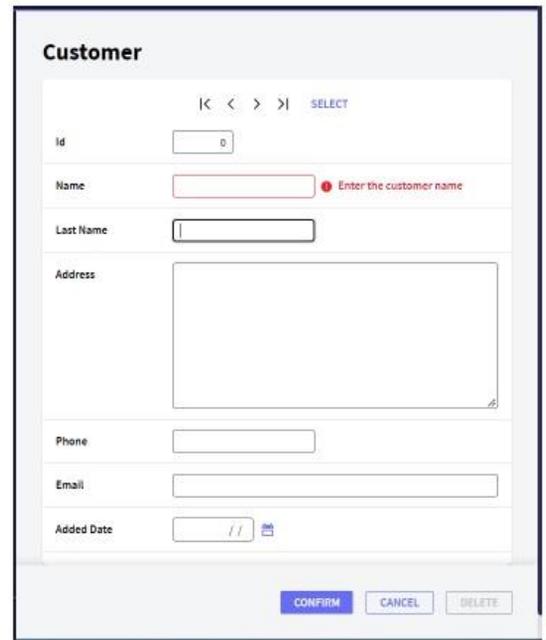
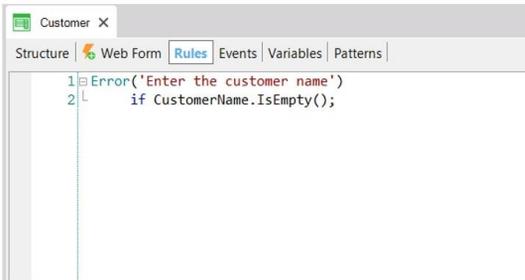
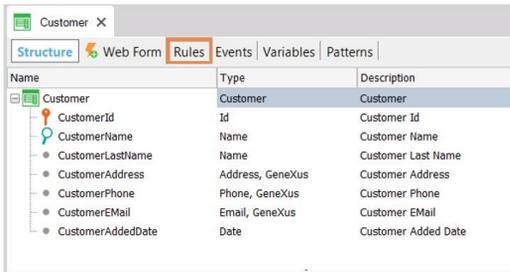


## Reglas: Repaso y Client-Side Validation

GeneXus™

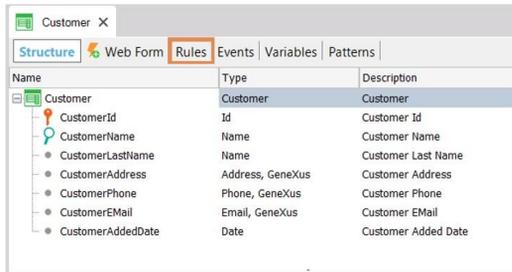
## Rules



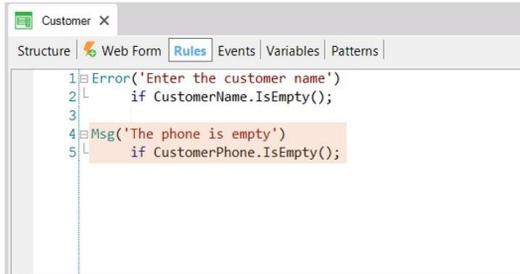
En el curso anterior aprendimos que el objeto Transacción cuenta con una sección llamada **Rules**, en la cual son definidos los controles que se deben efectuar o las reglas que se deben cumplir para una cierta realidad.

Nos centramos en la regla **Error**, que impide que se almacene en la base de datos un registro mientras cierta condición se cumpla: por ejemplo, si estamos ingresando un cliente y dejamos su nombre vacío, ese cliente no se podrá insertar hasta que el usuario haya escrito su nombre.

## Rules



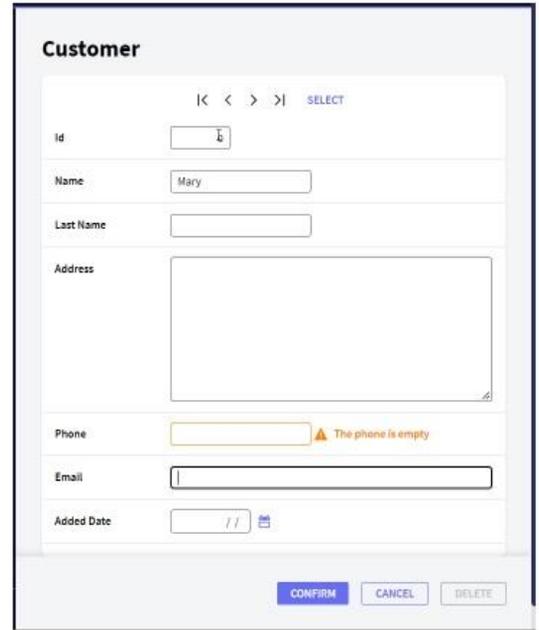
Name	Type	Description
Customer	Customer	Customer
CustomerId	Id	Customer Id
CustomerName	Name	Customer Name
CustomerLastName	Name	Customer Last Name
CustomerAddress	Address, GeneXus	Customer Address
CustomerPhone	Phone, GeneXus	Customer Phone
CustomerEMail	Email, GeneXus	Customer EMail
CustomerAddedDate	Date	Customer Added Date



```

1 Error('Enter the customer name')
2   if CustomerName.IsEmpty();
3
4 Msg('The phone is empty')
5   if CustomerPhone.IsEmpty();

```



**Customer**

Navigation: |< < > >| SELECT

Id:

Name:

Last Name:

Address:

Phone:  ⚠ The phone is empty

Email:

Added Date:

Buttons: CONFIRM CANCEL DELETE

Vimos también la regla **Message**, la cual únicamente informa al usuario mediante un mensaje pero permite efectuar la grabación;

# Rules

Customer X

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

Name	Type	Description
Customer	Customer	Customer
CustomerId	Id	Customer Id
CustomerName	Name	Customer Name
CustomerLastName	Name	Customer Last Name
CustomerAddress	Address, GeneXus	Customer Address
CustomerPhone	Phone, GeneXus	Customer Phone
CustomerEMail	Email, GeneXus	Customer EMail
CustomerAddedDate	Date	Customer Added Date

Customer X

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

```
1 Error('Enter the customer name')
2 | if CustomerName.IsEmpty();
3
4 Msg('The phone is empty')
5 | if CustomerPhone.IsEmpty();
6
7 Default(CustomerAddedDate, &Today);
```

Customer

|< > >| SELECT

Id

Name

Last Name

Address

Phone

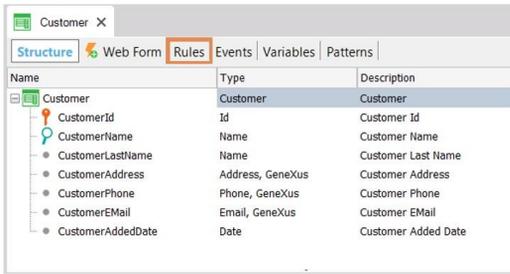
Email

Added Date  ←

CONFIRM CANCEL DELETE

la regla **Default**, que nos permite inicializar un atributo o variable con un valor cuando accedemos a la transacción en modo de inserción;

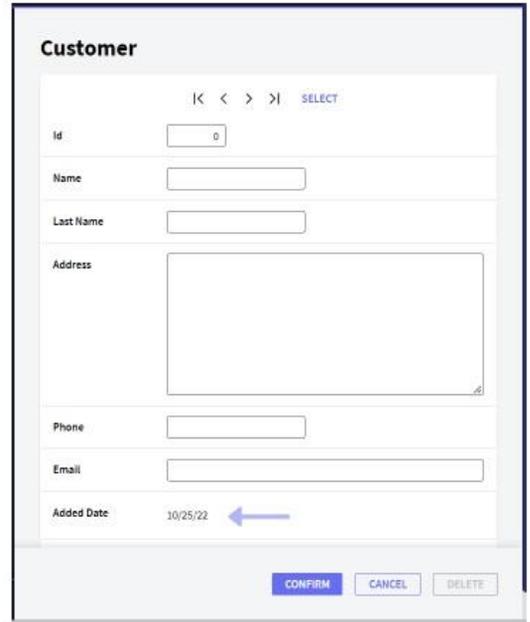
# Rules



Name	Type	Description
Customer	Customer	Customer
CustomerId	Id	Customer Id
CustomerName	Name	Customer Name
CustomerLastName	Name	Customer Last Name
CustomerAddress	Address, GeneXus	Customer Address
CustomerPhone	Phone, GeneXus	Customer Phone
CustomerEMail	Email, GeneXus	Customer EMail
CustomerAddedDate	Date	Customer Added Date



```
1 Error('Enter the customer name')
2   L   if CustomerName.IsEmpty();
3
4 Msg('The phone is empty')
5   L   if CustomerPhone.IsEmpty();
6
7 Default(CustomerAddedDate, &Today);
8
9 Noaccept(CustomerAddedDate);
```



**Customer**

Id:

Name:

Last Name:

Address:

Phone:

Email:

Added Date: 10/25/22

CONFIRM CANCEL DELETE

la regla **NoAccept**, que impide que el usuario pueda modificar un campo en el formulario: lo muestra deshabilitado,

# Rules

The screenshot shows the 'Rules' tab in the GeneXus IDE. A rule is defined as follows:

```
1 Serial(CityId, CountryLastLine, 1);  
2  
3  
4  
5
```

Below the code is a structure diagram for the 'Country' entity. It shows a tree view with the following nodes:

- Country
  - CountryId
  - CountryName
  - CountryLastLine
  - City
    - CityId
    - CityName

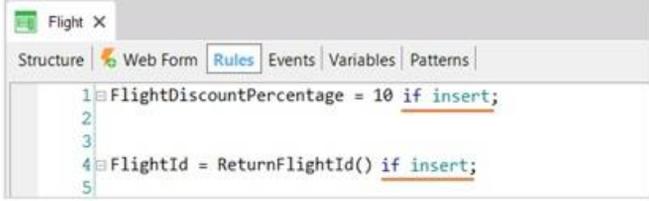
The screenshot shows the 'Country' web form. It has a 'Name' field with the value 'Brasil' and a 'Last Line' field with the value '1'. Below these is a table with the following data:

	Id	Name
<input type="checkbox"/>	1	San Paulo
<input type="checkbox"/>	0	Rio de Janeiro
<input type="checkbox"/>	0	

At the bottom of the table is a '+ [NEW ROW]' button. At the bottom of the form are 'CONFIRM' and 'CANCEL' buttons.

y por último la regla **Serial**, que sirve para autonumerar un segundo nivel, o tercero, u otro nivel anidado de una transacción.

## Rules



```
1 FlightDiscountPercentage = 10 if insert;  
2  
3  
4 FlightId = ReturnFlightId() if insert;  
5
```

*if insert*

*if update*

*if delete*

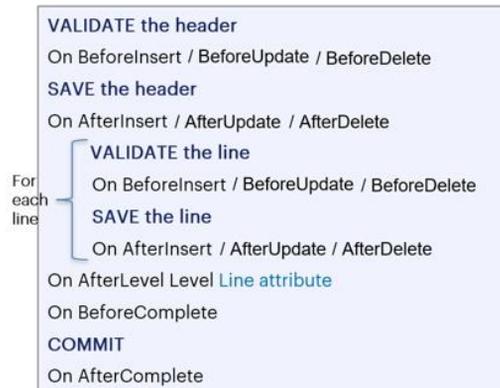
Estudiamos también que mediante las reglas podemos definir asignaciones de valores o invocar objetos, y que también podemos condicionarlas para que se ejecuten solo cuando se está insertando, modificando o eliminando.

## Rules

```

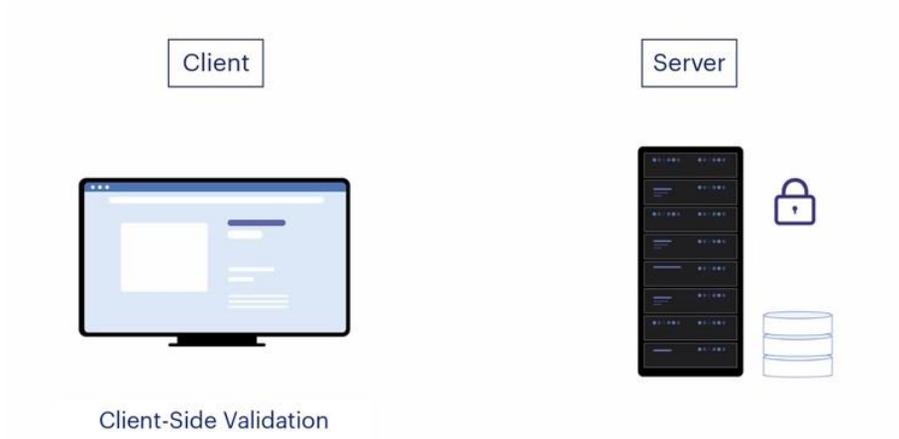
1 FlightDiscountPercentage = 10 if insert;
2
3
4 FlightId = ReturnFlightId() on BeforeInsert;
5

```



Además, si el momento elegido por GeneXus para ejecutar una regla no es el que necesitamos, podemos indicarle en qué momento preciso queremos que se ejecute, utilizando los eventos de disparo.

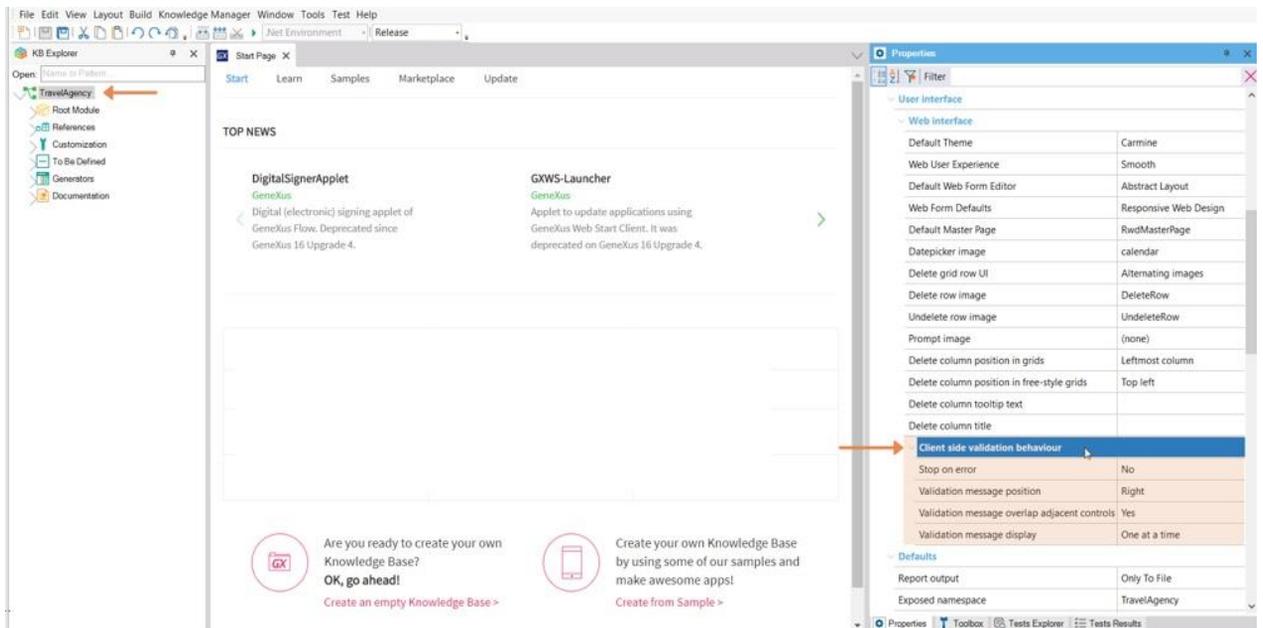
## Execution of rules



Recordemos que todas estas reglas que hemos estudiado, son validadas tanto en el cliente web como en el servidor.

La validación que ocurre en el cliente es llamada **Client-Side Validation**, y su objetivo es brindar una buena experiencia al usuario, haciéndole sentir que la aplicación está todo el tiempo interactuando con él; pero es el servidor quien realmente valida que toda la información enviada sea consistente y no viole la seguridad del sistema, y es el único que puede operar sobre la base de datos.

## Client side validation



Hasta aquí hemos visto el comportamiento por defecto de estas reglas. Pero es interesante saber que contamos con un grupo de propiedades llamado **Client Side Validation Behaviour** que encontramos a nivel de la versión, el cual nos permite personalizar el comportamiento y los mensajes de las reglas de muchas formas, permitiendo que la aplicación sea más atractiva para usuarios finales.

## Stop on error property

The image shows a 'Customer' form with several input fields: 'id' (0), 'Name' (empty), 'Last Name' (empty), 'Address' (text area), 'Phone' (empty), 'Email' (empty), and 'Added Date' (10/25/22). The 'Name' field has a red error message: 'Enter the customer name'. A configuration dialog titled 'Client side validation behaviour' is open, showing the 'Stop on error' property set to 'Yes'.

Client side validation behaviour	
Stop on error	Yes
Validation message position	Right
Validation message overlap adjacent controls	Yes
Validation message display	One at a time

Cuando utilizamos la regla Error, vemos que aunque la transacción no permite grabar, sí permite pasar al siguiente campo, luego de haber mostrado el mensaje de error. Si nos interesara que en caso de dispararse un error, se mantenga el foco en el control y se obligue al usuario a corregir el valor para poder pasar al siguiente campo, debemos cambiar el valor de la propiedad **Stop on error** para Yes.

## Validation message position property

The screenshot shows a 'Customer' form with several input fields: 'id' (containing '0'), 'Name' (with a red error message 'Enter the customer name'), 'Last Name', 'Address' (a large text area), 'Phone', 'Email', and 'Added Date' (10/25/22). At the bottom, there are 'CONFIRM', 'CANCEL', and 'DELETE' buttons. A configuration dialog titled 'Client side validation behaviour' is open, showing the following settings:

Client side validation behaviour	
Stop on error	No
Validation message position	Top
Validation message overlap adjacent controls	Yes
Validation message display	One at a time

También podremos definir el lugar donde va el mensaje con respecto al campo, utilizando la propiedad **Validation Message Position**, que cuenta con los valores Right (que es el valor por defecto), Left, Top y Bottom.

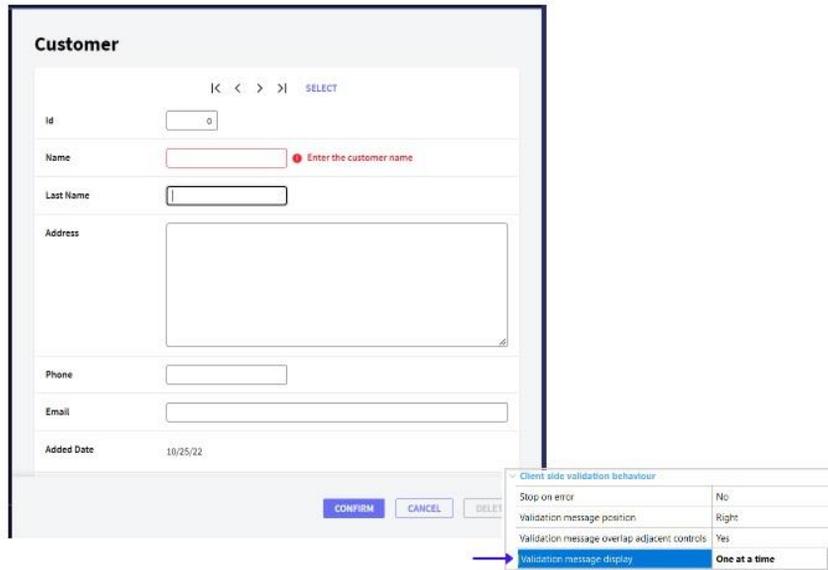
## Validation message overlap adjacent property

The screenshot shows a form titled "Customer" with the following fields: Id (0), Name (with a red error message "Enter the customer name"), Last Name, Address, Phone, Email, and Added Date (10/25/22). At the bottom, there are buttons for CONFIRM, CANCEL, and DELETE. A configuration dialog titled "Client side validation behaviour" is open, showing the following settings:

Client side validation behaviour	
Stop on error	No
Validation message position	Bottom
Validation message overlap adjacent controls	Yes
Validation message display	Yes
	No

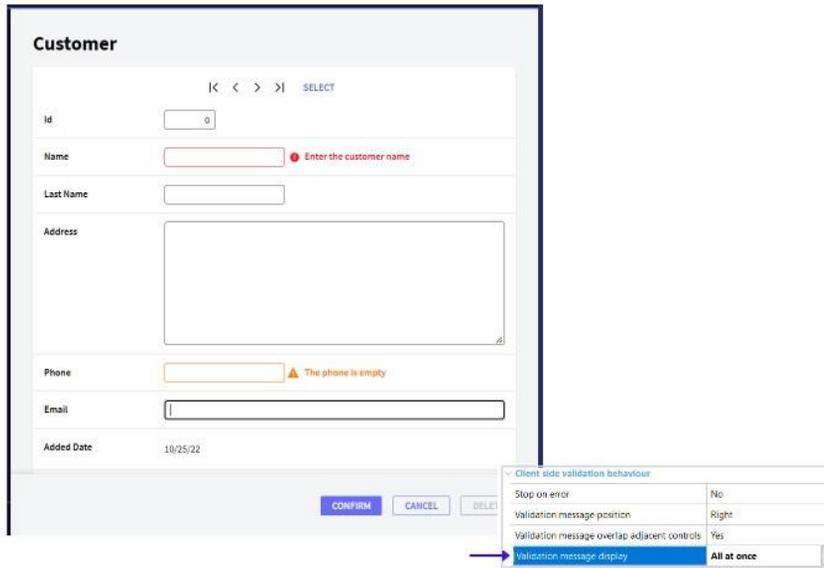
En caso de que el mensaje se muestre solapado a algún otro control, también debe configurarse la propiedad **Validation message overlap adjacent controls**, que controla si los mensajes deben superponerse o no.

## Validation message display property



La última propiedad del grupo es **Validation message display**, que cuenta con los valores One at a time (que es el valor por defecto) y cuyo objetivo es mostrar siempre el último mensaje de validación que activa la aplicación,

## Validation message display property



y All at once, que muestra al mismo tiempo cada mensaje de validación que debería estar en la pantalla.

Como estas propiedades modifican el comportamiento relativo a la interacción en los forms, se deben volver a generar todos los objetos que tienen form para que los cambios sean efectuados. Para ello, debe utilizar la opción **Rebuild All**, que genera absolutamente todos los objetos.

En el siguiente video agregaremos algunas reglas interesantes a las ya conocidas y en otros estudiaremos cómo se evalúan para determinar el orden de ejecución, así como profundizaremos un poco más en los eventos de disparo.

*GeneXus*<sup>TM</sup>

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