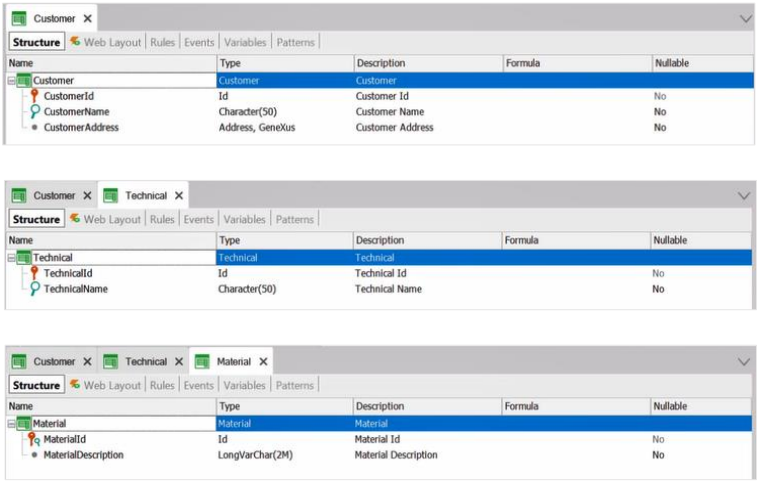
Transacciones paralelas

GeneXus™

GeneXus

Parallel Transactions

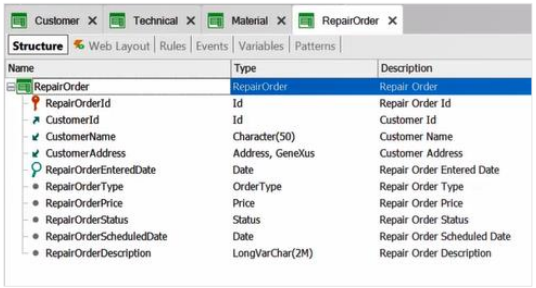


Supongamos que estamos diseñando una pequeña aplicación, para una empresa que brinda servicios técnicos a domicilio para sus clientes. Desde la cual, entre otras cosas, podremos generar las ordenes de reclamo de reparación realizadas por parte de los clientes.

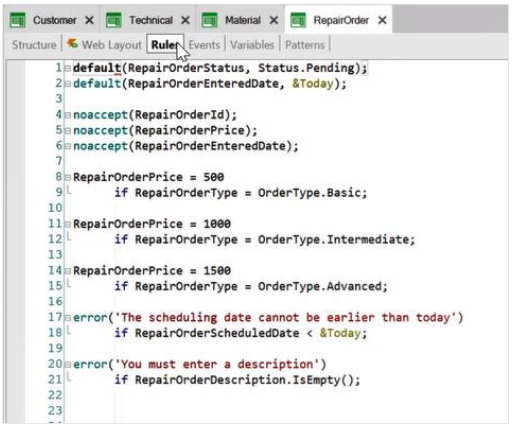
Veamos sólo las transacciones que nos interesan para nuestro ejemplo. Tenemos una transacción para registrar los clientes. Otra transacción para registrar los técnicos de la empresa. Y una transacción donde se podrán registrar los materiales, los cuales utilizan los técnicos para solucionar los diferentes inconvenientes de los clientes.

RepairOrder Trn

Structure



Rules



Luego tenemos esta transacción (RepairOrder), donde el operador que recibe el reclamo por parte del cliente, ingresará todos los datos necesarios para generar la orden de reclamo.
En la misma, tenemos como clave primaria al atributo RepairOrderId, el cual tiene asignado un tipo de datos auto numerado. Cada orden tendrá un cliente asociado, por lo que tenemos en esta transacción inferidos atributos de la tabla Customer.
Y por último tenemos estos otros atributos propios de la transacción. Para registrar la fecha de ingreso, el tipo de orden (cuyo tipo de datos será un enumerado que contiene estos valores), el precio, un estado (cuyo tipo de datos también será un enumerado, que aceptará los siguientes valores), la fecha de agendado y una descripción.

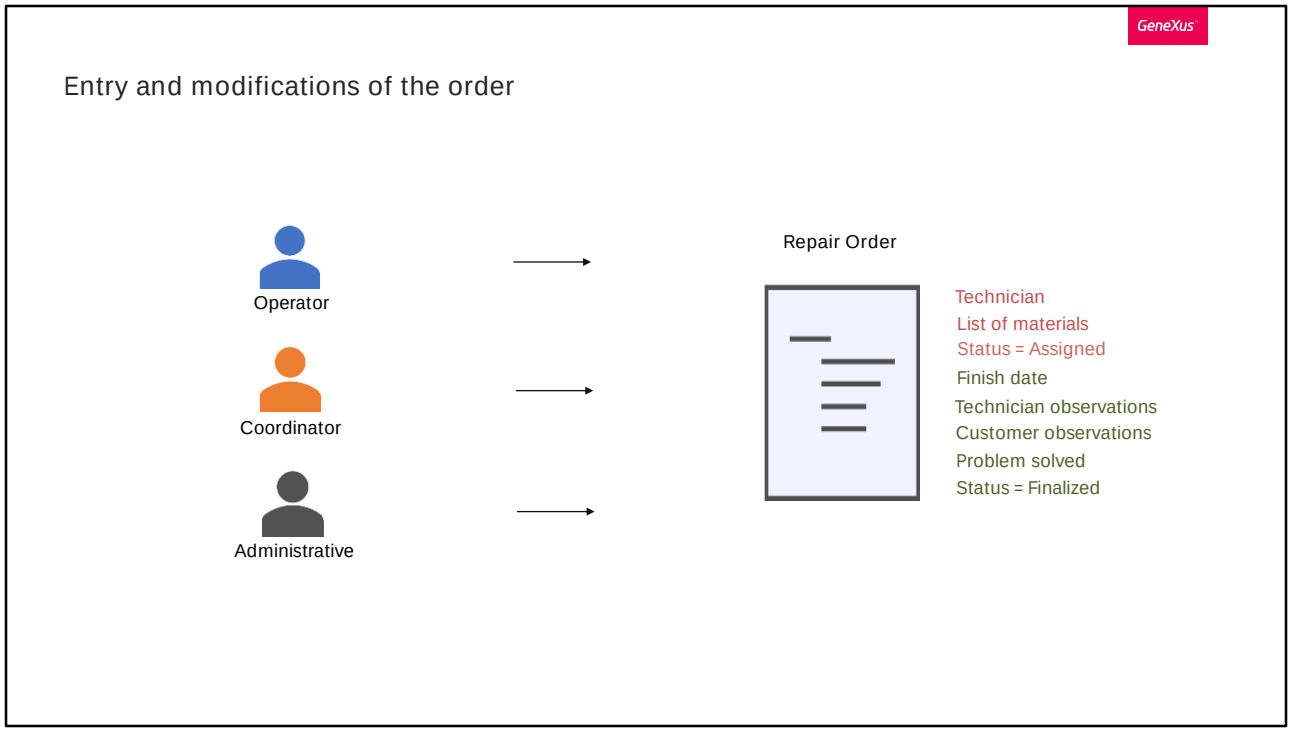
Veamos ahora las reglas que ingresamos.

Tenemos un par de reglas “default” y varias reglas “noaccept”.
En las reglas default asignamos un valor al atributo que guardará el status de la orden, en este caso “P”, de pendiente. Y en este otro caso, le asignaremos la fecha del día al atributo que guardará justamente la fecha de ingreso de la orden de trabajo.

También vemos, que este atributo (RepairOrderPrice) de la transacción,

será calculado a partir del valor ingresado en este otro atributo (RepairOrderType), el cual registrará el tipo de orden.
Y por último declaramos un par de reglas “error”.

Veamos en ejecución esta transacción, desde la cual ingresamos una orden de reclamo.

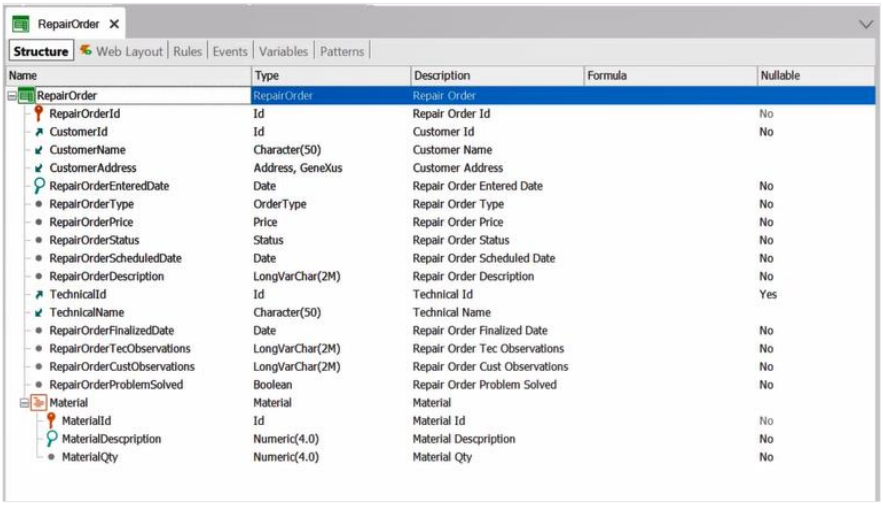


Ahora supongamos, que en nuestra realidad, luego de ingresada una orden, un coordinador deberá asignarle un técnico y los materiales que se le proporcionan al mismo, para poder llevar un control. Y cambiar el status de la misma para que quede como asignada.

Y a su vez, necesitamos que cuando se finalice la tarea por parte del técnico. Pueda un administrativo asignarle a la orden una fecha de finalización, observaciones del técnico y del cliente, registrar si la falla fue solucionada o no, y cambiar el status de la orden a finalizado.

¿Cómo podemos hacer esto?

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
MaterialDescription	Numeric(4.0)	Material Description		No
MaterialQty	Numeric(4.0)	Material Qty		No

Una opción sería agregar todos estos atributos a la transacción, e ir completando los que necesitemos en cada momento.
La transacción nos quedaría así.

Veámoslo en ejecución.

Como vemos, esta solución no presenta una interfaz amigable para el usuario, ya que siempre habrá campos que no interesan ni deban completarse.

Por ejemplo, cuando el operador recibe el reclamo por parte del cliente e ingresa por primera vez la orden, antes de que la misma sea asignada a un técnico, veremos varios campos que no se completarán, todos estos. En este caso son relativamente pocos, pero si la cantidad de atributos aumenta esto impactará cada vez más en el usuario y su experiencia con el sistema.
Lo mismo sucede cuando el coordinador ingresa a la transacción, selecciona la orden que desea modificar, por lo que esta pasa a modo Update, se le asigna un técnico a la misma y carga los materiales.

Ya al buscar la orden y cargarla en pantalla, se nos muestran atributos que tal vez no nos interesen visualizar. Luego, para cargarle los materiales,

deberá dejar campos sin completar, que serán para un posterior ingreso. Esto evidentemente no es nada cómodo para el ingreso. Y cómo dijimos recién, al ser pocos atributos en este ejemplo, no impacta tanto, pero a medida que los atributos se incrementen, ya que podemos necesitar registrar más datos, el ingreso se comienza a complejizar.

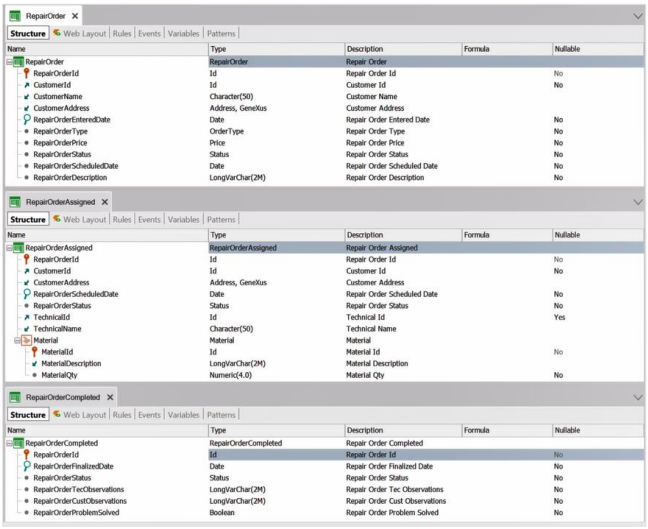
O incluso cuando el administrativo quiera finalizarla con las observaciones correspondientes. Deberá seleccionar la orden sobre la cual desea trabajar, pasando la transacción a modo Update, y verá todos los atributos de esta transacción, cuando en realidad sólo necesita ver unos pocos datos.

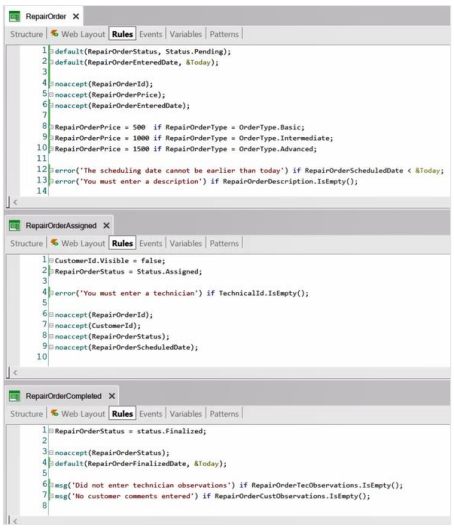
Otra desventaja es la complejidad que se puede llegar a tener para programar el comportamiento de esta transacción. Ya que como vimos, la misma pasará por diferentes estados, y modificaciones con ingresos posteriores, pudiendo necesitar cada momento diferentes controles que queramos hacer, incluso de los mismos atributos. Pero, dado este diseño, todo será programado en el mismo objeto. No pudiendo personalizar reglas o eventos propios de cada uno de los momentos que acabamos de ver.

Estas desventajas podríamos solucionarlas de una forma sencilla mediante el uso de lo que llamamos transacciones paralelas.

GeneXus

Parallel transactions



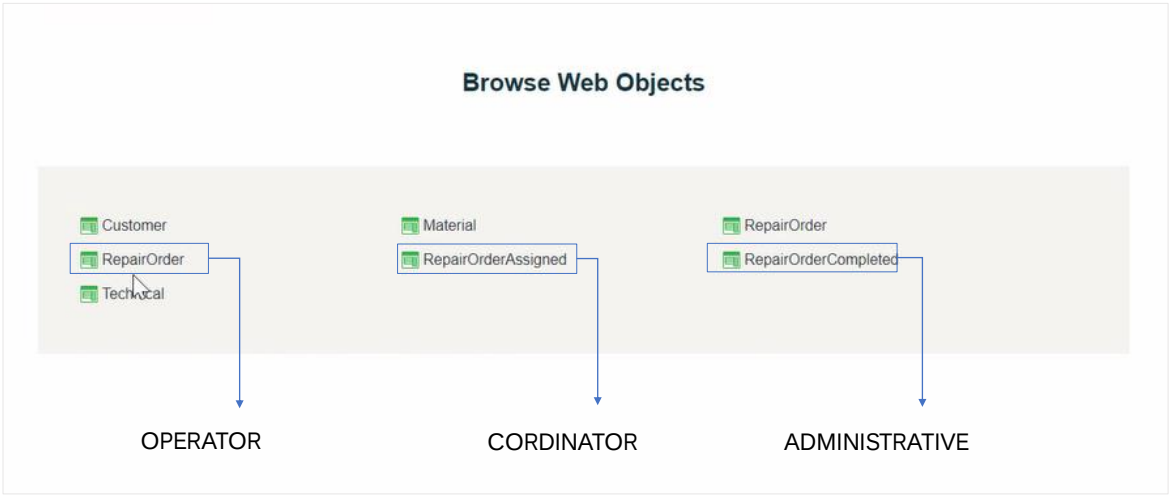


De esta forma tendríamos una transacción para el ingreso de la orden en primera instancia por parte del operador. Otra transacción, donde el coordinador asignará esa orden al técnico y cargará los materiales que se le darán al mismo. Y por último una transacción, donde el administrativo podrá cargarle las observaciones del técnico y del cliente si las hubiera, y si el problema fue solucionado o no. Quedando cada una, con los atributos que realmente nos interesan para cada instancia.

Para esto, las tres transacciones deberán tener el mismo atributo como clave primaria. Esto es lo que las convierte en transacciones paralelas. Una cosa importante a mencionar es que la paralelización de las transacciones es por nivel. En este ejemplo, son paralelos los niveles principales de estas transacciones.

Si observamos la sección reglas de las transacciones, vemos que cada una tiene las suyas propias, totalmente independientes unas de otras. Esta es una de las grandes ventajas que tiene el trabajar con transacciones paralelas. Lo mismo sucede con los eventos que podamos programar.

Veamos la aplicación en ejecución con estas modificaciones.

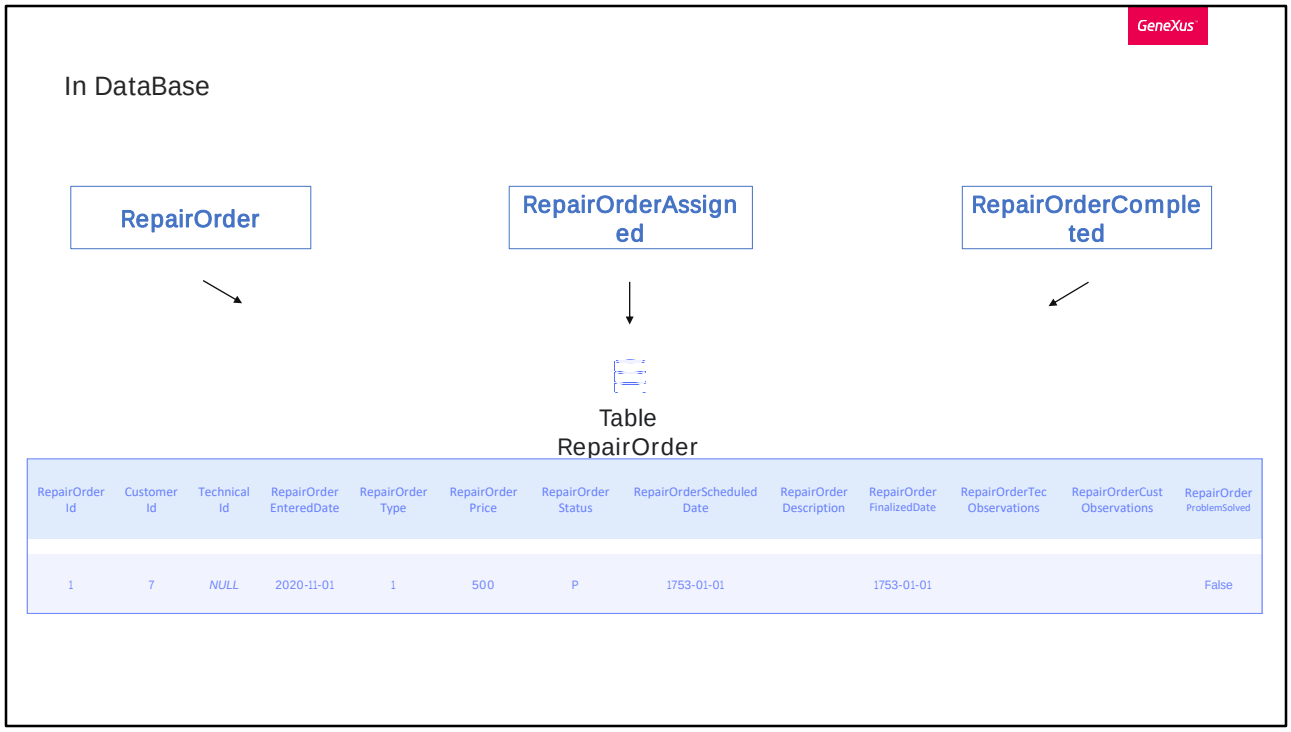


Desde esta transacción (RepairOrder) un operador recibe el reclamo del cliente y completa todos los campos. Vemos que se asigna automáticamente el valor “P” de pendiente, por tenerlo definido en la regla default.

En algún momento posterior, el coordinador ingresa a esta otra transacción (RepairOrderAssigned), busca la orden, y completa los datos necesarios. Cambiando automáticamente el valor que registra el status de la orden a “A” de asignado.

Por último, luego de realizada la tarea y de recibido el parte técnico, el administrativo lo gestionará desde la tercera transacción creada (RepairOrderCompleted), completando los campos correspondientes, y se cambiará automáticamente el status asignándole el valor “F”, de finalizado.

Vemos que, de esta manera, se hace mucho mas simple, claro e intuitivo para los usuarios. Y ayuda a que la aplicación sea escalable, para enfrentar futuros cambios que puedan existir en el procedimiento de ingreso de las órdenes.



A nivel de base de datos, a partir de los niveles principales de estas tres transacciones se generará una única tabla física, ya que como sabemos, GeneXus diseña la base de datos siguiendo criterios de normalización.

La tabla generada contendrá los atributos que resulten de la unión de las tres transacciones.
Cuando ingresamos un registro desde la primera transacción, ¿qué sucede entonces en la tabla de nuestra base de datos, con los atributos a los que aún no les asignamos ningún valor? ¿Con qué valor quedarán?

Se asignarán valores vacíos por defecto: “0” en campos numéricos, cadena vacía en los campos del tipo character, una fecha por defecto que representa el vacío para los del tipo Date, y false para los atributos booleanos.

Las transacciones paralelas nos ofrecen una forma limpia de mantener en la estructura de cada transacción estrictamente la información con la que se quiere trabajar dentro de ese programa, independientemente de la información de la misma tabla que pueda manejarse en otra de las transacciones

Hay una gran diversidad de realidades que nos pueden llevar a la

necesidad de utilizar este tipo de transacciones, las que no entraremos en detalle en este video. Lo invitamos a indagar en nuestra WIKI para más detalles sobre este tema.

GeneXus™

training.genexus.com
wiki.genexus.com