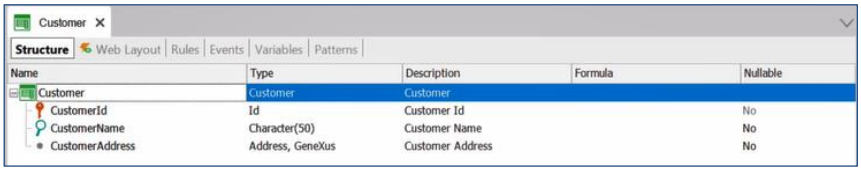
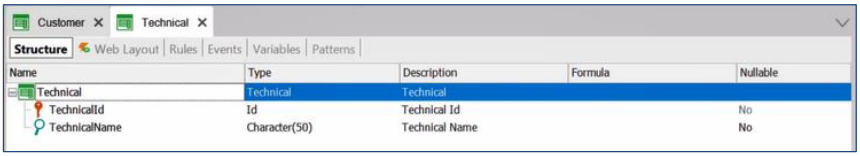
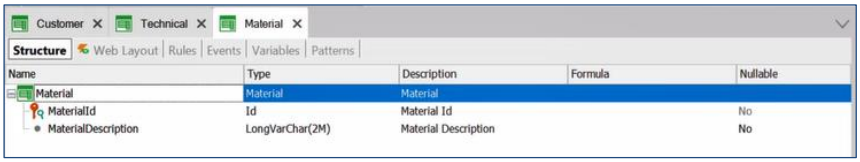
Parallel Transactions

GeneXus™

Parallel Transactions





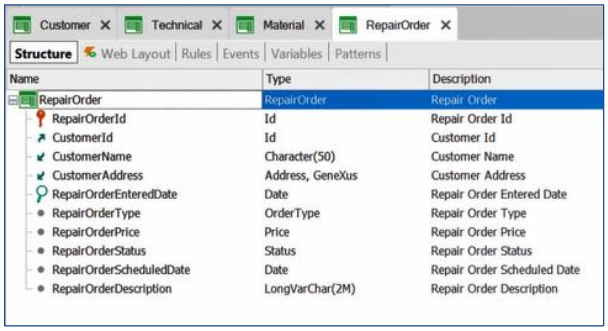


Let's suppose that we are designing a small application for a company that provides technical services at the customers' location. From this application, among other things, we will be able to generate the orders for repair requests made by customers.

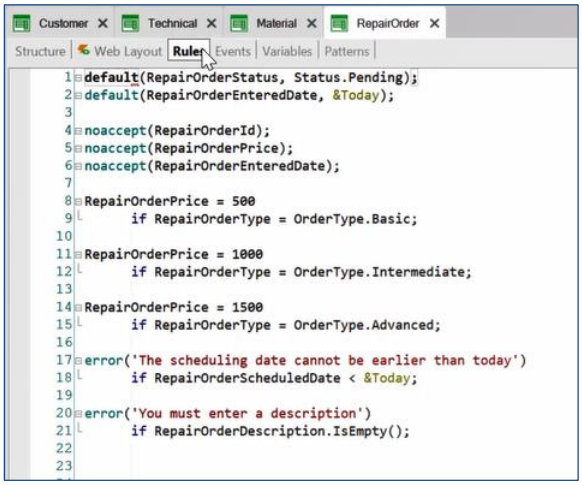
Let's look only at the transactions we're interested in for our example.
We have a transaction to record customers; another transaction to record the company's technicians; and a transaction where the materials can be recorded, which the technicians use to solve the different issues of clients.

RepairOrder Trn

Structure



Rules



Then we have this transaction (RepairOrder), where the operator who receives the request from the customer will enter all the necessary data to generate the service order.
In it, we have as a primary key the RepairOrderId attribute, which has an auto-numbered data type assigned to it. Each order will have an associated client, so in this transaction we have inferred attributes from the Customer table.
Lastly, we have these other attributes of the transaction. To record the date of entry, the type of order (whose data type will be enumerated containing these values), the price, a status (whose data type will also be enumerated, and will accept the following values), the date scheduled and a description.

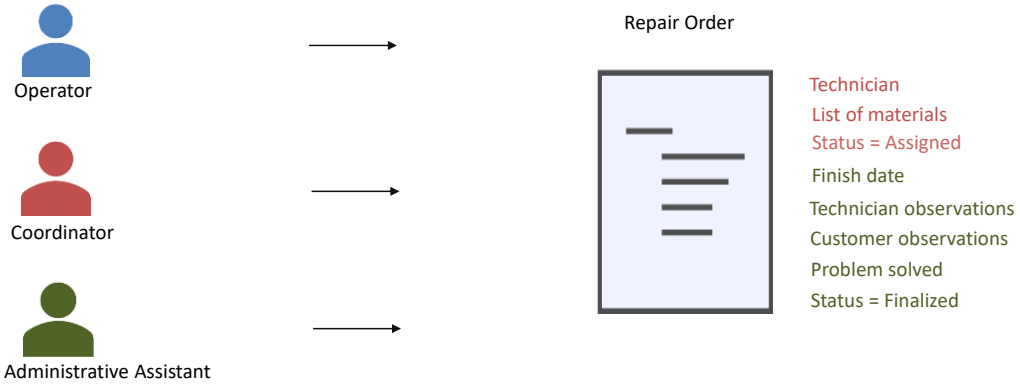
Let's look at the rules we entered.

We have a couple of “default” rules and several “noaccept” rules.
In the default rules we assign a value to the attribute that will keep the status of the order –in this case “P”, for pending. And in this other case, we will assign today's date to the attribute that will save precisely the date of entry of the work order.

We also see that this attribute (RepairOrderPrice) of the transaction will be calculated from the value entered in this other attribute (RepairOrderType), which will record the type of order.
Lastly, we declare a couple of “error” rules.

Let's see this transaction running, from which we enter a repair order.

Order entry and modifications

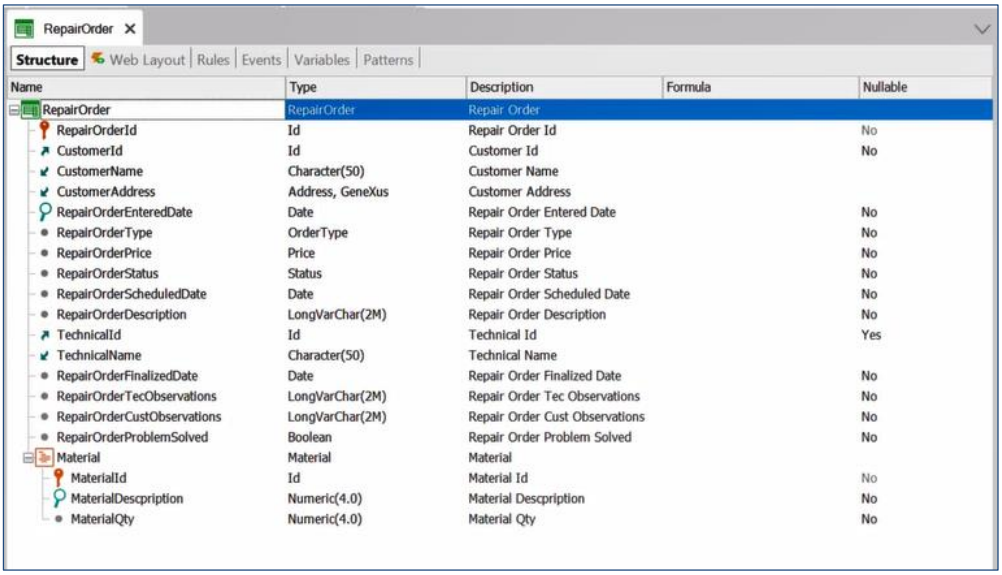


Now let's suppose that in our reality, after an order is entered, a coordinator must assign a technician to it and the materials provided to that technician, in order to monitor it. And change its status to assigned.

In addition, when the task is completed by the technician, an administrative assistant should be able to assign an end date to the order, enter comments from the technician and the customer, record whether the issue was solved or not, and change the status of the order to completed.

How can we do this?

RepairOrder Trn with all attributes



Name	Type	Description	Formula	Nullable
RepairOrder	RepairOrder	Repair Order		
RepairOrderId	Id	Repair Order Id		No
CustomerId	Id	Customer Id		No
CustomerName	Character(50)	Customer Name		
CustomerAddress	Address, GeneXus	Customer Address		
RepairOrderEnteredDate	Date	Repair Order Entered Date		No
RepairOrderType	OrderType	Repair Order Type		No
RepairOrderPrice	Price	Repair Order Price		No
RepairOrderStatus	Status	Repair Order Status		No
RepairOrderScheduledDate	Date	Repair Order Scheduled Date		No
RepairOrderDescription	LongVarChar(2M)	Repair Order Description		No
TechnicalId	Id	Technical Id		Yes
TechnicalName	Character(50)	Technical Name		
RepairOrderFinalizedDate	Date	Repair Order Finalized Date		No
RepairOrderTecObservations	LongVarChar(2M)	Repair Order Tec Observations		No
RepairOrderCustObservations	LongVarChar(2M)	Repair Order Cust Observations		No
RepairOrderProblemSolved	Boolean	Repair Order Problem Solved		No
Material	Material	Material		
MaterialId	Id	Material Id		No
MaterialDescpription	Numeric(4.0)	Material Descpription		No
MaterialQty	Numeric(4.0)	Material Qty		No

One option would be to add all these attributes to the transaction, and to complete those that we need at every moment.
The transaction would look like this.

Let's see it at runtime.

As we can see, this solution does not provide a friendly interface for the user, since there will always be fields that may not be of interest or may not have to be completed.

For example, when the operator receives the order from the customer and enters it for the first time, before it is assigned to a technician, we will see several fields that will not be completed. In this case they are relatively few, but if the number of attributes is increased this will impact more and more on the users and their experience with the system.
The same happens when the coordinator accesses the transaction, selects the order to be modified and so it is changed to Update mode, assigns a technician to it and loads the materials.

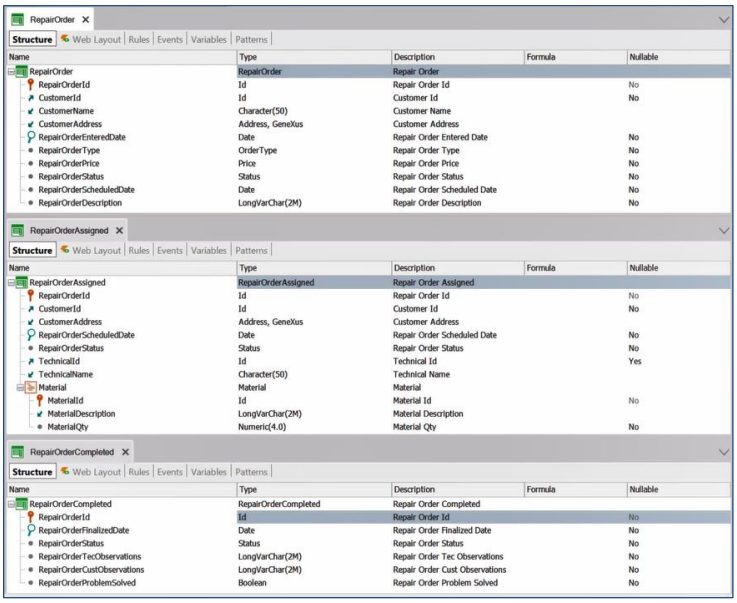
As soon as we search for the order and load it on the screen, we will be shown attributes that we may not be interested in viewing. Then, to load the materials, you will have to leave fields empty in order to complete them later.
Of course, this is not comfortable for entering data. And as we've just said, since there are few attributes in this example, it doesn't have a great impact. However, as the number of attributes increases, since we may need to record more data, the input process becomes more complex.

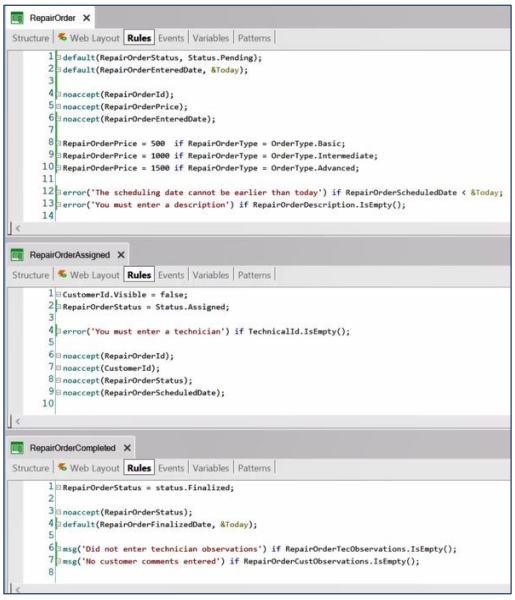
When the administrative assistant wants to complete it with the corresponding observations, he will need to select the order to work on, switching the transaction to Update mode, and will see all the attributes of this transaction, when in fact he only needs to see a few details.

Another disadvantage is how complex it can be to program the behavior of this transaction. As we saw, it will go through different states and modifications with later entries. Also, it may require different controls every time, even of the attributes themselves. But, given this design, everything will be programmed in the same object. It will not be possible to customize rules or events specific to each of the moments we have just seen.

These disadvantages could be solved in a simple way by using what we call **parallel transactions**.

Parallel transactions



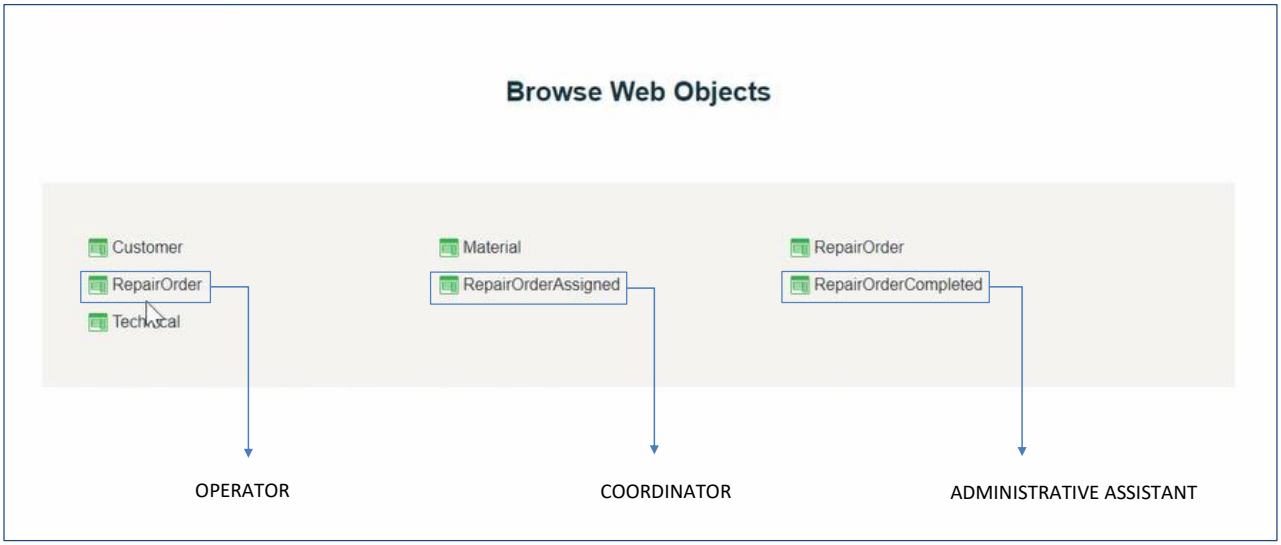


In this way, we would have a transaction for entering the order first by the operator. Another transaction, where the coordinator will assign that order to the technician and load the materials to be given to him. Lastly, a transaction, where the administrative assistant will be able to upload the technicians' and customer's comments, if any, and whether the problem was solved or not. Each one of these transactions will have the attributes that we are really interested in for each case.

For this, all three transactions must have the same attribute as the primary key. This is what makes them parallel transactions.
It is important to mention that the parallelization of transactions is done by level. In this example, the main levels of these transactions are parallel.

If we look at the rules section of the transactions, we see that each one has its own rules, totally independent from each other. This is one of the great advantages of working with parallel transactions. The same is true for any events we may program.

Let's see the application running with these modifications.



From this transaction (RepairOrder) an operator receives the customer's request and completes all the fields. We see that the value “P” –of pending– is assigned automatically, because it is defined in the default rule.

Some time later, the coordinator accesses this other transaction (RepairOrderAssigned), searches for the order, and completes the necessary data. The value that records the status of the order is automatically changed to “A” (assigned).

Finally, after the task is completed and the technical report is received, the administrator will manage it from the third transaction created (RepairOrderCompleted), filling in the corresponding fields, and the status will be automatically changed assigning it the value “F” (finished).

We see that, in this way, it becomes much simpler, clearer and more intuitive for users. And it helps to make the application scalable, to face future changes that may have to be made in the procedure for entering orders.

In Database

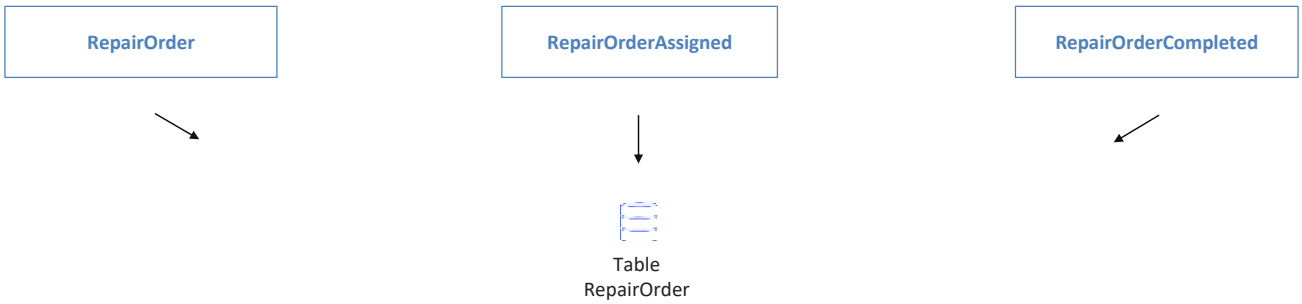


Table
RepairOrder

RepairOrder Id	Customer Id	Technical Id	RepairOrder EnteredDate	RepairOrder Type	RepairOrder Price	RepairOrder Status	RepairOrderScheduled Date	RepairOrder Description	RepairOrder FinalizedDate	RepairOrderTec Observations	RepairOrderCust Observations	RepairOrder ProblemSolved
1	7	*NULL*	2020-11-01	1	500	P	1753-01-01		1753-01-01			False

At the database level, from the main levels of these three transactions, a single physical table will be generated, since as we know, GeneXus designs the database following normalization criteria.

The generated table will contain the attributes resulting from associating the three transactions.
When we enter a record from the first transaction, what happens then in the table of our database, with the attributes to which we haven't assigned any value yet? What value will they have?

Empty values will be assigned by default: “0” in numeric fields, empty string in character type fields, a default date representing the empty value for those of Date type, and false for Boolean attributes.

Parallel transactions offer a clear way to keep in the structure of each transaction solely the information you want to work with within that program, regardless of the information in the same table that can be handled in another transaction.

There is a wide range of realities that can lead us to use this type of transaction, which we will not examine in detail in this video. For more details on this topic, please visit our Wiki.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications