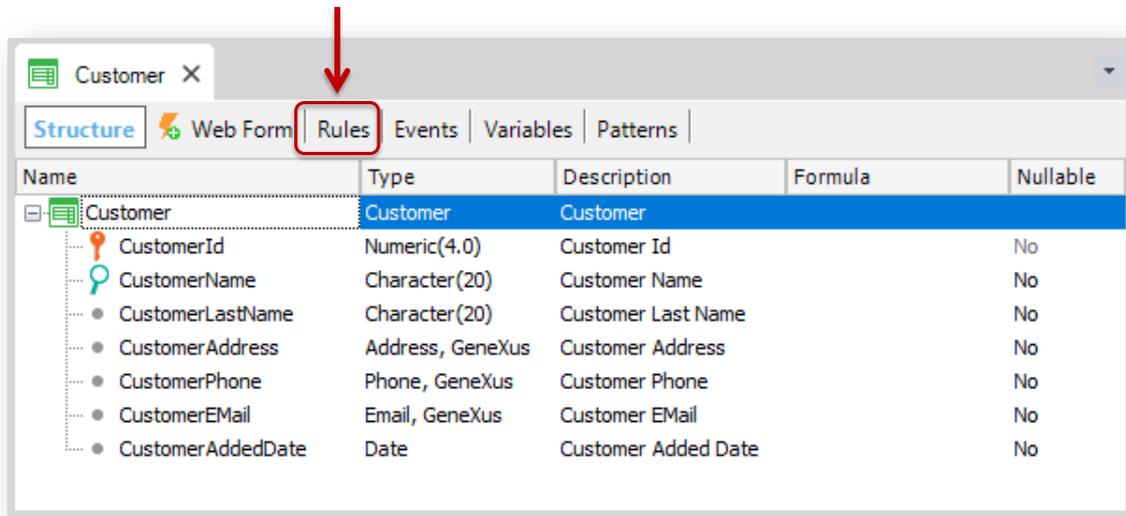


Definição de regras

*GeneXus*<sup>™</sup>

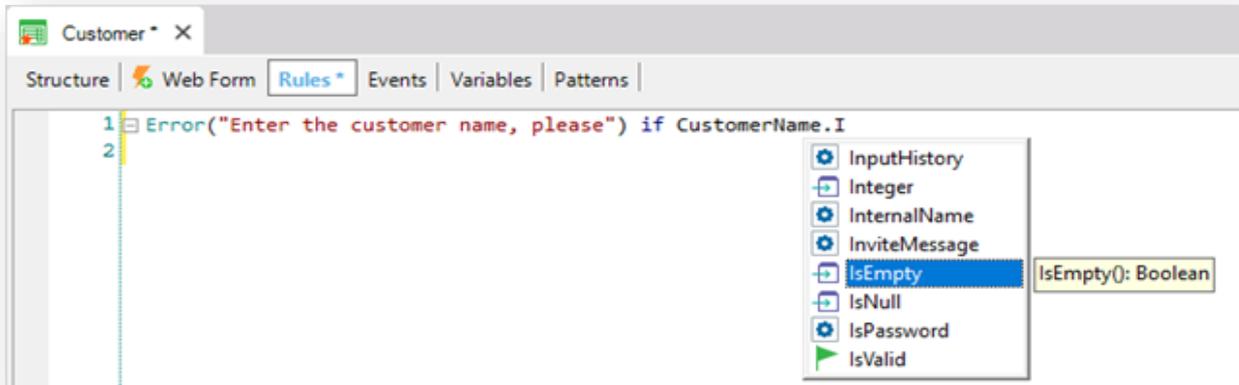
Alguns controles na entrada do cliente



Além de todos os controles automáticos que GeneXus inclui nas aplicações que gera, existem comportamentos específicos na programação que os analistas podem inserir.

Nas transações, as regras de negócio que definem seu comportamento são definidas na seção de Rules.

## Alguns controles na entrada do cliente



Por exemplo, se temos que definir um requisito do tipo : não permitir armazenar clientes sem nome... temos uma regra chamada Error que permitirá implementar esse controle.

Escrevemos "Error", abre parêntese, e entre aspas digitamos o texto que queremos visualizar quando o usuário tente deixar um nome de cliente sem preenchimento... fechamos o parêntese... e depois somente incluímos qual é a condição a ser cumprida para que a mensagem seja exibida.

A condição a ser cumprida é que o atributo CustomerName esteja vazio... então escrevemos "if CustomerName", ponto, e selecionamos o método: IsEmpty

Toda regra que programamos deve ser finalizada com um ponto-e-vírgula. Fazemos isso em nossa regra Error. Salvamos... e pressionamos F5 para ver em execução o funcionamento da regra definida.

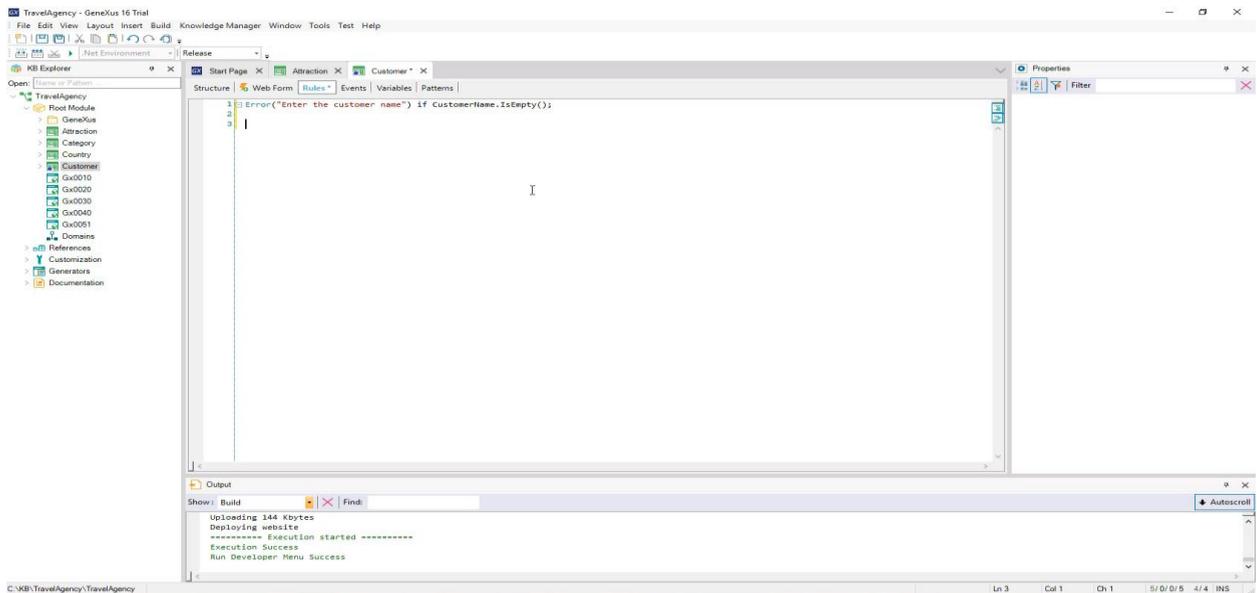
## Regra Error

The screenshot shows a web application interface. At the top, there is a red header bar with the text 'Application Name' on the left and 'by GeneXus' on the right. Below the header, there are two tabs: 'Recents' and 'Customer'. The 'Customer' tab is active, and the form is titled 'Customer'. The form contains several fields: 'Id' with a value of '0', 'Name' (which is empty and has a red border and a yellow tooltip that says 'Enter the customer name, please'), 'Last Name' (which is empty), and 'Address' (a large text area). At the top of the form, there are navigation arrows and a 'SELECT' button.

Executamos a transação Customer. Se deixarmos o nome do cliente vazio e sairmos do campo, aparecerá o texto que definimos.

A regra Error, mesmo permitindo passar para o próximo campo da tela quando a condição é válida, não permitirá gravar o registro ... [teste isso em execução] Esse é o comportamento padrão. Podemos configurar para que a regra Error nunca permita passar para o próximo campo. Isso é possível através de uma propriedade que não veremos aqui.

## DEMO

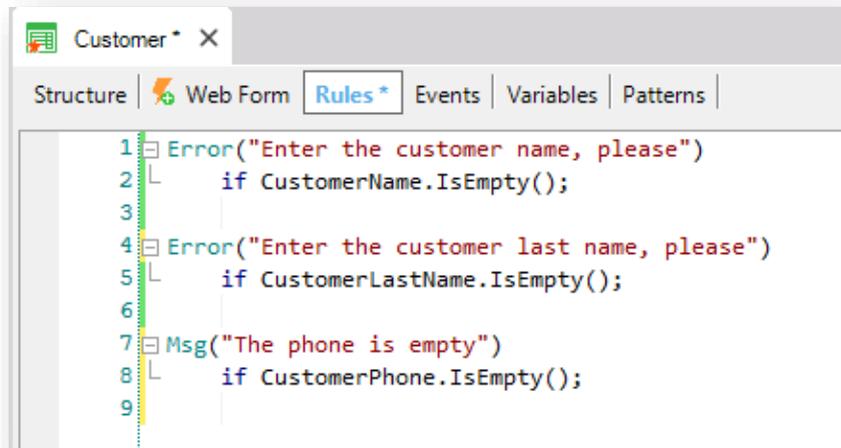


[ DEMO: <https://youtu.be/dhXcPuqxEPs> ]

Se for um requisito de negócio impedir que o sobrenome do cliente fique vazio, temos que definir uma regra análoga. Assim, basta copiarmos a regra Error existente... alterarmos o texto name por lastname e o atributo envolvido também:

Pressionamos F5... executamos a transação Customer... deixamos o nome do cliente vazio... é disparado o erro associado ao nome que ficou vazio... digitamos Paul... tentamos deixar o sobrenome vazio... e é disparado o erro associado ao sobrenome que ficou em branco.

## Regra Message



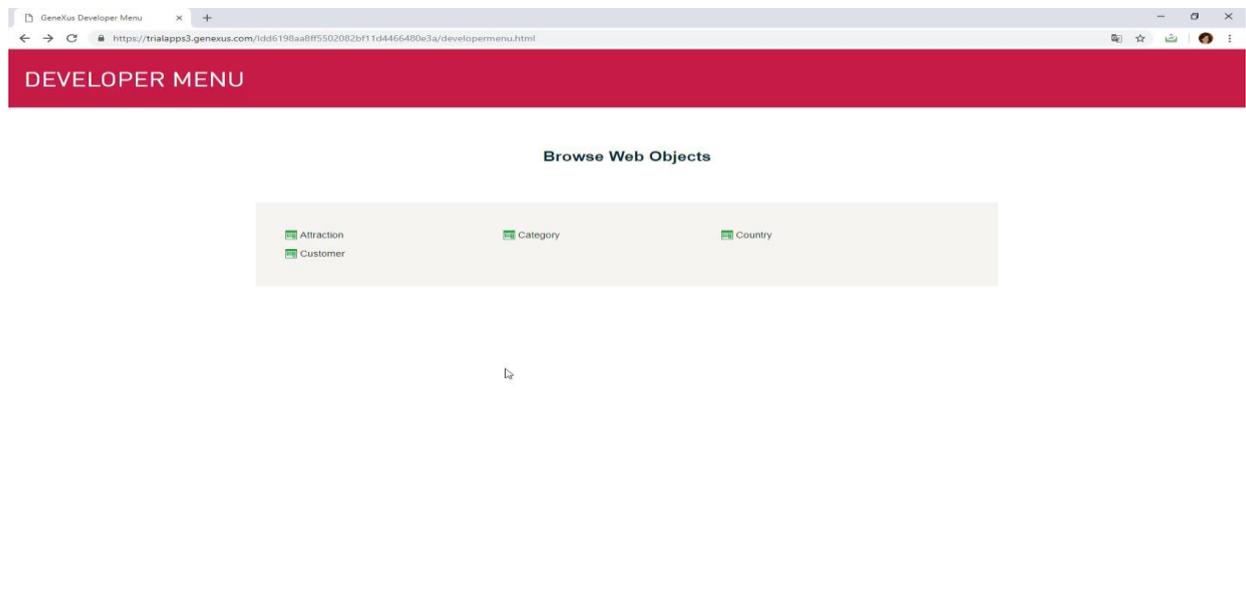
```
1 Error("Enter the customer name, please")
2   if CustomerName.IsEmpty();
3
4 Error("Enter the customer last name, please")
5   if CustomerLastName.IsEmpty();
6
7 Msg("The phone is empty")
8   if CustomerPhone.IsEmpty();
9
```

Agora... contamos com outra regra que tem praticamente a mesma sintaxe que o Error... seu nome é Msg... e a única diferença com relação ao Error é que ao ser disparada, essa regra exibirá um aviso ou advertência e o registro poderá ser gravado na tabela.

Se queremos, por exemplo, avisar que o telefone do cliente ficou em branco, porém sem obrigação de informá-lo, podemos definir uma regra Msg, abrir parêntese, e definir o texto entre aspas simples (ou dupla)... 'The phone is empty'.

Fechamos o parêntese... e definimos a condição para a regra ser executada: "if CustomerPhone" ... ponto ... IsEmpty. E terminamos com ponto-e-vírgula a definição da regra.

## DEMO



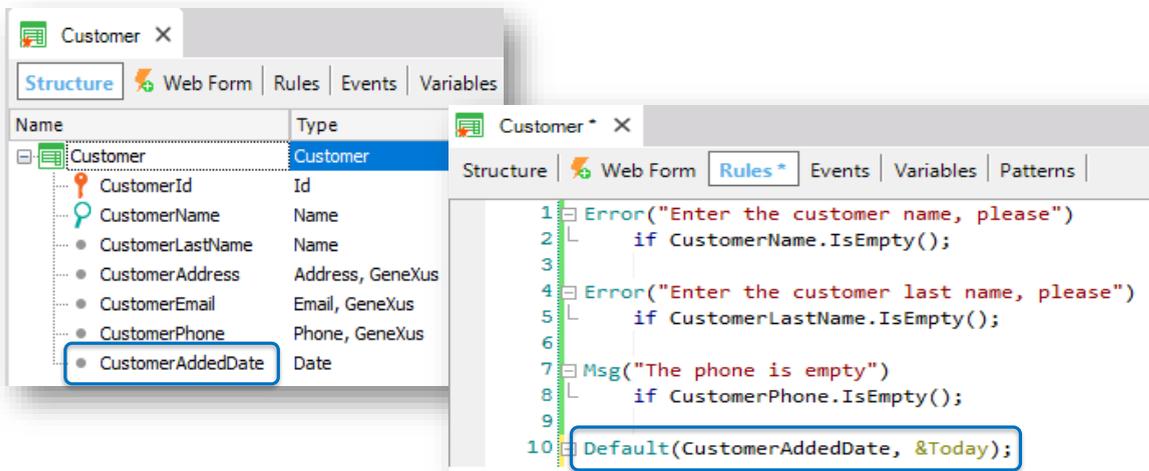
[ DEMO: <https://youtu.be/iN5j24xKqNo> ]

Pressionamos F5 para testar essa funcionalidade...

Podemos ver que se deixarmos em branco o telefone e tentarmos sair do campo, a mensagem que definimos é exibida, e se agora queremos gravar o registro, conseguiremos.

## Regra Default

- “Para cada cliente armazenar a data em que o cliente foi inserido no sistema” (atribuir valor predeterminado).



Agora... suponhamos que os usuários da agência de viagem requisitaram que interesse armazenar para cada clientes a data de cadastro.

Precisamos criar um novo atributo na transação Customer para armazenar a data. Definimos o CustomerAddedDate ... do tipo Date...

e com isso falta apenas atribuímos automaticamente a data do dia.

Vamos na seção Rules...lá contamos com uma regra chamada Default.

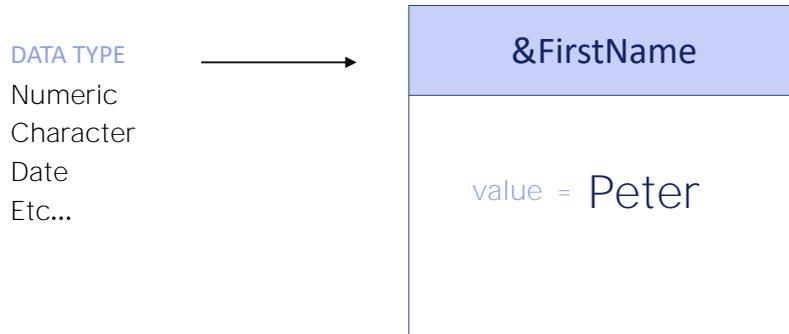
Essa regra permite inicializar um atributo ou variável com um valor:

Dessa maneira, foi inserida a sintaxe da regra Default, e agora vamos substituir dentro dos parênteses o atributo que queremos inicializar, que é o CustomerAddedDate e o valor com o qual queremos inicializar, que é a data de hoje.

“&Today” é uma variável padrão existente, que sempre tem carregada a data do dia para ser usada.

## Variáveis

- Valor temporariamente armazenado na memória com um nome associado e tipo de dados. Para fazer referência a uma variável, o símbolo "&" deve ser usado.



Exemplo: A variável &FirstName armazena na memória o valor "John"

Uma variável consiste em um espaço de memória, possui um nome simbólico, e um tipo de dados que pode armazenar (texto, números, datas, etc.). Essa variável tem um determinado valor armazenado (guardado). O nome da variável é a forma usual de referir-se ao valor armazenado.

## Atributos

- Valor fisicamente armazenado no banco de dados

CUSTOMER TABLE

CustomerId	CustomerFirstName	CustomerLastName
1	Peter	Smith
2	Susan	Parker

Em conra-partida um atributo é um valor armazenado fisicamente na base de dados.

## Definição de variáveis

Variáveis só podem ser definidas dentro de cada objeto onde elas são usadas.

### 1) Usando o seletor Variables

The screenshot shows the 'Variables' tab in the GeneXus IDE. It displays a table with columns: Name, Type, Is Collection, and Description. The table is divided into 'Standard Variables' and 'MyDate'.

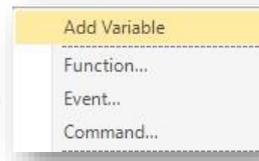
Name	Type	Is Collection	Description
<b>Standard Variables</b>			
GxRemove	Numeric(1,0)	<input type="checkbox"/>	Gx Remove
Mode	Character(3)	<input type="checkbox"/>	Mode
PgmDESC	Character(256)	<input type="checkbox"/>	PgmDESC
PgmNAME	Character(128)	<input type="checkbox"/>	PgmNAME
Time	Character(8)	<input type="checkbox"/>	Time
Today	Date	<input type="checkbox"/>	Today
<b>MyDate</b>			
MyDate	Date	<input checked="" type="checkbox"/>	My Date

Arrows point from the 'Standard Variables' section to the text 'Variáveis do sistema predefinidas' and from the 'MyDate' row to 'Variáveis definidas pelo desenvolvedor'.

### 2) Definindo-a ao declará-la:

```
Event Start
  &FromDate
Endevent
```

Botão direito



A maioria dos objetos GeneXus permitem definir variáveis. Essas variáveis são locais, o que significa que somente podem ser utilizadas dentro do respectivo objeto. Para referenciar uma variável devemos utilizar o símbolo "&". Por exemplo: &Total.

Se posicionamos, por exemplo, dentro da aba Variables dentro de uma transação, veremos que já existe um conjunto de variáveis definidas. São variáveis do sistema, como, por exemplo: &Today, &Mode, etc. Em particular, a variável &Today permite obter a data atual do sistema.

Além dessas variáveis do sistema, o desenvolvedor também pode definir suas próprias variáveis (variáveis do usuário). Por exemplo, uma variável myDate do tipo Date:

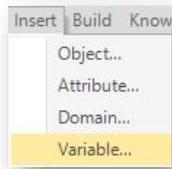
1) Defini-lás através do seletor Variables existente em todo objeto GeneXus, como acabamos de fazer.

2) Defini-lá no momento de usar, estando em qualquer parte do código. Por exemplo: escrever o símbolo "&" que identifica uma variável e definir um nome, depois posicionar o mouse sobre o nome da variável e pressionar o botão direito do mouse para escolher no menu contextual a opção: Add Variable.

Com isso, fica editável as propriedades da variável. Podemos atribuir seu tipo de dados. Se acessarmos o seletor de Variables, a variável estará lá.

## Definição de variáveis

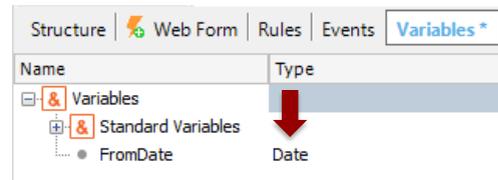
3) Insert \ Variable:



4) Automático, de acordo com o padrão de nome:

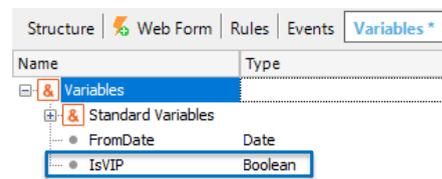
a) Domain : Name

**&FromDate** - É automaticamente definido como Date.



b) &IsXxx | &HasXxx

→ São automaticamente definidos como Boolean



3) A terceira opção é selecionar a opção Insert do menu e depois Variable... e New Variable...

No exemplo mostrado, através da regra Default, estamos atribuindo por padrão a data atual no atributo CustomerAddedDate.

Vamos salvar a transação... e pressionar F5.

## Análise de impacto

The screenshot shows the 'Impact Analysis' window for a 'Customer' table. The main heading is 'Database needs to be reorganized.' Below this, a message states: 'This report describes Database changes and how they will be handled by reorganization programs. Please select Reorganize to proceed or Cancel.' There are 'Reorganize' and 'Cancel' buttons.

The 'Warnings' section contains a warning icon and the text: 'rgz0007 Attribute CustomerAddedDate does not allow nulls and does'. Below this is the 'Table Structure' section, which lists attributes and their definitions:

Attribute	Definition
<a href="#">CustomerId</a>	Numeric (4), Not null
<a href="#">CustomerName</a>	Character (20), Not null
<a href="#">CustomerLastName</a>	Character (20), Not null
<a href="#">CustomerAddress</a>	Varchar (1024), Not null
<a href="#">CustomerEmail</a>	Varchar (100), Not null
<a href="#">CustomerPhone</a>	Character (20), Not null
New <a href="#">CustomerAddedDate</a>	Date, Not null

The 'Indexes' section shows a table with the following data:

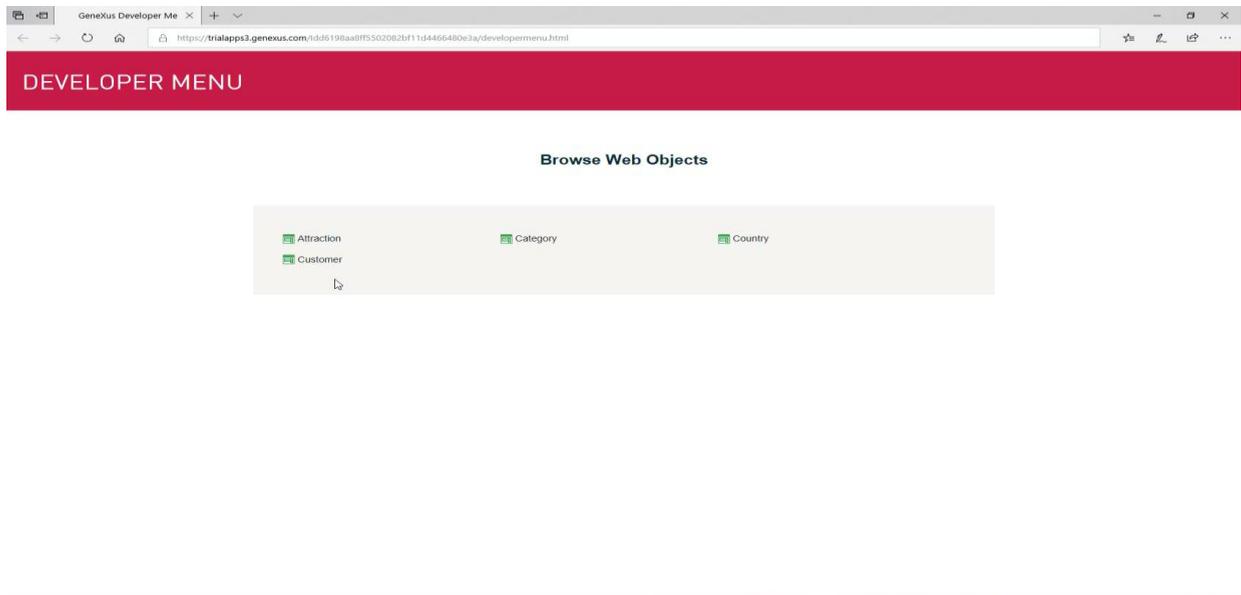
Name	Definition
ICUSTOMER	primary key Clustered

The 'Statements' section is currently empty.

Somos avisados que um novo atributo será adicionado CustomerAddedDate na tabela CUSTOMER:

Procedemos com a reorganização da tabela...

## DEMO



[ DEMO: <https://youtu.be/jqlcfWNDhOk> ]

E novamente estamos com a aplicação pronta para ser executada.

Acessamos a transação Customer...

E já podemos perceber o novo atributos “data do cadastro” inicializado com a data de hoje.

Se não tivéssemos programado a regra Default, o campo da data apareceria vazio como os demais campos.

Cadastramos um cliente, Robert... Hill... que mora na rua 81... seu telefone é 760 5100 e seu e-mail é [rhill@example.com](mailto:rhill@example.com) .  
E observamos que a data de hoje já é sugerida, mas podemos alterá-la.

Mais validações são necessárias ...

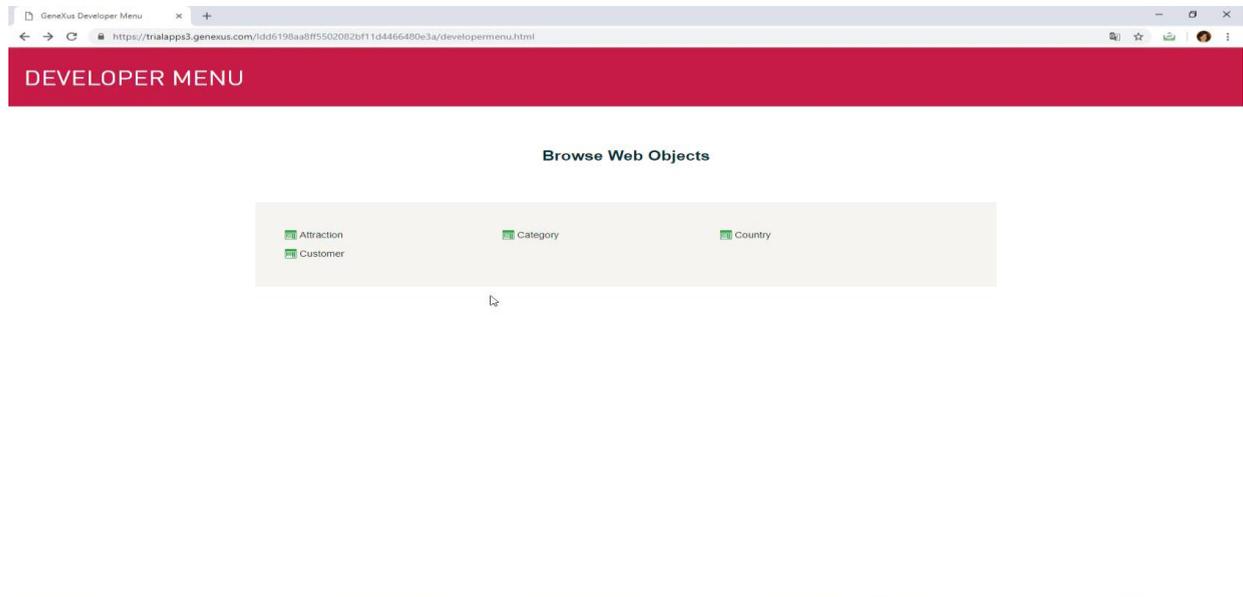
- deixe o campo de data editável + controle que “nenhuma data futura pode ser inserida”.

```
1 Error("Enter the customer name, please")
2   if CustomerName.IsEmpty();
3
4 Error("Enter the customer last name, please")
5   if CustomerLastName.IsEmpty();
6
7 Msg("The phone is empty")
8   if CustomerPhone.IsEmpty();
9
10 Default(CustomerAddedDate, &Today);
11
12 Error("The date must be lower or equal than today")
13   if CustomerAddedDate > &Today;
14
```

Se para os usuários da agência de viagens for interessante deixar a data editável, mas que seja controlado que não possa ser cadastrada datas futuras... poderíamos definir uma regra Error.

Abrimos parêntese, digitamos 'The date must be lower or equal than today', fechamos o parêntese... e colocamos a condição `if CustomerAddedDate > &today;`

## DEMO



[ DEMO: [https://youtu.be/EPiudriAo\\_M](https://youtu.be/EPiudriAo_M) ]

Vamos testar isso em execução, pressionamos F5...

Cadastramos Alex... Johnson...

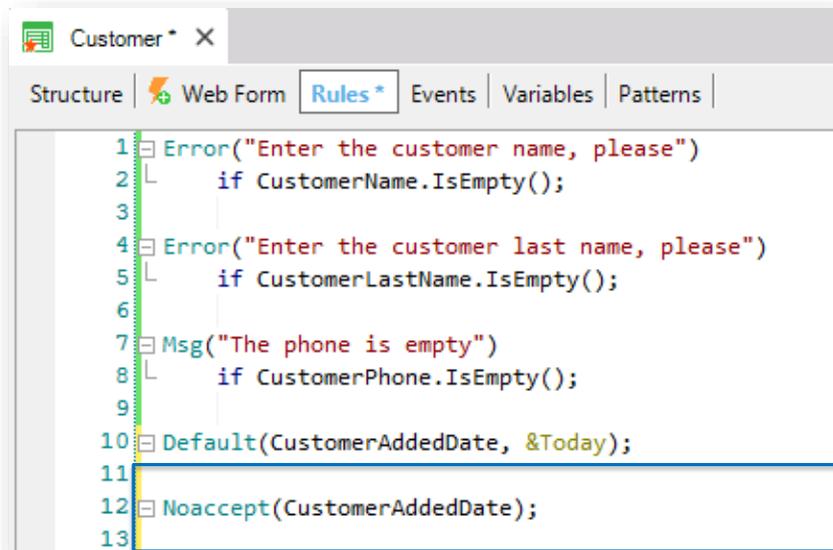
E se tentarmos colocar uma data maior que a data de hoje...

a condição que definimos é disparada, exibindo o erro associado.

Agora suponhamos que a agência de viagens informe que a data de cadastro do cliente não pode ser editada, que deve ficar desabilitada no formulário e ser gravada com a data fornecida pelo sistema...

Para realizar esse requisito, eliminamos essa regra, porque já não terá mais sentido.

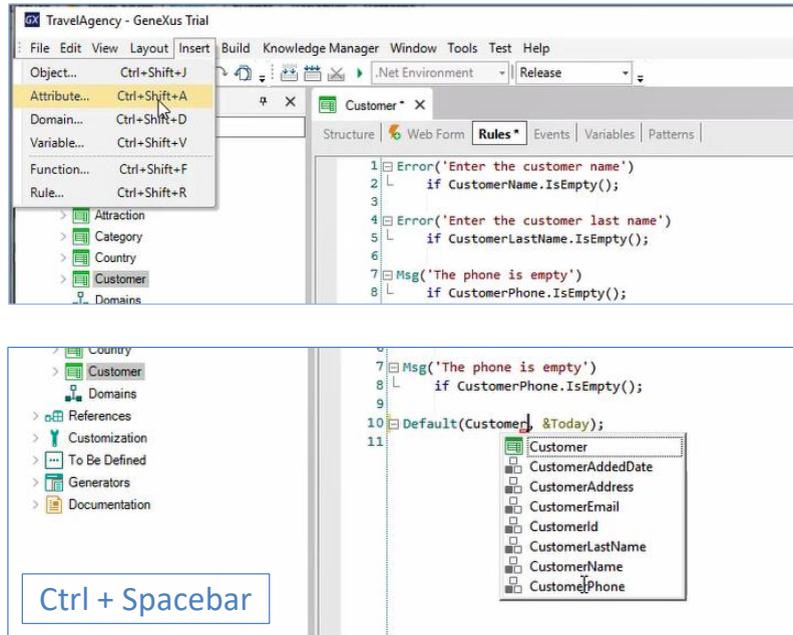
## Desativando um campo/atributo



```
1 Error("Enter the customer name, please")
2   |   if CustomerName.IsEmpty();
3
4 Error("Enter the customer last name, please")
5   |   if CustomerLastName.IsEmpty();
6
7 Msg("The phone is empty")
8   |   if CustomerPhone.IsEmpty();
9
10 Default(CustomerAddedDate, &Today);
11
12 Noaccept(CustomerAddedDate);
13
```

E teríamos que programar uma regra... Noaccept

Substituímos dentro do parênteses o texto que diz "attribute or variable" pelo atributo CustomerAddedDate e apagamos "if condition", porque queremos que a regra sempre seja executada.



Até agora, escrevemos de memória os nomes dos atributos e isto pode levar a erros. Para evitá-los, uma das opções que temos é inserir os atributos a partir do menu de opções.

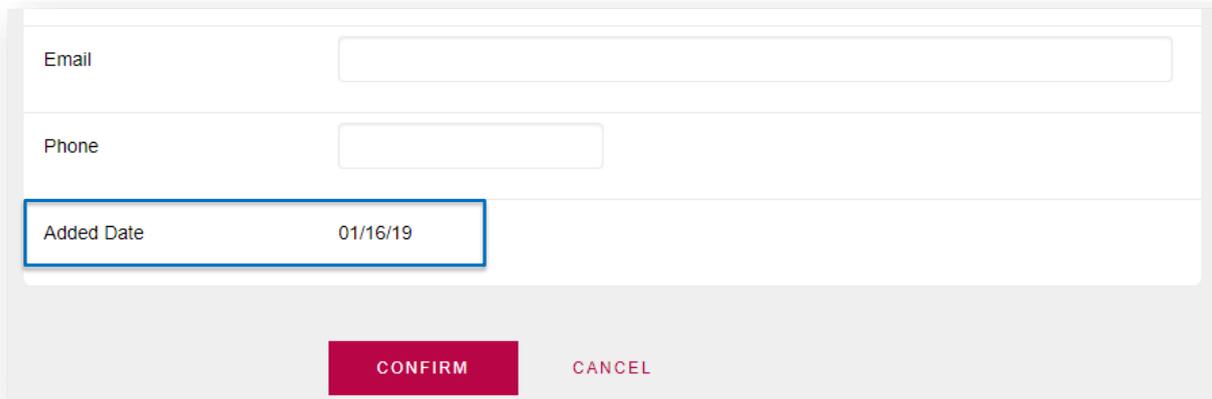
Para isto, selecionamos “Insert” e depois “Attribute”, ou poderíamos fazê-lo usando o atalho ctrl+Shift+A.

Acessaremos uma lista de todos os atributos declarados em nossa KB, onde podemos filtrar por nome e inserir o atributo escolhido.

A outra opção é começar a escrever o nome do atributo, por exemplo, escrevemos “Customer” e pressionamos ctrl e espaço. Por meio deste atalho, serão mostrados todos os objetos de nossa KB que iniciam com esse nome, e clicando no atributo que necessitamos, é inserido. Estas formas de selecionar o atributo em vez de digitá-lo nos permitem economizar tempo e evitar erros de digitação.

Vamos voltar.

## Desativando um campo/atributo



Email	<input type="text"/>
Phone	<input type="text"/>
Added Date	01/16/19

**CONFIRM**    CANCEL

Testamos o comportamento agora...

E podemos ver que a data aparece inicializada pela regra Default e desabilitada pela regra NoAccept. Muito bem...

Aprendemos que para inicializar o atributo CustomerAddedDate com a data do dia devemos programar a regra Default:

É importante saber que toda regra Default que programamos será executada somente quando estivermos inserindo registros.

Ou seja, se consultarmos um cliente que já está cadastrado, a regra Default não será executada... já que o cliente possui data de cadastro... e a regra Default não pode sobrescrever o valor.

## Regras de Atribuição

```
CustomerAddedDate = &Today
```

Sempre será executada, independentemente de se o usuário estiver inserindo ou atualizando dados.

```
CustomerAddedDate = &Today  
if Insert;
```

Só será executada quando um novo registro estiver sendo inserido. Esse comportamento é igual à regra Default

Agora suponhamos que no lugar de programarmos a regra Default tivéssemos definido uma regra de atribuição:

```
CustomerAddedDate = &Today;
```

Com essa definição de regra, o atributo CustomerAddedDate seria sempre atribuído com a data do dia. Essa é uma regra de atribuição, e será executada sempre, independentemente do usuário inserir, atualizar, etc.

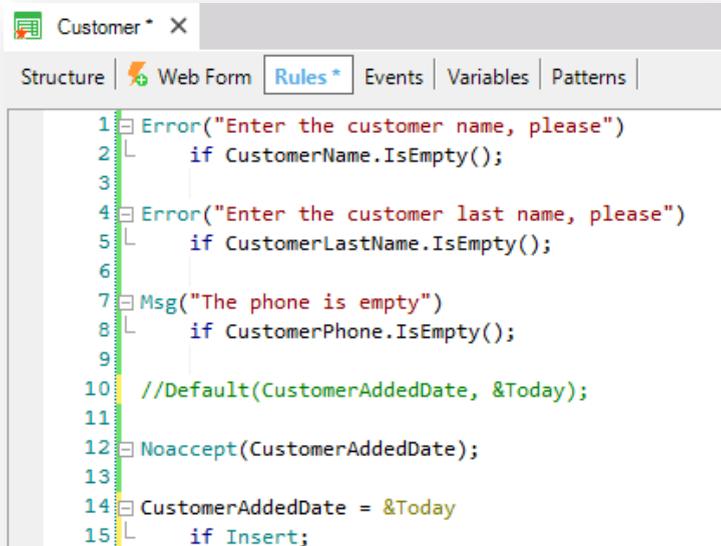
Podemos condicionar uma regra de atribuição para que seja executada somente quando o usuário está efetuando determinada operação na base de dados, como inserção, atualização ou eliminação de registro.

Vamos condicionar a regra. Adicionamos if insert :

O comportamento dessa regra definida será equivalente ao que já implementa a regra Default, já que agora estamos condicionando que a atribuição seja realizada somente se estivermos inserindo um registro. Exatamente o que faz a regra Default.

Da mesma forma como podemos condicionar uma regra com if insert, contamos com a possibilidade de definir if update ou if delete também.

## Ordem de execução das regras



```
1 Error("Enter the customer name, please")
2   | if CustomerName.IsEmpty();
3
4 Error("Enter the customer last name, please")
5   | if CustomerLastName.IsEmpty();
6
7 Msg("The phone is empty")
8   | if CustomerPhone.IsEmpty();
9
10 //Default(CustomerAddedDate, &Today);
11
12 Noaccept(CustomerAddedDate);
13
14 CustomerAddedDate = &Today
15   | if Insert;
```

A ordem em que as regras são definidas não corresponde necessariamente à ordem em que serão executadas.

O que é importante observar e saber, é que a ordem que definimos as regras não corresponde necessariamente com a ordem em que serão executadas.

Esse conjunto de regras poderia estar definido em qualquer outra ordem e o resultado em execução seria exatamente o mesmo, já que GeneXus decide em qual momento deverá ser disparada cada uma das regras definidas.

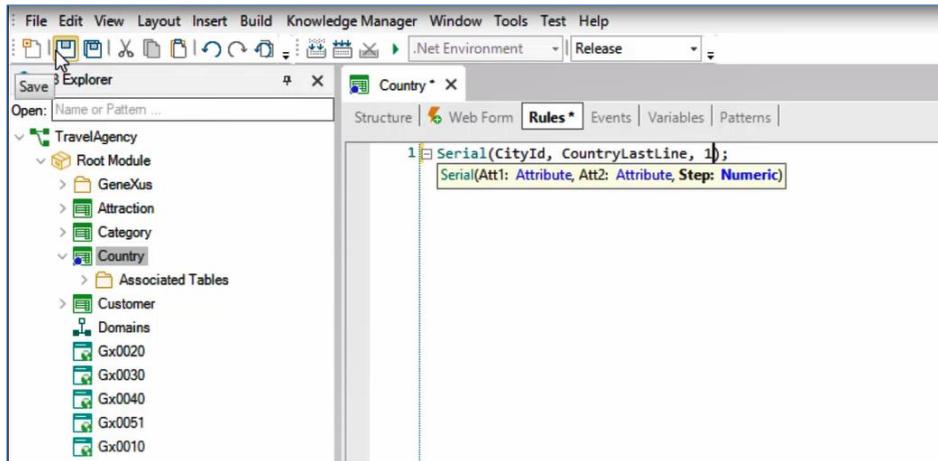
Para finalizar, lembremos que para cada transação teremos que programar suas próprias regras e comportamentos.

Nesse caso, programamos regras na transação de clientes para controlar suas particularidades requisitadas para quando os usuários interagirem com os dados do cliente. Muito provavelmente a agência pretenda controlar certas regras ou comportamentos para a transação de atrações também...

ou para outras transações. E para isso, cada transação conta com sua própria seção de regras.

Para finalizar, vamos enviar as alterações para o GeneXus Server.

## Serial Rule



Vejamos agora brevemente uma situação que nos ocorre na transação Country.

Em vídeos anteriores, quando criamos o segundo nível de nome City nesta transação, GeneXus gerou a tabela CountryCity na base de dados. Como vimos, a chave primária desta tabela é composta por dois atributos, CountryId e CityId. Lembremos que tanto ao atributo CountryId como a CityId atribuímos o domínio Id, que definimos como autonumerado. Ao criar esta tabela, GeneXus disparou um Warning indicando que a propriedade autonumber com valor True do atributo CityId seria ignorada.

Por que isto acontece?

Porque a propriedade autonumber só se aplica a chaves primárias simples, ou seja, àquelas formadas por um único atributo, não às chaves primárias compostas, como neste caso, formadas por dois atributos: CountryId e CityId.

Para numerar automaticamente um segundo nível, utilizaremos a regra Serial.

Para isto, vamos à seção rules da transação Country e inserimos a regra Serial.

Como primeiro parâmetro, nos pedirá para inserir um atributo, que será aquele que queremos numerar automaticamente. Então, inserimos CityId.

Como segundo parâmetro, nos pedirá a inserção de outro atributo, que será o encarregado de guardar o último valor atribuído ao atributo que queremos autonumerar, no nosso caso CityId. Para isto, criaremos um novo atributo, que deverá estar no nível acima daquele que queremos numerar.

Voltamos à estrutura da transação, e o criamos no primeiro nível dela, o chamaremos de CountryLastLine e será do tipo Numeric.

Para que nos permita salvar as alterações, comentaremos parcialmente o código inserido na seção Rules, pois estando ainda incompleto, não nos deixa salvar. Agora sim, salvamos, descomentamos a regra e inserimos como segundo parâmetro o atributo CountryLastLine.

Por último, esta regra nos pede para inserir um valor numérico, que corresponderá ao incremento automático da numeração. Inserimos o valor "1", desta forma o aumento da numeração de CityId será de um em um.

Salvamos as alterações que realizamos, e executamos a aplicação.

Nos pede para reorganizar a base de dados para adicionar o atributo CountryLastLine na tabela Country e confirmamos

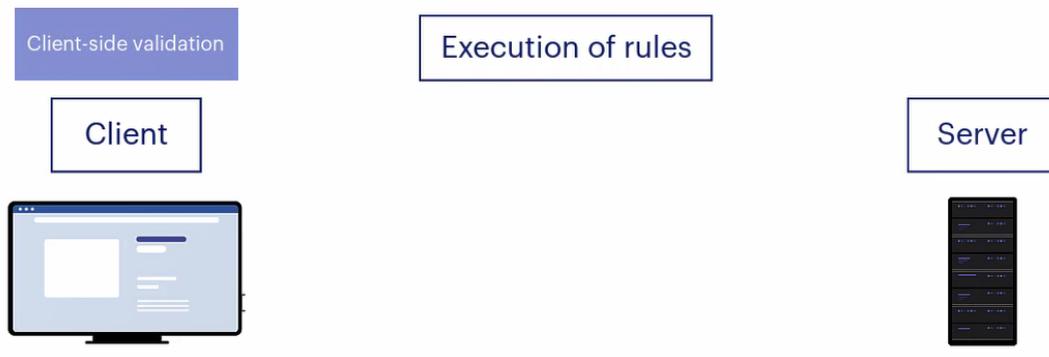
Acessamos a transação Country.

Inserimos o país Alemanha. E em cidade não inseriremos um valor no id, apenas inseriremos o seu nome, neste caso Berlin.

Quando nos posicionamos no Id da cidade, damos TAB e ao sair do campo, vemos que a esta cidade foi atribuído para CityId o valor 1. E o atributo CountryLastLine ficou com esse mesmo valor. Lembremos que este atributo é o encarregado de guardar o último valor inserido em CityId.

Vamos tentar inserir outra cidade, faremos isso com Hamburgo. Novamente vemos que ao sair do campo CityId com tab sem escrever nada, foi atribuído automaticamente um valor ao id, neste caso "2", e o atributo CountryLastLine novamente assumiu este último valor.

Se inseríssemos uma terceira cidade, com base na regra Serial que criamos, GeneXus tomará o valor do atributo CountryLastLine, acrescentará 1, pois é o valor de incremento que definimos, e o valor resultante o atribuirá a CityId e a CountryLastLine e assim por diante."



Se observamos o comportamento das regras que vimos até agora, poderíamos chegar a pensar que são executadas exclusivamente no cliente, pois, como vimos por exemplo com a de erro no nome do cliente, assim que o usuário sai do campo, deixando-o vazio, a mensagem é disparada. Porém, o usuário conseguiu continuar preenchendo os demais campos, e ao pressionar Confirm, lá quem assume o controle é a parte do programa associado à transação que executa no servidor e aí a regra volta a ser disparada, evitando, neste caso, por tratar-se justamente da regra Error, que o registro seja gravado na tabela da base de dados.

Portanto, as regras que vimos são validadas tanto no cliente (chamamos isto de Client side validation) quanto no servidor. Poderia ser desabilitada esta execução de regras no cliente, mas de forma alguma poderemos desabilitar a execução no servidor, que é quem realmente tem o controle do que será feito na base de dados. Vamos pensar que o browser é um meio hostil, poderia receber ataques que enviaram informações adulteradas ao Server. O Servidor é quem deve ter o controle definitivo sobre os dados.

As regras no cliente são executadas para proporcionar uma boa experiência ao usuário, que sinta que a aplicação está interagindo com ele o tempo todo.

Portanto, as mesmas regras estão incorporadas tanto no programa em execução no frontend (a parte que mantém o estado e a business logic, como representante do servidor no cliente) quanto no próprio programa no backend, ou seja, na validação da business logic.

Entender isto será importante mais tarde, quando tornarmos um pouco mais complexo o que podemos controlar a partir das regras.

Para terminar, vamos salvar as alterações no GeneXus server.

Até agora, vimos que:

- As regras permitem programar o comportamento da transação.
- As regras são acionadas dentro do escopo da transação em que foram declaradas.
- GeneXus determina a ordem em que as regras são acionadas, com certa independência da ordem em que foram escritas.
- As regras podem ser condicionadas, entre outras coisas, pelos modos de operação da transação (Insert, Update, Delete).

# GeneXus™

**The power of doing.**

Videos

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

Documentation

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

Certifications

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