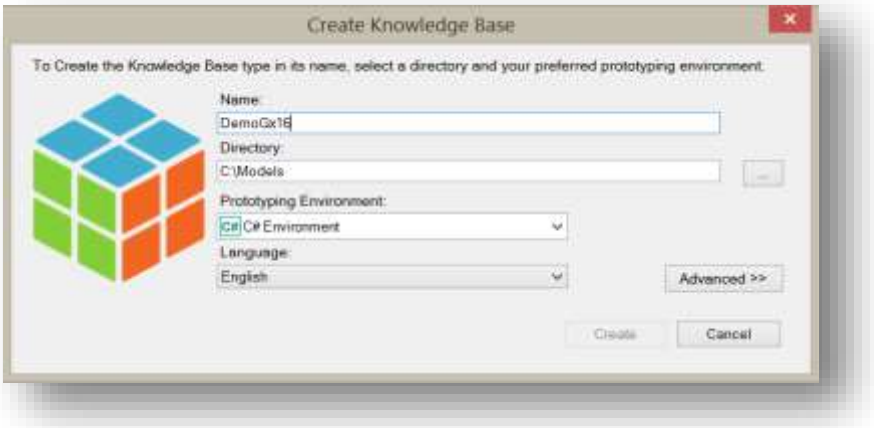
Demo de GeneXus

Comenzaremos creando una Knowledge Base:

Para eso vamos a la opción File/New/Knowledge Base.

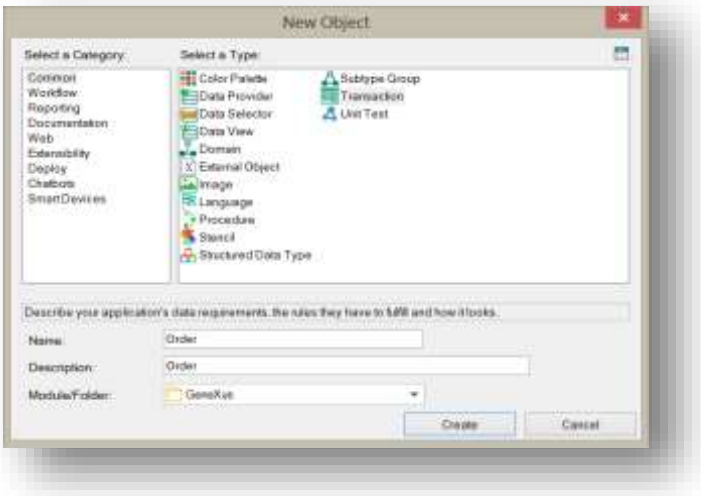
Y proponemos un nombre para esta Base de Conocimiento, por ejemplo DemoGX16.



Mientras se crea la KB es importante mencionar que los proyectos GeneXus reciben el nombre de "Base de Conocimiento", debido a a que en GeneXus las aplicaciones se describen en lugar de programarse. Se diseñan en base a la realidad del negocio que estemos modelando, y por ende el proyecto termina siendo una base del conocimiento del negocio.

Comenzaremos ahora a describir y modelar la realidad.

Vamos entonces a la opción ***File / New / Object*** *[o presionamos CTRL + N]*



Y vemos que en GeneXus hay varios tipos de objetos que en su conjunto nos permitirán modelar cualquier aplicación.

En particular, para representar las entidades de nuestra realidad utilizamos el objeto Transaction.

De las Transacciones definidas se infiere el modelo de datos de la aplicación en 3era forma normal y GeneXus utiliza también este objeto para generar el programa de la aplicación que le permitirá al usuario final, insertar, borrar y actualizar registros de la base de datos en forma interactiva.

Creamos entonces la Transacción Order.

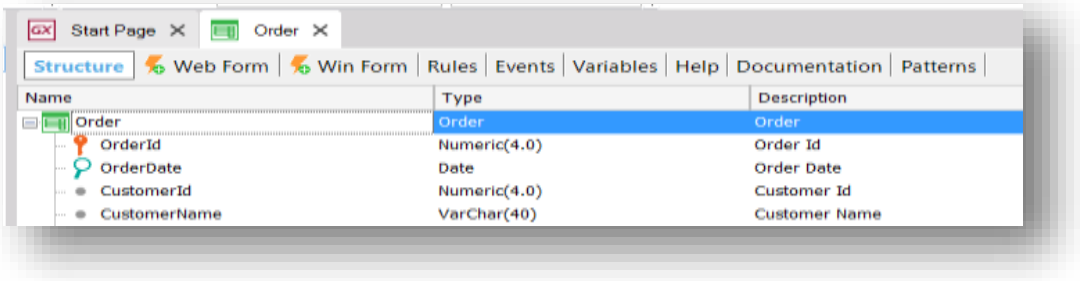
¿Qué nos va a interesar registrar de una orden de compra?

- Seguramente un número que la identifica.
- La fecha.
- Los datos del Cliente.
- Y el conjunto de productos con su precio, cantidad y el total de la línea.
- Y el total de la orden.

Vamos entonces a comenzar a definir estos datos, y para esto vamos a utilizar una nomenclatura en la que cada campo, o atributo, se nombra con un prefijo de la transacción a la que pertenece.
Para esto tendremos que indicar el atributo, por ejemplo OrderId. Y vamos entonces a presionar la tecla del ".", para que ya precargue el nombre de la transacción al momento de nombrar a cada atributo.

Damos Enter para ingresar el siguiente atributo que en esta caso va a ser la fecha de la orden, así que ponemos OrderDate.
Definimos ahora CustomerId y CustomerName para registrar al cliente de la orden, y presionamos tabulador para indicar el tipo de dato correspondiente.

Hasta aquí hemos ingresado los campos del cabezal de la orden de compra. Ahora ingresaremos las líneas.



Para eso presinamos Ctrl + la flecha derecha del teclado e ingresamos los campos del segundo nivel. Estos atributos del segundo nivel son los que corresponden a los productos.

Así que ingresamos el Id, damos Enter, ingresamos el nombre, tabulador para elegir el tipo de dato, Enter nuevamente, el precio..

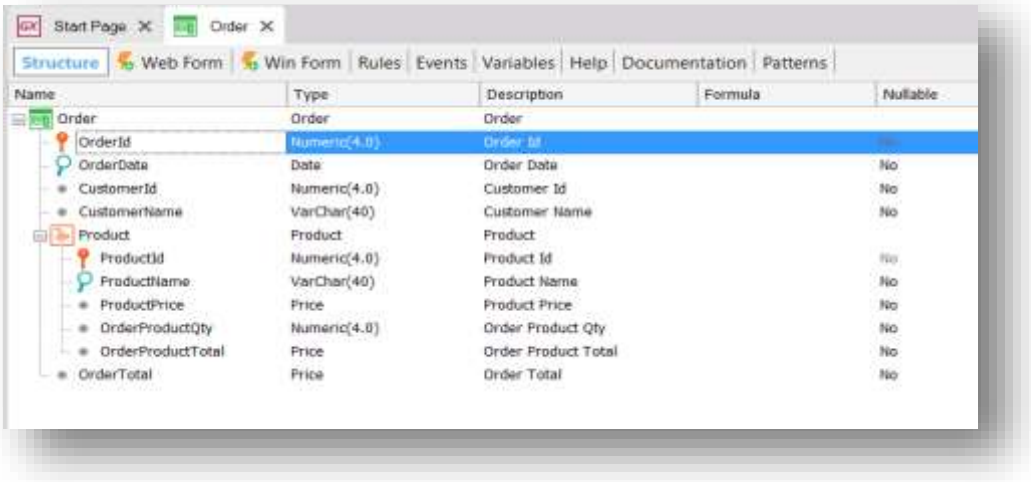
Y llegados a este punto vamos a definir el dominio Price.
Los dominios nos permiten definir tipos de datos de forma centralizada de manera que puedan ser reutilizados. Con esto ganamos en nivel de abstracción y en facilidad para adaptarnos ante el cambio. Por ejemplo: recordemos que en el año 2000 las fechas pasaron de tener el año modelado con 4 dígitos en lugar de 2.

Así que vamos a crear el dominio Price como numérico de 6 dígitos con dos decimales.

Escribimos el nombre Price, signo de "=" y el tipo de dato que va a almacenar. En nuestro caso, numérico, paréntesis, y los 6 dígitos con 2 decimales.

Ingresamos ahora la cantidad de unidades del producto, de tipo numérico, y el total de la línea basado en el dominio Price.

Una vez ingresados los atributos del segundo nivel, presionamos Ctrl y la flecha de la izquierda para volver al nivel del cabezal e ingresar los datos del pie de la orden de compra, también de tipo Price.



Nos posicionamos ahora en el atributo OrderId, y en sus propiedades indicamos que queremos que sea autonumerado. Así que indicamos el valor True en la propiedad Autonumber

Guardamos la Transacción.

Bien. Acabamos de crear la estructura de una Transacción compuesta de dos niveles:

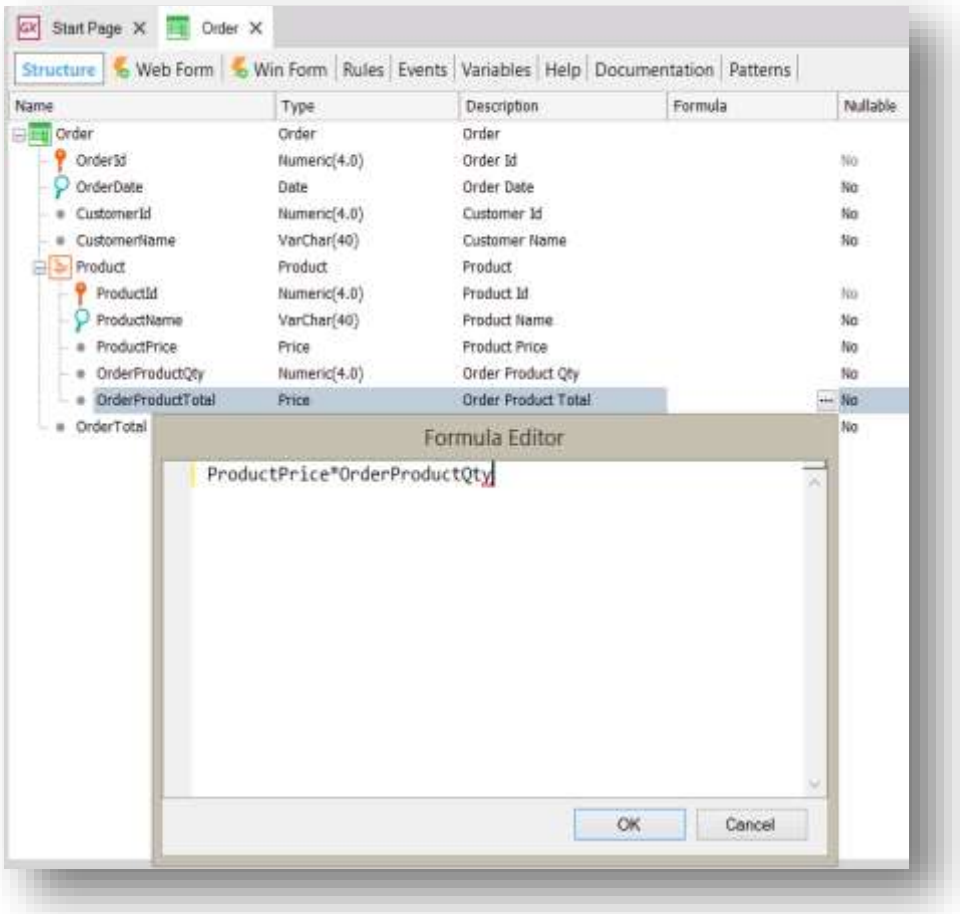
1. Un nivel básico (Order), donde se especifica toda la información necesaria para el cabezal de la orden de compra.

2. Y un nivel anidado correspondiente a los productos, donde se especifica la información de los ítems.

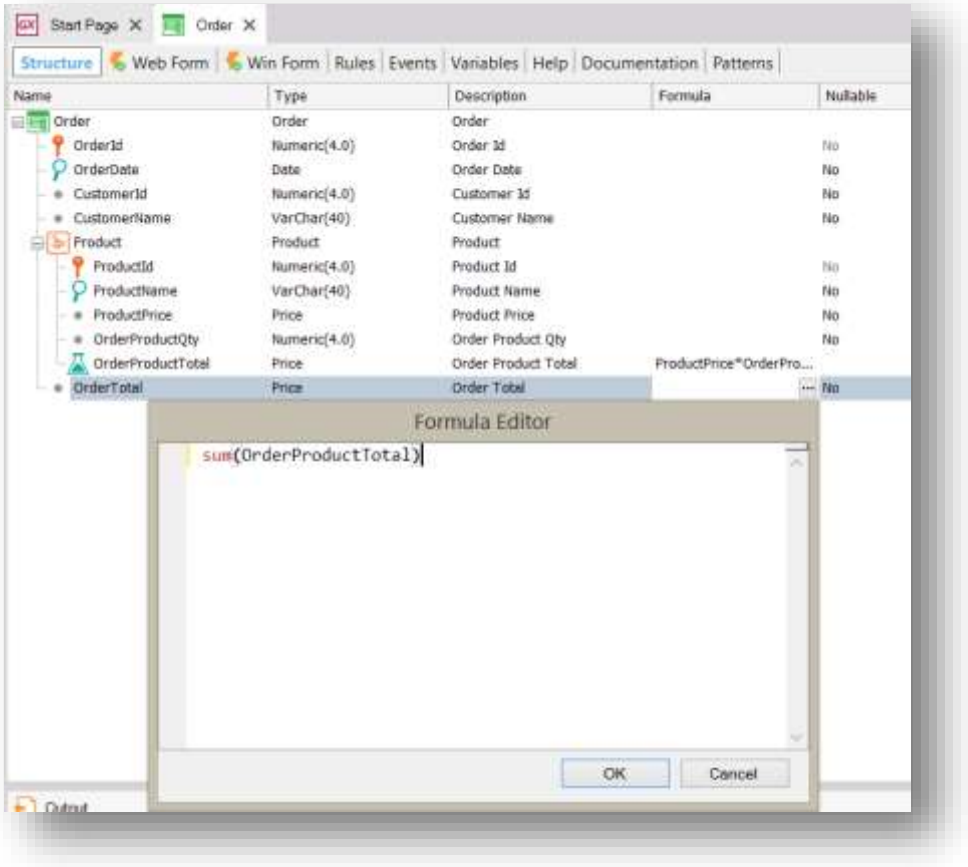
Observemos que tiene sentido que ciertos campos sean calculados en forma automática, como ser el total de la línea y el total de la orden.

Para esto le indicaremos a GeneXus la fórmula para el cálculo de cada uno de ellos.

Así que nos posicionamos en la columna fórmula del atributo que corresponde al total de la línea e indicamos que su cálculo corresponde al precio del producto por la cantidad de unidades.



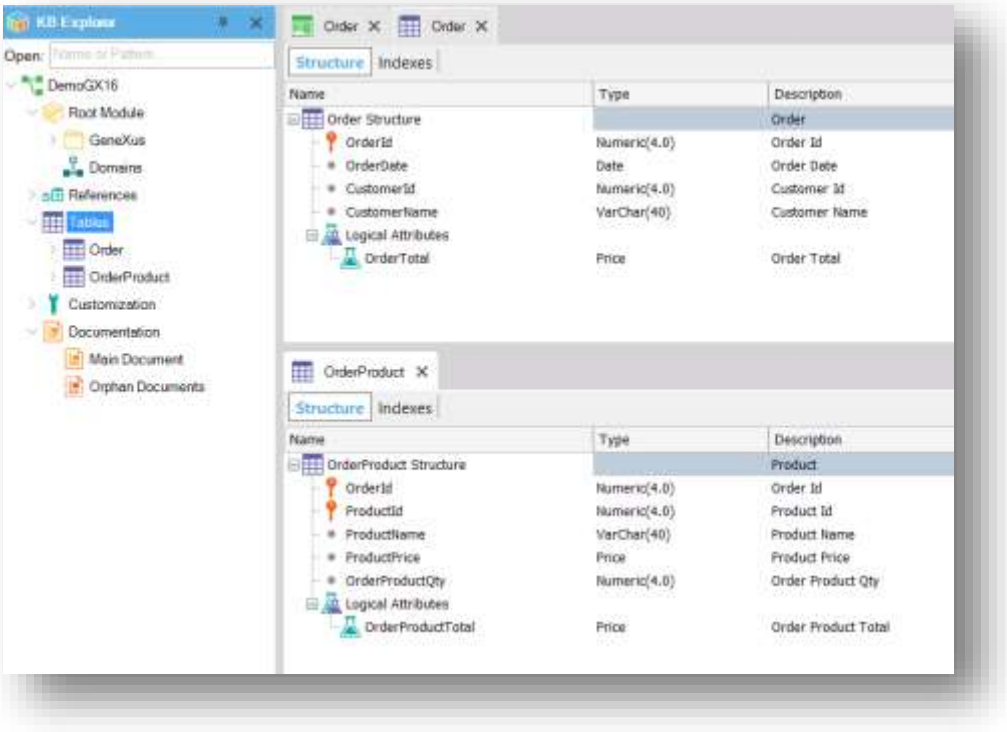
De igual forma, en el atributo que representa el total de la orden, indicamos que su cálculo corresponde a la suma de los totales de las líneas.



GeneXus generará de forma automática el mejor modelo de datos que represente las Transacciones definidas por el Analista GeneXus.

Siempre que se haga clic en el botón Save o Run, GeneXus inferirá el modelo de datos óptimo (3era forma normal sin redundancias) que soporte a las entidades del usuario final representadas por los objetos Transacción. En base a este modelo GeneXus generará una base de datos física para el DBMS definido en su Environment.

Si queremos ver el modelo de datos creado por GeneXus, vamos entonces al menú superior View, y elegimos Tables.



Podemos ver que GeneXus creó automáticamente un modelo de datos normalizado creando dos tablas para soportar el objeto Transacción Order:

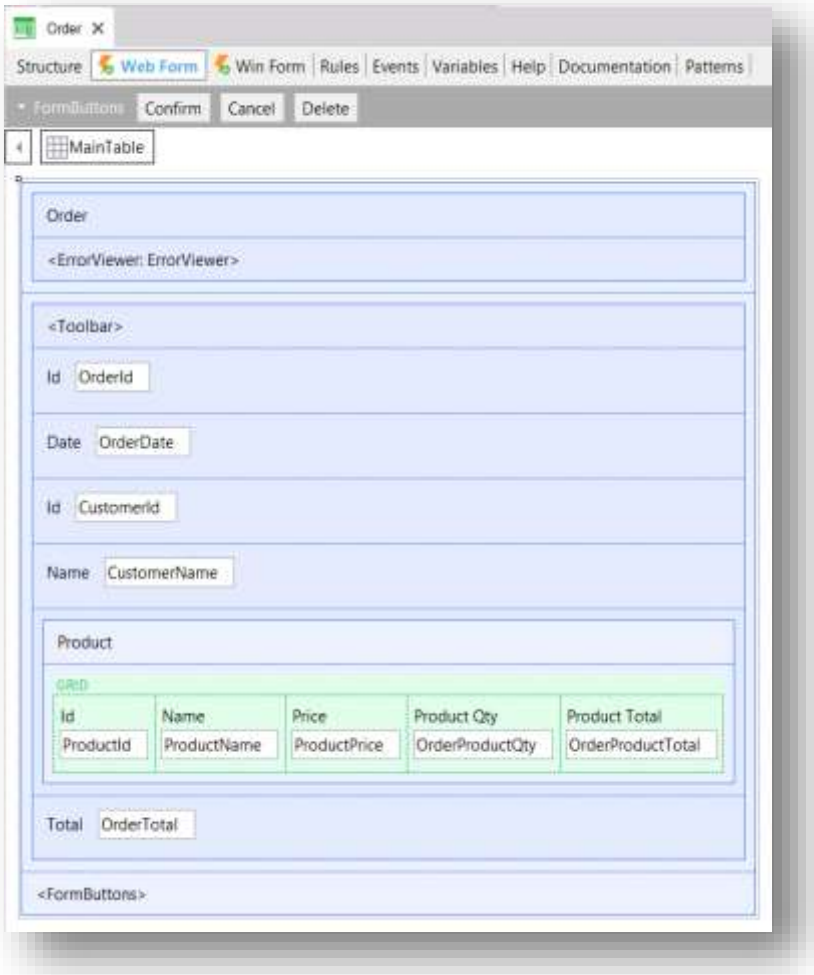
- Order, que corresponde al cabezal de la orden
- Y OrderProduct que corresponde a las de la orden)

Esto representa un beneficio en tiempo y costos gracias a la automatización mediante inteligencia artificial.

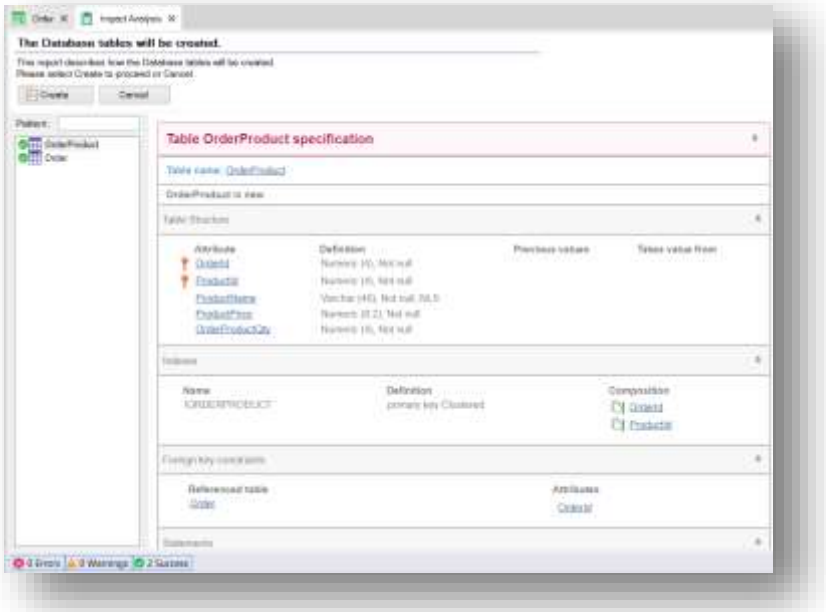
Antes de ejecutar la aplicación veamos el formulario web que fue creado automáticamente por GeneXus a partir de la estructura de la transacción. Dicho web form será el que el usuario final visualizará cuando ejecute la aplicación en una plataforma web y a través de él podrá ingresar, modificar y eliminar registros como veremos en unos minutos.

Estos formularios pueden ser personalizados por el analista de negocio en cualquier momento.

Los web forms están construidos en un editor web abstracto que nos permitirá crear Responsive Web Applications.



Para ejecutar la aplicación presionamos F5 o RUN desde la Toolbar.



Esta pantalla que aparece aquí es el reporte de creación de la base de datos. GeneXus nos declara cuáles son los scripts que ejecutará sobre la Base de Datos.

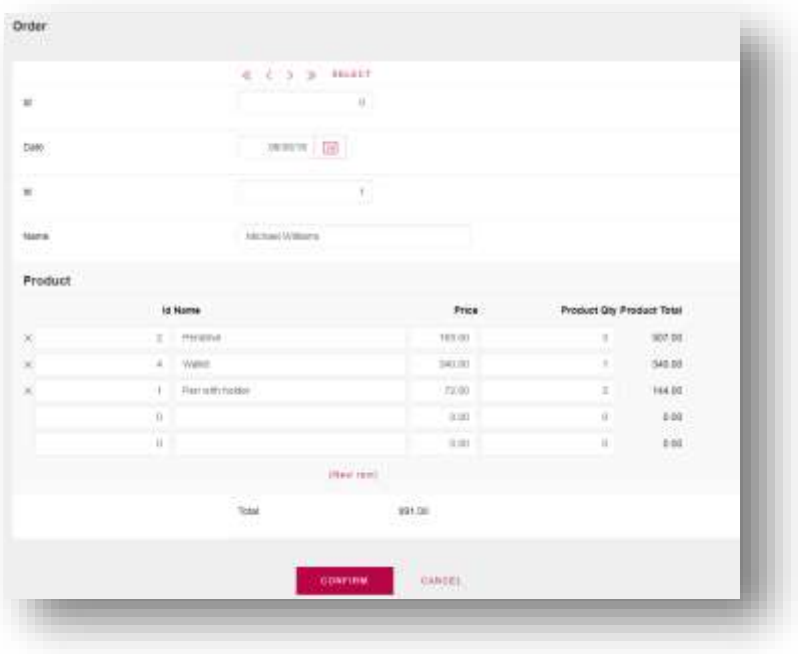
Hacemos clic en “Create”.

GeneXus generará entonces los programas necesarios para ejecutar la aplicación web, así como todos los programas para crear y mantener la base de datos normalizada.

Cuando termine este proceso se abrirá un navegador web y se cargará un menú con los objetos creados en GeneXus.

El Developer Menu es un archivo XML que incluye todos los objetos ejecutables. Es un menú auxiliar para que el Analista GeneXus pueda probar la aplicación.

Hacemos clic entonces en la opción Order para ingresar algunos registros de órdenes de compra.



Indicamos la fecha, e ingresamos algunos productos.

Una vez completado el ingreso de todos los datos presionamos Confirm.

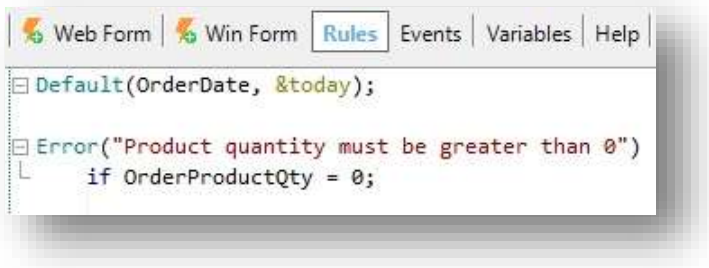
Las Reglas en GeneXus son el medio para definir la lógica del negocio asociada a cada objeto. Son escritas de forma declarativa y GeneXus decide de manera inteligente qué regla aplicar y cuándo hacerlo.

Estas Reglas juegan un rol muy importante en los objetos de tipo Transacción porque permiten describir su comportamiento, por ejemplo: asignando valores predeterminados, definiendo controles de datos, etc.

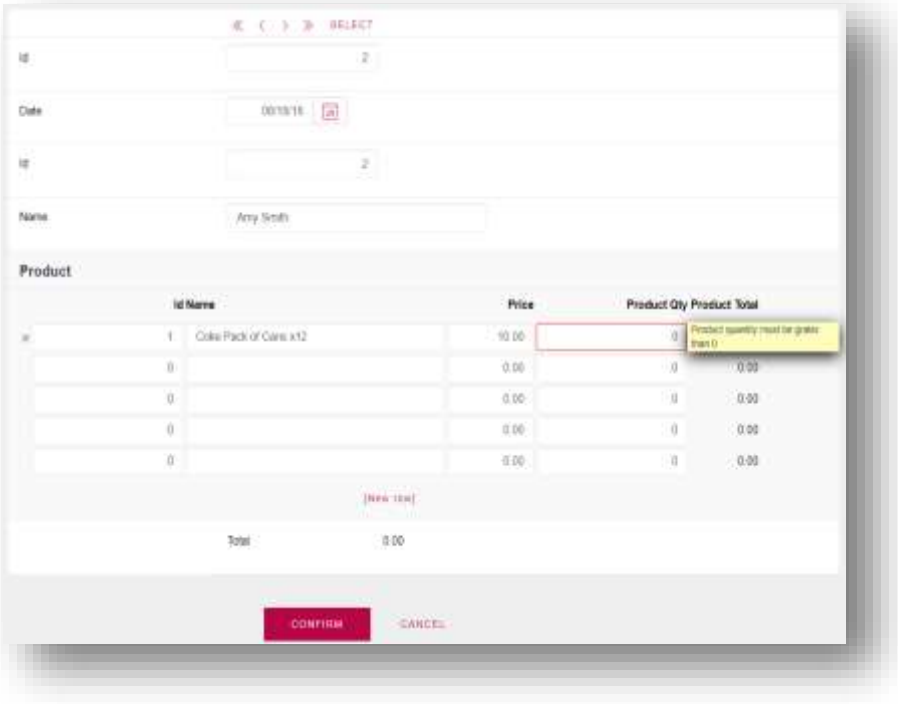
Supongamos entonces que nuestro cliente, nos solicita que la fecha de la orden de compra tome por defecto el valor de la fecha del día. Para lograr esto vamos al sector de las Rules de la transacción Order, y agregamos la regla Default desde el menú superior Insert / Default

Y completamos la declaración de esta regla de la siguiente forma…

A continuación agregamos otra regla que establece un error si la cantidad del producto ingresada es 0.



Vamos a guardar los cambios y a verlos en ejecución.



De esta forma hemos visto qué fácil es describir reglas de negocio en GeneXus.

En esta última interacción con el formulario, nos resultó raro tener que digitar nuevamente el nombre de los productos. ¿Acaso no sería mejor contar con un catálogo de productos en donde esté su definición y su precio? Esta observación es totalmente acertada por lo cual vamos a cambiar nuestro sistema para que cuente con un catálogo de productos.

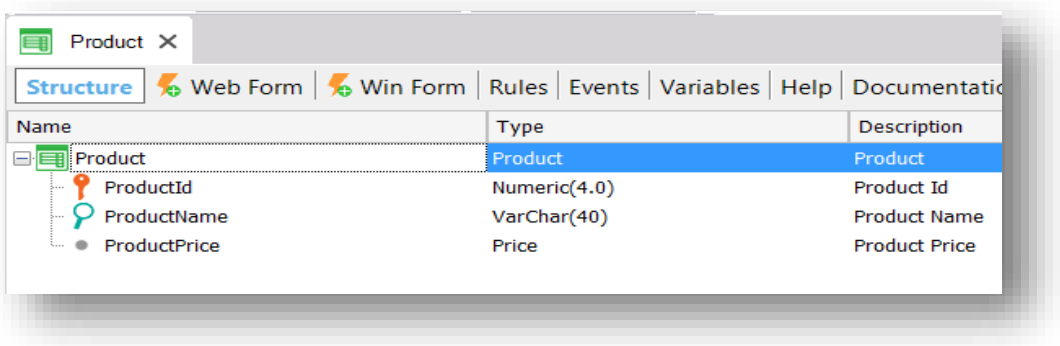
En el modelado actual, el producto formaba parte de la Transacción Order, pero entendemos que debería ser una entidad independiente de la orden, por lo tanto corresponde que sea definido como una Transacción en sí misma.

No hay ningún problema con esto, dado que GeneXus se encargará de mantener la base de datos normalizada en función de nuestro esquema de transacciones.

Así que procedemos a crear la Transacción Producto.

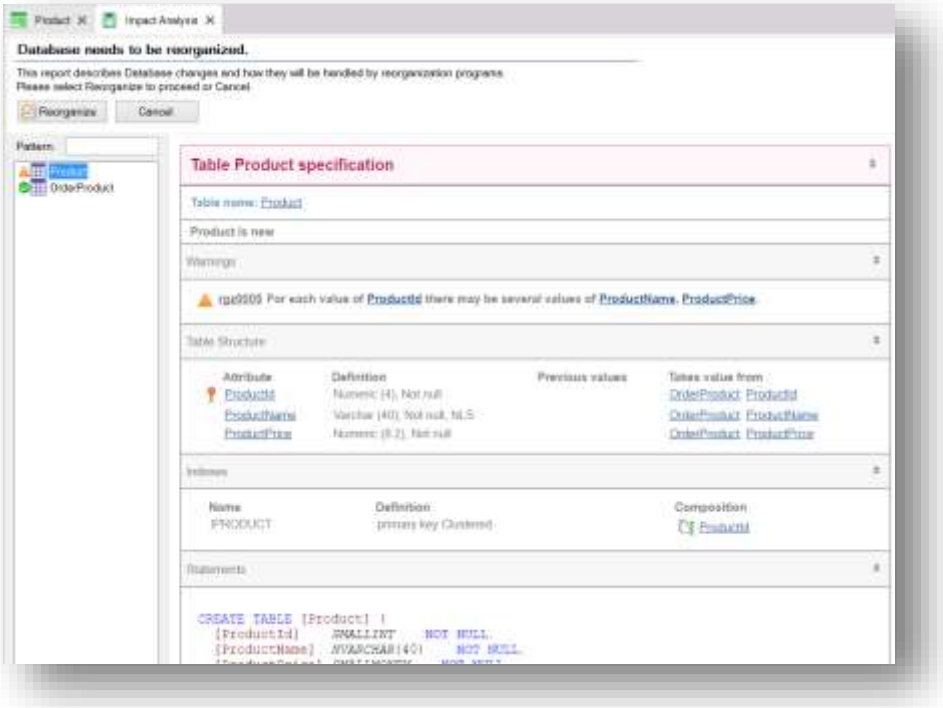
Los datos inherentes al producto que ya teníamos en la Transacción Order son los siguientes:

- El identificador
- El nombre
- Y el precio



Notemos que no fue necesario definir el tipo de datos de cada uno de los atributos, dado que GeneXus identificó que ya habían sido definidos previamente en la transacción Order.

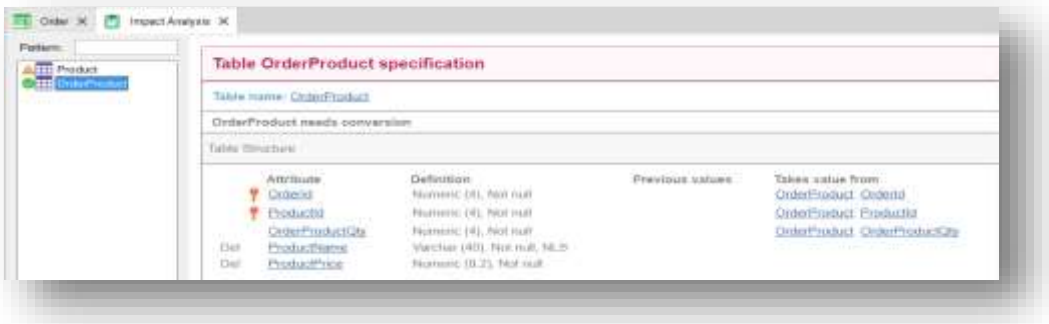
Guardamos y presionamos F5.



GeneXus realiza el análisis de impacto y analiza qué cambios debe realizar en la base de datos en función de los cambios que hayamos hecho en las transacciones y nos declara entonces por cada objeto, cuáles serán dichas modificaciones.

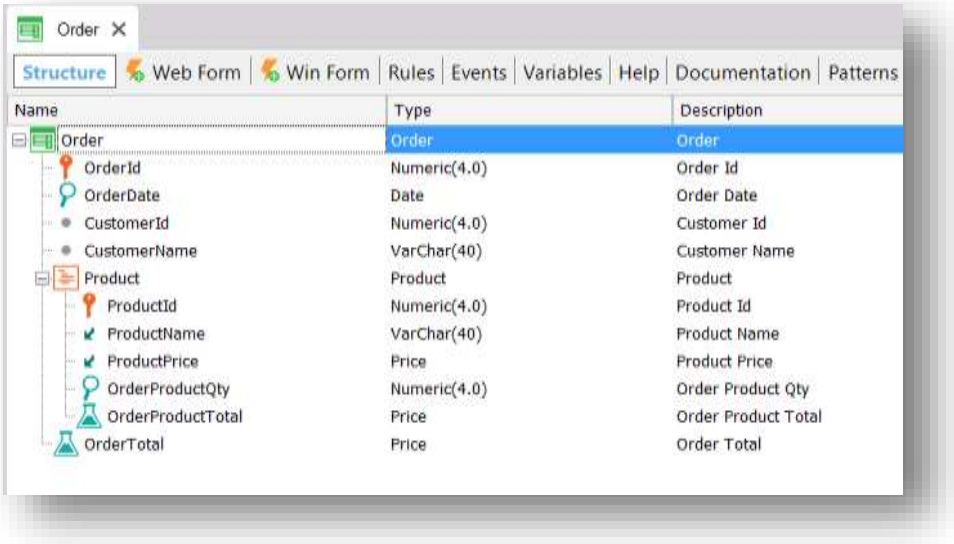
Vemos que GeneXus nos indica que deberá crear la tabla correspondiente al Producto y nos alerta en un warning de especificación, que para un mismo valor de ProductId pueden existir varios valores de ProductName y ProductPrice.

También nos indica que deberá eliminar los atributos ProductName y ProductPrice de la transacción Order, ya que los mismos pasarán a formar parte de la transacción Product y podrán ser inferidos a través del valor de ProductId. Recordemos que GeneXus nos asegura que mantiene la base de datos normalizada, por lo cual no podemos tener un atributo secundario en más de una tabla física.



Reorganizamos. Una vez finalizado la reorganización del modelo de datos, al navegar por los registros de la transacción Product, vemos que tenemos la información de los productos que habíamos ingresado a través de la Orden. Esta automatización de GeneXus es lo que hace viable el cambio en el modelo de datos. GeneXus se encargó de modificar la estructura y de la migración de datos de una tabla a la otra.

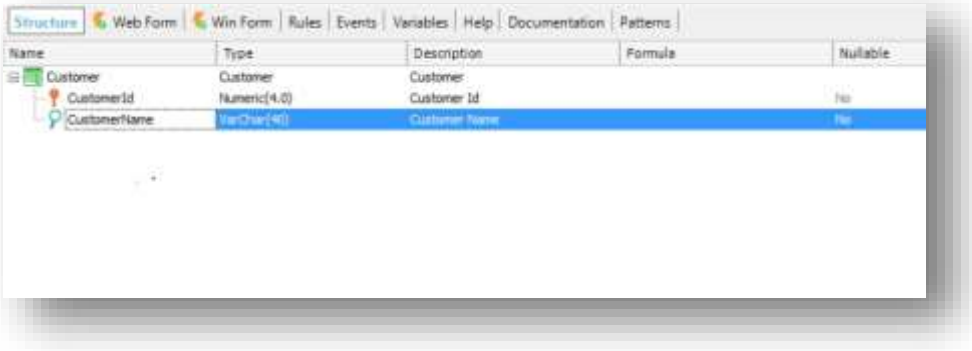
Pero… ¿qué sucedió con la información de Producto que estaba en la Transacción Order?



Veamos que los atributos que ahora pertenecen físicamente a la tabla de Producto, figuran con “una flecha hacía abajo” en la Transacción Order. Esto denota que están siendo inferidos a partir del atributo ProductId.

Avanzando un poco más con el relevamiento, nos indican que en la empresa cuentan con un maestro de clientes con lo cual no será correcto tener los datos de los clientes como propios de la transacción Order. En lugar de esto deberían ser modelados en una transacción independiente.

Creamos entonces la transacción Customer,…… y la guardamos.

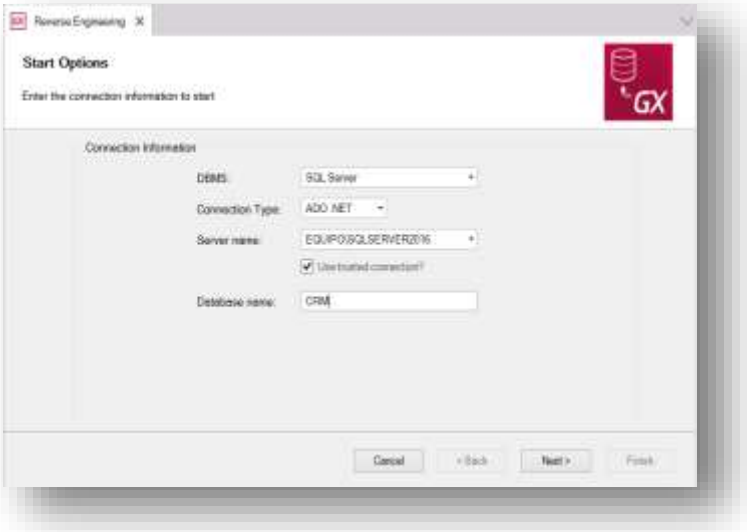


Los clientes están registrados en el CRM de la empresa y se necesita que sean tomados de allí para poder registrar las órdenes de compra.

Generalizando esto, diremos que GeneXus permite que nos integremos con cualquier sistema externo y tomar información de su base de datos para que sea utilizada en el sistema a desarrollar.

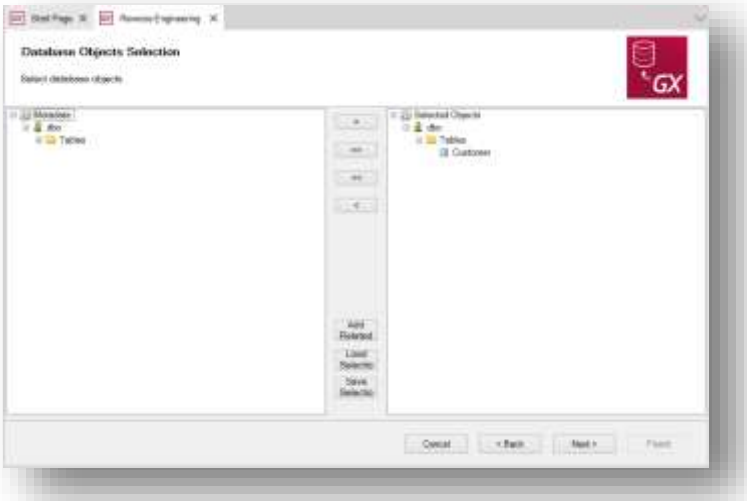
Para hacer esto vamos al menú *Tools* y seleccionamos la opción *Database Reverse Engineering*

Desde aquí indicamos el nombre del servidor de base de datos y el nombre de la base de datos a la cual deseamos integrarnos.

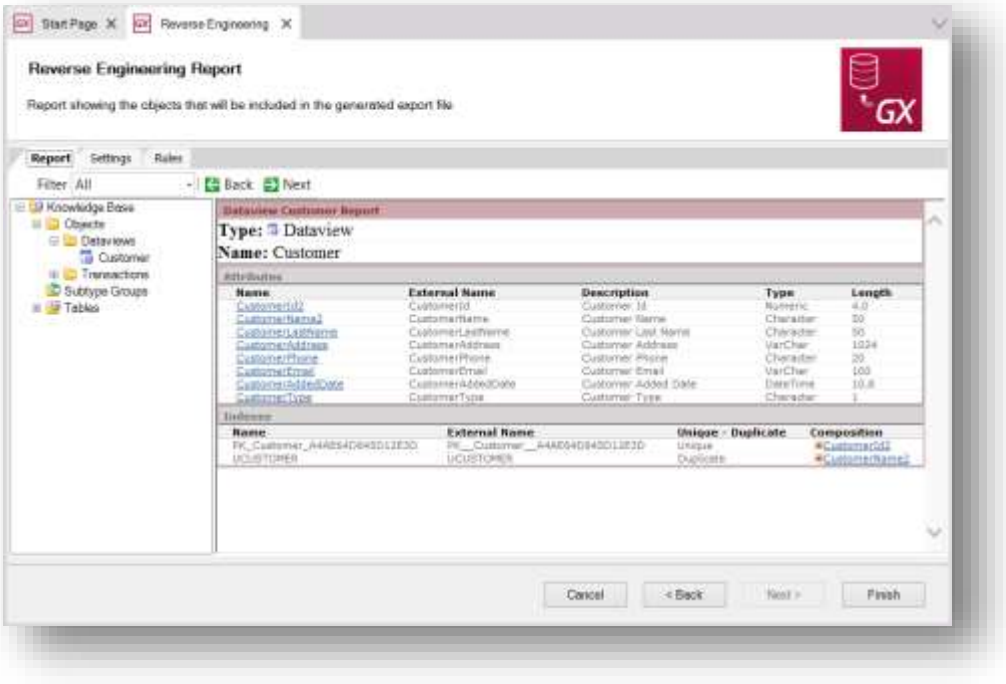


Al dar clic en el botón *Next* se nos despliega un listado con todas las tablas de la base de datos para que seleccionemos las que necesitamos.

Seleccionamos la tabla Customer.



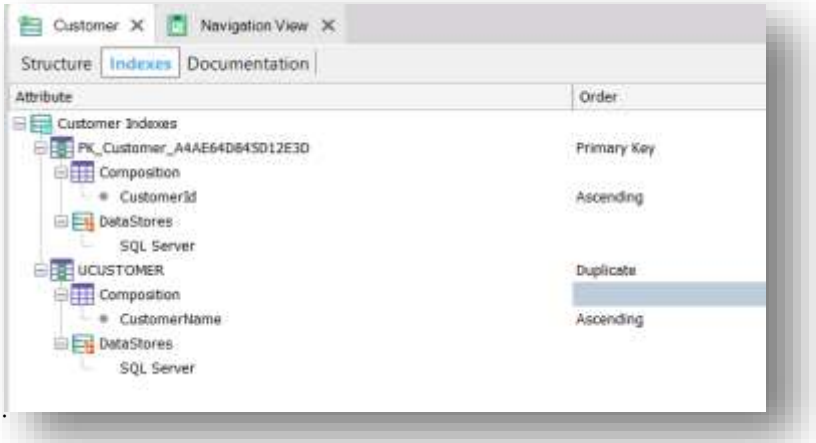
Al hacer clic nuevamente en el botón ***Next*** GeneXus nos indica que creará el Data View Customer

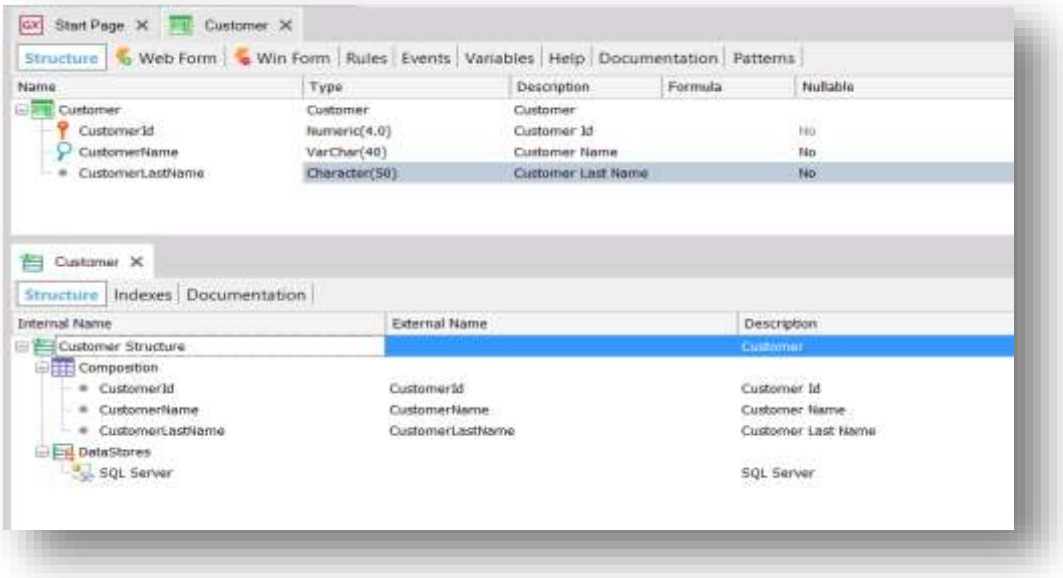


Vemos que a los atributos CustomerId y CustomerName les agregó el número 2, esto se debe a que ya existían en nuestro modelo. En caso de querer que sean los mismos, se lo debemos explicitar. Para esto vamos al Data View Customer y le cambiamos el nombre a estos atributos.

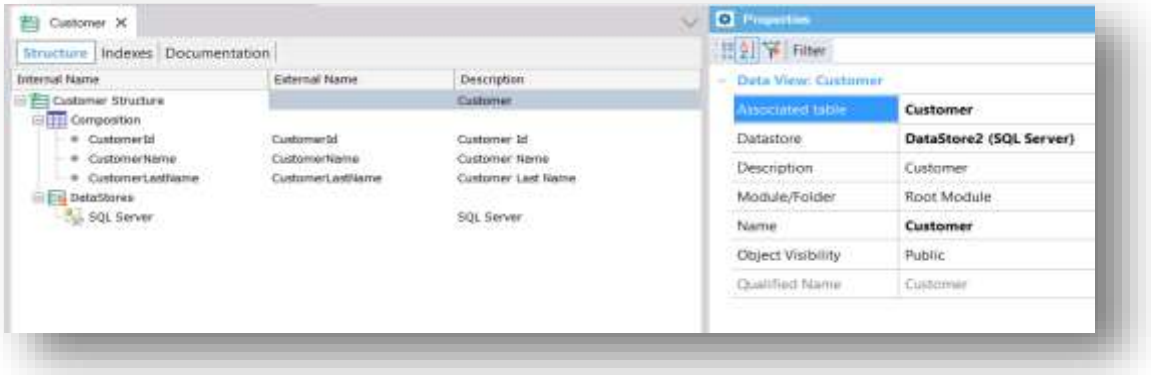
Eliminamos también los atributos que no vamos a utilizar. Dejamos solamente CustomerId, CustomerName y CustomerLastName. Así que eliminamos CustomerAddress, CustomerPhone y CustomerEmail.

También verificamos la definición de los índices para que el nombre de los atributos sean los correctos.



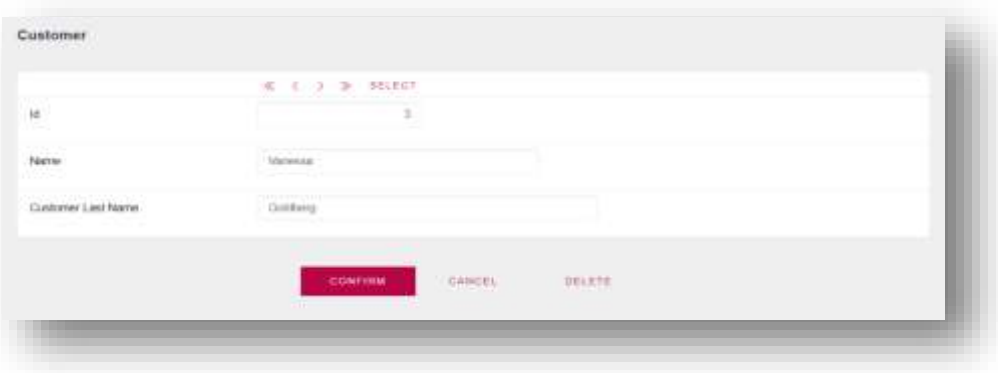


Al haber realizado el cambio de nombres, el Data View pierde el contenido de la propiedad *Associated table* , así que vamos a asignarle el valor Customer.



Presionamos F5 y vemos que en el Análisis de Impacto nos indica que la transacción Customer está asociada al Data View Customer y que no va a ser reorganizada, o sea, no nos va a crear una tabla física asociada.

Ejecutamos y vemos que contamos con la transacción Customer que toma datos de la tabla Customer del CRM.

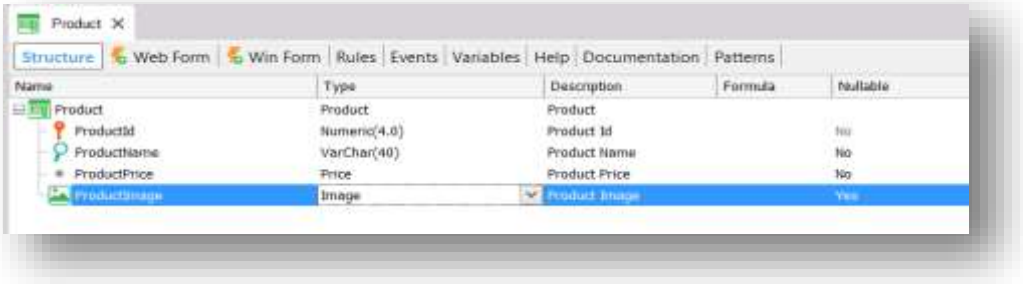


Mostraremos ahora cómo podemos generar aplicaciones más ricas a nivel de usabilidad de interfaz de usuario. Vamos a aplicar el pattern Work With for Web y vamos a ver en ejecución las facilidades que nos brinda.

Los patterns permiten definir y encapsular el comportamiento de forma automática y replicarlo en forma masiva en base a una plantilla de definición.

Veamos un ejemplo.

Para hacerlo más real vamos a agregarle la foto al producto y a continuación le aplicaremos el pattern Work With for Web.

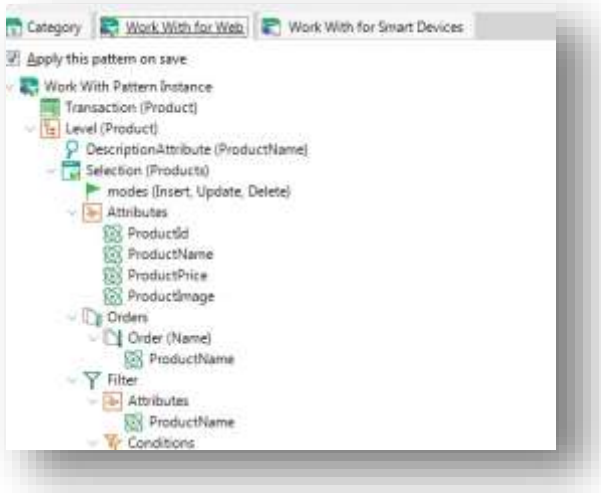


El tipo de datos Image es uno de los que llamamos dominios semánticos, que en su definición ya tiene implícito su comportamiento. Por ejemplo, este tipo de datos nos pondrá un selector de archivos en el formulario para poder subir una imagen. Hay otros dominios semánticos como ser Address, Phone, Email, etc.

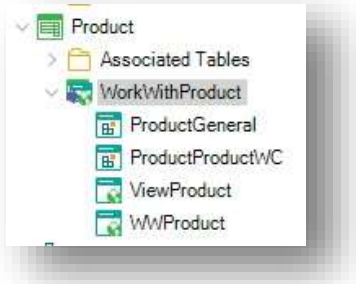
Para aplicar el pattern Work With for Web, vamos a la solapa Patterns, seleccionamos Work With for Web y hacemos clic en esta casilla que dice *Apply this pattern on save*.

Vale mencionar que lo que estamos viendo es lo que llamamos la instancia del pattern aplicado a la transacción.

Desde aquó podemos agregar nuevo órdenes, nuevos filtros, personalizar las columnas, agregar acciones, etc. Guardamos.

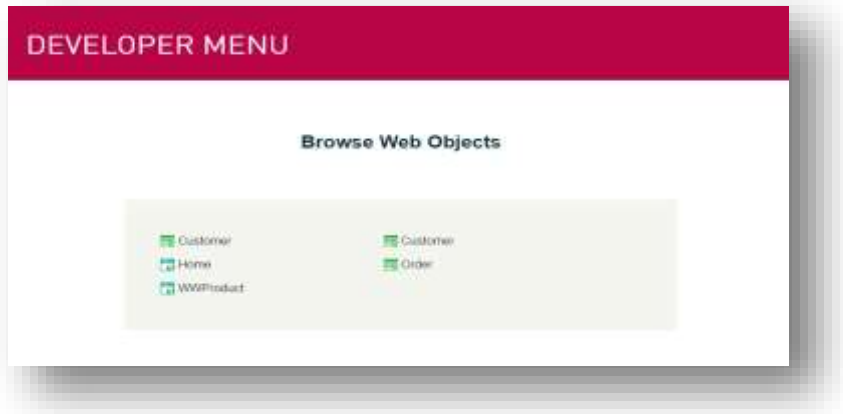


Si hacemos doble clic sobre la transacción Product vemos que automáticamente aparece la lista de objetos creados automáticamente por la aplicación de este pattern.

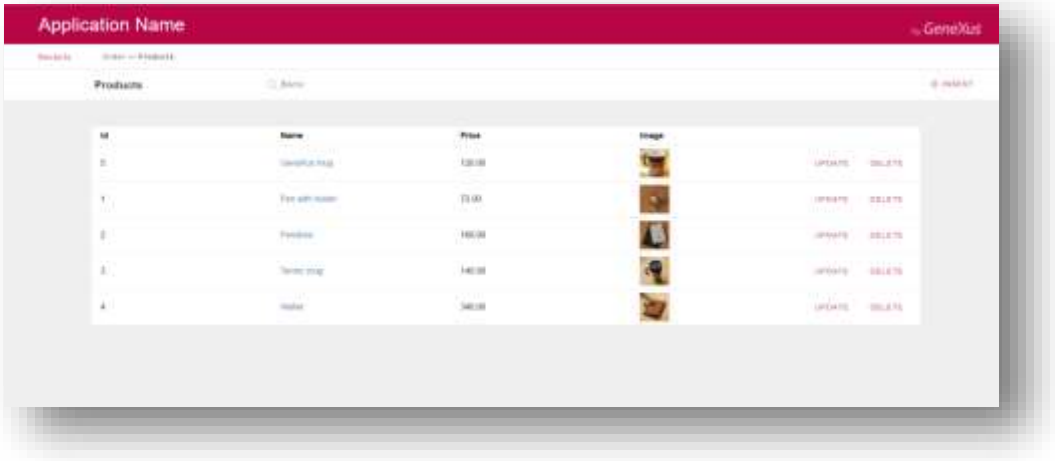


Vamos a ejecutar. Presionamos F5.

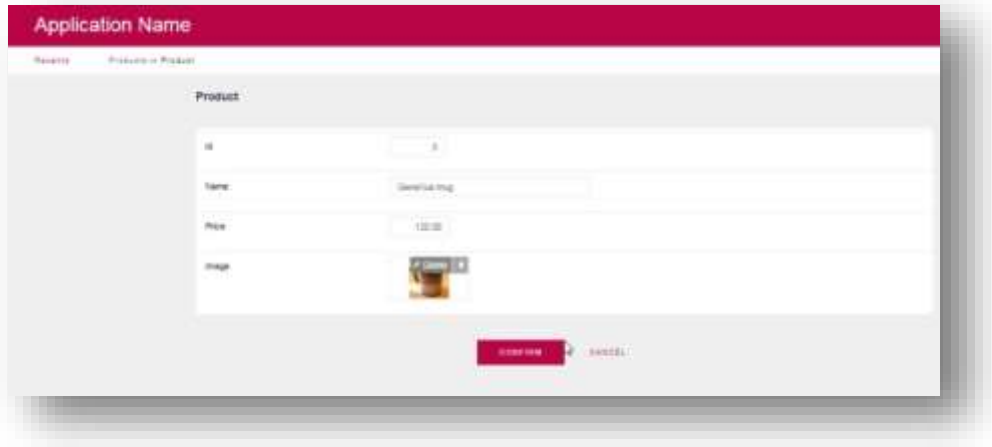
Al ejecutar entonces vemos que cambió el acceso directo a la transacción Product y que ahora se accede mediante WWProduct.



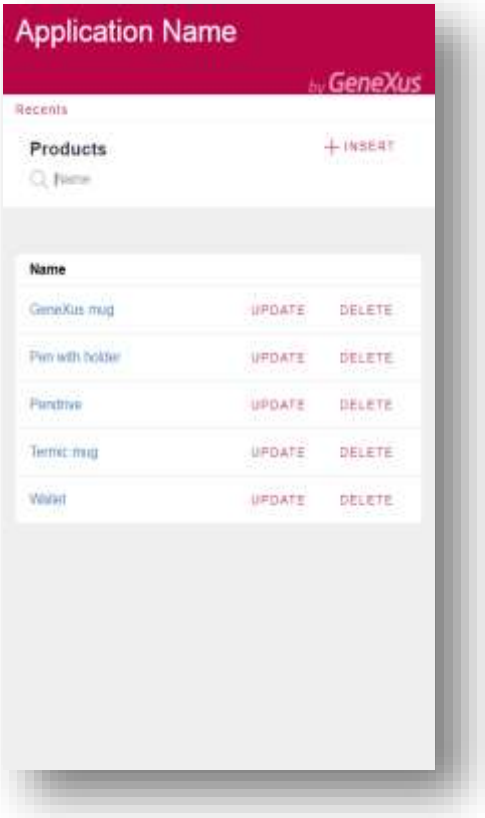
Vemos que GeneXus generó de forma automática una nueva página web con el listado de todos los productos, desde la cual podemos buscar, insertar, editar, eliminar y visualizar la ficha del producto.



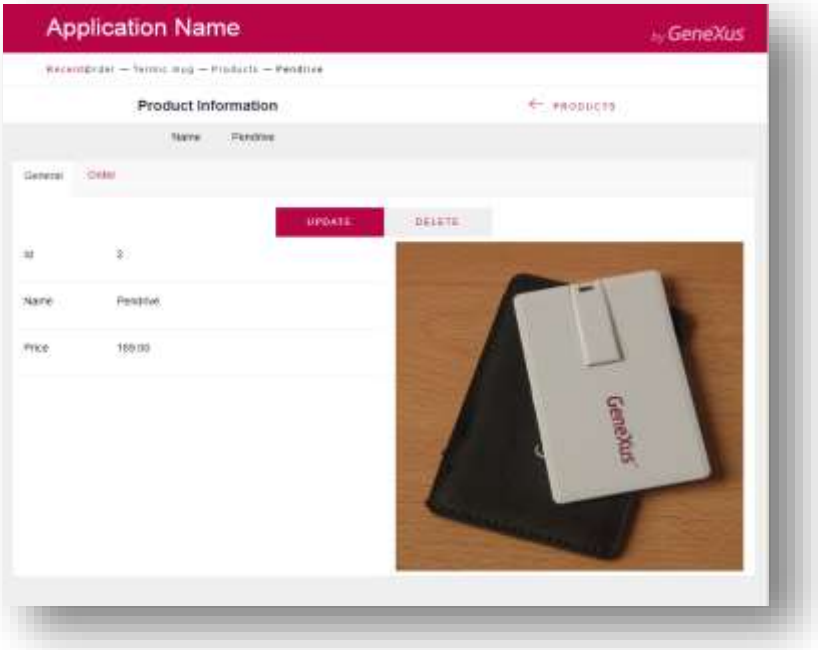
A modo de ejemplo, vamos a ingresar un nuevo producto. Presionamos *Insert*.



Si achicamos el navegador vemos que la aplicación es responsiva, adaptándose al tamaño de pantalla disponible y priorizando mostrar la información más relevante.



Si hacemos clic sobre el nombre de un producto accedemos a la ficha del mismo. En donde veremos toda su información así como también la información de las entidades relacionadas. Por ejemplo, un tab con las órdenes en donde ese producto fue incluido.



Esto hace que la aplicación sea muy amigable para navegar, teniendo al alcance de un clic toda la información que necesitamos referente a la entidad con la que estamos trabajando.

Tómese un segundo para darse cuenta que todo esto lo generó GeneXus con 1 solo clic.

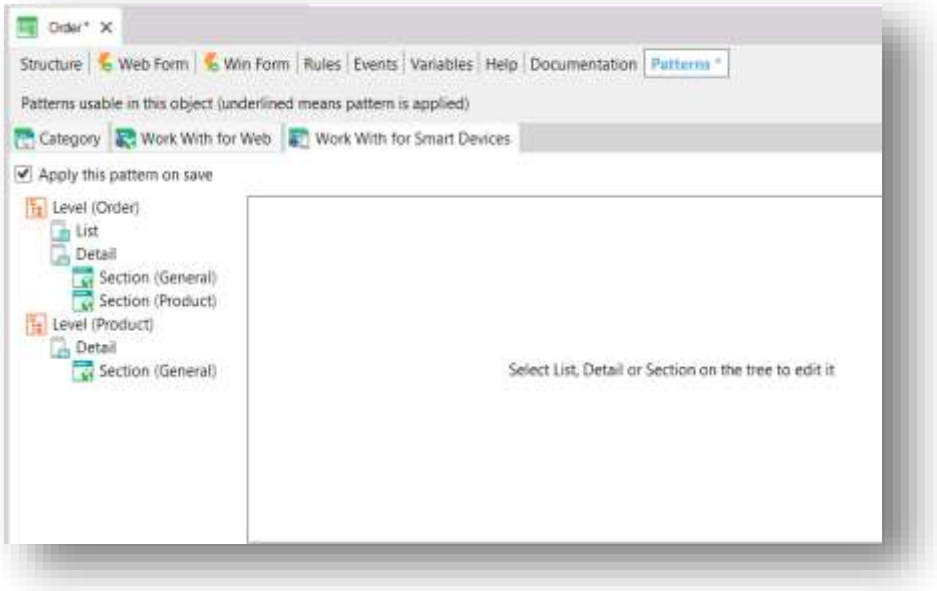
Vamos a proceder ahora a generar una aplicación para dispositivos móviles con sistema operativo Android. La aplicación será generada utilizando código nativo Android.

Para esta aplicación móvil nos va a interesar tener las siguientes pantallas:

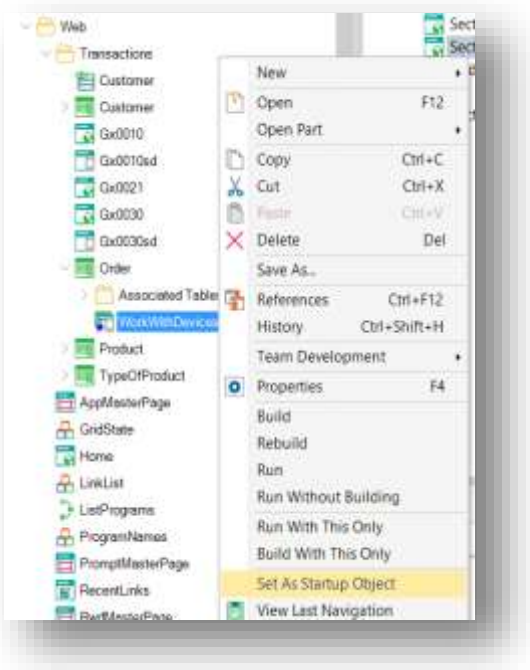
- Un menú de acceso a opciones
- El listado de órdenes de compra ordenadas por fecha en forma descendente.
- Para cada orden de compra, la lista de los productos incluidos con los totales y subtotales.
- Un catálogo de productos, con el detalle de cada producto.

Vamos a comenzar por el listado de órdenes de compra.

Para esto vamos a la solapa Patterns de la transacción Order y aplicamos el pattern Work With for Smart Devices.



Guardamos y tenemos que establecer un objeto de smart devices como start up object, así mientras estamos prototipando, al hacer clic en RUN automáticamente se levantará la aplicación en el emulador. Localizamos la transacción Order en la ventana de exploración objetos y hacemos clic derecho sobre el objeto WorkWithDevicesOrder. Lo seteamos como Start Up Object



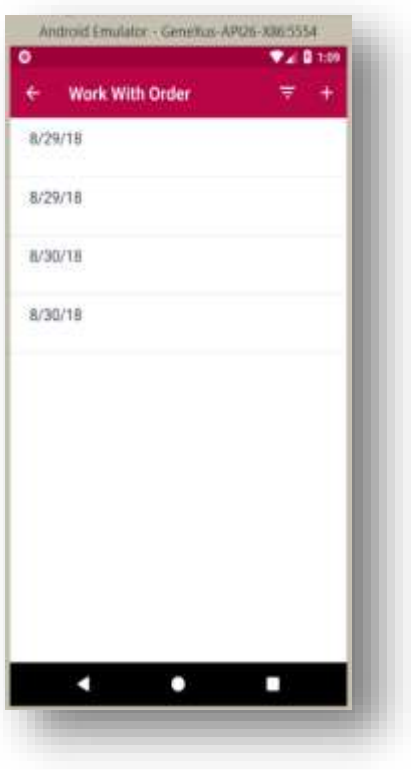
Cabe destacar que si se tiene un dispositivo Android conectado por usb al equipo, con el seteo de depuración por usb activada, al hacer clic en RUN la aplicación se instalará automáticamente en el dispositivo, y se podrá probar allí mismo. Esto es de gran valor para hacer el testing más real.

Presionamos RUN para que GeneXus genere la aplicación para Android.

GeneXus está generando todos los programas, pantallas y servicios web necesarios para crear la aplicación móvil utilizando código nativo, en este caso Android.

Como GeneXus es quien genera los programas, nos independiza de la tecnología. Yendo a un caso concreto, cuando iOS cambió su lenguaje de generación de Objective-C a Swift, mientras que muchos debieron reescribir sus aplicaciones, el usuario GeneXus no tuvo que hacer absolutamente nada, ya que el generador cambió para que ahora se generen las aplicaciones utilizando Swift.

Vemos que GeneXus creó una pantalla con el listado de las órdenes. Para cada una de ellas se muestra por defecto su fecha.



Sería deseable que:

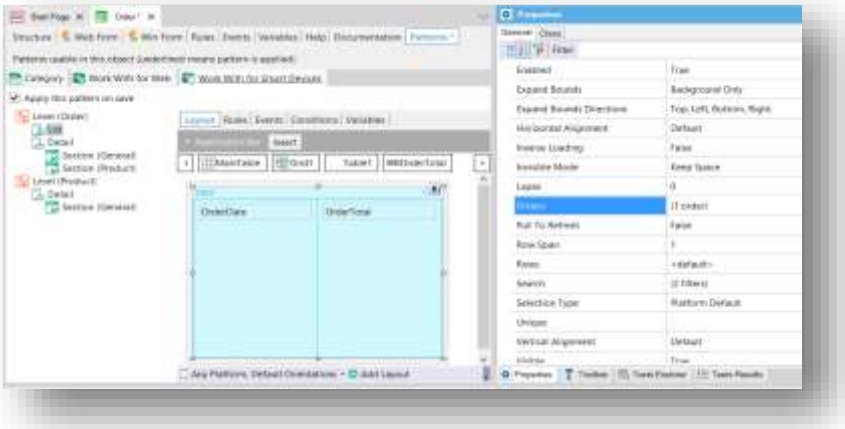
- el título de la pantalla sea *Orders*.
- Que para cada orden se muestre también el total.
- Que se listen en orden de fecha descendente.
- Y que el contenido de la línea aparezca centrado verticalmente.

Vamos a hacer estos agregados.

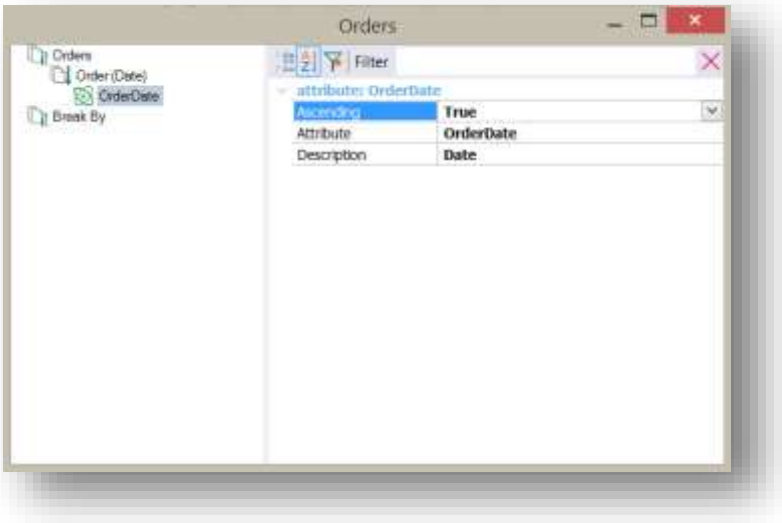
Nos posicionamos en el nodo List de la instancia Work With for Smart Devices de la transacción Order, y modificamos el valor de la property *Caption* para ponerle *Orders.*

Desde la Toolbox arrastramos el atributo OrderTotal a la grilla. Y modificamos la property *LabelPosition* con el valor *None* para que no se muestre la descripción.

A continuación hacemos clic en la Grilla y nos posicionamos en la propiedad *Orders*



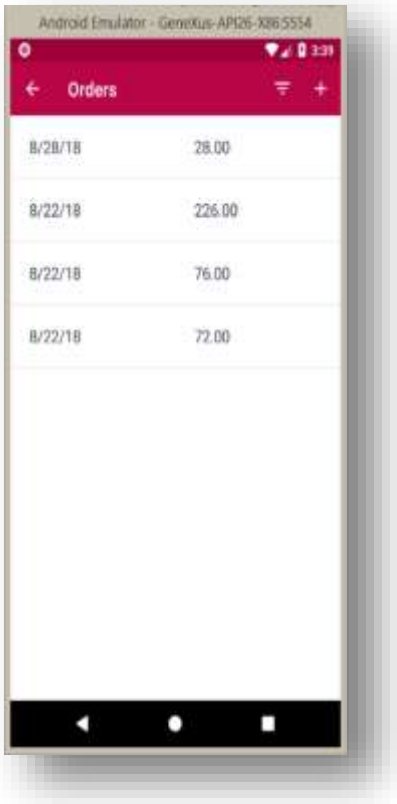
Al hacer clic se nos despliega una pantalla para cambiar los órdenes especificados.



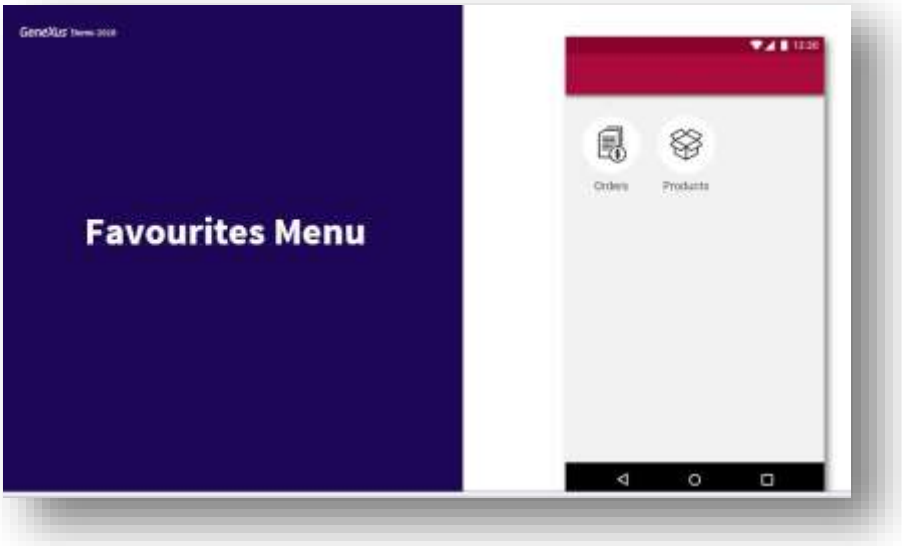
Seleccionamos OrderDate y para cumplir con nuestro requerimiento, cambiamos el valor de *Ascending* a *False*.

Por último, cambiamos el valor de la propiedad *Vertical Alignment* de ambos atributos a *Middle.*

Volvemos a presionar RUN para ver los cambios en ejecución.



Ya vimos lo fácil que es modificar el contenido de las pantallas. Ahora vamos a avanzar un poco más rápido y vamos a aplicar el diseño que nos envió el diseñador.



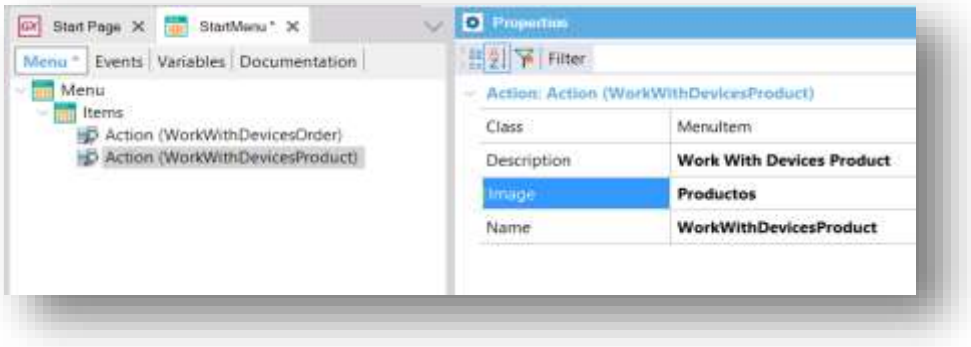
Vemos que la primera pantalla es un menú de acceso con opciones.

Para esto vamos a crear en GeneXus un nuevo objeto del tipo Menu for Smart Devices.

Ya hemos agregado el pattern Work With for Smart Devices a la transacción correspondiente al Producto.

Vamos ahora a crear el menú de acceso.

Agregaremos una acción al menú por cada acceso que deseemos tener.



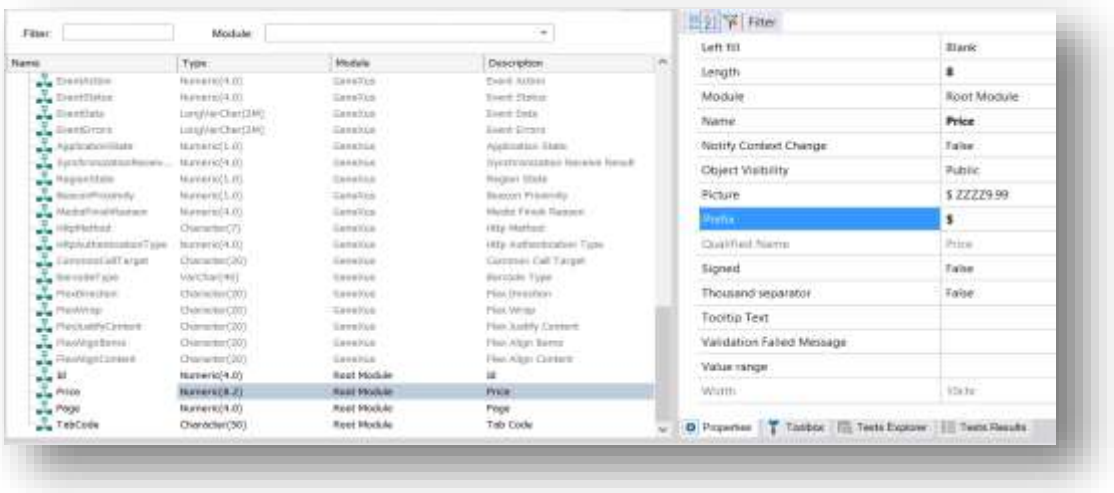
Así cargamos los accesos a las órdenes y los productos.

Podemos cargar también una imagen en la propiedad Image para incluir los íconos que nos envió el diseñador.

Y hacemos lo mismo para seleccionar el ícono correspondiente a los Productos.

Lo salvamos y lo seteamos como objeto Start up. Presionamos RUN.

Se desea que el total se muestre con el signo de $, y tiene sentido que este mismo criterio se utilice para todos los montos del sistema. Así que vamos a la definición del dominio Price y le vamos a agregar un prefijo utilizando la property *Prefix*.



También vemos que se desea que el total de la orden se vea en negrita.

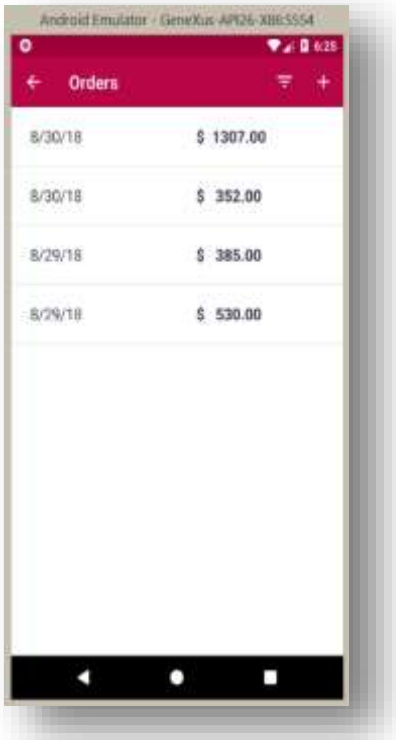
Para lograr esto, abrimos el objeto de nombre CarmineAndroid (del tipo Theme). Y mencionemos que los Themes encapsulan la definición de la estética de los componentes. Observemos que para cada tipo de componente (por ej: Atributos, Tablas, Grillas, Botones) tenemos varias clases definidas. Vamos a crear entonces una nueva clase de atributo con fuente bold para poder utilizarla en el total de las órdenes.

Así que vamos al nodo de Atributos, hacemos clic derecho y seleccionamos la opción *Add class*. Le ponemos como nombre AttributeBold.

GeneXus nos copia la definición de la clase padre y nos permite personalizar su estética a través de las propiedades. Vamos ahora a la propiedad *Font Weight* y elegimos *bold*. Guardamos.

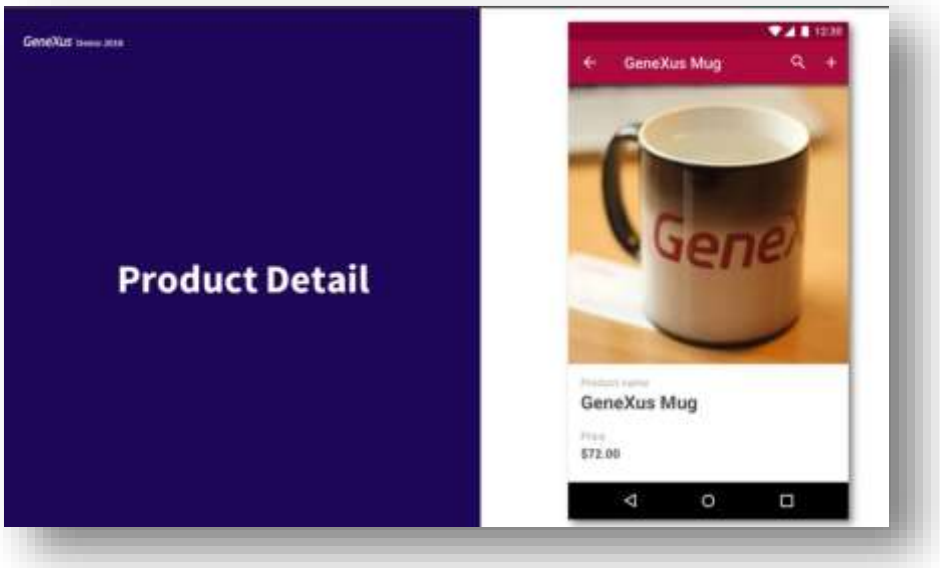
Por último, vamos al objeto OrderList para cambiarle la propiedad correspondiente al atributo del total de la orden, y tome la nueva clase definida.

Seleccionamos RUN para ver los cambios.



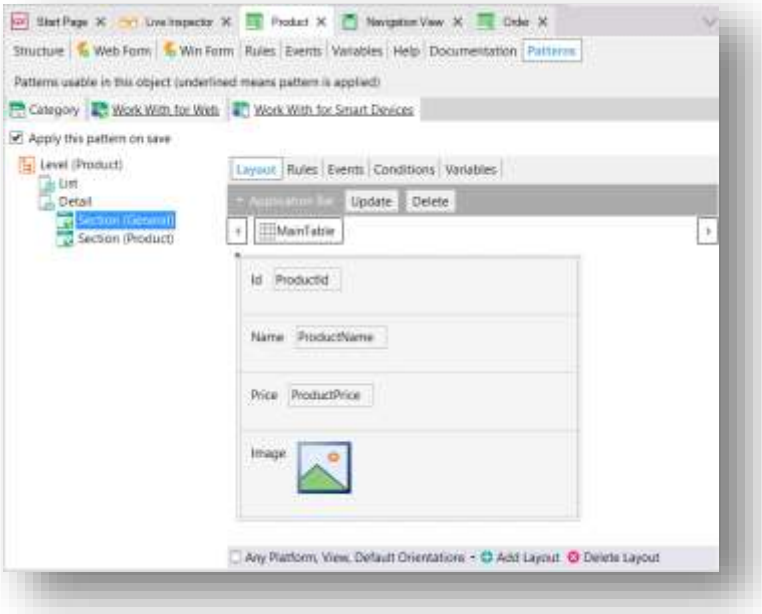
Vemos que efectivamente se aplicaron los cambios y esta pantalla ya cumple con los requerimientos estéticos solicitados.

Y vamos a nuestra última pantalla.



Hasta el momento, la pantalla creada por defecto por GeneXus para mostrar la información general del producto, luce así:

Desde el objeto Work With devices Product, vamos a la Sección general para editar esta pantalla:



Vamos a realizar los siguientes cambios:

Al atributo de la imagen le seteamos la propiedad *Label Position* con el valor *None* y lo posicionamos en la parte superior de la pantalla simplemente haciendo drag and drop.

Luego tendremos que editar las propiedades de la clase del theme que está utilizando la imagen. Vemos que la clase utilizada es ViewImage.

Así que vamos al theme y a esta clase ViewImage le vamos a cambiar las siguientes propiedades:

- Width: (Use default, sin especificar)
- Height: (Use default, sin especificar)
- Margin: (Use default, sin especificar)

Guardamos. Luego a la tabla MainTable que es la que contiene propiamente la ficha del producto, le vamos a cambar la clase para que no se muestren las líneas separadoras. Así que le seteamos la clase Table.

También modificamos la propiedad Label position para que no se vea la descripción.

Y vemos los cambios en el emulador.

Resumiendo, vimos que a partir de lo que ya definimos para el sistema web (estructura y comportamiento), podemos generar aplicaciones para dispositivos móviles reutilizando el conocimiento del negocio que encapsula la Knowledge Base.