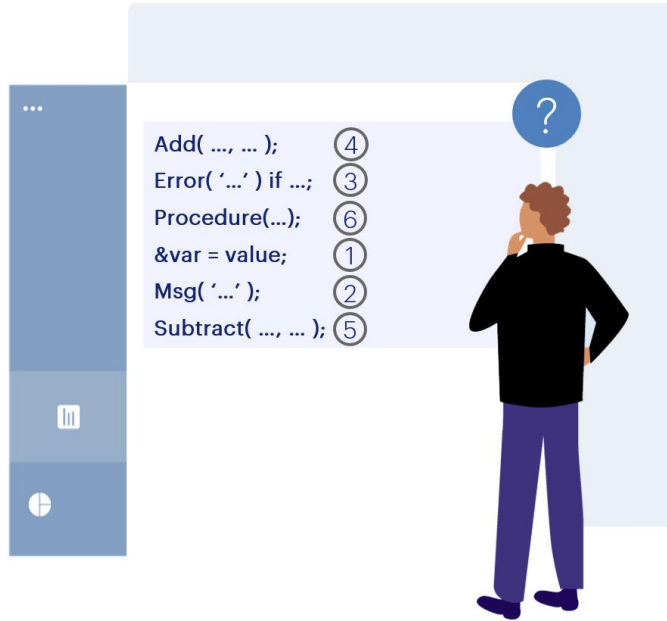


# Árbol de evaluación de disparo de reglas y fórmulas

GeneXus™

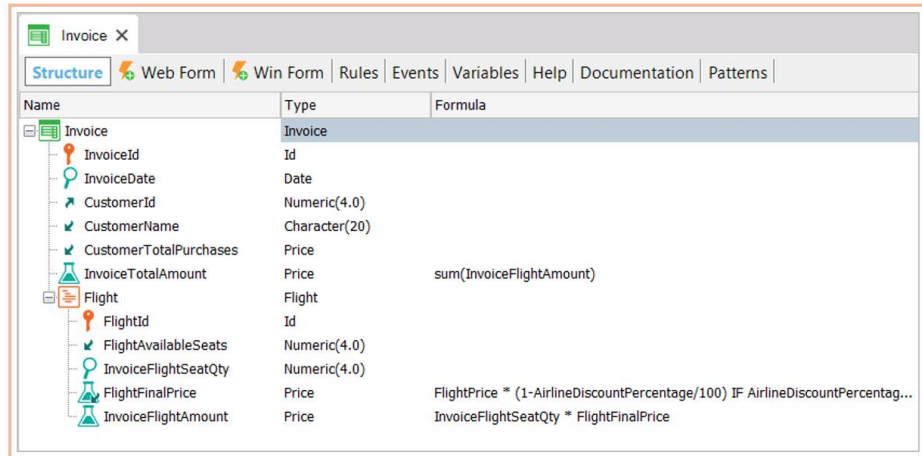
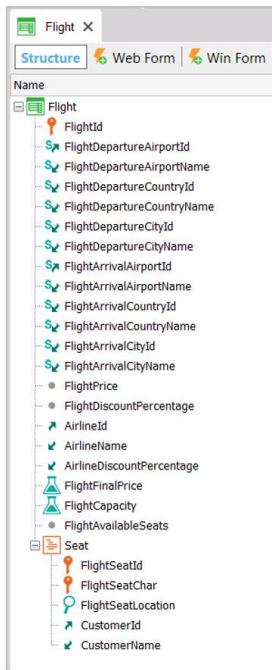
## Rules in Transactions



Sabemos que las reglas en una transacción se declaran en cualquier orden y es GeneXus quien determina el momento en que cada una se dispara. Esto a veces resulta confuso para los desarrolladores, porque sienten que pierden el control.

Pero en realidad se trata de una ventaja. Los desarrolladores solo debemos preocuparnos por declarar la lógica y GeneXus inferirá automáticamente, dónde y cuándo cada regla deberá dispararse.

## Reality



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.

Nos vamos a concentrar en esta transacción y en el disparo de sus reglas.

## Reality

The image displays two transaction structure windows in GeneXus. The left window shows the 'Flight' transaction structure, and the right window shows the 'Invoice' transaction structure. A diagram of an airplane cabin is positioned between the two windows, with an arrow pointing from the 'FlightAvailableSeats' attribute in the Flight structure to the cabin diagram.

**Flight Transaction Structure:**

- FlightId
- FlightDepartureAirportId
- FlightDepartureAirportName
- FlightDepartureCountryId
- FlightDepartureCountryName
- FlightDepartureCityId
- FlightDepartureCityName
- FlightArrivalAirportId
- FlightArrivalAirportName
- FlightArrivalCountryId
- FlightArrivalCountryName
- FlightArrivalCityId
- FlightArrivalCityName
- FlightPrice
- FlightDiscountPercentage
- AirlineId
- AirlineName
- AirlineDiscountPercentage
- FlightFinalPrice
- FlightCapacity
- FlightAvailableSeats
- Seat
  - FlightSeatId
  - FlightSeatChar
  - FlightSeatLocation
  - CustomerId
  - CustomerName

**Invoice Transaction Structure:**

Name	Type	Formula
Invoice	Invoice	
InvoiceId	Id	
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/100) IF AirlineDiscountPercentag...
InvoiceFlightAmount	Price	InvoiceFlightSeatQty * FlightFinalPrice

Observemos que hemos agregado a la transacción Flight el atributo **FlightAvailableSeats**, 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 asientos en el vuelo.

Lo hemos agregado, entonces, en este nivel. Será un atributo inferido.

## Reality

The image displays the GeneXus IDE interface with three entity structure windows: Flight, Invoice, and Customer. The Flight entity structure includes attributes such as FlightId, FlightDepartureAirportId, FlightDepartureCountryId, FlightDepartureCityId, FlightArrivalAirportId, FlightArrivalCountryId, FlightArrivalCityId, FlightPrice, FlightDiscountPercentage, AirlineName, AirlineDiscountPercentage, FlightFinalPrice, FlightCapacity, FlightAvailableSeats, and Seat. The Invoice entity structure includes InvoiceId, InvoiceDate, CustomerId, CustomerName, CustomerTotalPurchases, InvoiceTotalAmount, Flight, FlightId, FlightAvailableSeats, InvoiceFlightSeatQty, FlightFinalPrice, and InvoiceFlightAmount. The Customer entity structure includes CustomerId, CustomerName, CustomerAddress, CustomerPhone, CustomerEmail, CustomerAddedDate, and CustomerTotalPurchases. A formula for InvoiceFlightAmount is shown:  $\text{sum}(\text{InvoiceFlightAmount})$  and  $\text{FlightPrice} * (1 - \text{AirlineDiscountPercentage} / 100)$  IF AirlineDiscountPercentag...  $\text{InvoiceFlightSeatQty} * \text{FlightFinalPrice}$ . A diagram of an airplane cabin layout is also visible.

Hemos agregado también a la transacción Customer el atributo **CustomerTotalPurchases**, para registrar el total comprado por el cliente por concepto de pasajes de vuelos. También lo agregamos a nuestra transacción como atributo inferido.

Los atributos InvoiceTotalAmount, FlightFinalPrice e InvoiceFlightAmount de la estructura de Invoice han sido definidos como fórmulas.

## Invoice Rules

Name	Type	Formula
Invoice	Invoice	
InvoiceId	Id	
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/100) IF AirlineDiscountPercentag...
InvoiceFlightAmount	Price	InvoiceFlightSeatQty * FlightFinalPrice

```

1 Default(InvoiceDate, &Today);
2
3 Subtract(InvoiceFlightSeatQty, FlightAvailableSeats);
4
5 Error("There are no more seats for sale")
6   if FlightAvailableSeats < 0;
7
8 Add(InvoiceTotalAmount, CustomerTotalPurchases);
9
10

```

En la transacción Invoice, hemos definido las siguientes reglas para especificar su comportamiento:

La regla **Default**, que inicializa el atributo de la fecha de la factura con la fecha de hoy; la regla **Subtract**, que decrementa la cantidad de asientos disponibles del vuelo según la cantidad de asientos comprados en la factura -observemos que estará decrementando un atributo de la tabla Flight: FlightAvailableSeats es, aquí, inferido-; la regla **Error**, que muestra un mensaje de error si el vuelo ya no cuenta con asientos disponibles; y la regla **Add**, que suma el total de la factura al total de compras realizadas por el cliente -otra vez, atributo presente en una tabla de la extendida, Customer-.

## Evaluation Tree

(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 are no more seats for sale" ) if FlightAvailableSeats < 0;

The screenshot shows a form titled "Invoice" with the following fields and a table:

- Id: InvoiceId
- Date: InvoiceDate
- Customer Id: CustomerId
- Customer Name: CustomerName
- Customer Total Purchases: CustomerTotalPurchases
- Total Amount: InvoiceTotalAmount

Below the form is a table with the following columns:

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
FlightId	FlightAvailableSeats	InvoiceFlightSeatQty	FlightFinalPrice	InvoiceFlightAmount

Resumiendo, tenemos todas estas reglas y fórmulas definidas en la transacción Invoice:

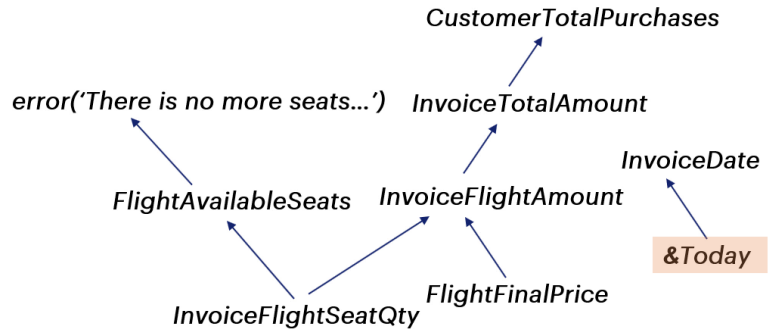
La gran pregunta es ¿cómo sabe GeneXus en qué orden debe dispararlas, y cuándo sí y cuándo no?

Por supuesto, hay un primer orden natural que es el que corresponde al ordenamiento de los atributos en la pantalla (de arriba hacia abajo y de izquierda a derecha).

## Evaluation Tree

```
(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 are no more seats for sale" ) if FlightAvailableSeats < 0;
```

Invoice	
	<< < > >> SELECT
Id	<input type="text" value=""/>
Date	<input type="text" value="10/13/20"/> <input type="button" value="29"/>
Customer Id	<input type="text" value="0"/> <input type="button" value="↑"/>
Customer Name	

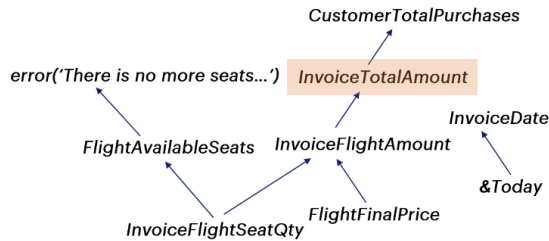
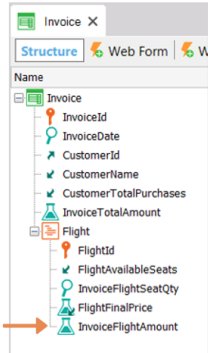


Una regla o fórmula se dispara ni bien se cuenta con la información que necesita. Por ejemplo, la regla Default solo necesita el valor de la variable &Today y saber que se está en modo Insert. Por eso apenas abrimos la pantalla en modo Insert ya vemos el valor en el campo, aún cuando ni siquiera hemos llegado a él (apenas estamos posicionados sobre Invoiceld).



## Evaluation Tree

- (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 are no more seats for sale" ) if FlightAvailableSeats < 0;



Invoice

Id:

Date: 10/13/20

Customer Id:

Customer Name:

Customer Total Purchases: 0.00

Total Amount: → 0.00

Flight

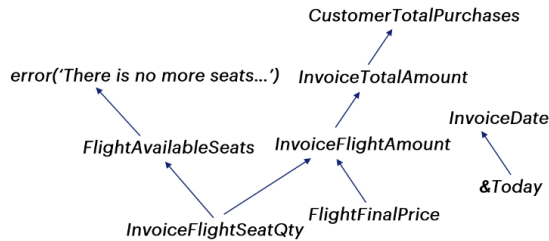
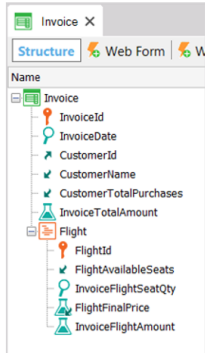
Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0.00	0.00

[New row]

Pensemos qué pasa con la fórmula del primer nivel: InvoiceTotalAmount, que es un sum de un atributo del segundo nivel. Como la fórmula Sum necesita solamente el atributo InvoiceFlightAmount, se va a disparar para el cabezal incluso antes de haber podido ingresar la primera línea, dando 0.

## Evaluation Tree

- (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 are no more seats for sale" ) if FlightAvailableSeats < 0;



Invoice

Navigation: << < > >> SELECT

Id: 0

Date: 10/13/20

Customer Id: 1

Customer Name: Joseph

Customer Total Purchases: 25110.00

Total Amount: 9900.00

Flight

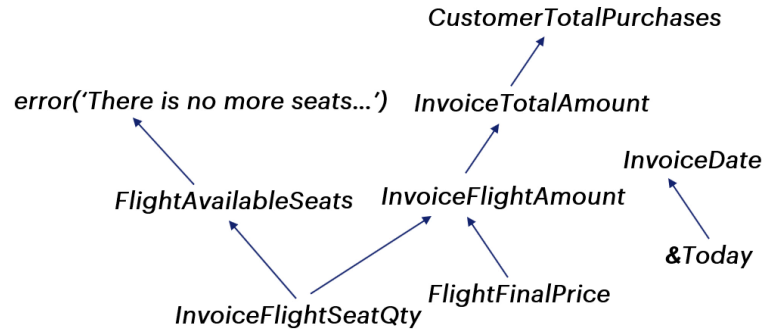
Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
1	148	2	2700.00	5400.00
2	497	1	4500.00	4500.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00

[New row]

Pero luego, a medida que vayamos ingresando líneas, por cada una se va a redisparar. Pero, ¿y qué pasa si entramos en modo Update a la transacción y, por ejemplo, modificamos algo de una línea que no modifique para nada el InvoiceFlightAmount? (en este caso no tenemos ningún atributo para modificar que no modifique ese valor, porque los únicos dos atributos editables son el id de la línea, que no se puede cambiar por ser parte de la clave primaria, y luego el InvoiceFlightSeatQty, que sí modifica el valor de InvoiceFlightAmount, pero imaginemos que lo hubiera, por ejemplo, que deba indicarse si alguno de los pasajeros es diabético, y en el línea en cuestión teníamos que sí, y ahora queremos cambiarlo por no). Obviamente, en ese caso, la fórmula del cabezal no se recalcularía.

## Evaluation Tree

```
(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 are no more seats for sale" ) if FlightAvailableSeats < 0;
```



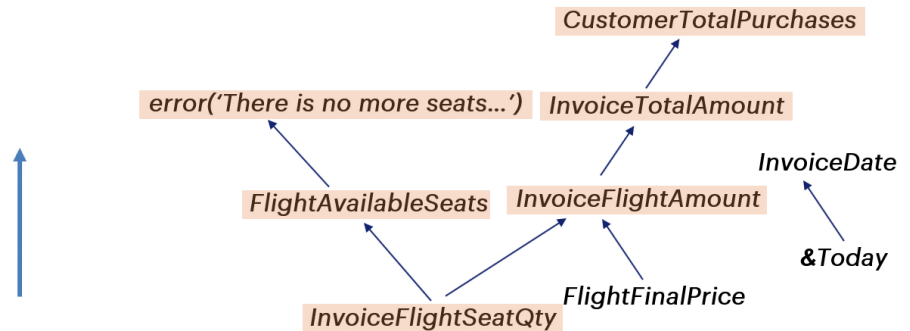
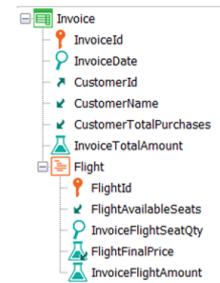
Este “obviamente” también lo es para GeneXus, que lo que hace internamente es extraer las dependencias existentes entre lugares que asumen los controles en la pantalla, las reglas y las fórmulas, para construir un árbol de dependencias (conocido como árbol de evaluación) que es el que determinará qué reglas y fórmulas deberán redispárarse ante cambios en atributos de la pantalla. En nuestro ejemplo será así:

## Evaluation Tree

```

(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 are no more seats for sale" ) if FlightAvailableSeats < 0;

```



Por ejemplo, atendamos lo que sucede con el atributo InvoiceFlightSeatQty. De él depende la actualización con subtract del atributo FlightAvailableSeat, asientos disponibles del vuelo, del que depende, a su vez, el disparo de la regla error. Por ese motivo la condición de la regla error debe escribirse sabiendo que siempre, por esta dependencia, ya se habrá ejecutado el subtract, y es por ello que se coloca la condición 'menor que cero'. Recuerde que, como ya hemos estudiado, de quedar asientos negativos, se deshará todo lo hecho en el árbol que condujo a producir este error.

A su vez, también de InvoiceFlightSeatQty depende la actualización de la fórmula InvoiceFlightAmount, de la que, por su parte, depende la del cabezal, InvoiceTotalAmount, de la que depende la actualización del total de compras del cliente.

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).

## Evaluation Tree

Invoice

« < > » SELECT

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

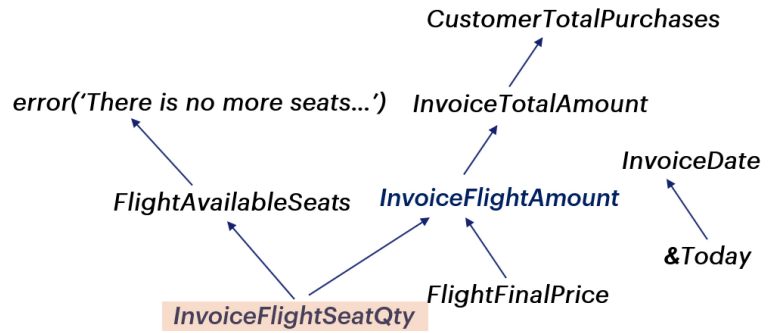
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
×	2	98	2	2700.00
	0	0	0	0.00
	0	0	0	0.00
	0	0	0	0.00
	0	0	0	0.00

[New row]



$$\text{InvoiceFlightAmount} = \text{FlightFinalPrice} * \text{InvoiceFlightSeatQty}$$

Sigamos el ejemplo anterior:

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.

## Evaluation Tree

Invoice

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

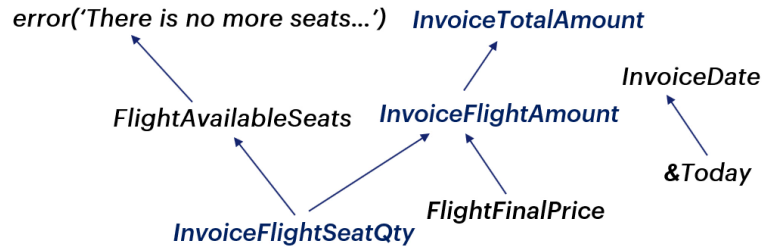
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
2	98	2	2700.00	5400.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00
	0	0	0.00	0.00

[New row]



$$\text{InvoiceFlightAmount} = \text{FlightFinalPrice} * \text{InvoiceFlightSeatQty}$$

$$\text{InvoiceTotalAmount} = \text{Sum}(\text{InvoiceFlightAmount})$$

Al redispárarse, también deberá hacerlo la fórmula correspondiente al total de la factura (InvoiceTotalAmount).

## Evaluation Tree

**Invoice**

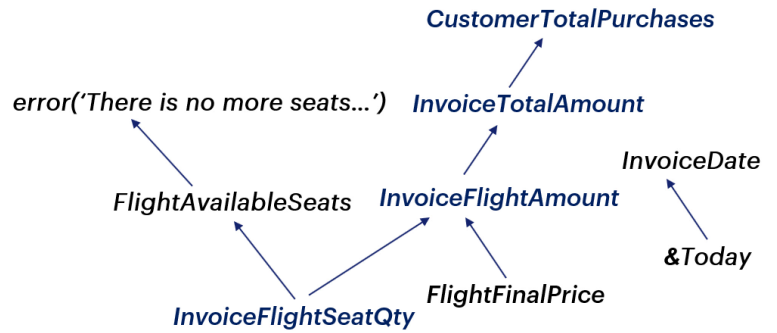
<< < > >> SELECT

Id	<input type="text" value="0"/>
Date	<input type="text" value="09/17/20"/> <span style="color: red; font-size: small;">[29]</span>
Customer Id	<input type="text" value="1"/> <span style="color: blue; font-size: small;">[v]</span>
Customer Name	Joseph
Customer Total Purchases	5400.00 <span style="color: orange; font-size: small;">←</span>
Total Amount	5400.00

**Flight**

	Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
×	<input type="text" value="2"/> <span style="color: blue; font-size: small;">[v]</span>	98	<input type="text" value="2"/> <span style="color: blue; font-size: small;">[v]</span>	2700.00	5400.00
	<input type="text" value=""/> <span style="color: blue; font-size: small;">[v]</span>	0	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0.00	0.00
	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0.00	0.00
	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0.00	0.00
	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0	<input type="text" value="0"/> <span style="color: blue; font-size: small;">[v]</span>	0.00	0.00

[New row]



**InvoiceFlightAmount = FlightFinalPrice \* InvoiceFlightSeatQty**

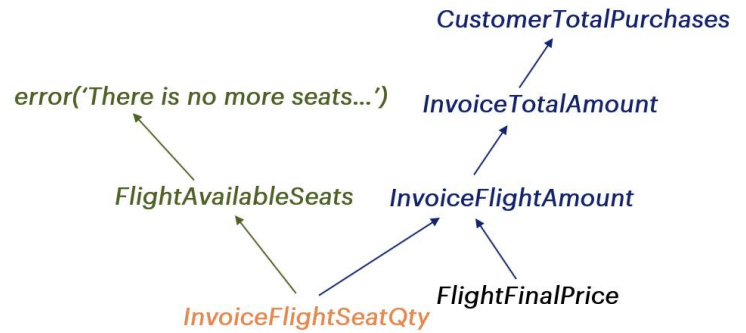
**InvoiceTotalAmount = Sum( InvoiceFlightAmount )**

**Add( InvoiceTotalAmount, CustomerTotalPurchases );**

Por último, por cambiar el total también se tendrá que disparar la regla `Add(InvoiceTotalAmount, CustomerTotalPurchases)` ya que deberá actualizarse el total de compras del cliente.

## Evaluation Tree

(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 are no more seats for sale" ) if FlightAvailableSeats < 0;



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.



## Evaluation Tree

Invoice

« < > » SELECT

Id: 0

Date: 09/17/20

Customer Id: 1

Customer Name: Joseph

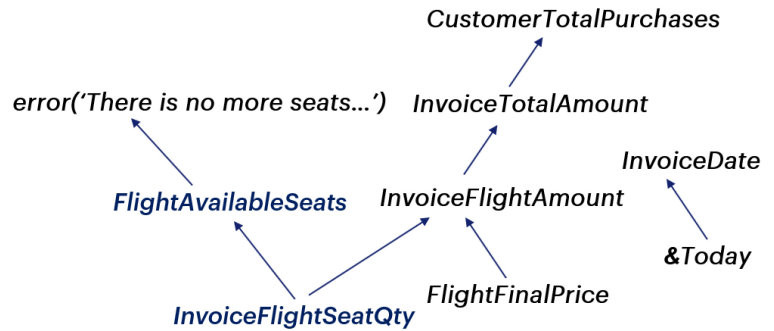
Customer Total Purchases: 5400.00

Total Amount: 5400.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
2	98	2	2700.00	5400.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00

[New row]



**Subtract( InvoiceSeatQty, FlightAvailableSeats );**

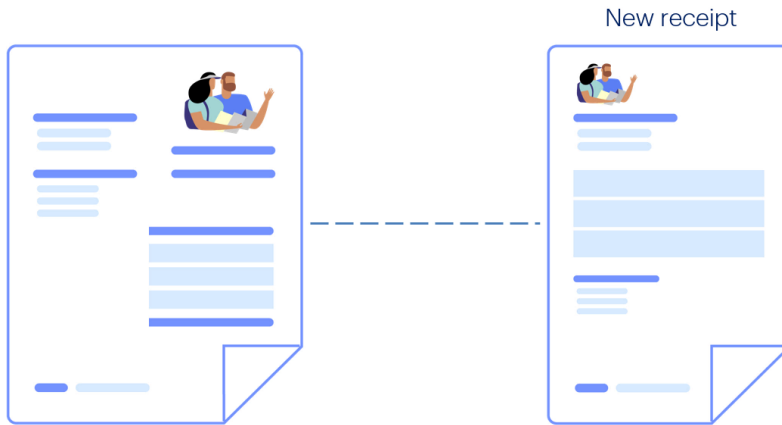
**Error( "There are no more seats for sale" )  
if FlightAvailableSeats < 0;**

Como ya vimos, 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 para saber si habrá que disparar la regla Error que indica que ya no hay asientos disponibles.

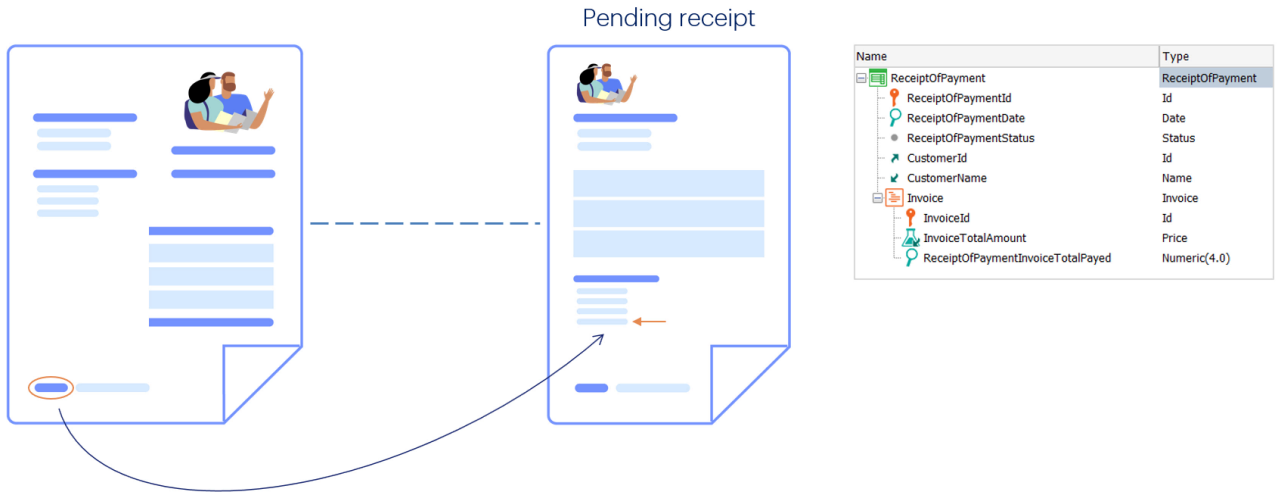
Si la condición para dispararse el error se satisface, todo lo hecho en el árbol desde el cambio en el atributo InvoiceFlightSeatQty será deshecho automáticamente, y los datos de la base de datos volverán al estado anterior a la ejecución de la regla error.

## Reality



Veamos ahora un ejemplo en el que una regla no siempre se dispara en el momento deseado. Supongamos que inmediatamente de realizarle una factura a un cliente por una cantidad de asientos de vuelos comprados, queremos generarle un recibo de pago. Si el cliente está al día con los pagos, entonces le generamos un recibo nuevo,

## Reality



pero si no lo está, entonces agregamos el importe de esta factura al recibo pendiente anterior. Hemos creado una transacción ReceiptOfPayment. La misma se compone por el identificador del recibo, la fecha, el estado (que es un dominio enumerado con los valores pendiente y completado), el cliente, y un segundo nivel para registrar las facturas para las cuales se emite el recibo de pago.

## Reality

Invoice

Id: 0

Date: 10/11/20

Customer Id: 1

Customer Name: Joseph

Customer Total Purchases: 14400.00

Total Amount: 0.00

Flight

Flight Id	Flight Available Seats	Seat Qty	Flight Final Price	Flight Amount
3	100	1	890.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00
0	0	0	0.00	0.00

[New row]

CONFIRM CANCEL

Diagram illustrating the data flow from the Invoice form to the RECEIPTOFPAYMENT Table. The table contains a search icon and the following SQL query:

```
where CustomerId = &CustomerId
where ReceiptOfPaymentStatus = Status.Pending
```

Entonces, cuando se ingresa una nueva factura, se busca por programa si ya existe un recibo de ese cliente en estado Pending.

# Reality

Selection List Invoice

id	Date	Customer Id	Invoice Total Amount
✓ 1	10/07/20	1	5400.00
✓ 2	10/09/20	1	9000.00
✓ 3	10/11/20	1	810.00

CANCEL



RECEIPTOFPAYMENT Table

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

if exists

where CustomerId = &CustomerId  
 where ReceiptOfPaymentStatus  
 = Status.Pending

Receipt Of Payment

Payment Id: [input]  
 Payment Date: 10/11/20 [calendar]  
 Payment Status: Pending [dropdown]  
 Customer Id: 1 [input]  
 Customer Name: Joseph

Invoice

Invoice Id	Invoice Total Amount	Total Payed
× 1	5400.00	0
× 2	9000.00	0
× 3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0

[New row]

Si existe, se agrega una nueva línea con esta factura.

# Reality

id	Date	Customer Id	Invoice Total Amount
✓ 1	10/07/20	1	5400.00
✓ 2	10/09/20	1	9000.00
✓ 3	10/11/20	1	810.00

CANCEL



RECEIPTOFPAYMENT Table

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

if not exists

where CustomerId = &CustomerId  
 where ReceiptOfPaymentStatus  
 = Status.Pending

**Receipt Of Payment**

<< < > >> SELECT

Payment Id	<input type="text"/>
Payment Date	10/11/20 <input type="button" value="20"/>
Payment Status	→ Pending <input type="button" value="v"/>
Customer Id	1 <input type="button" value="u"/>
Customer Name	Joseph

**Invoice**

Invoice Id	Invoice Total Amount	Total Paid
✕ 3 <input type="button" value="u"/>	\$10.00	0
0 <input type="button" value="u"/>	0.00	0
0 <input type="button" value="u"/>	0.00	0
0 <input type="button" value="u"/>	0.00	0
0 <input type="button" value="u"/>	0.00	0
0 <input type="button" value="u"/>	0.00	0

[New row]

Si no existe, se crea el cabezal y la línea, y el cabezal queda en estado Pending.

## Reality

Payment Id: 1

Payment Date: 10/11/20

Payment Status: Pending

Customer Id: 1

Customer Name: Joseph

Status Completed  
if InvoiceTotalAmount =  
ReceiptOfPaymentInvoiceTotalPaid

Invoice Id	Invoice Total Amount	Total Paid
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
[New row]		

Name	Type
ReceiptOfPayment	ReceiptOfPayment
ReceiptOfPaymentId	Id
ReceiptOfPaymentDate	Date
ReceiptOfPaymentStatus	Status
CustomerId	Id
CustomerName	Name
Invoice	Invoice
InvoiceId	Id
InvoiceTotalAmount	Price
ReceiptOfPaymentInvoiceTotalPaid	Numeric(4,0)

CONFIRM
CANCEL
DELETE

Luego viene el cliente a pagar, por lo que el empleado abre la transacción y modifica el atributo `ReceiptOfPaymentInvoiceTotalPaid` para las facturas que el cliente desee pagar.

El estado del recibo (`ReceiptOfPaymentStatus`) sólo podrá pasarse a `Completed` si coinciden para todas las líneas el valor de `InvoiceTotalAmount` con el de `ReceiptOfPaymentInvoiceTotalPaid`. Supongamos que esa modificación es realizada por el usuario, es decir, que es él quien cambia el valor de `ReceiptOfPaymentStatus`, por lo que debe corroborarse que no se permita cambiar a `Completed` si quedó alguna factura impaga o mal paga.

## Reality

Payment Id: 1

Payment Date: 10/11/20

Payment Status: Completed

Customer Id: 1

Customer Name: Joseph

**Invoice**

Invoice Id	Invoice Total Amount	Total Paid
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
[New row]		

CONFIRM CANCEL DELETE

```

1 Error('Incomplete payments')
2   if ReceiptOfPaymentStatus = Status.Completed
3     and InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPaid
4     and Update;

```

Podríamos pensar en colocar la siguiente regla de error, condicionándola a dispararse cuando estamos actualizando un recibo, el estado es Completed, y no coincide el valor del atributo que indica lo que debe pagarse y el que indica lo que se pagó para una línea:

Sin embargo... ¿cuándo se disparará esta regla?

Claramente si ingresamos a la transacción, cambiamos el estado a Completed y para una línea ingresamos un valor diferente del esperado, se disparará. Pero, ¿qué pasaría si para otra línea que tiene en 0 la cantidad pagada ni siquiera entramos? ¿Se disparará la regla?

La respuesta es no. Cuando ejecutamos una transacción en modo Update podemos no querer modificar el cabezal y cambiar solamente una línea. ¿En ese caso, qué se disparará? El cabezal se actualizará siempre, y luego solo la línea modificada. Para ella se disparará todo de acuerdo al árbol de evaluación.



## Reality

Payment Id: 1

Payment Date: 10/11/20

Payment Status: Pending

Customer Id: 1

Customer Name: Joseph

Invoice Id	Invoice Total Amount	Total Payed
1	5400.00	5400
2	9000.00	0
3	810.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0
0	0.00	0

CONFIRM CANCEL DELETE

Parm(ReceiptOfPaymentId);

```

if InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPayed
    &ok = false
else
    &ok = true
endif

```

```

1 &ok = CheckAllValid(ReceiptOfPaymentId)
2   if Update and ReceiptOfPaymentStatus = Status.Completed
3     on BeforeComplete; //equivalent with on AfterLevel event
4
5 Error('Incomplete payments')
6   if Update and ReceiptOfPaymentStatus = Status.Completed and not &ok
7     on BeforeComplete; //equivalent with on AfterLevel event
8

```

¿Cómo resolveríamos, entonces este caso?

Una solución es llamar a un procedimiento al que le enviamos el id del recibo, una vez que dimos tiempo a modificar todas las líneas, y una vez que estas modificaciones se realizaron en la base de datos, pero antes del commit, así podemos deshacer. Ese procedimiento recorre TODAS las líneas y se asegura de que no quede ninguna con valor diferente para los atributos que nos interesan.

Si bien para la regla de Error podría parecer que no necesitamos condicionar al evento BeforeComplete dado que la variable &ok para evaluarse ya lo está, en verdad sí es necesario. Si no condicionáramos el error a exactamente el mismo evento, rompería las dependencias entre ambas reglas.

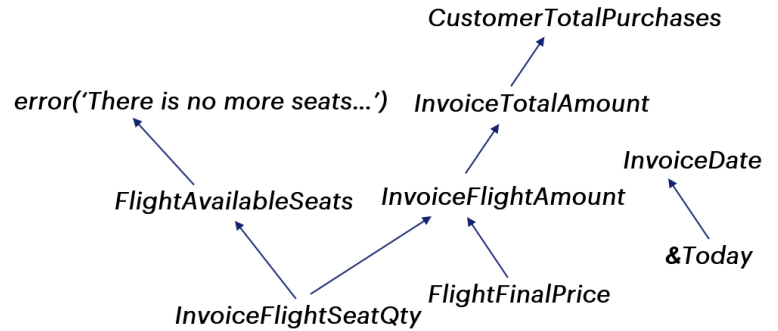
Cada evento de disparo tiene su propio árbol de evaluación, lo que significa que si condicionamos muchas reglas al mismo evento, las ordenará **al momento de ocurrir el evento** de acuerdo a sus dependencias, tal y como vimos antes.

## Evaluation Tree

```

(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 are no more seats for sale" ) 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.

## Reality

Payment Id

Payment Date

Payment Status

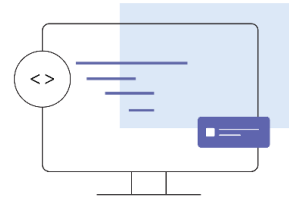
Customer Id

Customer Name Joseph

**Invoice**

Invoice Id	Invoice Total Amount	Total Payed
× 1 <input type="button" value="↑"/>	5400.00	5400
× 2 <input type="button" value="↑"/>	9000.00	9000
× 3 <input type="button" value="↑"/>	810.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0
<input type="text" value="0"/> <input type="button" value="↑"/>	0.00	0

[New row]



```

if InvoiceTotalAmount <> ReceiptOfPaymentInvoiceTotalPayed
    &ok = false
else
    &ok = true
endif

```

```

ReceiptOfPayment * X
Structure | Web Form | Win Form | Rules * | Events | Variables | Help | Documentation | Patterns
1 &ok = CheckAllValid(ReceiptOfPaymentId)
2   if Update and ReceiptOfPaymentStatus = Status.Completed
3     on BeforeComplete; //equivalent with on AfterLevel event
4
5 Error('Incomplete payments')
6   if Update and ReceiptOfPaymentStatus = Status.Completed and not &ok
7     on BeforeComplete; //equivalent with on AfterLevel event
8

```

En ocasiones, el árbol de evaluación no determina el orden de ejecución que deseamos: un claro ejemplo de ello es el que acabamos de ver, donde debimos retrasar el momento de disparo del procedimiento que chequea sobre los registros de las líneas y el error subsiguiente.

## Navigation Report

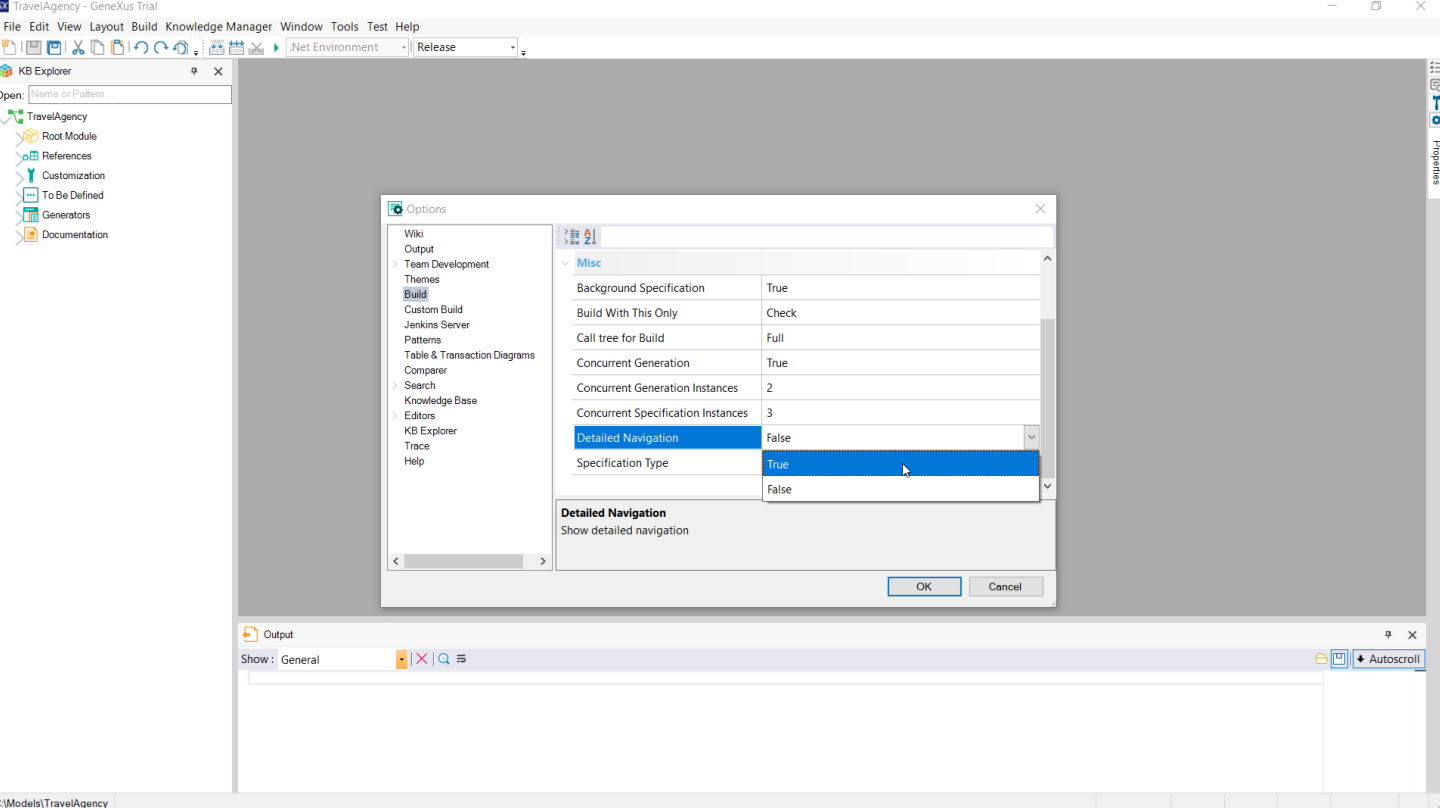
## Detailed Navigation Report

Table	Program	In Parameters	Out Parameters
Flight	Gx00B0		FlightId
InvoiceFlight	Gx00F1	InvoiceId	FlightId
Customer	Gx0010		CustomerId
Invoice	Gx00E0		InvoiceId

Si desea ver con más nivel de detalle el orden de las evaluaciones que dispara GeneXus, puede utilizar el listado de navegación detallado.

Aquí vemos la diferencia entre ambos... veamos, por ejemplo, que en la navegación detallada se nos muestran las reglas y los momentos en que se dispararán, lo que no sucede en el otro caso.

La navegación detallada puede ser útil en los casos en los que necesitamos entender bien dónde se está disparando una fórmula o regla, pero para lo normal lleva más tiempo de especificación, por lo que muchas veces no lo necesitamos.



Para habilitarlo, debe ir al menú **Tools > Options**, y dentro de la categoría **Build** activar la propiedad **Detailed Navigation**.

# GeneXus™

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