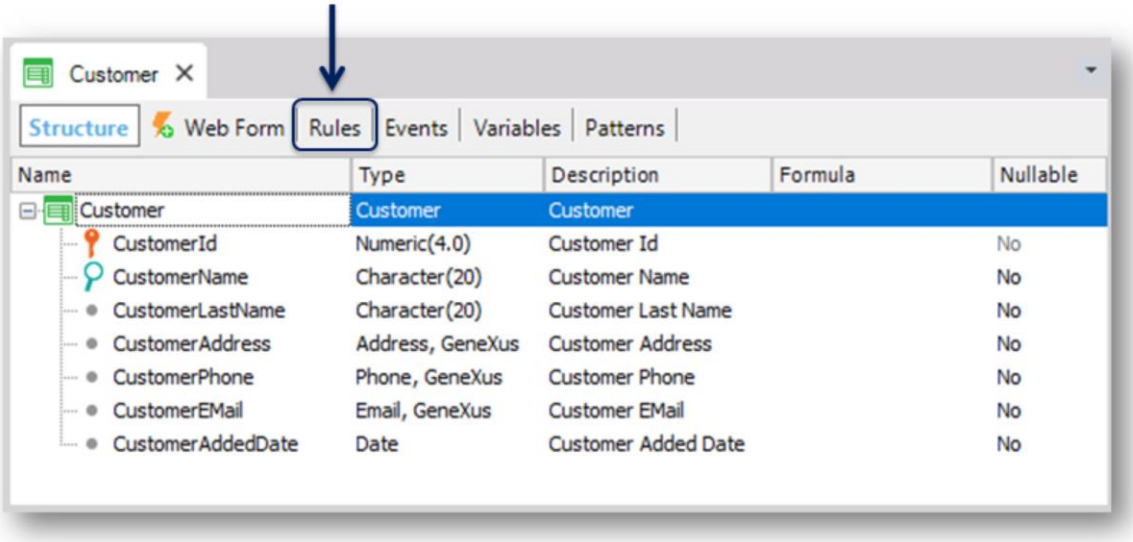
Defining controls requested by users

Rules in Transactions

GeneXus 16

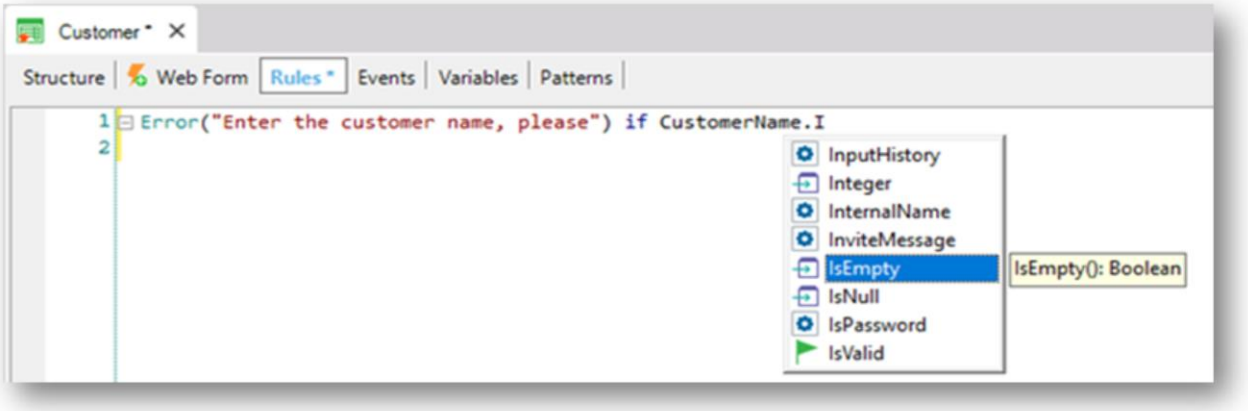
Some controls in the customer entry



In addition to all the automatic controls included by GeneXus in the applications it generates, sometimes users request that we make some specific controls.

In transactions, the rules that must be complied with, or the controls that we are asked to validate, are defined in the Rules section.

Some controls in the customer entry



The order in which rules are defined does not necessarily correspond to the order in which they will be executed.

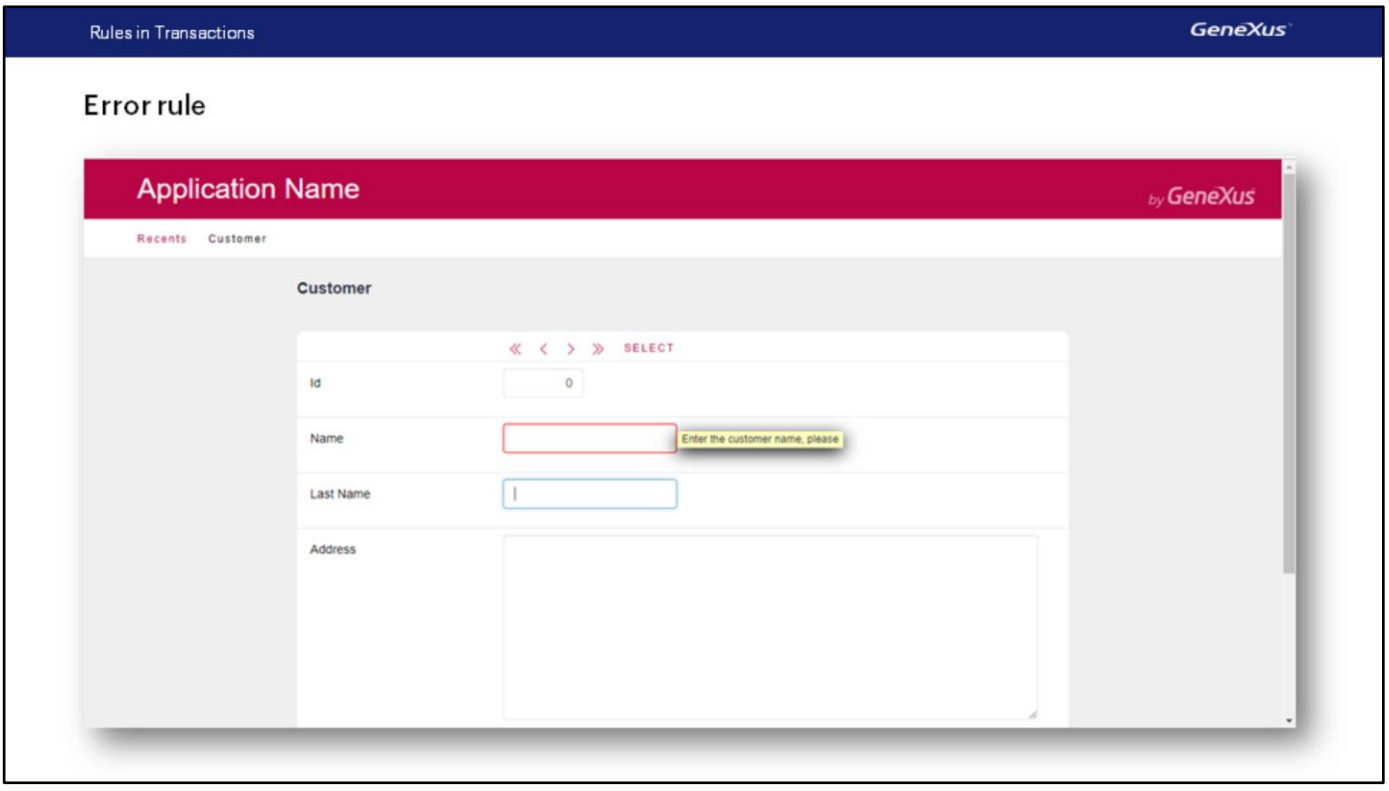
If, for example, a requirement is **not to allow storing of clients without a name...** we have a rule called **Error** that will enable us to avoid that.

We type “Error”, open brackets and between single quotation marks we enter the text we want to have displayed in the event that the user tries to leave a client name blank... we close brackets... and now we only have to indicate **the condition that must be met** for the text to be displayed.

The condition is that the **CustomerName** attribute is empty… so we write “if CustomerName”, period, and here we select: IsEmpty.

All the rules we state must end with a semicolon, so we include it.

We save... and press F5 to see this rule at runtime.

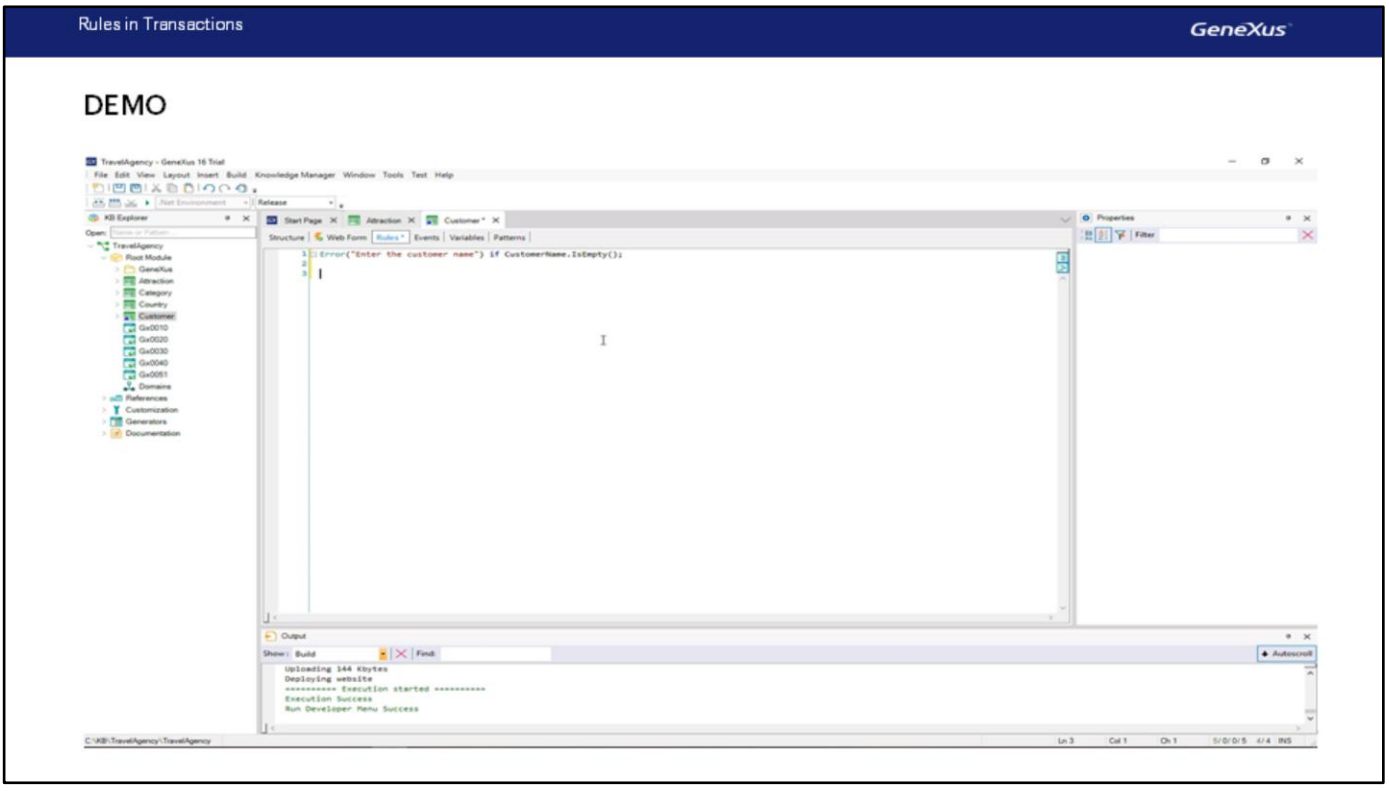


We run the Customer transaction, and if we leave the client name blank and leave the field, the text we defined is displayed.

The **Error rule doesn't allow us to move on if the condition continues to be met**… so, the user either enters a client name to be able to continue or cancels.

Note that if we try to click on the client's surname, the message stays on screen because the condition continues to be met.

We enter a name... and see that we can continue entering the rest of the client's details.



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

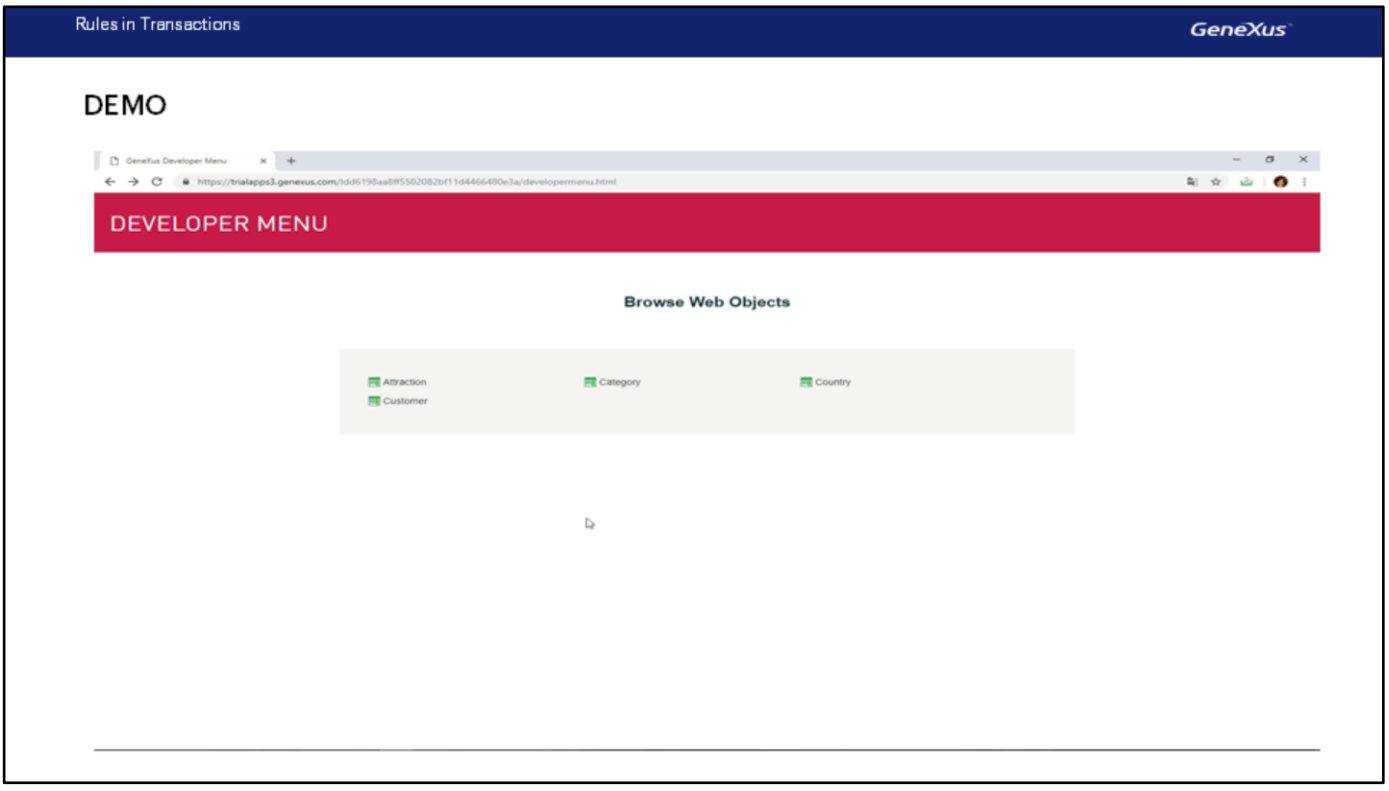
If another requirement involved preventing the client's surname from being left blank, a similar rule would also have to be stated. So, we copy and paste this rule definition... replace "name" with **lastname** and change the attribute involved.

We press F5... run Customer... and leave the client's name empty... the error associated with a blank name is displayed... we enter Paul... and try to leave the surname empty.... the error associated with a blank surname is displayed.

Message rule

There is another rule whose syntax is very similar to that of the **Error** rule… it is called **Message**… and the only difference with **Error** is that if the condition is met, the message is displayed as a notice or warning, and the user **can continue working**. That is to say, it doesn't prevent users from moving on, as does the Error rule.

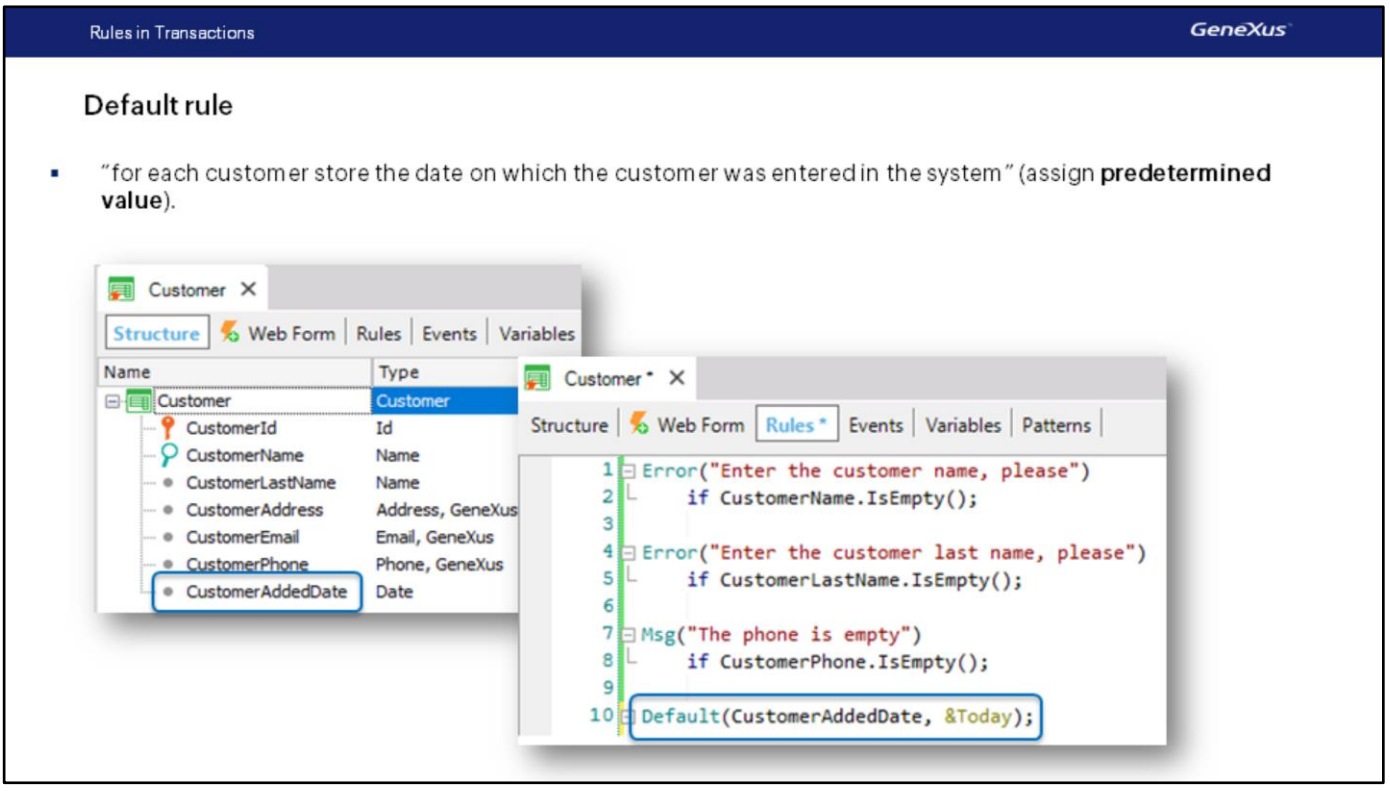
If, for example, we want to inform that the client's phone number has been left blank without forcing the user to enter it, we can create a **Message** rule, open brackets, enter the text between single (or double) quotation marks... 'The phone is empty'. We close brackets... and next we define the condition for the rule to be executed: "if CustomerPhone"… period… IsEmpty. And we type a semicolon to complete the rule definition.



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

We press F5 to try this functionality...

Note that if we leave the phone blank and try to leave the field, the message that we created is displayed. In this case it is orange and we can move on.



Let's suppose that our users at the travel agency have told us **that they are interested in storing the date on which each customer is added.**

So, we need to create a new attribute in the Customer transaction to store this date. We enter **CustomerAddedDate** … of Date type… and now we would only have to automatically assign it today's date.

We go to the Rules section... and select a rule called **Default.**

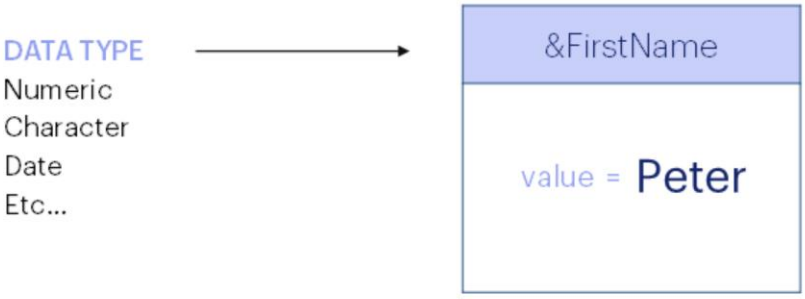
This rule allows us to start an attribute or variable with a value.

In this way, the syntax of the Default rule has been inserted. Now we will replace the attribute between brackets with the attribute that we want to initialize, which is **CustomerAddedDate**, and we will also add the value that we want it to use, which is today's date.

"Ampersand today" is a predefined variable that always has today's date loaded in order to use it.

Variables

- Value temporarily stored in memory with an associated name and data type. To make reference to a variable, the "&" symbol must be used.



Example: The variable &FirstName stores in memory the value "John"

A variable is one space in memory with an associated symbolic name and a data type that can be stored (text, numbers, dates, etc.). This variable has a certain value stored. In general, the variable name is used to make reference to this stored value.

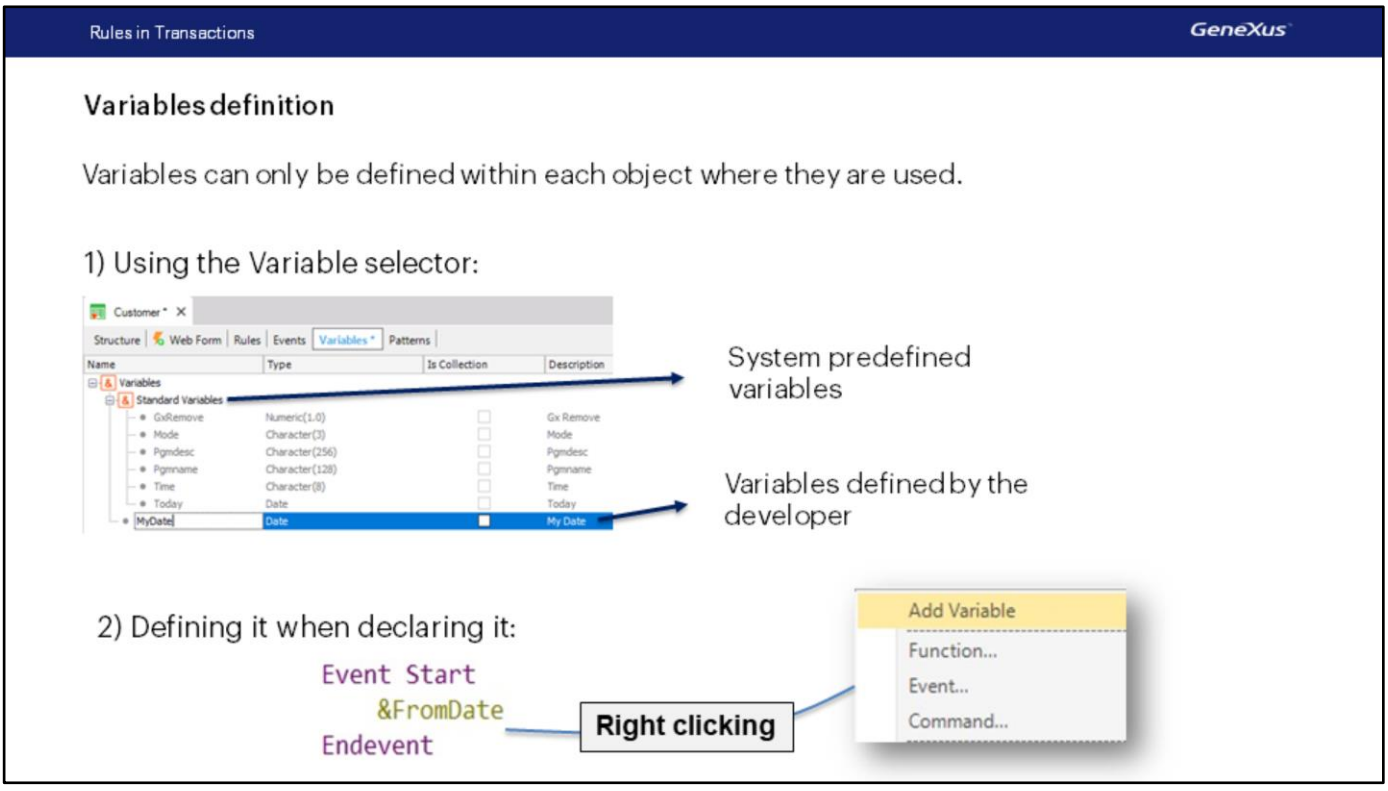
Attributes

- Value physically stored in the database

CUSTOMER TABLE

CustomerId	CustomerFirstName	CustomerLastName
1	Peter	Smith
2	Susan	Parker

An **attribute**, on the other hand, **is one** value physically stored in the database.



Most GeneXus objects allow defining variables. These variables are **local**, which means that they can only be used within each object. To make reference to a variable, the "&" symbol must be used. For example, &Total.

If we open the *Variables* selector within a transaction we will see that a set of variables has already been defined. They are system variables, such as for example, &Today, &Mode, etc. In particular, the &Today variable allows obtaining the current date taken from the system.

In addition to these system variables, the developer can also create his/her own variables (user variables). For example, a MyDate variable of Date type.

When creating variables, these options are available:

1. Create them through the Variables selector included in all GeneXus objects, as we've just done.

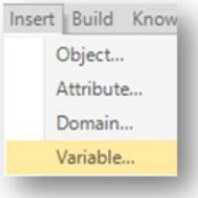
2. Create them at the time they will be used in the place they are needed. For example: type the ampersand symbol to indicate that a variable name comes next, followed by the variable name. Lastly, click on the name given to the variable and right-click to select this option from the context menu: **Add Variable**.

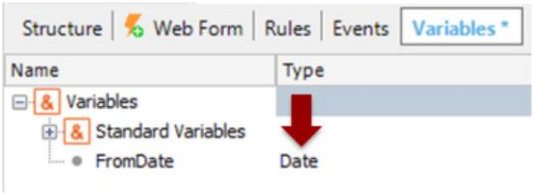
We can see that the variable properties are edited. Now we can assign it a data type.

It has been added to the Variables selector.

Variables definition

3) Insert \ Variable:





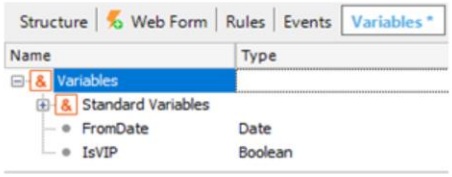
4) Automatic, according to the name pattern:

a) Domain : Name

&FromDate - It is automatically defined as Date.

b) **&IsXxx** | **&HasXxx**

→ They are automatically defined as Boolean

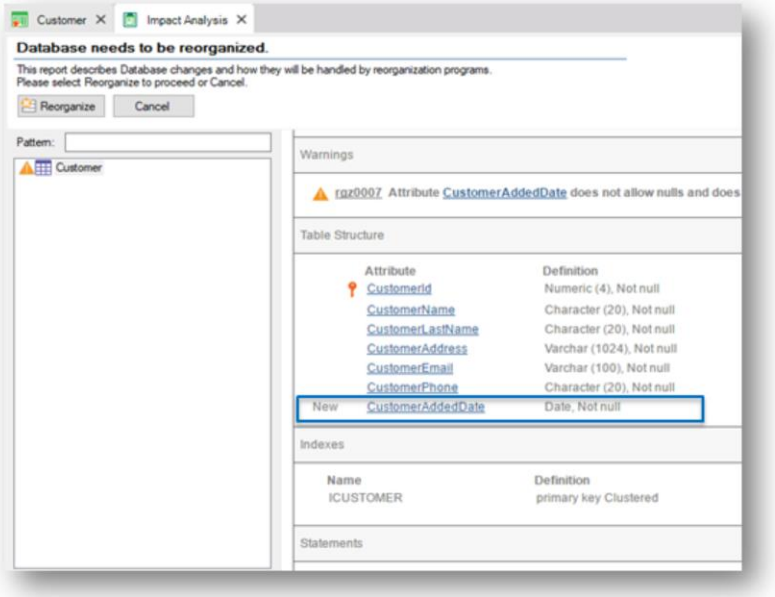


3. The third option is to select **Insert** from the menu bar and then **Variable…** and **New Variable…**

In the example displayed, the Default rule is used to assign the current date to the CustomerAddedDate attribute by default.

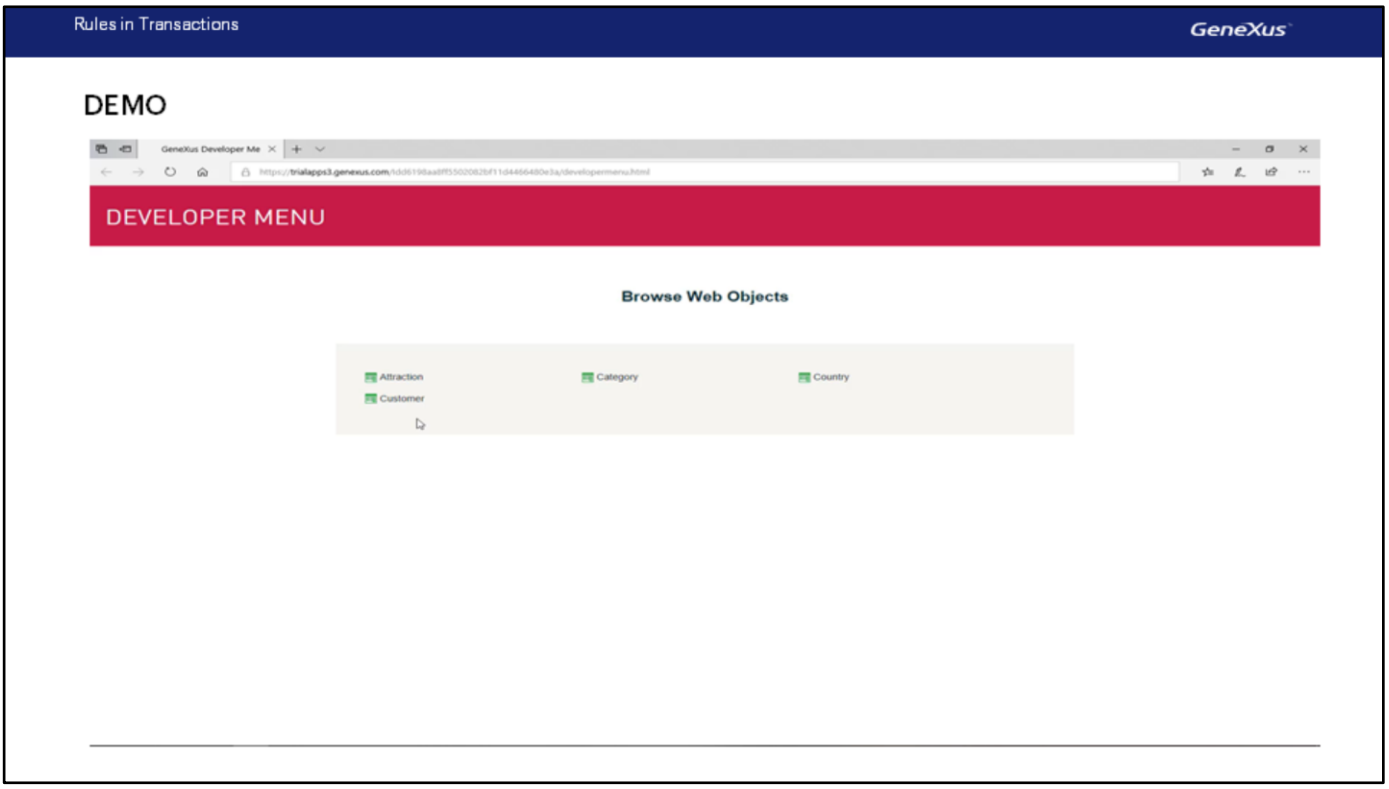
Now we save... and press F5.

Impact Analysis



We are notified that the new attribute **CustomerAddedDate** will be added to the CUSTOMER table.

We click on Reorganize...



[DEMO: https://youtu.be/jqIcfWNDhOk]

And once again we have the application ready to be executed.

We click on Customer...

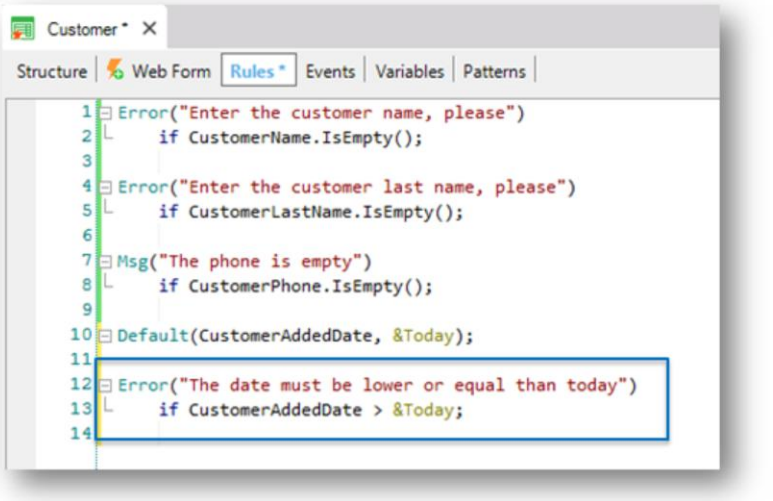
And we can see that the new “Added Date” attribute is already started with today’s date.

If we hadn’t entered the Default rule, the date field would be empty as the other fields.

We enter a customer… Robert… Hill… who lives on 81st Street…. his phone number is 760 5100 and his email address is Rhill@hotmail.com … note that today’s date is **suggested, but we can change it**.

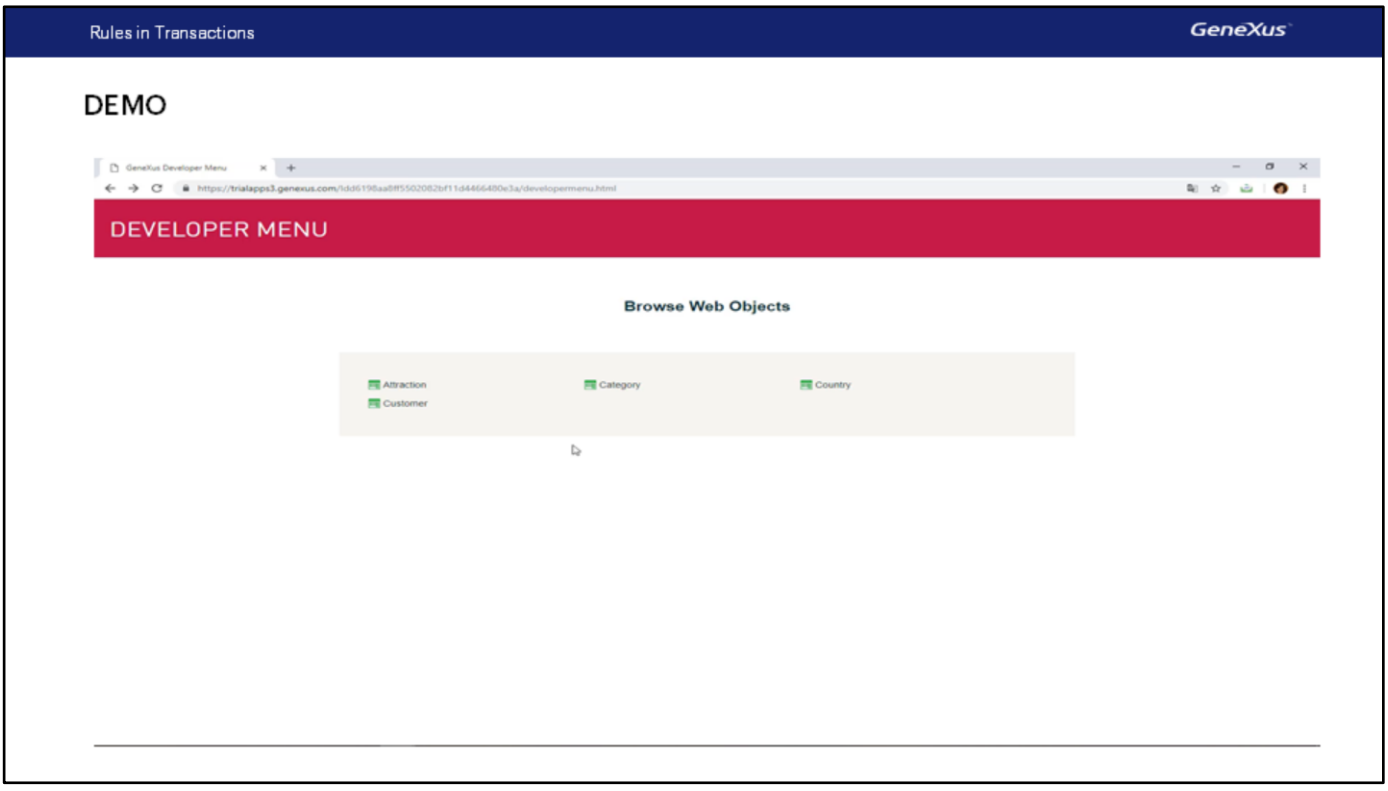
More controls are required...

- leave the date field editable + control that "no future dates can be entered".



If the users at the travel agency were interested in allowing the date to be edited, **preventing future dates from being entered**... we could create an **Error** rule.

We open brackets and type 'The date must be lower or equal than today', close brackets … and add the condition **if CustomerAddedDate > &today;**



[DEMO: https://youtu.be/EPiudriAo_M]

To try this at runtime... we press F5...

We enter Alex… Johnson…

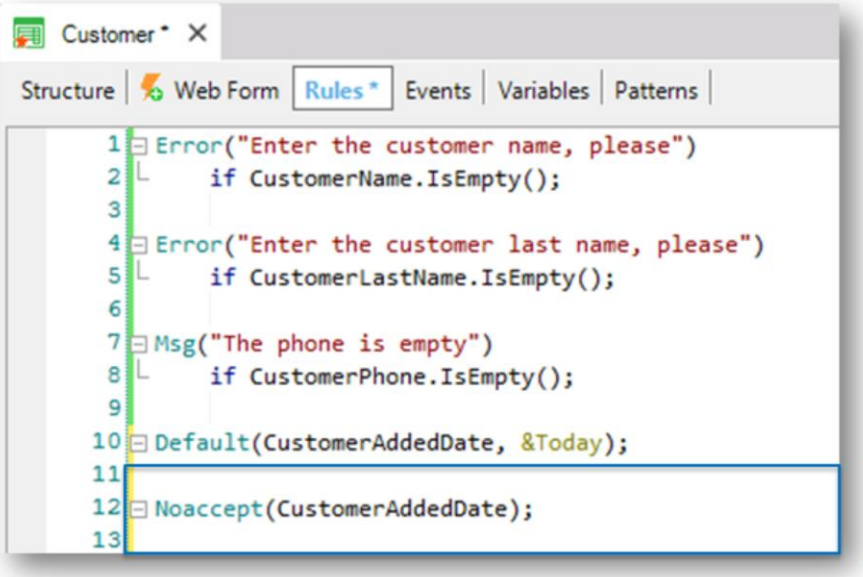
And if we try to enter a date higher than today...

The condition that we defined is met and the associated error is displayed.

Now, let's suppose that our users at the travel agency have told us that the date a client was added **cannot be edited**. It must be shown as disabled in the form and saved as suggested by the application...

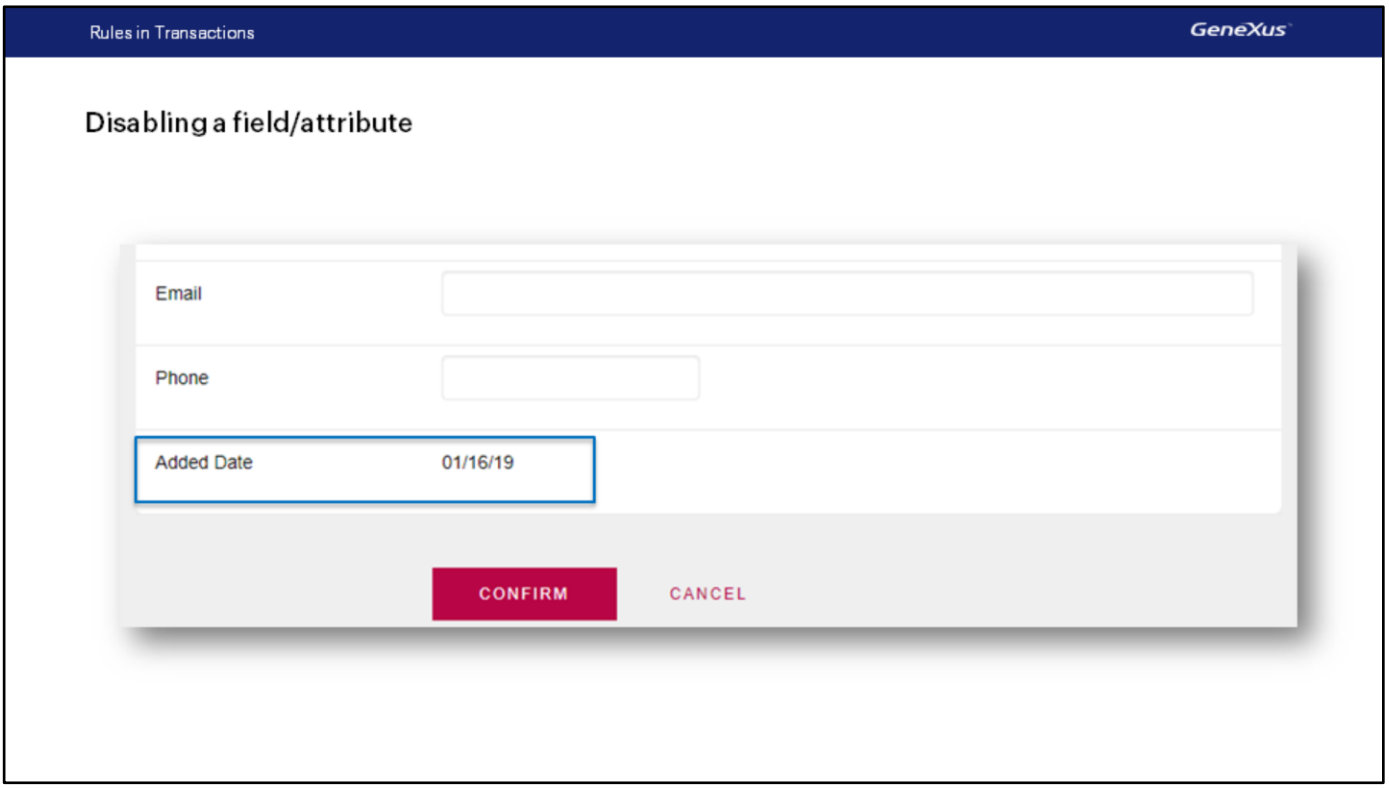
To meet this request, we would have to remove this rule because it no longer makes sense.

Disabling a field/attribute



Also, we would have to state a new rule... Noaccept.

We replace the text “attribute or variable” between brackets with the attribute **CustomerAddedDate** and delete “if condition”, because we want this rule to always be executed.



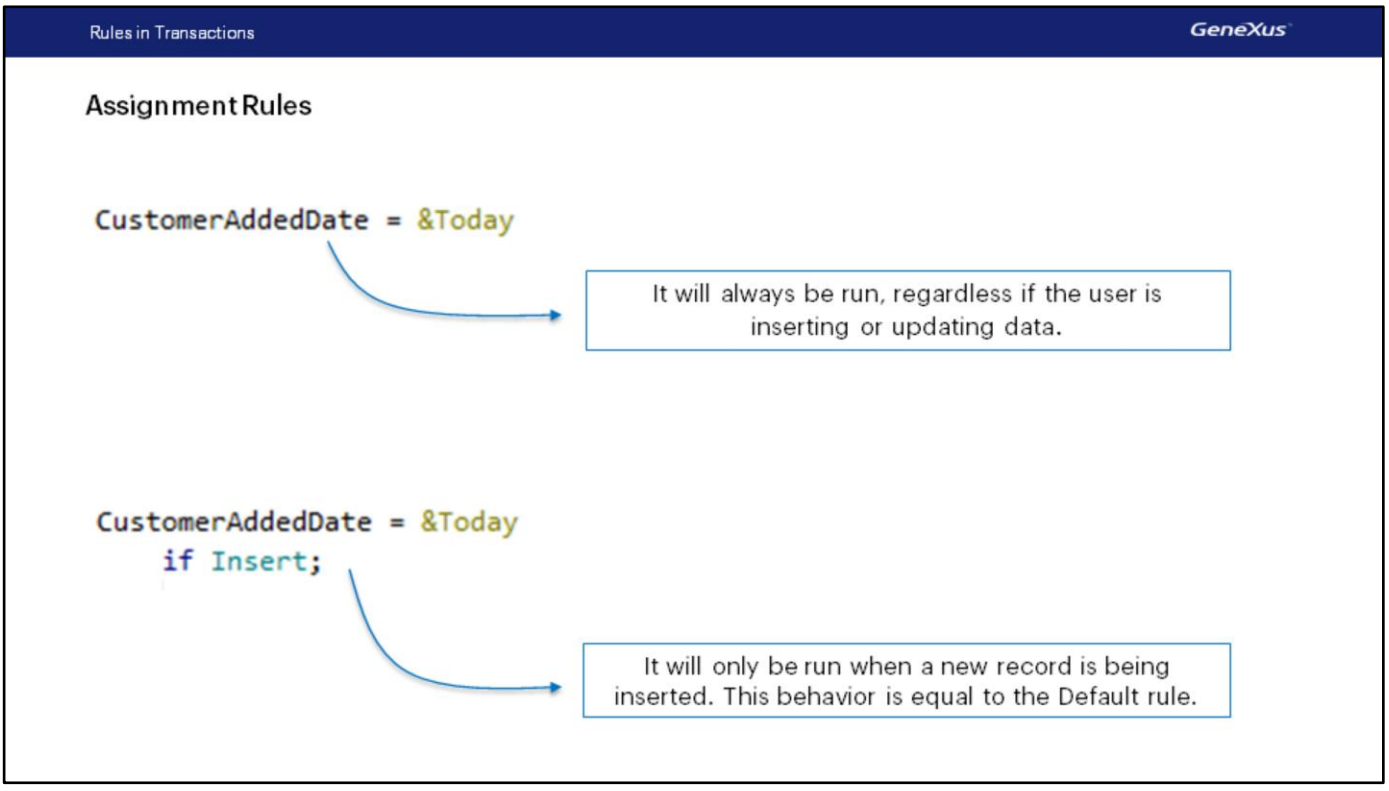
Now let’s try this behavior... F5...

As we can see, the date is **initialized** by the Default rule and **disabled** by the Noaccept rule.

We've seen that to initialize the CustomerAddedDate attribute with today's date, we have to define this Default rule.

It's important to know that every Default rule that we define will be executed **only when we're adding records.**

That is to say, if we query a customer that was already saved, the Default rule will not be executed... because this customer **already has an insertion date…** and the Default rule doesn't overwrite it.



Now, let's suppose that instead of defining the Default rule we made this assignment: *CustomerAddedDate = &Today;*

By defining this rule, the CustomerAddedDate attribute **would always be assigned** with today's date. **This is an assignment rule**, and it will always be executed, regardless if the user is inserting or updating data, etc.

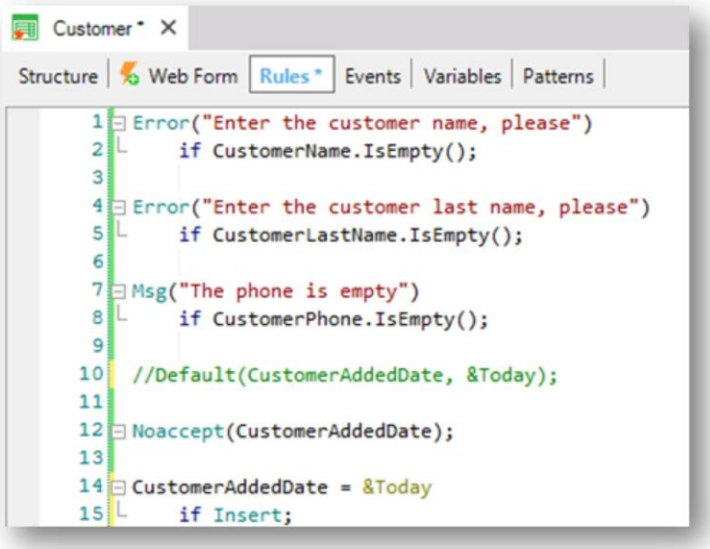
A condition can be added to an assignment rule so that it is executed only when the user is performing a certain action in the database, such as an insertion, update or deletion.

Let's do it. We type **if insert**:

The behavior of the rule defined in this way will be the same as that of the Default rule, because now we have conditioned the assignment to be made only if a record is being inserted. This is what the Default rule does.

Just like a rule can be conditioned with **if insert**, we can condition rules to be executed **if update** or **if delete** as well.

Triggering order of rules



Something important to note and learn is that **the order in which rules are defined doesn't necessarily match the order in which they will be executed**.

This set of rules, could be stated in any other order and the result at runtime would be exactly the same, because GeneXus decides when each one of the defined rules should be triggered.

Finally, remember that each transaction may need to have its own behavior rules defined.

In this case, we've defined rules in the Clients transaction to check the behavior that we were asked to control when users interact with the clients' details. Most likely, the agency wants to control certain rules or behavior for attractions as well... or for another transaction. To this end, **each transaction has its own rules section**.

Lastly, we save the changes in GXserver.

So far, we have seen that:

- Rules allow programming transaction behavior.
- Rules are triggered within the scope of the transaction where they have been stated.
- GeneXus determines the order in which rules are triggered, with certain independence from the order in which they have been written.
- Rules can be conditioned to, among other things, the operation modes of the transaction (**Insert**, **Update**, **Delete**).

GeneXus™

The power of doing.

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