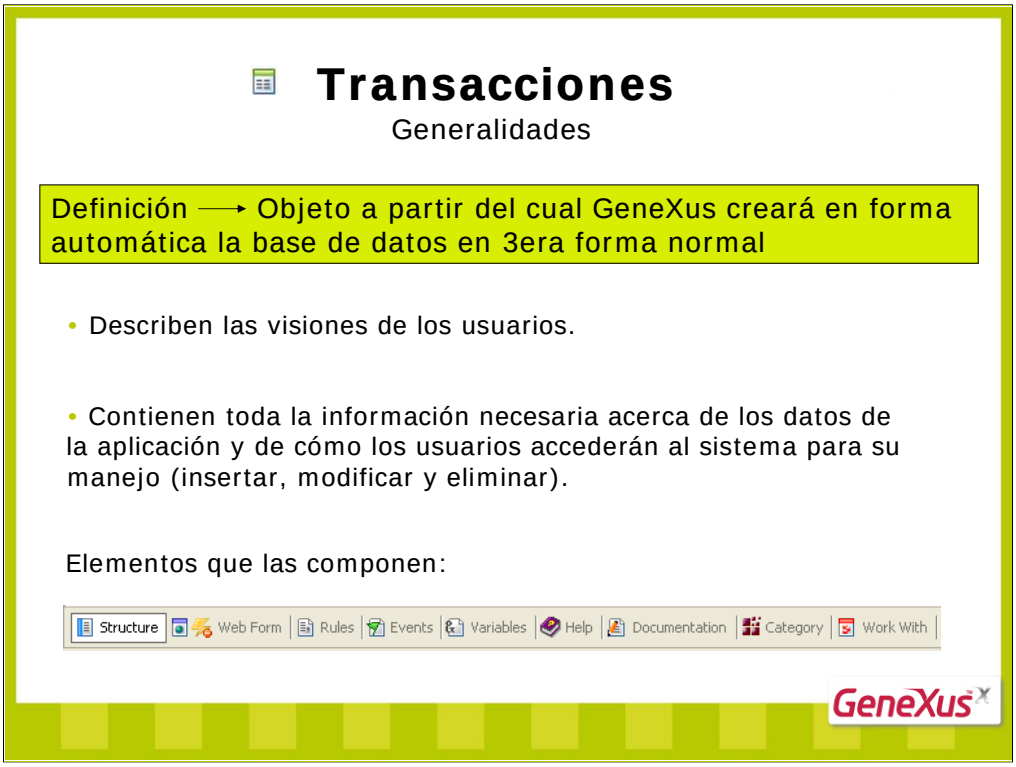
El análisis de toda aplicación GeneXus comienza con el diseño de las transacciones.

Las transacciones permiten definir los objetos de la realidad.

Para identificar cuáles transacciones deben crearse, se recomienda prestar atención a los sustantivos que el usuario menciona cuando describe la realidad.

Además de tener por objetivo la definición de la realidad y la consecuente creación de la base de datos normalizada, las transacciones, al igual que la mayoría de los objetos GeneXus que estudiaremos, provocan la generación de programas. En particular los programas correspondientes a las transacciones tienen por objeto permitir dar altas, bajas y modificaciones en forma interactiva en las tablas que tengan implicadas, controlando estos programas la integridad referencial de los datos.



Algunos elementos de las transacciones, que iremos viendo son:

Estructura: Permite definir los atributos (campos) que componen la transacción y la relación entre ellos. A partir de la estructura de las transacciones, GeneXus inferirá el diseño de la base de datos: tablas, claves, índices, etc.

Web Form: Cada transacción contiene un Form (pantalla) Web mediante el cual se realizarán las altas, bajas y modificaciones en ambiente Web.

Reglas: Las reglas permiten definir el comportamiento particular de las transacciones. Por ejemplo, permiten definir valores por defecto para los atributos, definir chequeos sobre los datos, etc.

Eventos: Las transacciones soportan la programación orientada a eventos. Este tipo de programación permite definir código ocioso, que se activa en respuesta a ciertas acciones provocadas por el usuario o por el sistema.

Variables: Permite la definición de variables que serán locales a la Transacción.

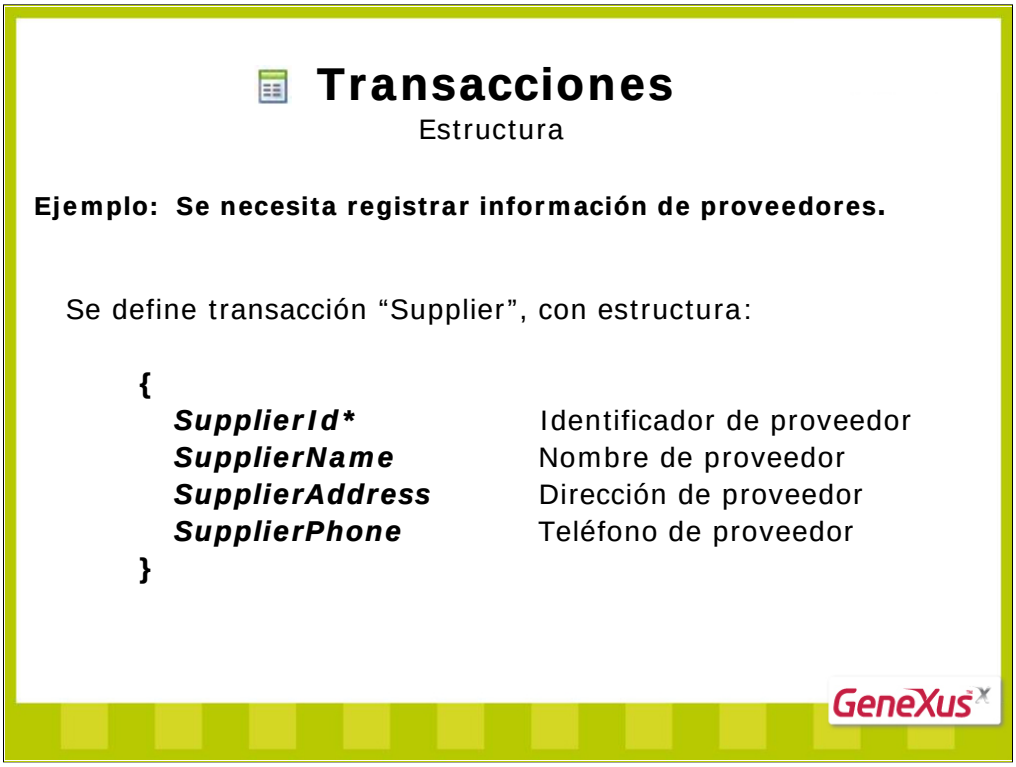
Propiedades: Permiten definir ciertos detalles referentes al comportamiento de la transacción.

Documentación: Permite la inclusión de texto técnico, para ser utilizado como documentación del sistema.

Ayuda: Permite la inclusión de texto de ayuda, para ser consultado por los usuarios en tiempo de ejecución de la transacción.

Category y Work With: Patterns (patrones) que pueden ser aplicados a la Transacción con el fin de implementar en forma automática cierta funcionalidad.

Algunos de estos elementos también están asociados a otros tipos de objetos GeneXus.



La estructura de una transacción permite definir qué atributos la integran y cómo están relacionados.

A modo de ejemplo, si en una aplicación se necesita registrar información de proveedores, claramente habrá que definir una transacción, a la que podemos dar el nombre “Supplier”, y su estructura podría ser la siguiente:

{*SupplierId**
SupplierName
SupplierAddress
SupplierPhone }

Esta lista de nombres (uno de los cuales está sucedido del símbolo asterisco) corresponde a los atributos que interesa mantener acerca de los proveedores.

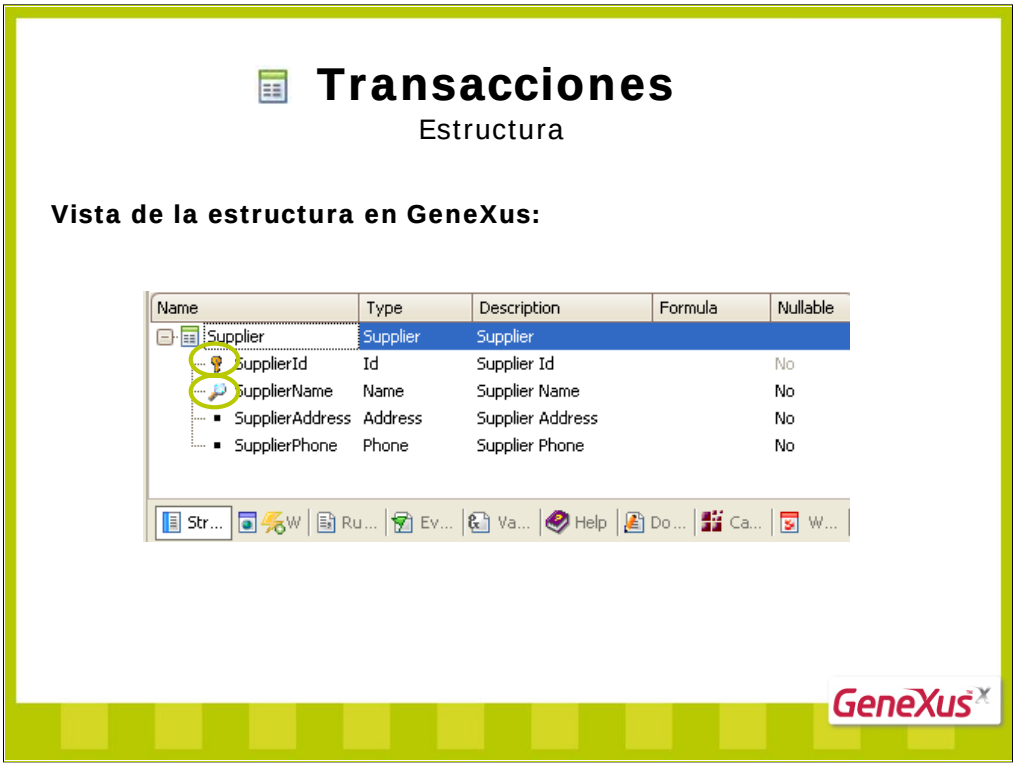
Entonces, creamos una transacción de nombre “Supplier” cuya estructura se compone de los atributos *SupplierId, SupplierName, SupplierAddress* y *SupplierPhone.*

Esto significa que cada proveedor se identificará por un código *SupplierId* (lo que queda determinado por el asterisco a continuación del atributo[1]), tendrá un nombre *SupplierName*, una dirección *SupplierAddress* y un teléfono *SupplierPhone*.

Para cada atributo definido en la estructura, deberemos indicar cosas tales como su tipo de datos, descripción y algunos detalles más que veremos.

---

[1] El asterisco corresponde a una notación teórica que utilizamos para indicar que el atributo es identificador. Como veremos, nuestro asterisco en GeneXus aparece representado por un ícono de llave y el usuario podrá configurarlo mediante un menú contextual que le ofrecerá esta posibilidad.



Atributos Clave

En la página anterior hemos explicado que el asterisco a continuación del atributo *SupplierId* indica que se trata del identificador de la transacción. Toda transacción debe tener un identificador, esto es, un atributo o conjunto de atributos que definan la unicidad.

En el ejemplo no podrán existir dos proveedores con el mismo valor de *SupplierId*. En definitiva se trata del concepto de clave primaria, en tanto, para hacer la elección de los atributos que la componen, se deben tener en cuenta los requisitos del objeto de la realidad.

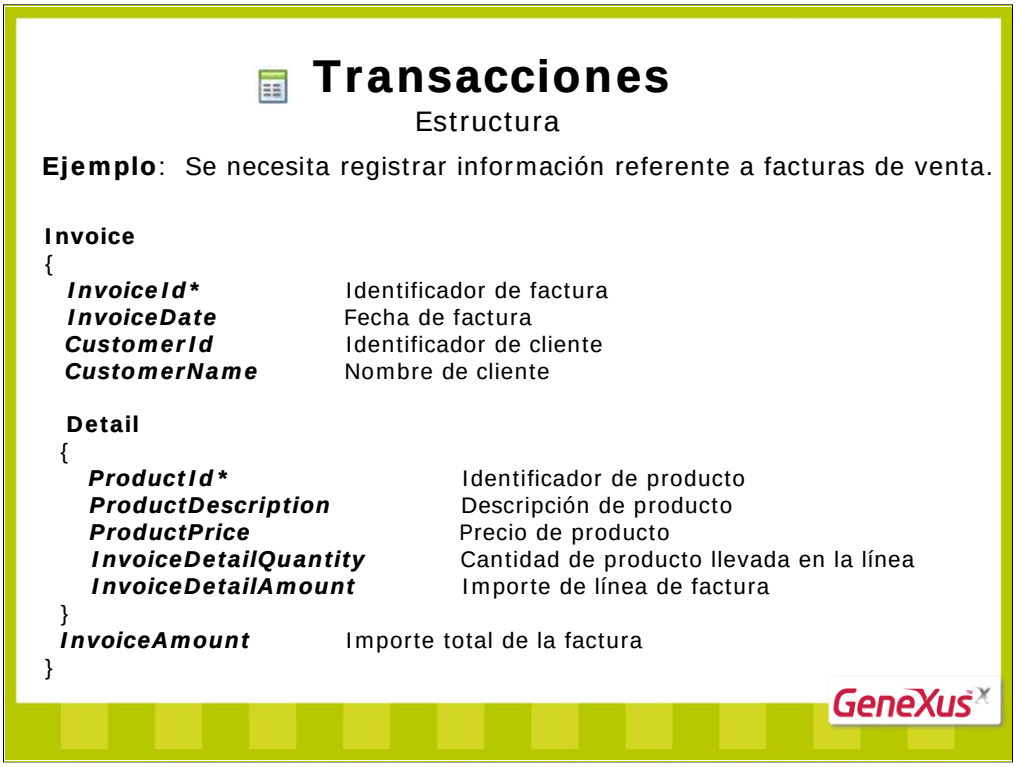
En los casos en los cuales no se pueda determinar un identificador, se debe optar por crear un atributo artificial (no existente en la realidad), y que su valor se asigne internamente, por ejemplo, en forma correlativa.

Como se puede observar en el editor de transacciones de GeneXus, un ícono de llave representa el asterisco que nosotros utilizamos como notación teórica.

Atributo “descriptor”

El ícono con una lupa representa al atributo que mejor describe o representa a la transacción. En otras palabras sería el atributo que tiene mayor carga semántica en la transacción.

Por defecto el primer atributo en la estructura de la transacción que sea de tipo de datos character, se definirá como “atributo descriptor”. Es posible definir a otro atributo como descriptor utilizando el menú popup correspondiente, así como no definir ninguno.



Niveles de una transacción

La transacción “Invoice” consta de dos niveles: el primer nivel queda implícito por lo que no necesita un juego de llaves; el segundo nivel corresponde al conjunto de atributos que se encuentra entre llaves[1].

El hecho de definir un segundo nivel significa que existen varias instancias del mismo, para cada instancia del nivel anterior. En el ejemplo, un cabezal de factura tiene varios productos.

Cada nivel de una transacción define un grupo de atributos que deben operar en conjunto, es decir, se ingresan, se eliminan o se modifican conjuntamente en la base de datos.

Llamaremos transacción plana a una transacción de un solo nivel. Así, la transacción “Supplier” es una transacción plana.

En cambio, la transacción “Invoice” tiene dos niveles. Es común hablar de “cabezal” para referirnos al primer nivel y de “líneas” para referirnos al segundo.

Para cada nivel de la transacción, se debe indicar cuáles de sus atributos actúan como identificador. El identificador de cada nivel puede estar compuesto de un solo atributo, como es el caso de las transacciones que hemos visto hasta ahora, o puede estar conformado por varios atributos.

En la transacción “Invoice” el atributo *InvoiceId* es el identificador del primer nivel, y el atributo *ProductId* es el identificador del segundo nivel. Esto último significa que para un número de factura dado, *InvoiceId*, no puede repetirse el valor del atributo *ProductId* en distintas líneas.

Una transacción puede contener varios niveles paralelos, así como anidados.

---

[1] Al igual que el asterisco es una notación teórica que representa que el atributo que lo antecede es identificador en la transacción, el juego de llaves también es utilizado como notación teórica, para representar que los atributos contenidos forman parte de un nivel anidado, y que tiene una representación visual en GeneXus distinta, pero que indica lo mismo.

En GeneXus queda visualmente claro el nivel correspondiente a las líneas de la factura.

A cada nivel de una transacción se le debe asignar un nombre, tipo[1] y descripción (salvo al primer nivel, que recibe como nombre el de la transacción).

Niveles paralelos y anidados

Una transacción puede tener varios niveles de anidación, así como niveles paralelos.

Por ejemplo, en el hipotético caso de que una factura pueda abonarse en varios pagos, podríamos definir dos tipos de estructura, dependiendo de lo que se quiera representar:

```
Invoice {
  InvoiceId*
 InvoiceDate
 CustomerId
 CustomerName
 InvoiceAmount
    Detail
    {ProductId*
     ProductDescription
     ProductPrice
     InvoiceDetailQuantity
     InvoiceDetailAmount}
     Payment
    {InvoicePaymentDate*
     InvoicePaymentAmount}
  }
```

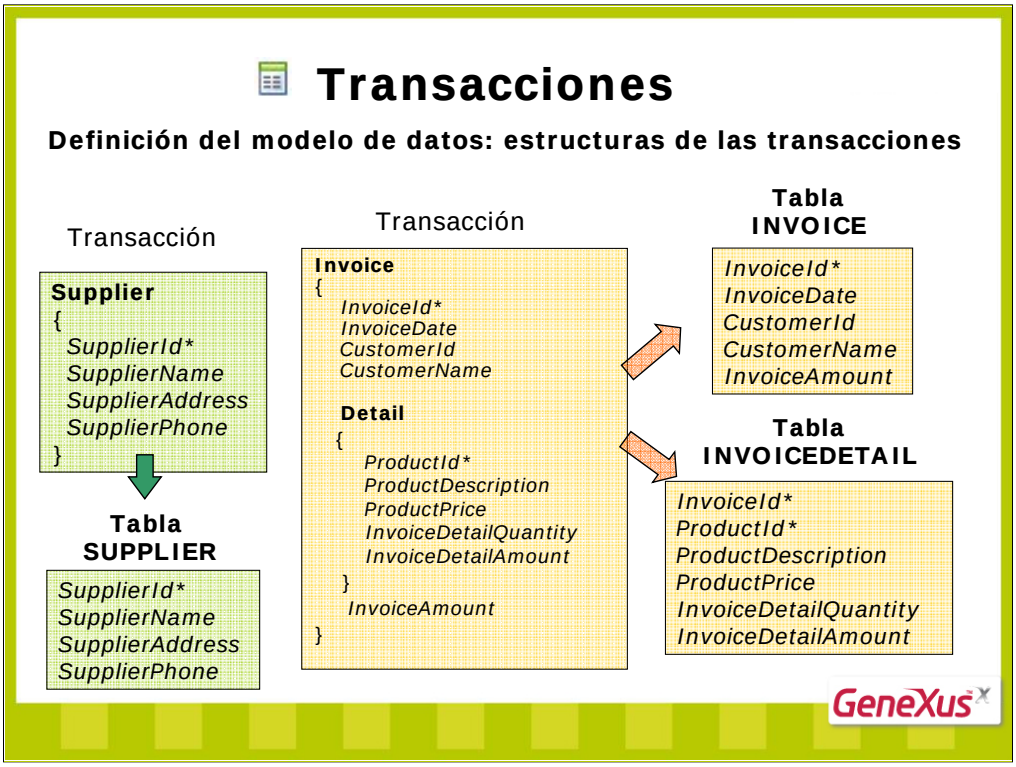
```
Invoice {
 InvoiceId*
 InvoiceDate
 CustomerId
 CustomerName
 InvoiceAmount
   Detail
   {ProductId*
    ProductDescription
    ProductPrice
    InvoiceDetailQuantity
    InvoiceDetailAmount
        Payment
        {InvoicePaymentDate*
        InvoicePaymentAmount}}
 }
```

Con la estructura de la izquierda se define que una factura tiene muchos productos y muchos pagos, pero no hay una relación directa entre los productos y los pagos (a no ser el hecho de pertenecer a la misma factura). En la estructura de la derecha se registran los pagos por producto llevado.

Es sencillo comprender que el segundo y tercer nivel de la transacción de la izquierda, son **paralelos**. Ambos se encuentran anidados al primer nivel, pero entre ellos, son paralelos. En la estructura de la derecha, son todos niveles **anidados**.

---

[1] Como veremos luego, el tipo que se define para un nivel de una transacción, será utilizado para trabajar con business components, concepto relacionado a las transacciones.



GeneXus utiliza la estructura de las transacciones para capturar el conocimiento necesario para definir automáticamente cuál es el modelo de datos que debe crear.

Para poder realizar la normalización de la base de datos llevándola a 3era. forma normal, GeneXus debe extraer las dependencias funcionales existentes entre los atributos definidos en la base de conocimiento.

En la base de conocimiento de nuestro ejemplo, hemos definido a la transacción “Proveedores” y de su estructura GeneXus extrae las siguientes dependencias funcionales:

SupplierId → *{SupplierName*, *SupplierAddress*, *SupplierPhone}*

Dadas estas dependencias funcionales, GeneXus determina que debe crear una tabla que tendrá por defecto el mismo nombre que la transacción (SUPPLIER)[1], y que estará conformada ni más ni menos que por los cuatro atributos anteriores, siendo *SupplierId* la clave primaria de la misma:

SUPPLIER	*SupplierId*	*SupplierName*	*SupplierAddress*	*SupplierPhone*

Diremos que la transacción “Supplier” tiene asociada la tabla SUPPLIER en el entendido de que cuando se ingresen, modifiquen o eliminen datos por medio de la transacción, éstos se estarán almacenando, modificando o eliminando físicamente en la tabla asociada.

---

[1] En la documentación, para distinguir el nombre de una tabla del nombre de una transacción escribiremos el nombre de la tabla todo en mayúscula.

A partir de la estructura de la transacción “Invoice”, GeneXus determina que debe crear dos tablas:

Tabla INVOICE, correspondiente al **primer nivel** de la transacción:

INVOICE	*InvoiceId*	*CustomerId*	*CustomerName*	*InvoiceDate*	*InvoiceAmount*

Clave primaria: *InvoiceId*

Tabla INVOICEDETAIL correspondiente al **segundo nivel** de la transacción:

INVOICEDETAIL	*InvoiceId*	*ProductId*	*ProductDescription*	*ProductPrice*
	InvoiceDetailQuantity		*InvoiceDetailAmount*	

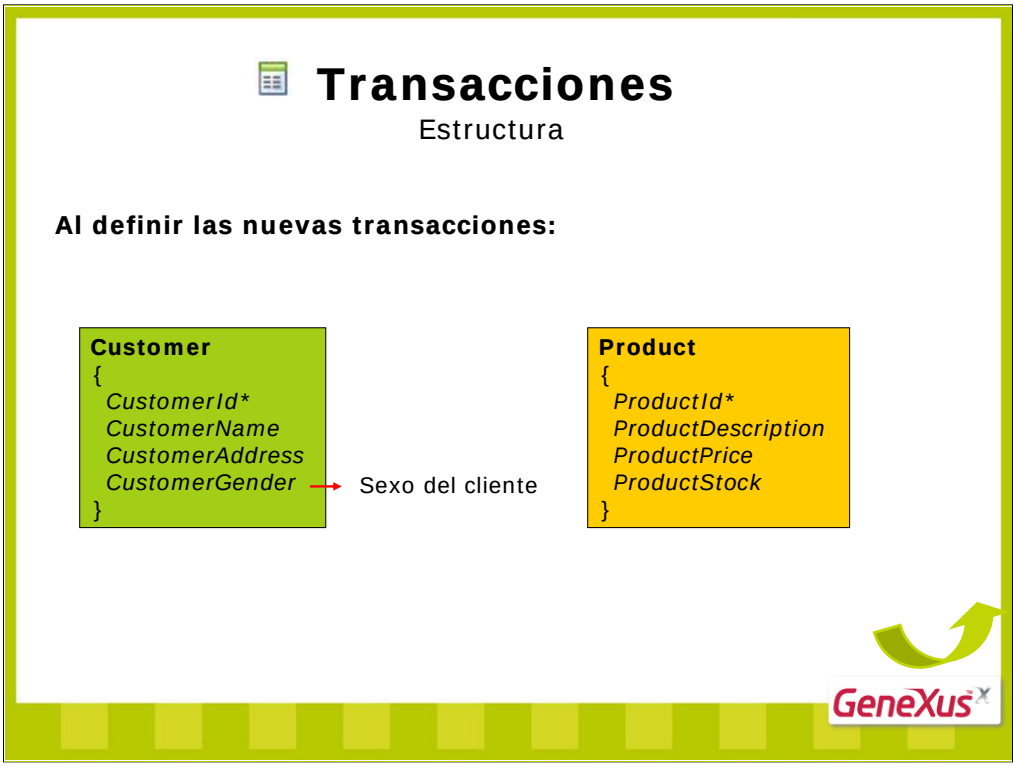
Clave primaria: {*InvoiceId*, *ProductId*}
Clave foránea: *InvoiceId*

ya que las dependencias funcionales son:

InvoiceId → *{CustomerId, CustomerName, InvoiceDate, InvoiceAmount}*
{InvoiceId, ProductId} → *{ProductDescription, ProductPrice, InvoiceDetailQuantity, InvoiceDetailAmount}*

Observemos que la clave primaria de la tabla INOVICELINE es la concatenación del identificador del primer nivel, *InvoiceId*, con el identificador del segundo nivel, *ProductId*. El caso es general: la clave primaria de la tabla correspondiente a un nivel n de una transacción se obtiene de concatenar los identificadores de los n-1 niveles anteriores anidados, con el identificador de ese nivel.

GeneXus asigna un nombre predeterminado a las tablas que crea. A la tabla asociada al primer nivel de una transacción le asigna el mismo nombre que el de la transacción; y a las tablas de niveles subordinados les asigna la concatenación de los nombres de los niveles. Por esto es que la tabla asociada al segundo nivel de la transacción “Invoice” recibe el nombre INVOICEDETAIL, dado que el nombre del primer nivel es el de la transacción, INVOICE, y el del segundo nivel es DETAIL. Los nombres de las tablas pueden ser modificados por el analista GeneXus cuando así lo desee.



Luego de haber modelado la transacción “Invoice”, nos damos cuenta que hay información de clientes y de productos que nos interesa mantener independientemente de las facturas. Es decir, los clientes y los productos son dos objetos de la realidad independientes de las facturas, por lo tanto creamos las dos nuevas transacciones “Customer” y “Product” detalladas arriba.

Dependencias funcionales

Con estas nuevas transacciones definidas, aparecen nuevas dependencias funcionales:

CustomerId → {CustomerName, CustomerAddress, CustomerGender, CustomerStatus}

ProductId → {ProductDescription, ProductPrice, ProductStock}

que conducen directamente a la creación de dos nuevas tablas:

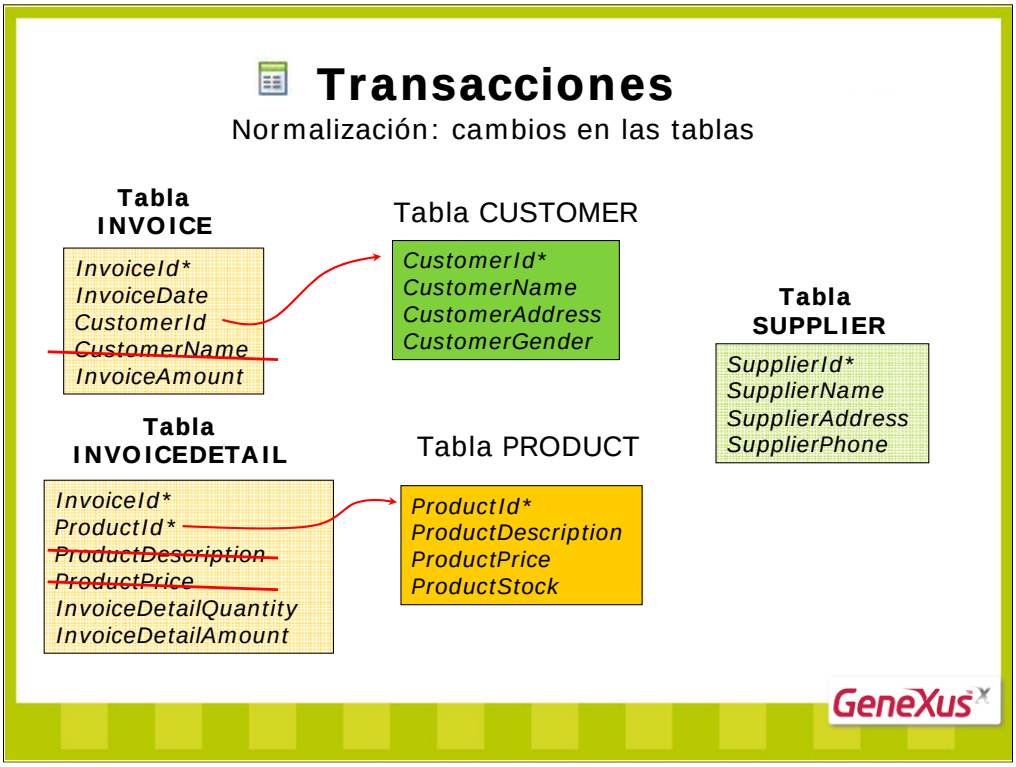
CUSTOMER	*CustomerId*	*CustomerName*	*CustomerAddress*	*CustomerGender*

Clave primaria: *CustomerId*

y:

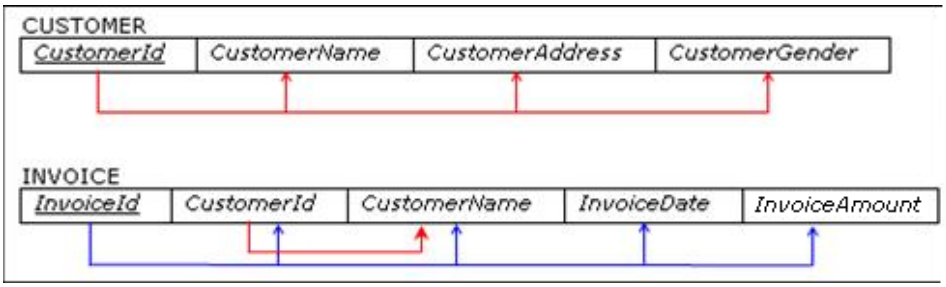
PRODUCT	*ProductId*	*ProductDescription*	*ProductPrice*	*ProductStock*

Clave primaria: *ProductId*



El conjunto total de dependencias funcionales existentes requiere que deban realizarse algunas modificaciones en las tablas INVOICE e INVOICEDETAIL diseñadas previamente para que el diseño de la base de datos permanezca en 3era. forma normal[1].

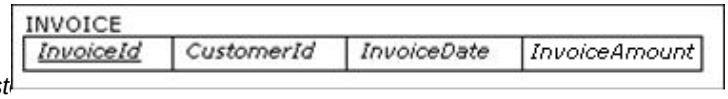
Si representamos sobre las tablas CUSTOMER e INVOICE las dependencias funcionales encontradas para sus atributos:



podemos ver claramente que INVOICE viola la 3era. forma normal (existe una dependencia funcional transitiva):

InvoiceId → *CustomerId*
CustomerId → *CustomerName*
InvoiceId → *CustomerName*

por lo tanto GeneXus normaliza, quitando el atributo *CustomerName* de la tabla INVOICE:

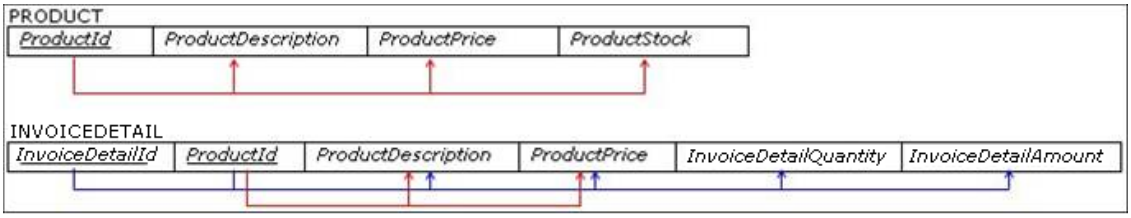


Ahora *Cust*

---

[1] Por más información sobre las formas normales (3era. forma normal, etc.) le recomendamos la lectura del documento "Fundamentos de bases de datos relacionales", el cual está incluido en el capítulo de "Anexos" del curso GeneXus no presencial. De lo contrario, puede pedírselo al docente.

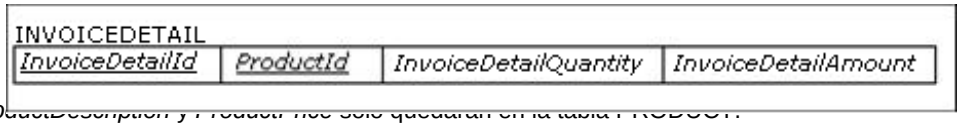
Ahora veamos las dependencias funcionales encontradas en las tablas PRODUCT e INVOICEDETAIL:



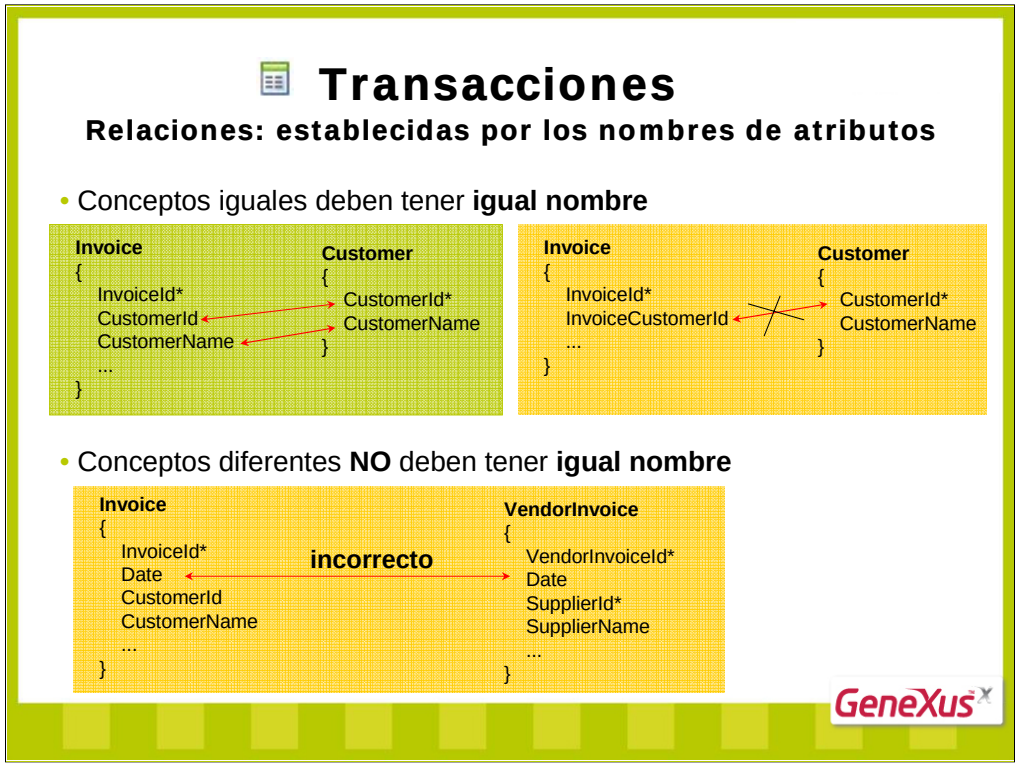
podemos percibir claramente que la tabla INVOICEDETAIL está violando la 3era. forma normal (existen atributos que están dependiendo en forma parcial de la clave primaria):

ProductId → ProductDescription
{InvoiceId, ProductId} → ProductDescription
ProductId → ProductPrice
{InvoiceId, ProductId} → ProductPrice

Por lo tanto GeneXus normaliza, quitando los atributos *ProductDescription* y *ProductPrice* de la tabla INOVICEDETAIL:



ProductDescription y ProductPrice solo quedarán en la tabla PRODUCT.



Conceptos iguales deben tener el mismo nombre y conceptos diferentes deben ser nombrados diferentes.

GeneXus establece las relaciones a través de los nombres de los atributos, de modo que cuando encuentra atributos de igual nombre en distintas transacciones, entiende que se trata del mismo concepto, y mediante dicho conocimiento es que puede normalizar.

En el ejemplo que venimos viendo, cuando GeneXus encuentra el atributo de nombre *CustomerId* tanto en la transacción “Customer” como en la transacción “Invoice”, analiza que: el atributo se llama igual en ambas transacciones, así que se refiere al mismo concepto. En la transacción “Customer”, *CustomerId* está marcado como identificador, lo que significa que será **clave primaria** en la tabla física CUSTOMER; entonces en la tabla física INVOICE será **clave foránea**.

El atributo *CustomerName,* por su parte, también se encuentra tanto en la transacción “Customer” como en la transacción “Invoice”, pero a diferencia de *CustomerId,* no está marcado como identificador en ninguna transacción del modelo; por tanto GeneXus entiende que se trata de un atributo **secundario**. Las dependencias funcionales indican que a *CustomerName* lo determina *CustomerId:*

InvoiceId → *CustomerId*

CustomerId → *CustomerName*

así que GeneXus incluirá *CustomerName* en la tabla física CUSTOMER (y no en la tabla física INVOICE).

Atributos primarios y secundarios

Un atributo se califica como **primario** cuando el mismo es identificador en alguna transacción del modelo. En el ejemplo que venimos viendo, *CustomerId* es un atributo primario ya que es identificador en la transacción “Customer”.

CustomerName, en cambio, es un atributo **secundario** ya que no es identificador en ninguna transacción del modelo.

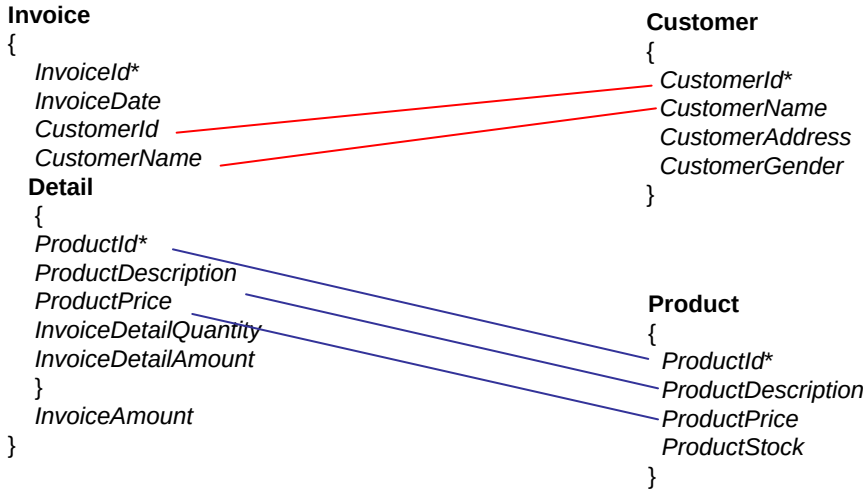
Atributos almacenados e inferidos

Al definir las transacciones “Customer” y “Product”, hemos incluido en ellas ciertos atributos que no hemos eliminado de la transacción “Invoice”.

Los atributos *CustomerId* y *ProductId,* se incluyeron en las transacciones “Customer” y “Product” respectivamente, y al ser denotados como identificadores de las mismas, pasaron a ser atributos primarios.
El atributo *CustomerName,* por su parte, se agregó en la transacción “Customer”; y los atributos *ProductDescription* y *ProductPrice* se incluyeron en la transacción “Product”. Estos son atributos secundarios.

Todos estos atributos han quedado en más de una transacción: se han dejado en la transacción “Invoice” tal como se habían definido en un principio, y se han incluido en las transacciones “Customer” y “Product” respectivamente, porque nos hemos percatado de la necesidad de crear estos objetos.

A continuación presentamos las 3 estructuras de las transacciones en cuestión, para poder visualizarlas juntas:



Probablemente usted no comprenda la razón por la cual los atributos secundarios *CustomerName, ProductDescription* y *ProductPrice* se han dejado en la estructura de la transacción “Invoice”.

La explicación es la siguiente: **las estructuras de las transacciones no son equivalentes a estructuras de tablas físicas.** En las estructuras de las transacciones se pueden incluir ciertos atributos que no estarán en la o las tablas físicas asociadas, ya que a la hora de reorganizar la base de datos GeneXus analizará el conjunto total de dependencias funcionales existentes en la base de conocimiento, y creará -o modificará, según el caso- las tablas físicas, dejándolas en 3ª forma normal.

Ahora, ¿con qué finalidad hemos dejado los atributos secundarios *CustomerName, ProductDescription* y *ProductPrice* en la estructura de la transacción “Invoice”? Los hemos dejado para poder incluirlos en alguno de los forms (GUI-Windows y/o Web) asociados a la transacción “Invoice” y así poder visualizar los valores de dichos atributos en tiempo de ejecución.

Dichos atributos, como hemos explicado, no quedarán almacenados ni en la tabla INVOICE, ni en la tabla INVOICEDETAIL; sin embargo, en tiempo de ejecución cuando el usuario ingrese a través de alguno de los forms (GUI-Windows y/o Web) un valor de *CustomerId* (atributo que sí se almacenará en la tabla INVOICE siendo clave foránea), a continuación se mostrará el *CustomerName* correspondiente. Decimos que el atributo *CustomerName* es un **atributo inferido** en la transacción “Invoice”, ya que su valor no se encuentra almacenado en ninguna de las tablas asociadas a la transacción, sino que **se infiere** –es decir, se obtiene- de la tabla CUSTOMER, dado el valor del atributo *CustomerId*.

Análogamente, los atributos *ProductDescription* y *ProductPrice* también son **inferidos** en la transacción “Invoice”, ya que no se encuentran almacenados en las tablas asociadas a la misma, sino que sus valores **se infieren** de la tabla PRODUCT, para ser mostrados en pantalla.

Transacciones

Estructura: nombrado de atributos

Es conveniente usar padrones para los nombres de los atributos.

- Facilitan la tarea de nombrado.
- Facilitan la tarea de integración de bases de conocimiento.
- Facilitan la lectura del código generado.

Artech ha definido un estándar para la nomenclatura de atributos: el **G**eneXus **I**ncremental **K**nowledge Base (GIK) que es utilizado por la comunidad de usuarios GeneXus.

En esta nomenclatura, el nombre de un atributo se forma con 4 componentes (algunos opcionales, señalados entre paréntesis rectos):

Componente de Entidad + Categoría [+ Calificador + Complemento] [1]

A continuación describimos en qué consiste cada componente:

Componente de Entidad (u Objeto):Una Entidad es un actor (ej: Customer), objeto o evento (ej: Vendor Invoice, factura de venta) que interviene en una aplicación dada, representado por una transacción Genexus[2]. Un Componente de Entidad, representa a cualquier nivel subordinado o paralelo que defina la entidad.

Ejemplo: la factura tiene los siguientes componentes, Invoice (cabezal), InvoiceDetail (líneas de la solicitud).

Se sugiere un largo de un entorno de 10 caracteres para representar el componente de la Entidad.

Categoría: Es la categoría semántica del atributo, es decir, define el rol que el mismo asume dentro del objeto y dentro del entorno de la aplicación. Se sugiere que no supere los 10 caracteres.

Ejemplos: Id (identificador), Code (código), Name (nombre), Date (fecha), Description (descripción), Price (precio), Stock.

---

[1] Para países que utilicen lenguas en las que los adjetivos suelan colocarse después del sustantivo. En el inglés esto es al revés, por lo que la categoría (el sustantivo) va al final.

[2] O un conjunto de Transacciones paralelas y/o subordinadas, de las que hablaremos más adelante.

Calificador: Es cualquier adjetivo o adverbio, en el entorno de 10 caracteres, que agregue diferenciación conceptual al nombre del atributo para hacerlo único.

En general refiere al texto que califica la categoría: Fecha **de vencimiento**,

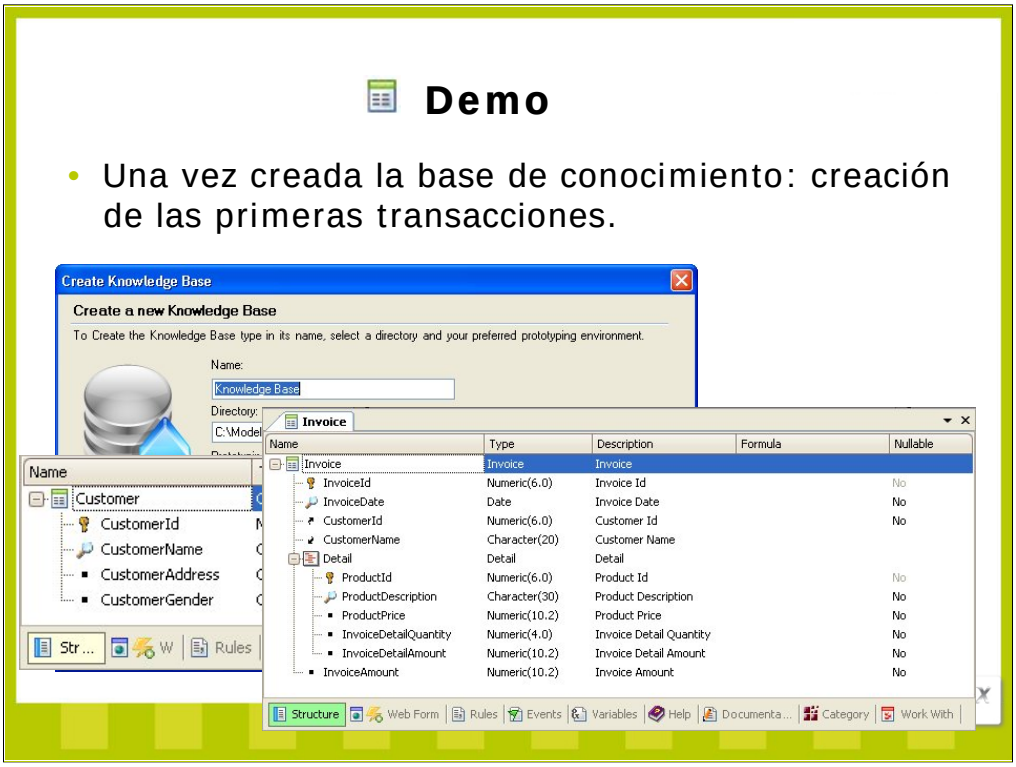
Ejemplos: Start (inicial), End (final), Due (vencimiento)

Complemento: Texto libre hasta completar la cantidad de caracteres significativos (30) para el nombre.

En la transparencia se muestran algunos ejemplos de nombres de atributos.

Nota 1: Las letras mayúsculas permiten establecer fronteras entre los componentes que forman a los nombres de atributos.

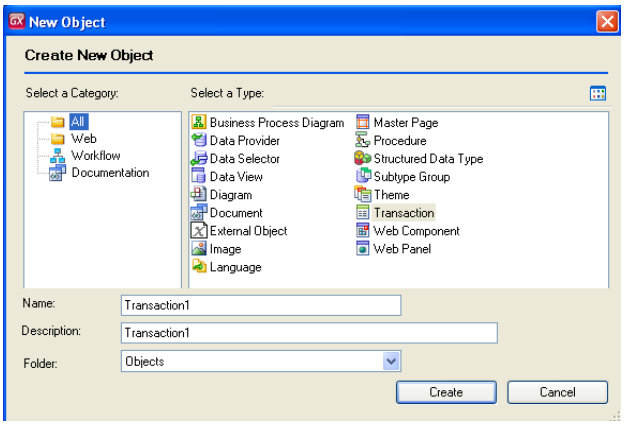
Nota 2: Para cada componente se pueden utilizar la cantidad de caracteres que se deseen, aunque se sugiere utilizar 10 y que el nombre completo del atributo no supere los 30.



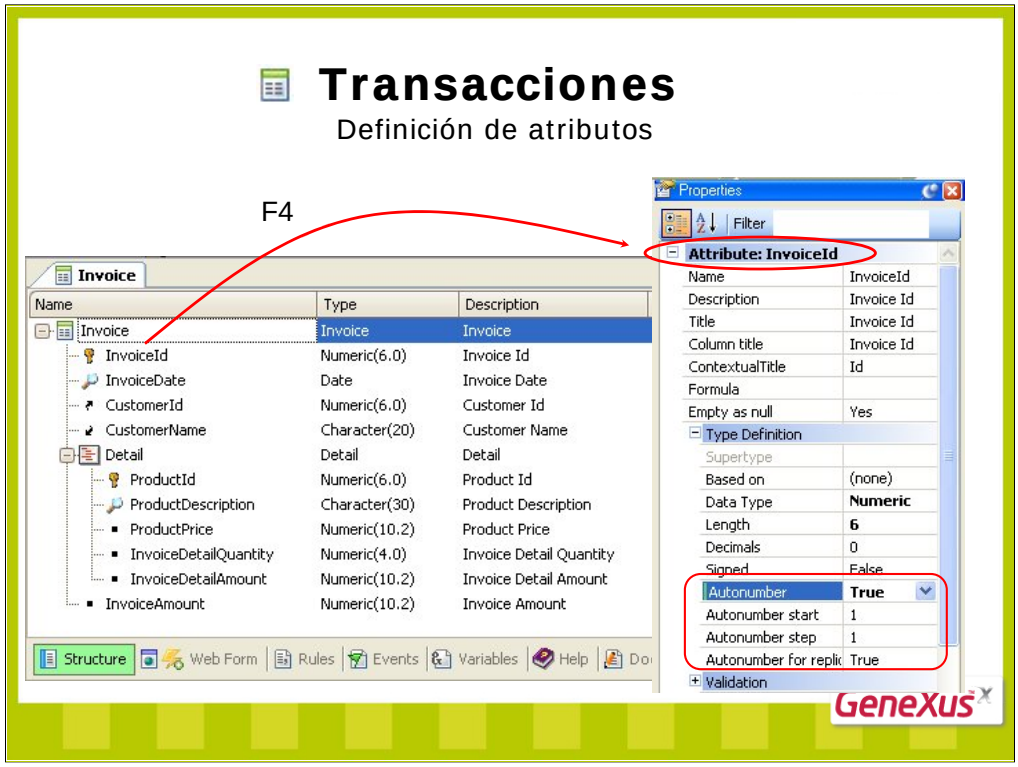
Una vez creada la base de conocimiento (ver introducción teórica), la misma quedará abierta para que se empiecen a crear las transacciones.

La creación de objetos, se realiza presionando Ctrl+N. Los objetos creados quedarán en la carpeta Objects que se puede ver en la Folder View de la ventana KB Navigator.
Si se desea que el objeto a crear quede guardado en otra carpeta, se deberá posicionar en dicha carpeta y luego hacer clic con el botón derecho del mouse. En el menú elegir New → Object.
También se podrá indicar la carpeta en el cuadro de creación de un objeto como se ve en la imagen.

Se desplegará un diálogo en el cual se deberá elegir el tipo de objeto que se desea crear (en este caso el tipo de objeto: Transaction), dar un nombre al objeto que se está creando (por ejemplo: “Customer” o “Invoice”), una descripción larga, y la carpeta en la cual guardar el objeto.



Una vez creada la transacción, la misma quedará abierta para que se defina su estructura y dentro de ella sus atributos.



Para definir un atributo, simplemente se debe digitar en el primer campo de una línea (o rama) de la estructura, el nombre del atributo que se desea crear. Mediante la tecla de tabulación se puede pasar a los siguientes campos para indicar el tipo de datos del atributo, así como su descripción, si admitirá valores nulos de la base de datos, y en el caso que el mismo vaya a ser una fórmula (concepto que veremos en breve), cómo calcularla. Con la tecla Enter se puede pasar a la siguiente línea, para definir otro atributo.

Una vez definidos los atributos en la estructura de la transacción, solamente restará guardar / salvar los cambios.

Si se necesita modificar el nombre de un atributo, su tipo de datos, descripción, nulabilidad, o fórmula, bastará con hacer doble clic sobre el campo implicado en la estructura, y el mismo se habilitará y se podrá editar. Luego se deberán guardar los cambios, nuevamente.

Para indicar que uno o más atributos son identificadores en la transacción, se los debe seleccionar y presionar CTRL + K; en consecuencia aparecerán con un símbolo de llave.

Para definir que comienza otro nivel en la transacción, se debe digitar CTRL + L, y automáticamente se producirá una indentación y el usuario deberá darle nombre a ese nivel, así como definir el nombre de su tipo de datos[1] y una descripción para el nivel.

Para indicar que un atributo de uno de los niveles de la transacción será el atributo “descriptor”, se lo debe seleccionar y presionar CTRL+D.

GeneXus cuenta con menús pop up[2], que son menús que se abren cuando el usuario está posicionado en determinado contexto y da clic con el botón derecho del mouse. Por ejemplo, al hacer clic con el botón derecho del mouse **sobre un atributo de la estructura**, se abre un menú pop up sobre el atributo seleccionado, que ofrece varias utilidades, como por ejemplo indicar que el atributo es clave (opción “Toggle key”), quitarlo de la estructura, pasarlo a un siguiente nivel de anidación, etc.

Cada atributo tiene propiedades. Algunas de ellas (las fundamentales y obligatorias) son las que se ofrecen directamente en la estructura para su ingreso “inline”. Para acceder al diálogo que permite configurar todas las propiedades de un atributo, se debe seleccionar el ítem **Properties** del menú contextual**,** o presionar la tecla F4. En la figura hemos cambiado la propiedad Autonumber que como veremos oportunamente, permite autonumerar atributos clave primaria.

---

[1] Veremos su utilidad cuando estudiemos los “business components”.
[2] También llamados “contextuales” dado que varían según el contexto.

Name: Es el nombre del atributo. Se utiliza para identificarlo.

Description: La "Descripción" o más propiamente "Nombre largo" es una descripción ampliada del atributo. Debe identificar bien al atributo, con independencia del contexto, y principalmente debe ser entendible por el usuario final. Debe ser un identificador global del atributo, es decir, no se le debe asignar a dos atributos en la base de conocimiento la misma descripción, ya que será a través de esta descripción que el usuario final podrá seleccionar atributos para definir sus propias consultas a la base de datos, con el objeto de tipo query, ejecutando "reportes dinámicos" (tema bastante simple, pero que no se abordará en este curso).

Relacionadas a esta propiedad, están las propiedades **Title**, **Column Title** y **Contextual Title**. Las 2 primeras por defecto toman el mismo valor que se especifica en **Description**, pudiéndose modificar:

Title: La descripción que se ingrese aquí será colocada por defecto al lado del atributo cada vez que se utilice en salidas "planas" como en el primer nivel de los forms de las transacciones (veremos en breve los forms).

Column Title: La descripción que se ingrese aquí será colocada por defecto como título del atributo cada vez que se lo incluya en la columna de un grid (grilla) (en el caso de una transacción solo si se trata de un atributo inferido, como por ejemplo el atributo *ProductDescription* en la transacción Invoice)[1].

Contextual Title: Cuando el atributo pertenece al segundo nivel de una transacción, y no es inferido (ejemplo *InvoiceDetailQuantity* en Invoice), el nombre de la columna del grid de la transacción será tomado de esta propiedad. Obsérvese que por defecto se toma de la propiedad Description, pero quitando el nombre de la transacción, pues se asume del "contexto".

Type Definition

Based on: Permite asociarle un dominio[2] al atributo. Al hacerlo, ciertas propiedades del atributo se deshabilitarán (como **Data Type** y **Length**) tomando los valores del dominio. En caso de que el atributo no pertenezca a un dominio, el programador dejará el valor '(none)' en esta propiedad, y las correspondientes al tipo de datos del atributo estarán habilitadas para ser ingresadas.

Data Type: Permite indicar el tipo de datos del atributo. Aquí se podrá elegir uno de los tipos de datos soportados por GeneXus. Dependiendo del tipo de datos que se seleccione habrá ciertas propiedades, u otras, para configurar.

Length: Permite indicar el largo del atributo. Si en la propiedad **Data Type** se indica que el atributo es numérico, entonces se deberá tener en cuenta que el largo incluya las posiciones decimales, el punto decimal y el signo. Esta propiedad estará deshabilitada cuando el tipo de datos elegido no requiera establecer un largo (por ejemplo, si se trata del tipo de datos Date).

Decimals: Si en la propiedad **Data Type** se indica que el atributo es numérico, se ofrecerá esta propiedad, para que se especifique la cantidad de decimales. El valor 0 en este campo, indicará que se trata de un entero.

Signed: Si en la propiedad **Data Type** se indica que el atributo es numérico, se ofrecerá esta propiedad para que se indique si manejará signo o no. El valor por defecto es "False", lo que indica que no se representarán valores negativos.

Validation

Value Range: Permite indicar un rango de valores válidos para el atributo. Por ejemplo:

- 1:20 30: - significa que los valores válidos son entre 1 y 20; y 30 o mayor.
- 1 2 3 4 - significa que los valores válidos son 1, 2, 3 o 4.
- 'S' 'N' - significa que los valores válidos son 'S' o 'N'.

Picture

Permite indicar el formato de edición para la entrada y salida del atributo. Dependiendo del tipo de datos del atributo, aparecerán determinadas propiedades bajo esta categoría.

GeneXus provee más propiedades para los atributos que las recién mencionadas. Dependiendo del valor que se elija para determinada propiedad, se ofrecerán ciertas propiedades relacionadas, u otras. Recomendamos para la lectura de otras propiedades, acceder al Help de GeneXus.

---

[1] El atributo también podrá estar en un objeto Web Panel. En ese caso, de aparecer como columna, siempre se ofrecerá por defecto para el nombre de la misma el de su propiedad ColumnTitle

[2] Los dominios se abordarán unas páginas más adelante en el curso, pero a modo de resumen, el objetivo de los dominios es realizar definiciones de datos genéricas. Por ejemplo: se puede definir un dominio de nombre Precio y tipo de datos Numeric(10,2) y luego asociarle este dominio a todos los atributos que almacenan precios. Esto tiene la ventaja de que si después se desea modificarlo a por ejemplo Numeric(12,2), hay que modificar solamente la definición del dominio y automáticamente todos los atributos basados en ese dominio heredan el cambio.

Control Info

A un atributo se le puede asociar un **tipo de control**. Los tipos de controles posibles son:

- Edit
- Radio Button
- Check Box
- Combo Box
- List Box
- Dynamic Combo Box
- Dynamic List Box

La asociación de cierto tipo de control a un atributo, sirve para especificar el tipo de control por defecto que se utilizará para el atributo cada vez que se lo incluya en un form.

Cuando se define un atributo con un tipo de datos básico, el tipo de control que tiene asociado es Edit, pudiendo el programador cambiarlo a cualquiera de los otros tipos. En general GeneXus elige el tipo de control para un atributo dependiendo de su tipo de datos. Si es un dominio enumerado, como veremos, elegirá Radio Button o Como Box, dependiendo de la cantidad de valores del dominio.

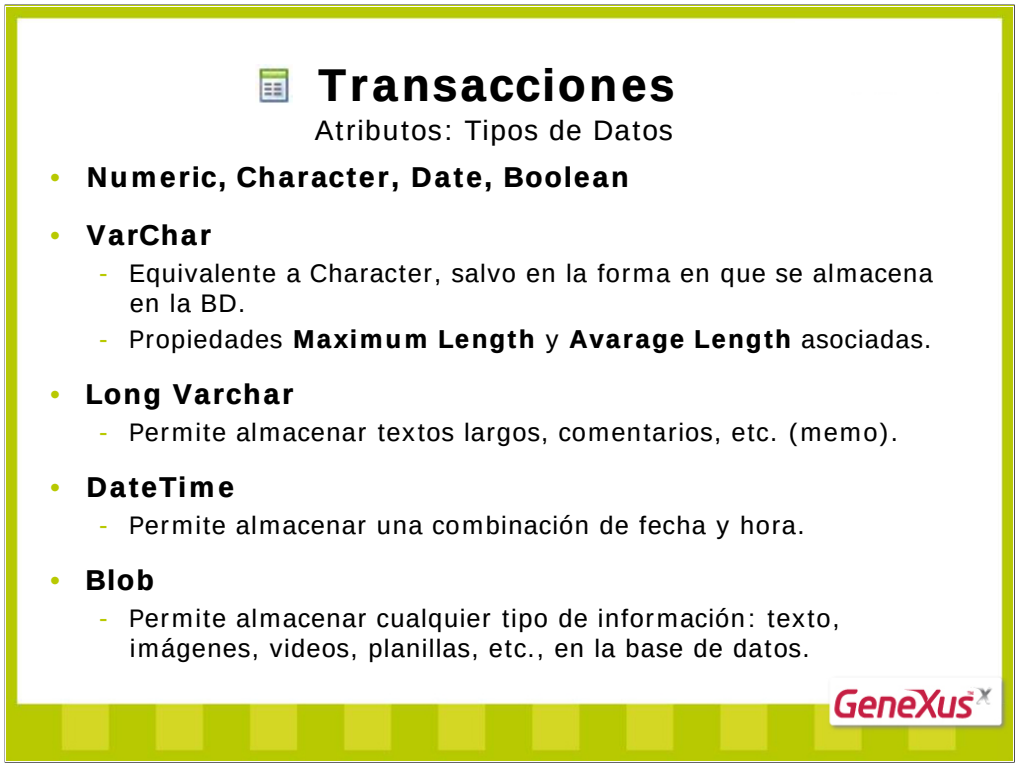
En el grupo **Control Info** del diálogo de edición de las propiedades de un atributo es donde el programador podrá cambiar el tipo de control asociado al atributo y dependiendo del tipo de control seleccionado, se solicitará distinta información complementaria.

Luego, cada vez que se agregue el atributo en un form se presentará automáticamente con el tipo de control que tenga asociado aquí.

Nota

En caso de asociar al atributo el tipo Edit, se podrá especificar que “disfrace” sus valores, mostrando los de otro atributo (propiedad **InputType**), e incluso que sugiera los valores posibles al usuario, a medida que éste vaya digitando (propiedad **Suggest**), entre otras, como veremos luego, cuando estudiemos “Descripciones en lugar de códigos”.

También se puede elegir el color de la fuente de letra que se desea asociar por defecto al atributo, así como el color de fondo, de modo que cada vez que se agregue el atributo en un form, se presente automáticamente con los colores que se le hayan asociado (ver grupo Appearance).



▪ **Numeric**: Permite almacenar datos numéricos. Cuando se selecciona este tipo de datos se debe indicar la cantidad total de dígitos del número, la cantidad de posiciones decimales, y si permite signo o no.

Ejemplo:
Si definimos que el tipo de datos del atributo *InvoiceAmount* es numérico de largo 10, con decimales 2, y sin signo, estamos especificando que representará números con hasta 7 dígitos en la parte entera y 2 decimales (7 dígitos en la parte entera + punto + 2 dígitos para los decimales = 10 dígitos).

▪ **Character**: Permite almacenar cualquier tipo de texto (caracteres y dígitos). Para este tipo de datos, se debe indicar el largo.

Ejemplo: El atributo *CustomerName* que utilizamos para almacenar el nombre de un cliente, es de tipo Character y si sabemos que nunca un nombre tendrá más de 20 caracteres, podemos fijar el largo: 20.

▪ **Date**: Permite almacenar una fecha.

Ejemplo: El atributo *InvoiceDate* que utilizamos para almacenar la fecha de una factura, será de este tipo de datos.

El formato a utilizar para las fechas (día-mes-año, mes-día-año), se configura en forma genérica para todo el modelo dentro de un tipo especial de objeto, el objeto Language correspondiente al lenguaje en el que se generará el modelo. El objeto “language” existe para poder tener una misma aplicación traducida en cualquier lenguaje.

La elección de presentar el año con 2 dígitos o 4, se configura con la propiedad **Picture** de cada atributo.

• **Boolean**: permite que un atributo o variable asuma los valores lógicos: True, False.

• **VarChar**: Permite almacenar texto de largo variable. Su función, en contraposición al Character, es optimizar el almacenamiento en la base de datos.

Cuando se selecciona el tipo de datos VarChar en el diálogo de definición del atributo se agregan las 2 siguientes propiedades: **Maximum Length** y **Average Length**.

La primera es para indicar el largo máximo de caracteres que se podrán almacenar, mientras que la segunda es para indicar el largo promedio de caracteres que se suele almacenar por lo general.

Ejemplo: Cuando se define un atributo de tipo Character y largo 60, si se le ingresa un dato que ocupa 25 caracteres, la capacidad restante de almacenamiento del atributo (35 caracteres) se rellena con blancos. El tipo de datos Varchar, en cambio, optimiza el almacenamiento de la siguiente forma: se le define **Average Length** (por ejemplo: 25), y **Maximum Length** (por ejemplo: 60); entonces, si el dato tiene largo menor o igual que 25, se lo almacena en el campo (rellenado con blancos) mientras que en los casos que el dato sea de largo mayor que 25, se almacenan los primeros 25 caracteres en el campo, y el resto en un área de overflow.

Como contrapartida a la ventaja de la optimización del almacenamiento, para los casos en que se utilice el área de overflow, será necesario realizar un acceso adicional tanto para la lectura como para la escritura del dato.

El tipo de datos Varchar es equivalente al tipo Character en todos los sentidos, salvo en la forma en que se almacena en la base de datos. Se le pueden aplicar todas las funciones y operadores existentes para el tipo de datos Character. Si el DBMS no soporta este tipo de datos, se creará el atributo de tipo Character.

▪ **Long Varchar**: Permite definir un atributo memo; es decir, se utiliza normalmente para almacenar textos largos, comentarios, etc. Al seleccionar este tipo de datos, se debe indicar un largo máximo.

Existen dos funciones para manipular este tipo de datos: **GXMLines** y **GXGetMLi**.

GXMLines retorna la cantidad de líneas que tiene un atributo Long Varchar, teniendo como parámetros el nombre del atributo, y la cantidad de caracteres que se desea considerar por línea.

Ejemplo: *&cantlin* = GXMLines(*AtribMemo*, 40).

GXGetMLi por su parte, extrae una línea del atributo Long Varchar (para luego imprimirla, por ejemplo); teniendo como parámetros el nombre del atributo, el número de línea deseado, y la cantidad de caracteres que se desea considerar por línea.

Ejemplo: *&txt* = GXGetMLi(*AtribMemo*, 1, 40).

Generalmente se usan estas 2 funciones en combinación, ya que primero se suele consultar la cantidad de líneas que tiene cierto atributo Long Varchar, indicando la cantidad deseada de caracteres por línea, y luego se prosigue extrayendo el contenido de las líneas, utilizando un bucle hasta llegar a la última línea, y de esta forma se imprimen, por ejemplo.

▪ **DateTime**: Permite almacenar una combinación de fecha y hora.

La propiedad **Picture** de este tipo de datos, permite elegir qué se desea mostrar de la fecha, y qué se desea mostrar de la hora.

Acerca de la fecha se puede elegir: no manejarla, manejarla y presentar el año con 2 dígitos, o manejarla y presentar el año con 4 dígitos. Acerca de la hora se puede elegir: manejar solo 2 dígitos para la hora (no manejando minutos ni segundos), manejar 2 dígitos para la hora y 2 dígitos para los minutos (no manejando segundos), o manejar 2 dígitos para la hora, 2 dígitos para los minutos y 2 dígitos para los segundos.

Los valores anteriores **no** afectan la forma de almacenar el tipo de datos sino específicamente la forma de **aceptar o mostrar su contenido**.

Nota: En caso de no manejar la fecha, sino solo la hora, el valor de fecha que quedará en el campo será el mínimo soportado por el DBMS, y será reconocido por GeneXus como fecha vacía o nula. En lo que respecta a la hora, los valores no aceptados (minutos y/o segundos) serán almacenados con valor cero.

▪ **Blob**: Ante el creciente manejo de imágenes digitalizadas, videos, planillas así como documentos de todo tipo, las aplicaciones requieren cada vez más mantener y trabajar con este tipo de información.

El tipo de datos Blob permite almacenar esta diversidad de información en la propia base de datos, aprovechando así los diferentes mecanismos de integridad y control que proveen los DBMSs.

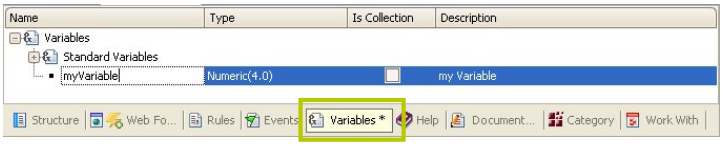
Este tipo de datos solamente se puede utilizar cuando se cuenta con un DBMS.

Para profundizar en el conocimiento de este tipo de datos puede dirigirse al Help de GeneXus.

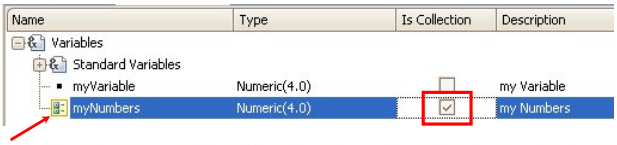
Transacciones

Definición de variables

- En todo objeto GeneXus es posible definir variables.
- Las variables son únicamente visibles dentro del objeto; es decir, son locales.
- Editor similar al de la estructura de las transacciones:



- Es posible definir variables colección (de cualquier tipo de datos).

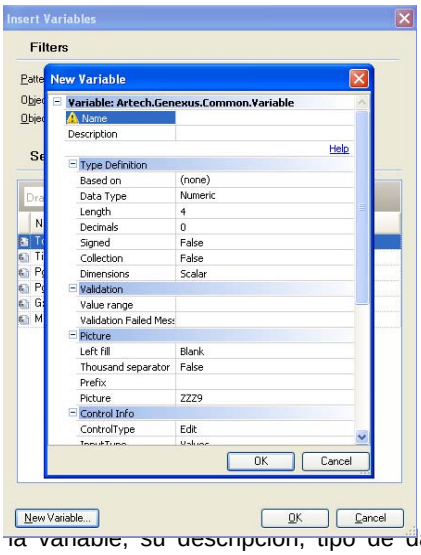


GeneXus

Para definir variables en determinado objeto, se debe seleccionar el selector Variables, como se muestra en la figura.

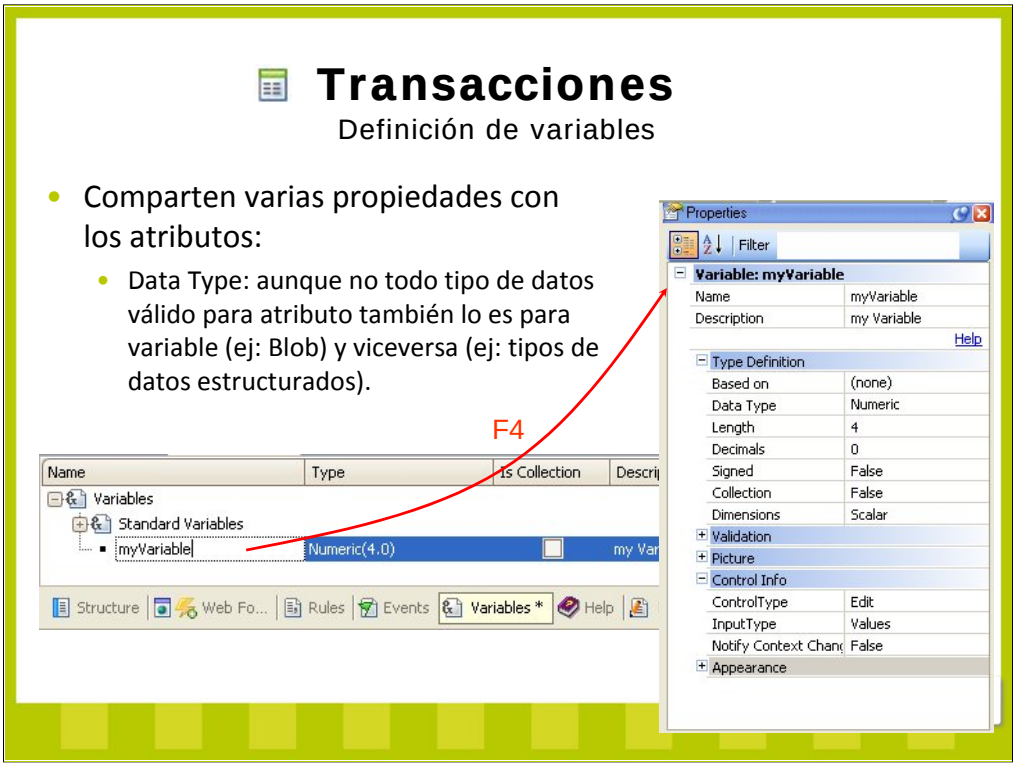
Este selector muestra variables definidas por defecto (Standard variables, como por ejemplo la variable Today que contiene la fecha del sistema) para el objeto, y permite definir variables nuevas a través de un editor similar al de las transacciones.

También se puede ir a la opción Insert de la menubar y elegir Variables. En el cuadro de diálogo que se desplegará seleccionar New Variable.



Este diálogo solicita el nombre de la variable, su descripción, tipo de datos y largo, o dominio, de forma análoga a cuando se define un atributo.

La propiedad **Dimensions** permite indicar si la variable será escalar o si se tratará de un vector (1 dimensión) o matriz (2 dimensiones). El valor que ofrece por defecto esta propiedad es escalar.



Based On

Ofrece una forma rápida de definir una variable. Cuando se selecciona, se despliega un diálogo que muestra todos los atributos (y dominios) definidos en la base de conocimiento; en dicho diálogo, simplemente se debe seleccionar un atributo, y a continuación se definirá automáticamente una variable con el mismo nombre y la misma definición que el atributo.

Por otra parte, es posible definir variables dentro del editor de código de cada objeto (source, eventos, etc.), haciendo uso del menú contextual (botón derecho) luego de escribir el nombre de la variable. Esto es, al escribir &nombreDeVariable y presionar botón derecho del mouse. En el menú contextual luego elegir Add Variable.

Variables colección

Es posible definir variables colección sobre cualquier tipo de datos ofrecido por GeneXus.

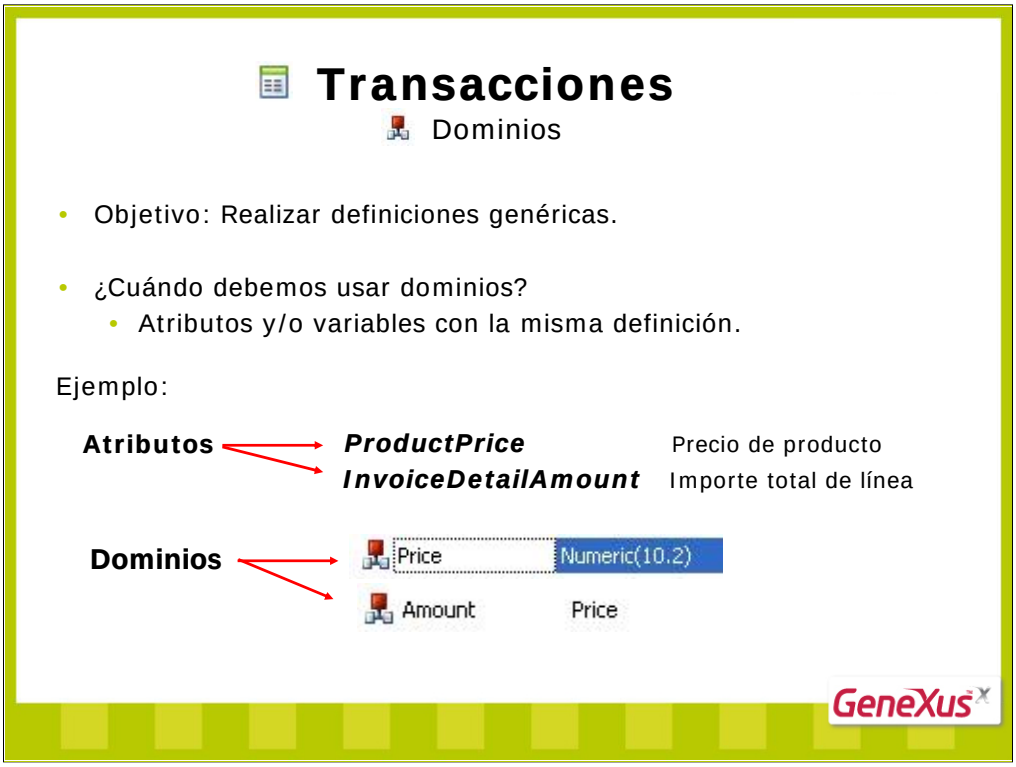
Para eso alcanza con clickear la correspondiente casilla en el editor de variables, o indicar el valor True en la propiedad Collection asociada a las variables.

Para recorrer luego los items coleccionados en dicha variable, se deberá utilizar la estructura de control “For in”.

Por ejemplo, hemos definido la variable &myNumbers de tipo Numeric(4.0)

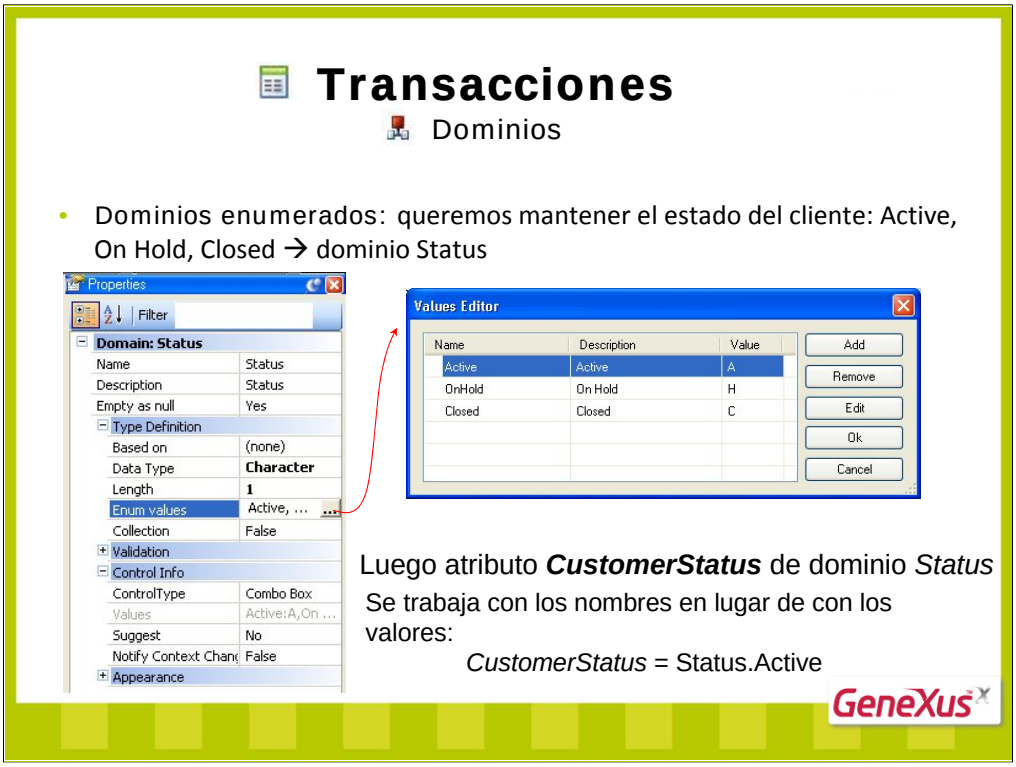
Para recorrer los valores almacenados se deberá crear una nueva variable de tipo Numeric(4.0). Creamos entonces la variable &OneNumber, y declaramos:

```
For &OneNumber in &myNumbers
----------
Endfor
```



Es común tener en una base de conocimiento atributos que comparten definiciones similares pero que no tienen relación entre sí. Por ejemplo, es común definir todos los nombres como atributos de tipo character y largo 20; o todos los importes, como atributos de tipo numérico y largo 10.2.

El objetivo de los dominios es permitir realizar definiciones genéricas. A modo de ejemplo, el atributo *InvoiceDetailAmount* es de tipo numérico y largo 10.2, y al mismo tiempo, el atributo *ProductPrice* es del mismo tipo y largo, en la misma base de conocimiento. De modo que podríamos definir un dominio de nombre *Price,* que sea de tipo numérico con largo 10.2, y luego uno Amount de tipo de datos el dominio Price anterior; a cada uno de los atributos anteriores le asignaríamos estos dominios. La ventaja de hacerlo así es que si en el futuro surge la necesidad de cambiar la definición de los atributos que representan importes, haríamos el cambio una sola vez (en el dominio *Price)*, propagándose éste automáticamente a los atributos *InvoiceDetailAmount* y *ProductPrice*.



Existe la posibilidad de trabajar con dominios enumerados (aquellos que representan valores finitos y particulares. Son muy conocidos en los lenguajes de programación. El caso de uso típico es para los días de la semana: cuando se necesita que una variable tome uno de los siguientes valores: Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo).

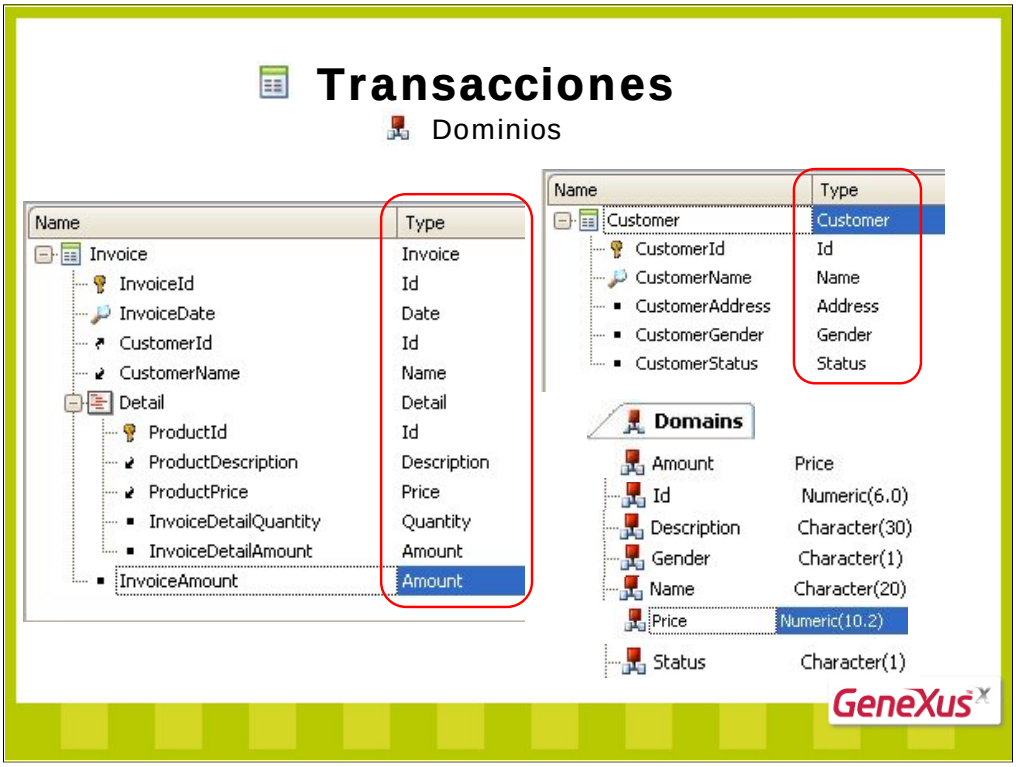
En nuestro caso, agregaremos a la estructura de la transacción Customer, un atributo que representa el estado del cliente en el sistema en un momento dado. Puede estar active, on hold o closed.

Una forma de representarlo es definiendo un dominio Status, que corresponde al tipo de datos Character(1) pero para el que decimos explícitamente que no queremos que pueda tomar todos los valores de ese tipo de datos, sino solo 3: A, H y C, y además le decimos que queremos trabajar dándole nombres a estos 3 valores y trabajar con sus nombres. Así, a A le asociamos el nombre 'Active', a H 'OnHold' y a C 'Closed'.

Al asociarle al atributo *CustomerStatus* el dominio Status (observar que GeneXus automáticamente lo sugiere al ingresar el atributo en la estructura luego de haber definido el dominio, debido a la terminación del nombre del atributo) internamente en el atributo de la tabla se guardará uno de los 3 valores: A, H o C, pero nosotros no tendremos que recordar esos valores, sino sus códigos: Active, OnHold y Closed.

Ya teníamos otro atributo con dominio enumerado: CustomerGender. En ese caso se componía de solamente 2 valores: Female y Male.

Observar cómo el Control Info ofrece para ControlType un Combo Box en lugar de un Edit. Esto tomará sentido cuando veamos el Form de una transacción un poco más adelante.



Así como podemos asociarle a un atributo un dominio (y a otro dominio), también lo podemos hacer para una variable. Un mismo dominio puede asignarse tanto a atributos como a variables, ya que su objetivo es exactamente el mismo.

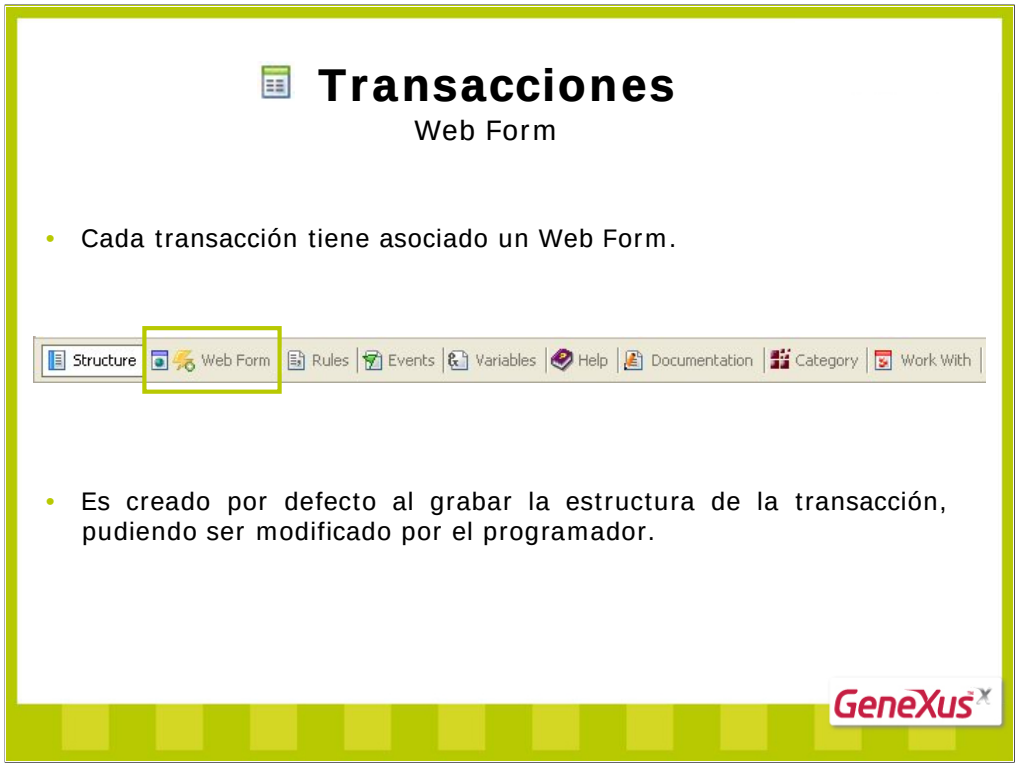
¿Cómo definir un dominio?

Existen varios caminos:

1) Opción Domains de el Folder View: Mediante un editor similar al de las variables, se podrá definir un nuevo dominio.

2) Dado que en la pantalla de configuración de las propiedades de un atributo es donde se le asigna a un atributo un dominio existente, en dicha pantalla se ofrece un botón para crear un dominio nuevo. Ídem con el diálogo de definición de variables.

3) En la estructura de la transacción es posible definir un nuevo domino en el campo de definición del tipo de datos de un atributo, simplemente escribiendo sobre esa misma línea, el nombre del dominio, y asignándole el tipo de datos. Por ejemplo, digitando Price = Numeric(10.2) sobre la columna Type del atributo que se está definiendo, queda también definido el dominio de nombre Price, con el tipo de datos Numeric(10.2).



Para cada transacción, GeneXus crea un form web, que será la interfaz con el usuario.

Es creado por defecto por GeneXus al momento de grabar la estructura de la transacción, y contienen todos los atributos incluidos en la misma, con sus respectivas descripciones, además de algunos botones.

Si bien es creado por defecto, es posible modificarlo para dejarlo más vistoso, cambiar por ejemplo controles de tipo edit a otros tipos de controles, agregar y/o quitar botones, etc.

En el ejemplo se muestra el form Web correspondiente a la transacción “Invoice”. A través de este form el usuario final podrá ingresar, modificar y eliminar facturas en la aplicación Web.

El control Error Viewer se utiliza para poder ubicar y programar el lugar donde queremos que los mensajes generales le sean desplegados al usuario final de una transacción Web.

Podremos especificar entre otras cosas el lugar donde queremos que este control se ubique dentro del form, su tipo de letra, color, etc.

Observar que el segundo nivel de la transacción aparece en el form como un control grid.

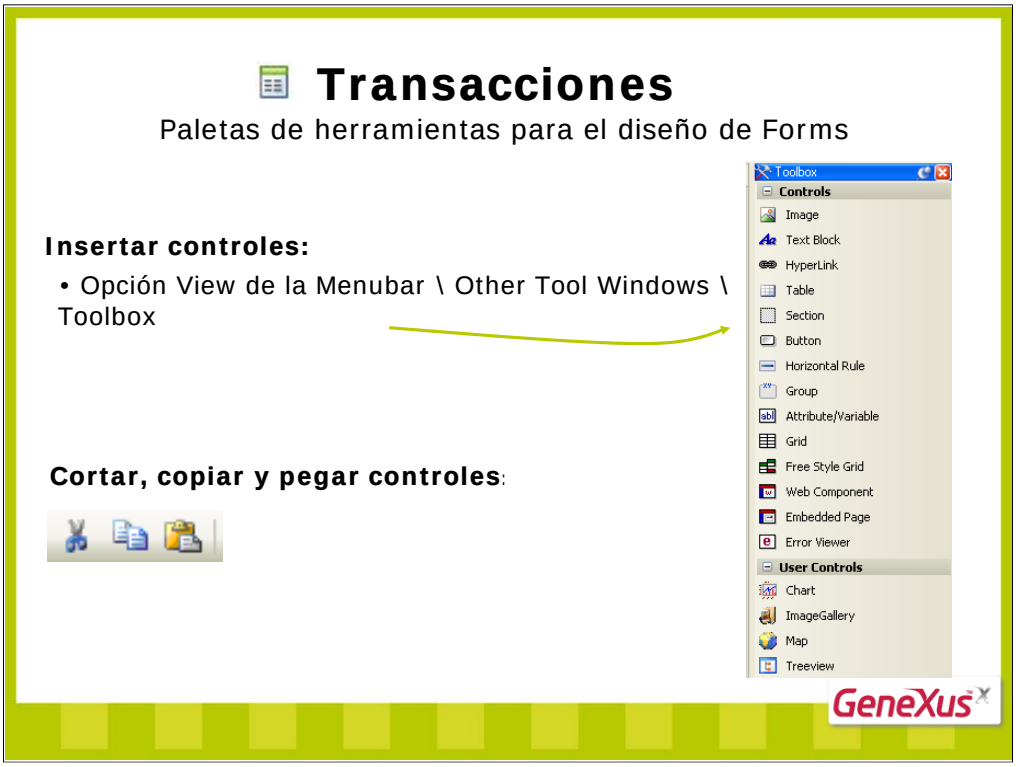
Transacciones

Web Form de la transacción "Customer"

Dominios básicos → controles **Edit**

Dominios enumerados → controles **Combo Box**

GeneXus



Podemos definir un **control** como **un área de la interfaz con el usuario, que tiene una forma y un comportamiento determinado**.

Existen distintos controles:

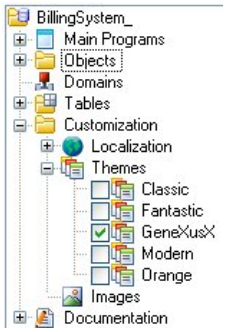
- **Texto:** Permite colocar texto fijo
- **Atributo/Variable:** Permite colocar atributos o variables.
- **Línea horizontal**
- **Error Viewer**
- **Table:** Insertar tablas en el form
- **Grid:** Permite definir grillas de datos.
- **Botón:** Permite incluir botones en los forms.
- **Bitmap:** Permite definir bitmaps estáticos, etc.

Paleta de herramientas para insertar controles: Cuando se está editando un form, se encuentra disponible una paleta de herramientas (Toolbox) que ofrece los controles posibles de insertar en el mismo. Simplemente se deberá seleccionar el control y arrastrarlo sobre el form.

Transacciones

Controles en Web Form

- Cada control del Web form podrá tener una **clase** asociada, de un objeto **theme** (tema) determinado, asociado al objeto.
- Al crear una KB, se crea por defecto el tema "GenexusX" y todos los objetos que se creen tendrán este tema asociado.
- Esto permitirá independizar el diseño de la interfaz, de la programación.
  - Cada tema tendrá definidas muchas clases para cada tipo de control.
  - El analista solo asocia un tema al objeto, y una clase a cada control del form y puede olvidarse del diseño de los mismos.



- El control hereda el diseño de la clase del tema al que esté asociado.

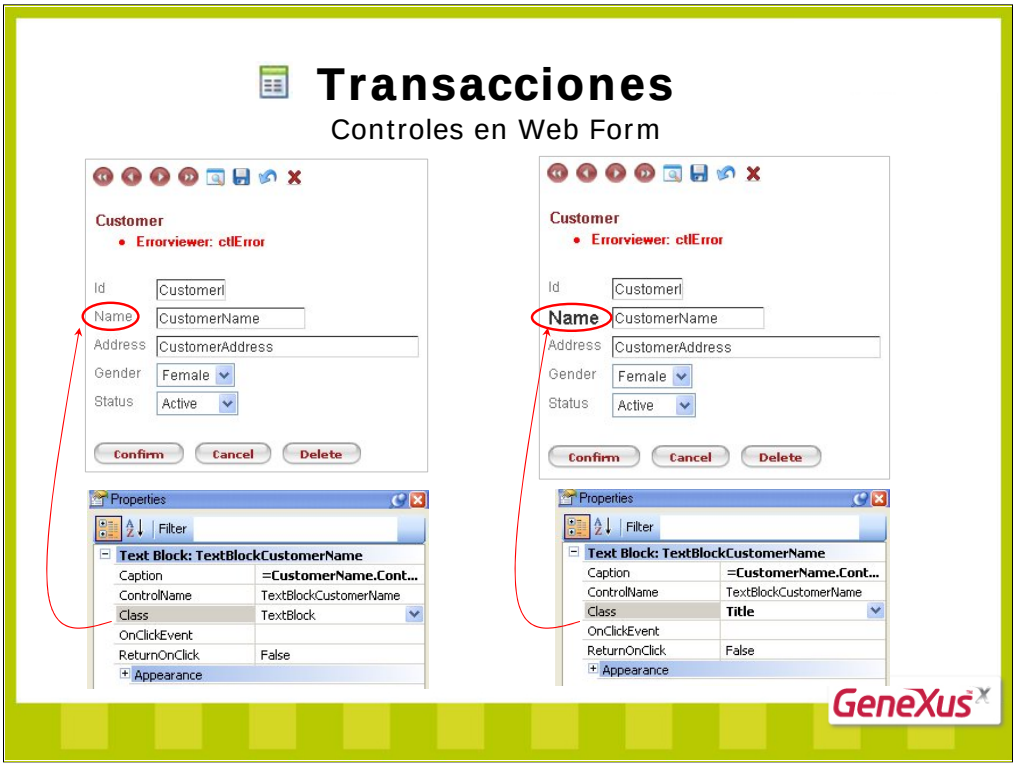
Para lograr separar los aspectos de diseño gráfico de un sitio web, de los aspectos de programación propiamente dichos, existe un tipo de objeto llamado Theme (Tema, en español).

El objetivo de esta separación es poder paralelizar el desarrollo de un sitio Web, permitiéndole al programador abocarse a las tareas de programación, y apoyarse en un diseñador gráfico para que defina las cuestiones de diseño. De esta manera el diseñador gráfico configurará el "tema" elegido por el programador, y el programador sólo deberá aplicarlo a los objetos de su base de conocimiento.

Un objeto "tema" contendrá un conjunto de clases, para cada tipo de control de los que pueden aparecer en el form Web de un objeto GeneXus. Por ejemplo, tendrá varias clases para el control botón, algunas para el control atributo, una clase para el control Error Viewer, etc.

Cuando se crea una base de conocimiento, automáticamente aparecerá dentro del Folder View la carpeta "Customization" con objetos Theme predefinidos que podrán importarse en la KB. Por defecto se importa el de nombre "GeneXusX". Este **tema** contiene un conjunto de **clases** asociadas a los distintos controles existentes.

Los objetos con form Web que se vayan creando tendrán por defecto asociado este Theme, y **cada control** que aparezca en sus forms tendrá **asociado una clase** de ese tema, para este control.

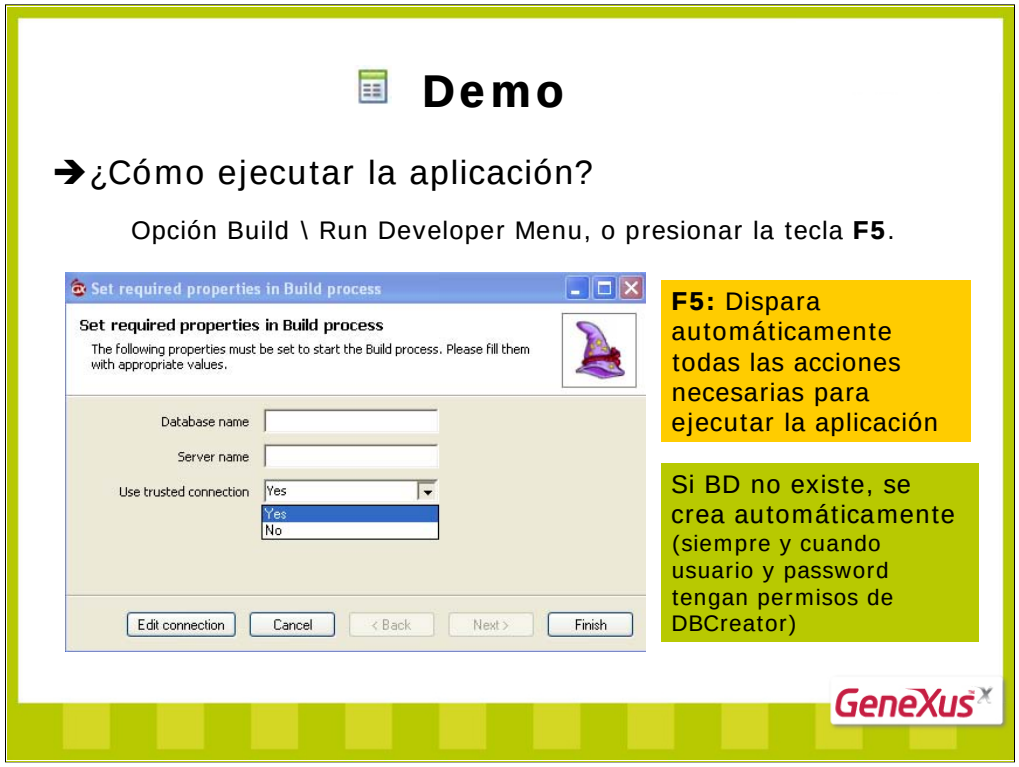


De este modo, cuando el analista crea la transacción “Customer” e ingresa su estructura, al grabar podrá apreciar que el form Web tendrá automáticamente el aspecto que se muestra en la pantalla de arriba a la izquierda. ¿Quién dio formato a todos los controles si no lo hizo el analista explícitamente?

Pues bien, todas las transacciones tendrán asociado por defecto el theme “GeneXusX” y todos los controles que se coloquen en el form, tendrán clases de este tema asociadas.

Si sobre el Text Block (control de texto) que se muestra arriba (Name), se editan sus propiedades (F4), se puede observar la **propiedad Class**, que tiene configurada la clase **TextBlock**. Esta propiedad puede ser cambiada por el analista. Al cliquear el combo box podremos ver una lista de clases posibles para ese control (son las que figuran en el tema asociado). A la derecha hemos cambiado la clase por una de nombre Title y en el form podemos ver la repercusión en el diseño del TextBlock.

No entraremos en detalle en este tema en el presente curso.



Recordemos que al momento de creación de la KB, se indicó el generador por defecto a utilizar. Ahora se debe completar la información necesaria para terminar de definir el ambiente de implementación.

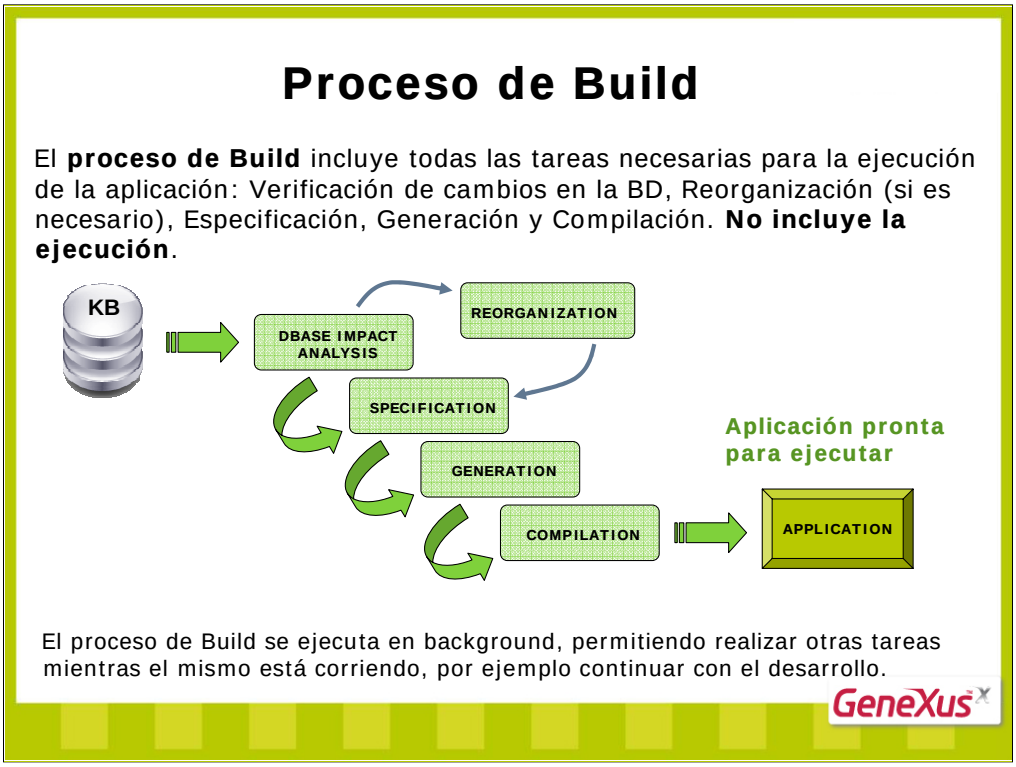
Database name: Nombre de la base de datos que estará asociada a la aplicación.

Server name: Nombre del servidor de base de datos que la manejará.

Use trusted connection: Yes
No (deberá indicarse usuario y contraseña válido en el DBMS)

Consideraciones:

Si la base de datos no existiera, GeneXus la creará, siempre y cuando el usuario y contraseña configurados en la ventana de diálogo tengan permiso de DBCreator en el DBMS.



Cada vez que se ejecuta la aplicación (F5), GeneXus realiza una comparación entre las definiciones actuales de todos los objetos y las definiciones de la última ejecución. Esta comparación se llama **análisis de impacto**. Este nombre es autodescriptivo: GeneXus analiza el impacto causado por las nuevas definiciones, y el resultado del análisis de impacto es un **reporte de análisis de impacto** (IAR: Impact Analisis Report) que informa al programador qué cambios físicos o estructurales habría que realizar en la base de datos asociada.

Si por ejemplo se creó una nueva transacción, esto provocará (tal como es el objetivo de las transacciones) que se creen las tablas correspondientes en la base de datos (ver siguiente página).

En el ejemplo, tras crear las 2 transacciones "Customer" e "Invoice" y dar F5, aparece el reporte de análisis de impacto que puede verse, donde se listan entre otras cosas, la estructura que tendrá cada tabla que se creará, con sus atributos, tipos de datos, los índices que se crearán sobre las tablas, las claves foráneas (observar que el reporte de la tabla INVOICE dice que se referenciará la tabla CUSTOMER (esto se debe al atributo CustomerId de la transacción Invoice que se llama igual que el atributo CustomerId de CUSTOMER, constituyendo por tanto una clave foránea a dicha tabla).

Si el programador está de acuerdo con los cambios estructurales informados en el reporte de análisis de impacto, podrá proseguir, pasando a **reorganizar la base de datos**. El término reorganizar refiere a efectuar cambios físicos en la base de datos.

Si en cambio el programador encuentra que algo de lo informado en el reporte de análisis de impacto no era lo que pretendía lograr, podrá no efectuar la reorganización, y efectuar en cambio las redefiniciones que crea convenientes.

Cuando se decide efectuar una **reorganización** GeneXus genera programas (en el lenguaje elegido cuando se creó la KB) que implementan las modificaciones a realizar en la base de datos. La ejecución de estos programas tiene por resultado la obtención de una nueva versión de la base de datos con los cambios efectuados.

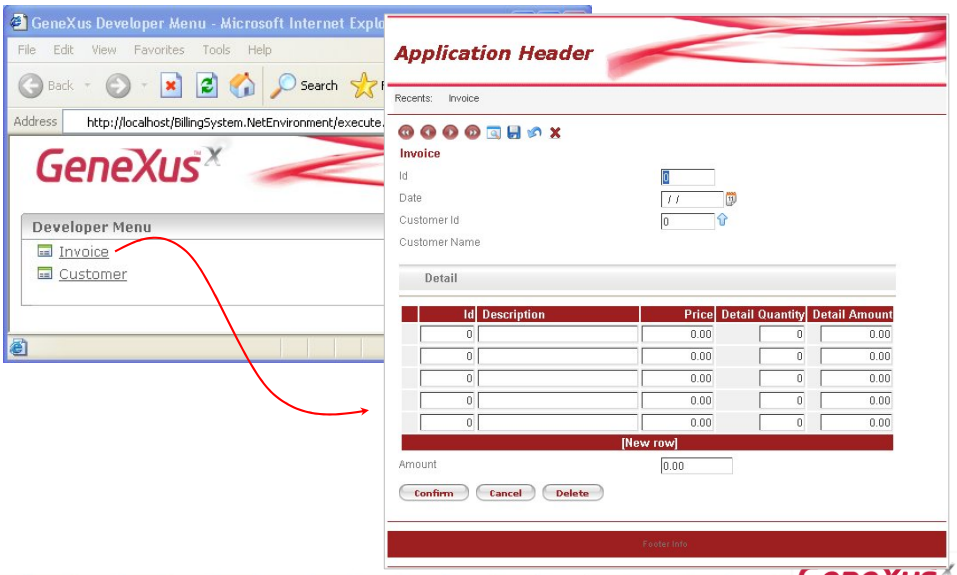
El siguiente paso será obtener los programas de aplicación asociados a los objetos GeneXus. Para ello GeneXus deberá **especificar, generar y compilar los programas de aplicación**.

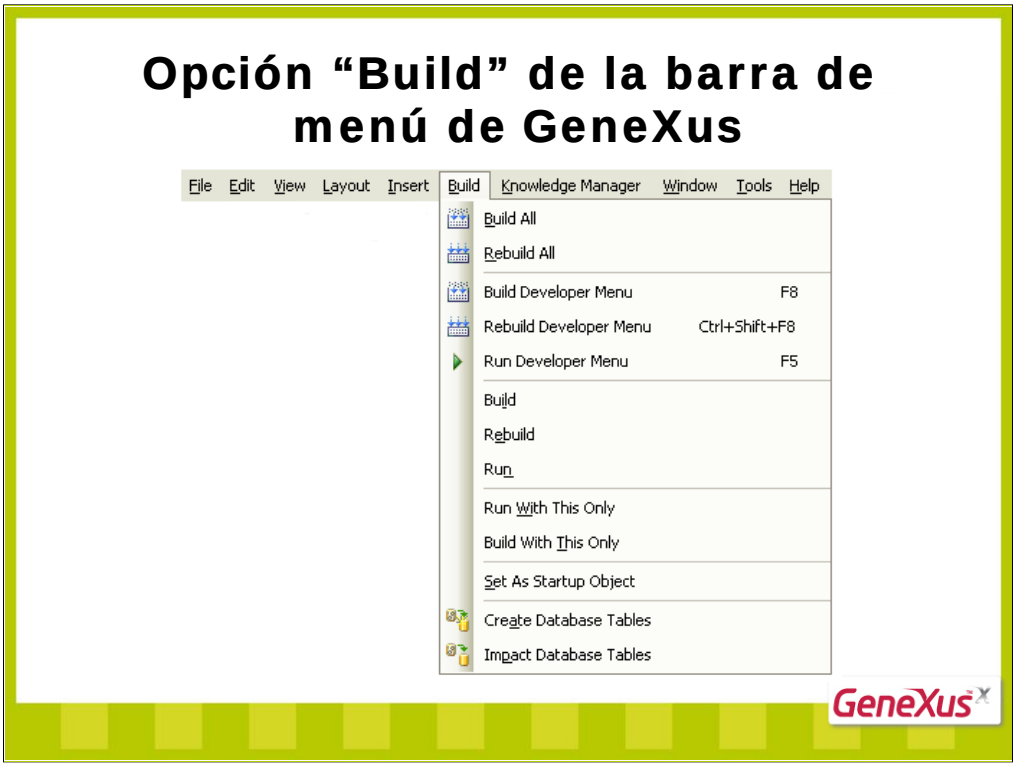
Especificar un objeto significa que GeneXus analizará todo lo definido en cada uno de los elementos que lo componen: estructura, forms, u otros elementos según corresponda. GeneXus verificará la sintaxis de las definiciones, la validez de lo definido, y como resultado de la especificación mostrará al usuario un **listado de navegación**, en el cual informará la lógica que ha interpretado, y si hay alguna advertencia o error.

Generar un objeto, significa que GeneXus escribirá líneas de código que implementen la programación del mismo, en el lenguaje elegido.

Compilar los programas significa que el código escrito (generado) sea convertido a lenguaje máquina (binario) para que puedan ser ejecutados.

Ejecución


GeneXus Developer Menu - Microsoft Internet Explo
File Edit View Favorites Tools Help
Back Search
Address http://localhost/BillingSystem.NetEnvironment/execute.
GeneXus
Developer Menu
Invoice
Customer
Application Header
Recents: Invoice
Invoice
Id
Date
Customer Id
Customer Name
Detail
Id Description Price Detail Quantity Detail Amount
[New row]
Amount
Confirm Cancel Delete
Footer Info
GeneXus



Opciones del ítem “Build”:

• **Build All y Rebuild All:** Estas opciones se utilizan cuando no se está seguro del impacto de los cambios y se necesita tener actualizadas las últimas definiciones. La opción Build All realiza todas las acciones que estén pendientes, mientras que Rebuild All forzará todas ellas.

Las acciones son:

- Salvar todos los objetos que no estén salvados.
- Reorganizar la base de datos, si es necesario.
- Especificar solo los objetos que han sido modificados (si se seleccionó Build All), o forzar la especificación de todos (si se seleccionó Rebuild All).
- Generar los objetos
- Compilar los objetos definidos Main
- Compilar el Developer Menu

• **Build / Rebuild Developer Menu:** Similar a las opciones anteriores pero aplicadas solamente al Developer Menu. No lo ejecuta.

• **Run Developer Menu:** Ejecuta todas las acciones que estén pendientes y ejecuta el Developer Menu.

- Salva todos los objetos que no estén salvados.
- Reorganizarla base de datos, si es necesario.
- Especifica solo los objetos que han sido modificados.
- Genera los objetos
- Compila y ejecuta el Developer Menu

• **Build / Rebuild / Run** options: Opciones que aplican a objetos definidos como Main. Disparan las siguientes acciones:

- Salvan los objetos que no estén salvados.
- Reorganizan la base de datos, si es necesario.
- Se especifican solamente los objetos que sufrieron cambios (si se seleccionó Build), o se fuerza la especificación de todos los objetos pertenecientes al árbol de llamadas del objeto Main (si se seleccionó Rebuild).
- Generación
- Compilación del objeto Main
- Ejecución del objeto Main (si se seleccionó la opción Run)

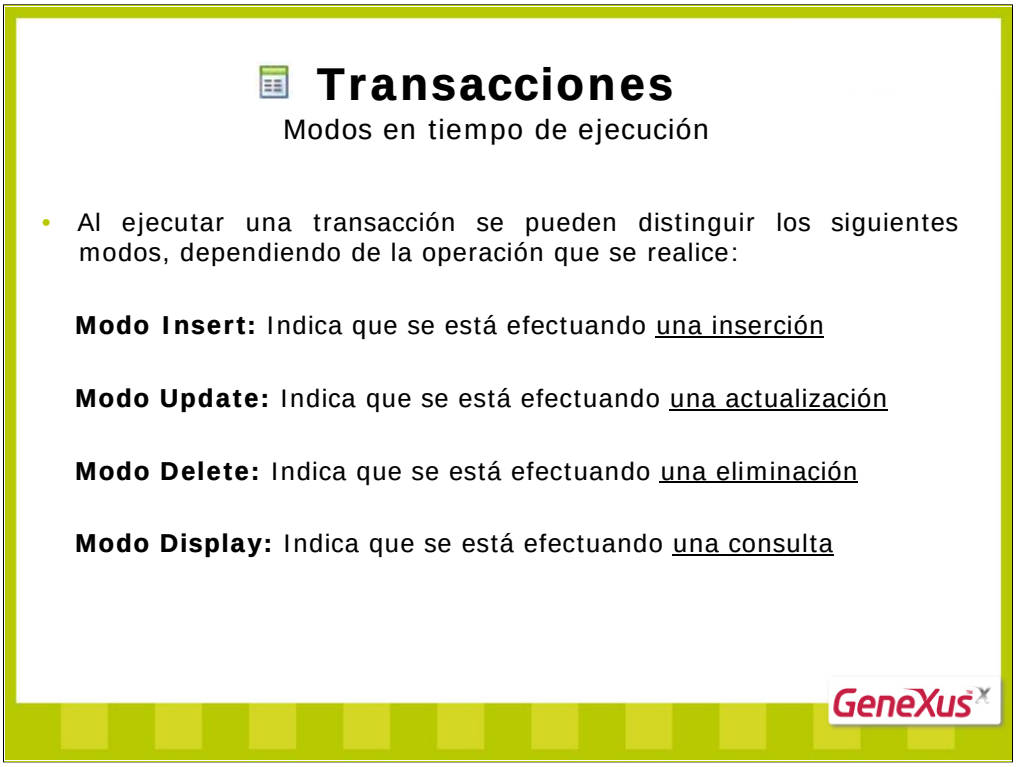
• **Build / Run with this only** options: Build y ejecución del objeto definido como Startup. Por defecto el objeto Startup es el Developer Menu.

- Salvan todos objetos que no estén salvados.
- Reorganizan la base de datos, si es necesario.
- Especifican solamente el objeto seleccionado.
- Generan solamente el objeto seleccionado.
- Compilan el objeto Startup
- Se ejecuta el objeto Startup (si se seleccionó Run).

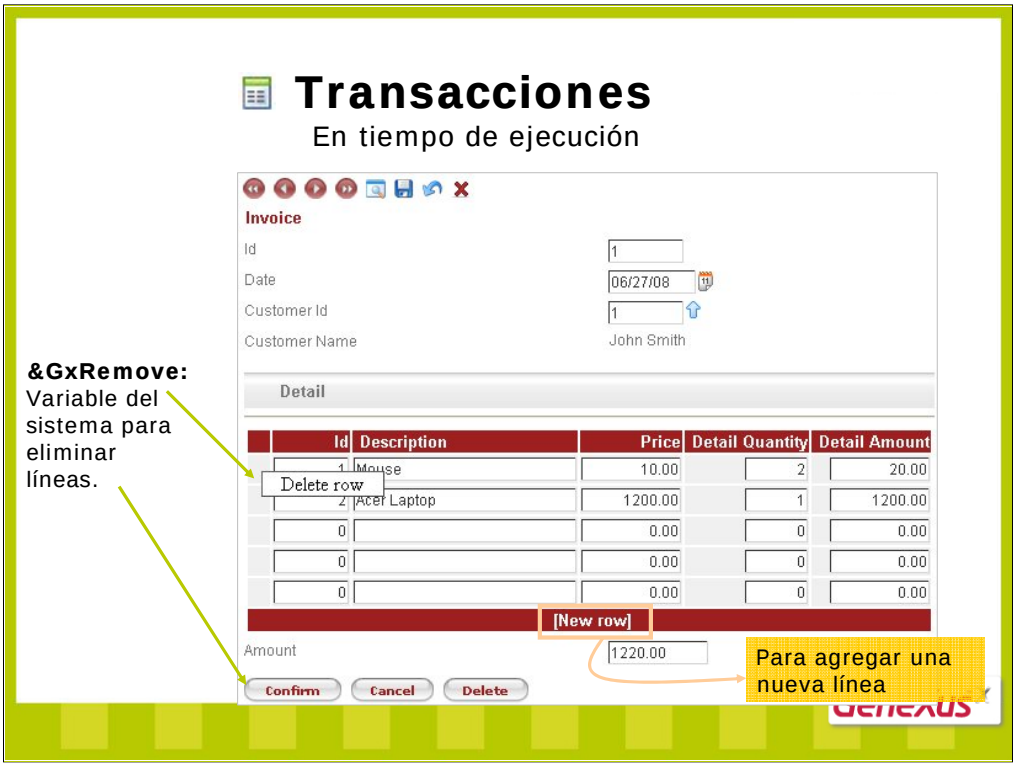
• **Set As Startup Object**: El objeto indicado pasará a ser el objeto Startup de la aplicación, o sea el objeto que finalmente se ejecutará cuando se presione la tecla F5. Por defecto el objeto Startup es el Developer Menu.

• **Create Database Tables**: Crea nuevamente las tablas. Se pierden los datos que pudieran estar almacenados previamente.

• **Impact Database Tables**: Se realiza un impacto sobre las tablas de la base de datos.



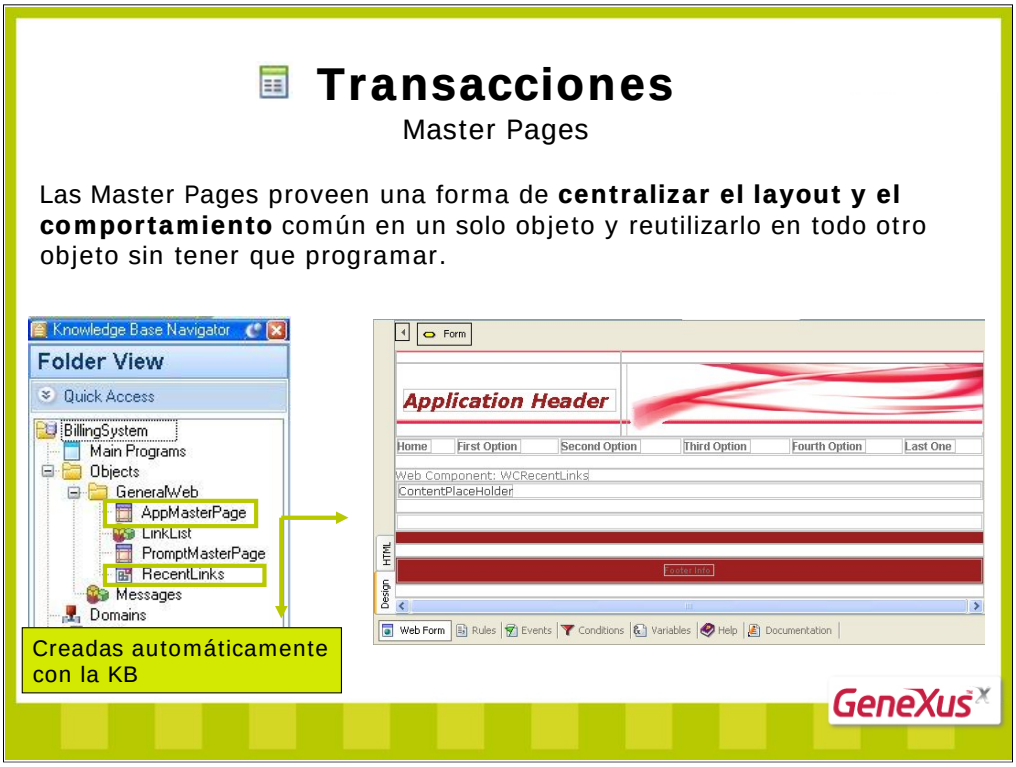
Dependiendo del ambiente de generación, habrá algunas diferencias en lo que se refiere a la operativa de las transacciones en tiempo de ejecución. No obstante, más allá de la plataforma, cada vez que se realice una operación de inserción, actualización, eliminación, o consulta a la base de datos a través de una transacción, habrá un modo asociado.



Siempre se mostrará un número de líneas vacías fijas para ser ingresadas por el usuario (valor configurable por el analista a nivel del grid, en su propiedad “Rows”). Por defecto se muestran 5 líneas vacías.

Si el usuario ingresó las 5 líneas y necesita ingresar más, le bastará con presionar New Row., o simplemente presionar Enter y se agregará una nueva línea.

Para poder eliminar líneas en ejecución, GeneXus incorpora automáticamente en el grid del form Web una primera columna conformada una variable del sistema de nombre *&GxRemove*, en forma de check box. Para visualizar este check box, se debe presionar el botón derecho del mouse sobre la línea a eliminar. Luego se deberá presionar el botón Confirm.



Tener un look & feel consistente es hoy en día un deber de toda aplicación Web.

Crear y mantener cada página de una aplicación Web asegurando la consistencia con el resto del sitio toma gran tiempo de programación.

Al crear una base de conocimiento GeneXus X creará también dos objetos de tipo Master Page:

AppMasterPage: Para la aplicación.

PromptMasterPage: Para los prompts.

Las Master Pages proveen una forma de centralizar el layout y el comportamiento común en un solo objeto y reutilizarlo en todo otro objeto sin tener que programar. Esto significa que la modificación de alguna parte del layout o del comportamiento común es tan fácil como modificarla en un único objeto y ¡listo!.

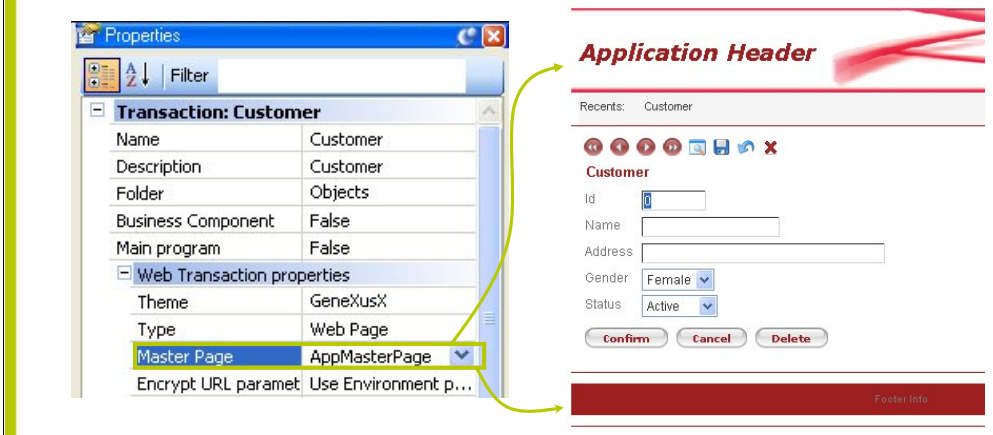
En una misma base de conocimiento se pueden definir tantas Master Pages como se desee.

Cuando estudiemos el objeto Web Panel comprenderemos que una Master Page será en particular un Web Panel categorizado como “Master Page” con todo lo que sea el Layout y comportamiento común a todas las páginas del sitio; dentro de este objeto se dejará un espacio para cargar en cada oportunidad la página que corresponda (el contenido variable del sitio). Eso se hará en el control presente en el form de nombre Content Placeholder.

Transacciones

Master Pages

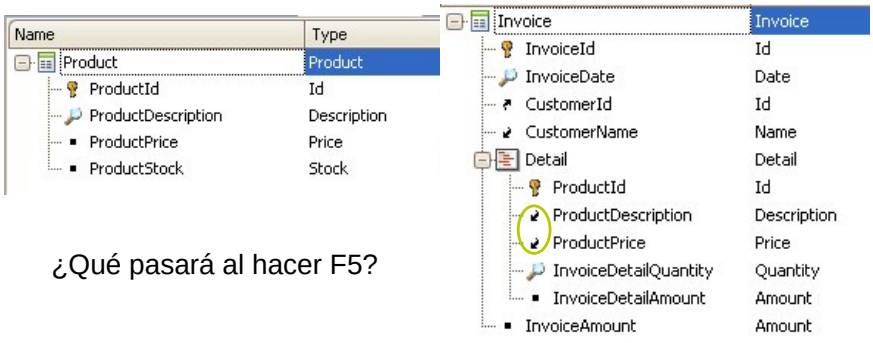
- Propiedad Master Page



Las páginas web que implementan el contenido variable, se asocian a la Master Page, de manera que cada vez que se ejecuten, se carguen con ese “contexto” (el de la Master Page).

Demo

- Agregar la transacción “Product” faltante y ver cómo se modifican los íconos en “Invoice”



¿Qué pasará al hacer F5?

Obsérvese cómo luego de crear la transacción “Product” GeneXus normaliza e indica gráficamente en la estructura de la transacción “Invoice” que los atributos, antes propios de la tabla INVOICEDETAIL, ahora serán inferidos, a través de la nueva clave foránea *ProductId*.

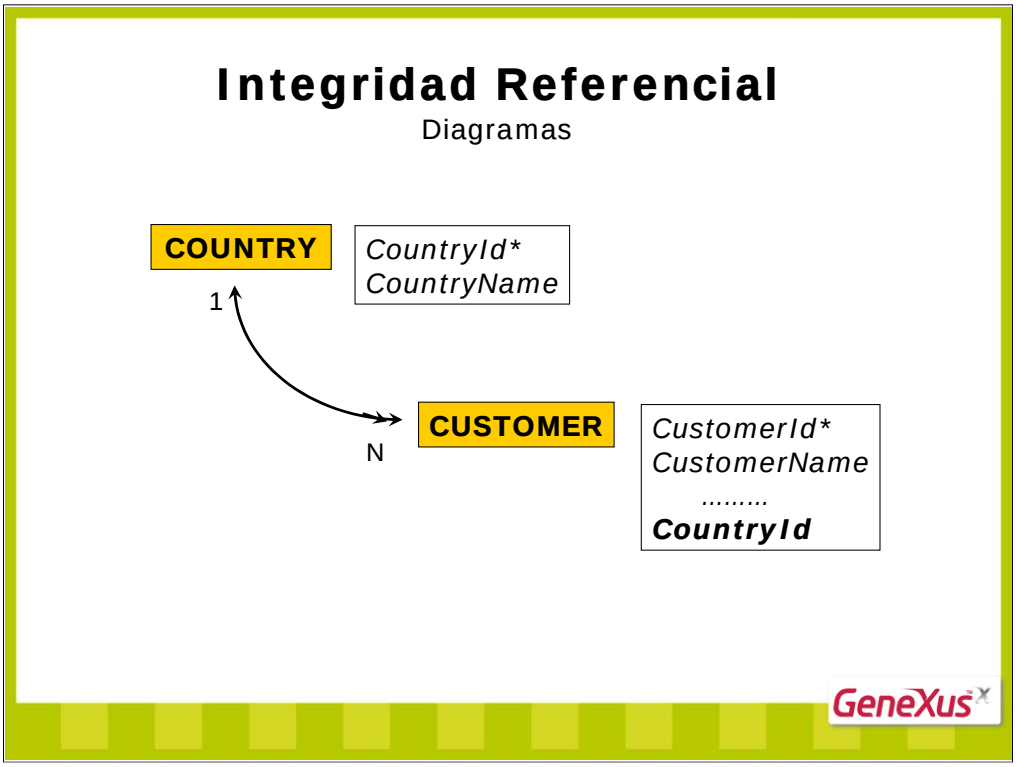
La clave primaria de la tabla INVOICEDETAIL seguirá siendo compuesta: {*InvoiceId*, *ProductId*}, pero además, el atributo *ProductId* sólo, será una FK a la tabla PRODUCT, a partir del cuál se inferirán los atributos *ProductDescription* y *ProductPrice*.

Luego del F5, naturalmente se informará de la necesidad de reorganizar la base de datos, creándose la tabla PRODUCT y modificándose la tabla INVOICEDETAIL de la manera que indicamos en los párrafos anteriores.

¿Qué pasará con los registros ya existentes en la tabla INVOICEDETAIL, para los que existían valores de *ProductDescription* y *ProductPrice*?

Como se verá cuando se ejecute, el programa de reorganización agrega una rutina para crear los registros de productos en la tabla PRODUCT con esa información pre-existente. ¿Qué pasará si existía el mismo producto en varias líneas de distintas facturas?

Integridad referencial



El concepto de integridad referencial es un concepto que tiene que ver con las bases de datos relacionales.

Se refiere a que debe haber consistencia entre los datos de las distintas tablas de una base de datos relacional.

Las tablas de una base de datos relacional se encuentran relacionadas por atributos que tienen en común. Estas relaciones implican que los datos de las tablas no son independientes, sino que al insertar, modificar y eliminar registros en una tabla, se deben tener en cuenta los datos de las otras tablas para que siempre se conserve la consistencia de la información en la base de datos.

Si tenemos un modelo de datos relacional con las tablas:

COUNTRY (*CountryId*, *CountryName*)
Clave Primaria: *CountryId*

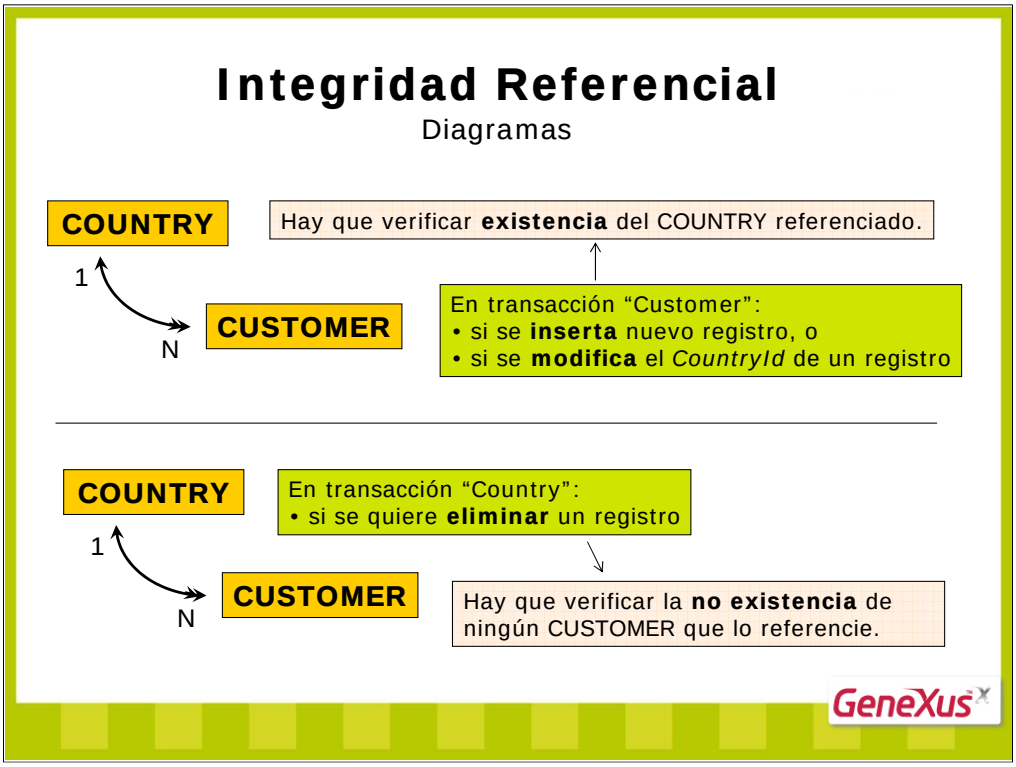
CUSTOMER (*CustomerId*, *CustomerName*, *CountryId*, *CustomerAddress*, *CustomerGender*, *CustomerStatus*)
Clave Primaria: *CustomerId*
Clave Foránea: *CountryId* (COUNTRY)

El hecho de que el atributo *CountryId* en la tabla CUSTOMER sea una clave foránea con respecto a la tabla COUNTRY, establece una relación entre ambas tablas. La relación entre ellas puede verse en el diagrama que mostramos arriba.

En dicho diagrama, la flecha simple representa la existencia de una instancia de la tabla apuntada, para cada instancia de la otra (es decir, que para cada cliente existe un y solo un país). La flecha doble representa la ocurrencia de varias instancias de la tabla apuntada, para cada instancia de la otra tabla (es decir, que para cada país, existen muchos clientes).

Se dice que la relación entre la tabla COUNTRY y la tabla CUSTOMER es 1 a N (1 a muchos).

Recíprocamente, la relación entre CUSTOMER y COUNTRY es N a 1 (muchos a 1).



En la terminología GeneXus, decimos que existe una relación de subordinación entre ambas tablas. Decimos que:

COUNTRY está **superordinada** a CUSTOMER

CUSTOMER está **subordinada** a COUNTRY

Significando que:

• Cuando se **crea** o **modifica** un registro en la **tabla subordinada** (CUSTOMER), **debe existir** el registro relacionado en la tabla **superordinada** (COUNTRY).

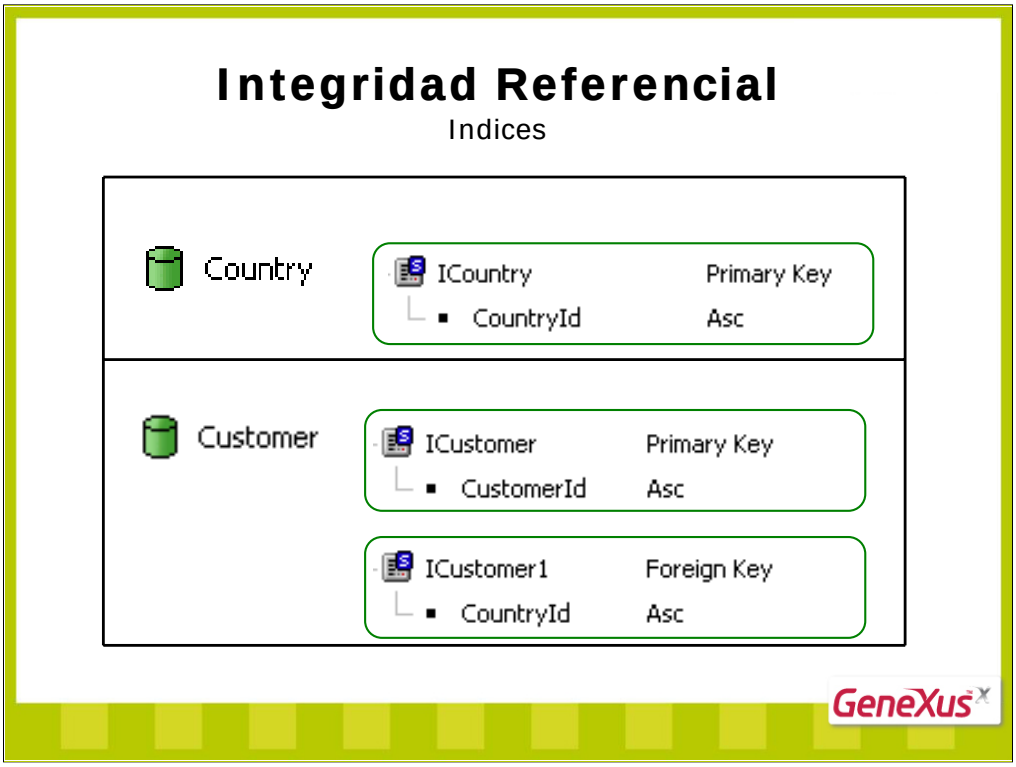
• Cuando se elimina un registro en la tabla **superordinada** (COUNTRY), **no deben existir** registros relacionados en la tabla **subordinada** (CUSTOMER).

Debido a esta relación entre las tablas, la información contenida en ellas no es independiente, y es necesario realizar controles para que los datos sean consistentes. A estos controles se les llama de **Integridad Referencial** y básicamente son los siguientes:

• Cuando se **inserta** o **modifica** un registro en la tabla CUSTOMER, el valor ingresado en el atributo que es clave foránea (*CountryId*), **debe existir** como valor de clave primaria de un registro en la tabla COUNTRY.

• Cuando se **elimina** un registro en la tabla COUNTRY, **no deben existir** registros en la tabla CUSTOMER cuyos valores de la clave foránea (*CountryId*), sean iguales al valor de la clave primaria del registro que se desea eliminar.

GeneXus genera los programas asociados a las **transacciones**, incluyendo en el código generado estos controles de **Integridad Referencial**. Por esta razón, si el usuario final inserta (o modifica) un cliente a través de la transacción "Customer", se validará automáticamente que el valor ingresado en el código de país *CountryId*, exista como clave primaria de un registro en la tabla COUNTRY. En caso de fallar este control de integridad referencial, un mensaje se le desplegará al usuario indicándole que no se encontró ese país.



Los índices son vías de acceso eficientes a las tablas.

GeneXus crea automáticamente algunos de ellos, y los otros deberán ser creados por el programador cuando éste así lo determine, basándose en criterios de optimización.

Existen cuatro tipos de índices en GeneXus:

- Primarios
- Foráneos
- De usuario
- Temporales

De todos ellos, los únicos que no son creados automáticamente por GeneXus son los “de usuario”.

En cuanto a los tipos de índices que son creados por GeneXus, la diferencia que hay entre ellos es el momento en que son creados y el tiempo durante el cual se mantienen.

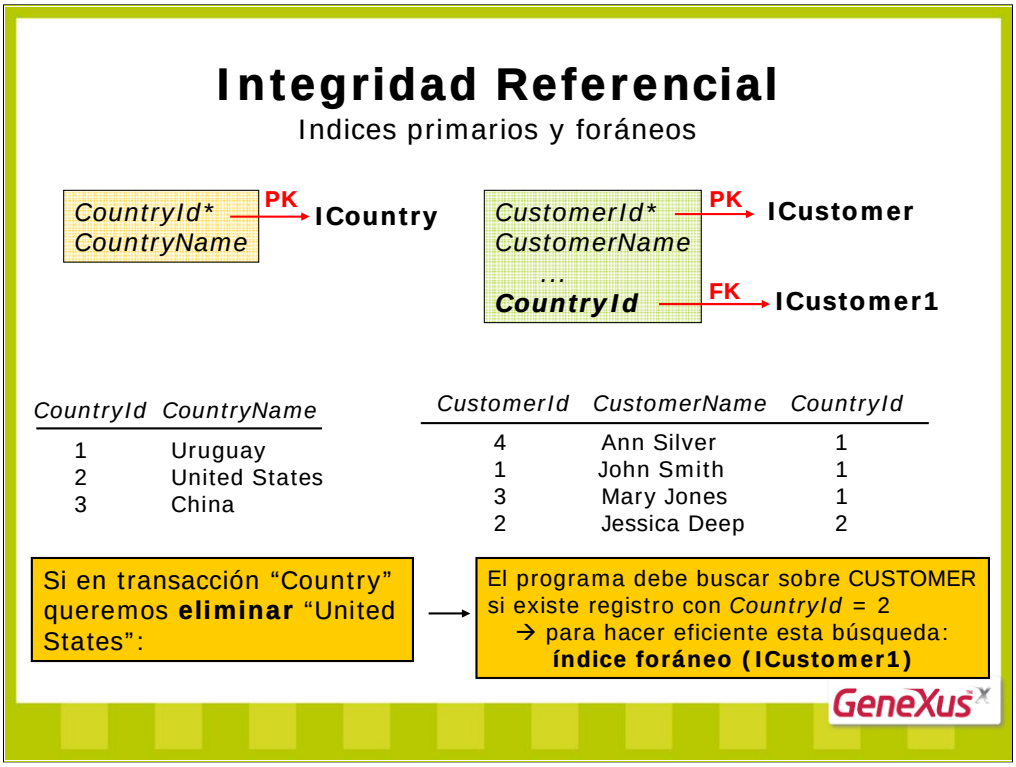
- Los índices primarios y foráneos son creados al momento de crear o reorganizar las tablas que componen la base de datos, y de ahí en más son mantenidos automáticamente por GeneXus.

- Los índices temporales, en cambio, son creados al ejecutar las aplicaciones, para acceder a tablas ordenadas por algún atributo o conjunto de atributos para el/los que no existe un índice de usuario creado; éstos se crean en tiempo de ejecución de las aplicaciones, se utilizan, y luego se eliminan.

ÍNDICES PRIMARIOS Y FORÁNEOS

GeneXus crea para cada tabla un índice por su clave primaria (**índice primario**), y un índice por cada clave foránea que la tabla tenga (**índices foráneos**). ¿Por qué crear índices primarios y foráneos para las tablas desde el comienzo en forma automática, siendo que luego deben ser mantenidos?

Sean las tablas COUNTRY y CUSTOMER, que vimos un par de páginas atrás, creadas en nuestro modelo de datos a partir de las estructuras de las transacciones "Country" y "Customer”. Existe entre estas tablas una relación 1-N, que viene dada por el hecho de que el atributo *CountryId* en la tabla CUSTOMER es una clave foránea con respecto a la tabla COUNTRY.



Por existir esta relación, GeneXus incluye en los programas asociados a las transacciones "Country" y "Customer", los controles de integridad referencial pertinentes. Estos controles son:

• Si el usuario final **inserta** o **modifica** un cliente a través de la transacción "Customer", se validará automáticamente que el valor ingresado en la clave foránea *CountryId* exista como clave primaria de un registro en la tabla COUNTRY. En caso de fallar este control de integridad referencial, se le indicará al usuario que no se encontró ese país.

Para controlar esto, se debe buscar en la tabla COUNTRY la existencia de un registro que tenga ese valor de *CountryId* como clave primaria; dado que se debe consultar la tabla COUNTRY, buscando por la clave primaria, resulta evidente que la búsqueda puede optimizarse si existe un índice por la clave primaria en dicha tabla.

• Si el usuario final intenta **dar de baja** un país a través de la transacción “Country”, se validará automáticamente que no existan clientes con dicho país asociado, como clave foránea; en caso de encontrar un registro en la tabla CUSTOMER, cuyo valor de clave foránea *CountryId* sea el que se desea eliminar, se le indicará al usuario que no es posible eliminar el país (ya que de lo contrario quedarían datos inconsistentes en la base de datos).

Para controlar esto se debe consultar la tabla CUSTOMER, buscando por la clave foránea *CountryId*. Esta búsqueda será óptima si existe un índice por *CountryId* en la misma.

Control de unicidad de clave primaria

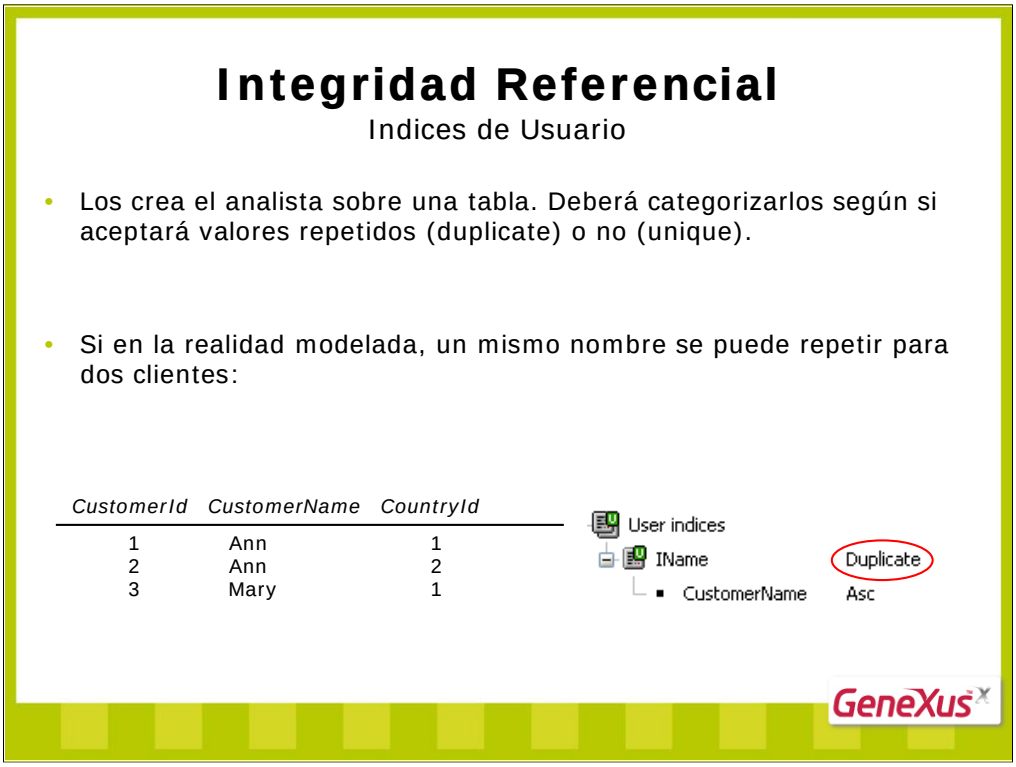
Otro control que GeneXus también incluye en los programas asociados a las transacciones es la unicidad de la clave primaria; esto es, en ninguna tabla podrán existir dos registros con el mismo valor en la clave primaria.

Para controlar esto, cuando el usuario final intenta insertar un registro se valida automáticamente que el valor ingresado para la clave primaria no exista ya como clave primaria de otro registro en la tabla.

Para hacer esta búsqueda con eficiencia, se debe utilizar el índice primario de la tabla.

Concluyendo, GeneXus al crear cada tabla de la base de datos crea también su índice primario, y un índice foráneo por cada clave foránea que la tabla contenga.

La creación de estos índices permite realizar los controles de integridad referencial y de unicidad de clave primaria accediendo a las tablas de forma eficiente.



Índices de usuario

Estos índices deben ser definidos explícitamente por el analista. **No son definidos automáticamente**. Se dividen en **duplicate** y **unique.**

Un índice de usuario **duplicate** es aquel que se define para atributos de una tabla para los que pueden existir varios registros con el mismo valor en los mismos (es decir, se define para atributos que no son una clave candidata).

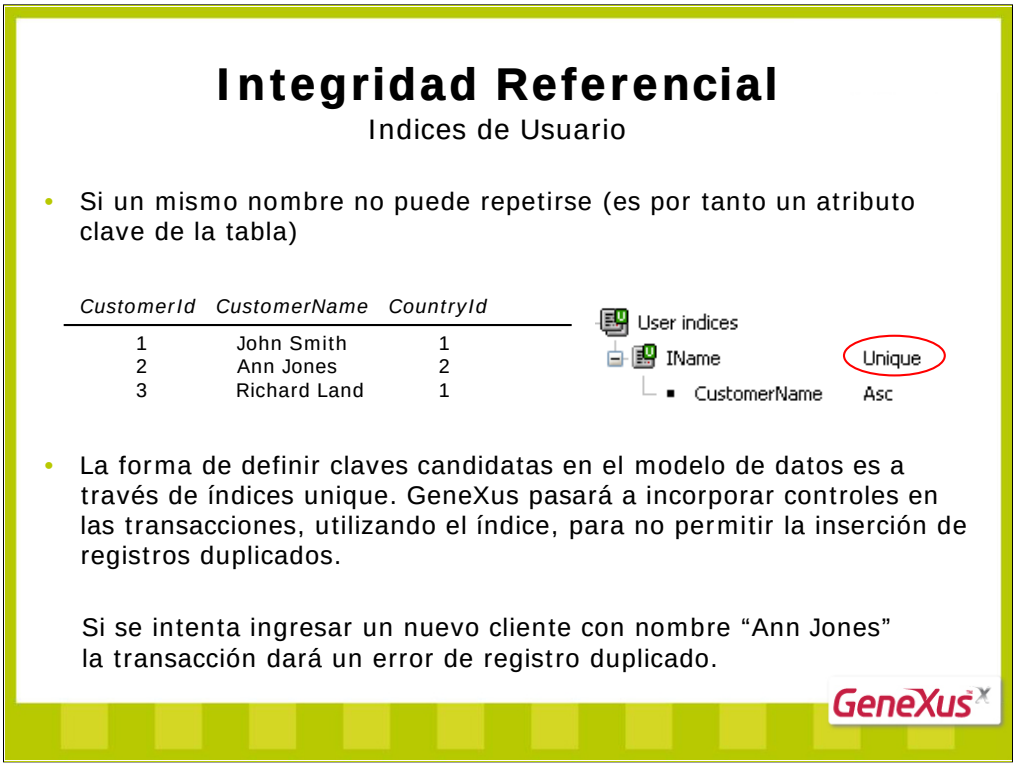
Este tipo de índices se define fundamentalmente para acceder a los datos ordenados por determinados atributos de forma eficiente.

A modo de ejemplo, suponiendo que el nombre de cliente no es clave en la tabla CUSTOMER (sus valores se pueden repetir) podremos definir un **índice de usuario duplicate** por el atributo *CustomerName*, siendo muy útil para realizar consultas y listados que se necesite salgan ordenados por nombre.

Un índice de usuario **unique** se utiliza para especificar que un conjunto de atributos es clave candidata en una tabla (diferente de la clave primaria).

Esta es la forma de representar claves candidatas en el modelo de datos. Con ello logramos que GeneXus incorpore automáticamente el **control de unicidad** correspondiente en las transacciones asociadas.

A modo de ejemplo, si el nombre de cliente no se puede repetir, la forma de representarlo y lograr que GeneXus lo controle automáticamente es definiendo en la tabla CUSTOMER un **índice de usuario unique** por el atributo *CustomerName*.



Índices unique (claves candidatas)

En una tabla de la base de datos pueden existir varios conjuntos de atributos cuyos valores sean únicos en la realidad. Se dice que cada uno de esos conjuntos es una clave de la tabla. Luego, el analista elige una de las claves como la clave primaria.

GeneXus identifica la clave primaria de la tabla de acuerdo a los atributos que fueron calificados por el analista con el símbolo de llave.

Supongamos que en la realidad además de poder identificar a un cliente por su código, también se lo puede identificar por su cédula de identidad. En este caso tanto el atributo *CustomerId* como el atributo *CustomerSSN* (donde se almacena la cédula de identidad) serían claves de la tabla CUSTOMER. Al indicar que *CustomerId* es el identificador de la transacción, GeneXus creará automáticamente un índice primario por dicho atributo y controlará la unicidad de los valores ingresados para el mismo.

¿Y qué sucederá con la cédula de identidad del cliente? Al ser este atributo clave, quisiéramos que GeneXus obrara de igual manera, no permitiendo que se ingrese un registro, si es que ya existe otro con el mismo valor de cédula de identidad (*CustomerSSN*). Para poder controlar esto de forma eficiente, GeneXus debería contar con un índice por cada atributo clave.

La forma de definir en GeneXus que un atributo o conjunto de atributos es clave alternativa o candidata y que por lo tanto se debe chequear su unicidad, es definiendo un **índice de usuario** compuesto por ese atributo o conjunto de atributos, y calificándolo de **“unique”** en lugar de “duplicate”, que es el valor por defecto de un índice de usuario.

A partir de allí, GeneXus incluirá en la lógica de la transacción ese control de unicidad utilizando ese índice definido por el usuario.

En resumen, las transacciones GeneXus realizan automáticamente los siguientes controles:

• Integridad referencial

• Unicidad de clave (tanto primaria como candidatas)

Índices temporales

Si se desea acceder a los datos ordenados por determinados atributos, pero no se desea crear un índice permanente para ello (por ejemplo, porque se trata de una consulta que se realiza con muy poca frecuencia), entonces, dependiendo de la plataforma, se creará un índice temporal.

Integridad Referencial

Manejo de nulos

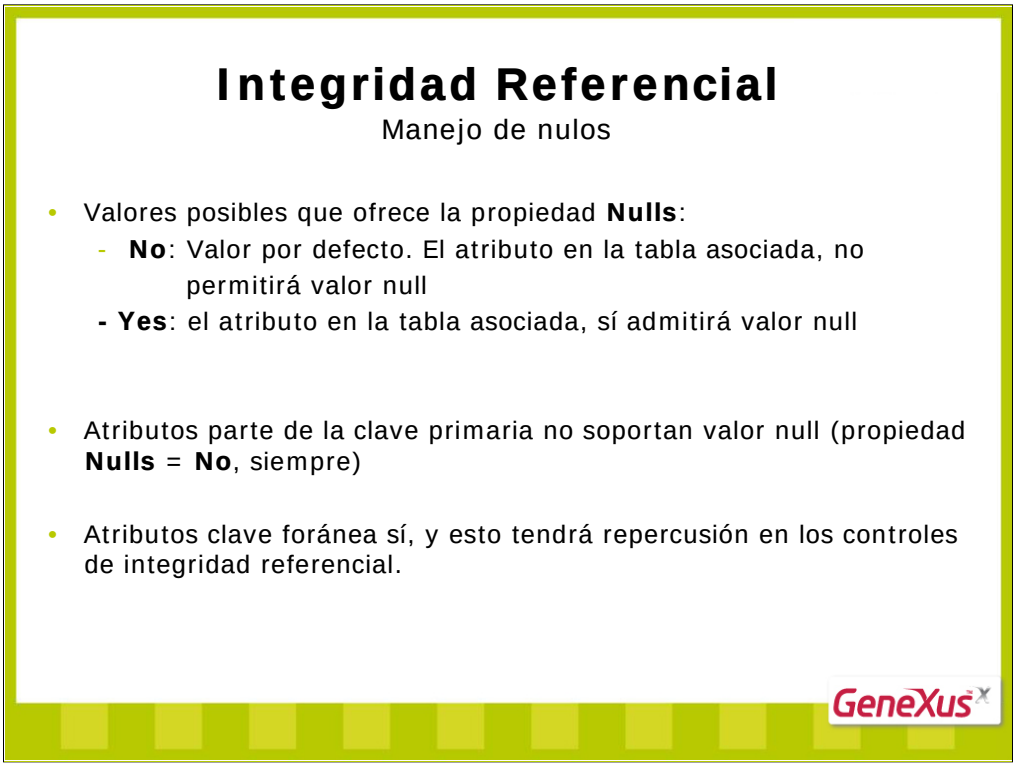
Name	Type	Description	Formula	Nullable
Customer	Customer	Customer		
CustomerId	Id	Customer Id		No
CustomerName	Name	Customer Name		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		
CustomerAddress	Address	Customer Address		No
CustomerGender	Gender	Customer Gender		No
CustomerStatus	Status	Customer Status		No

- Para cada atributo no inferido, ni identificador en la estructura de una transacción, es posible definir si admitirá nulos o no, en la tabla asociada.

La permisión o no del valor <NULL> en la base de datos es muy importante en el modelo relacional. Permitir el “valor” null para un atributo dado, significa que puede, bajo ciertas circunstancias, ser “ignorado” dado que es “valor no especificado”. Por otro lado, si un atributo no permite valor null, un valor válido siempre deberá asignarse a él.

La propiedad **Nulls** (presentada como columna en el editor de estructuras de transacciones), permite configurar para cada atributo si admite o no valor null en la tabla asociada. Los atributos para los cuales puede configurarse esto, es para aquellos almacenados en la(s) tabla(s) asociada(s) a la transacción (es decir, no inferidos) siempre y cuando no sean atributos primarios en dichas tablas (ya que por definición las claves primarias no soportan valor null).

Resumiendo: el objetivo de esta propiedad es definir qué valor se va a almacenar en la base de datos cuando no digitemos nada en el campo (el valor **<NULL>** o el valor **empty**).



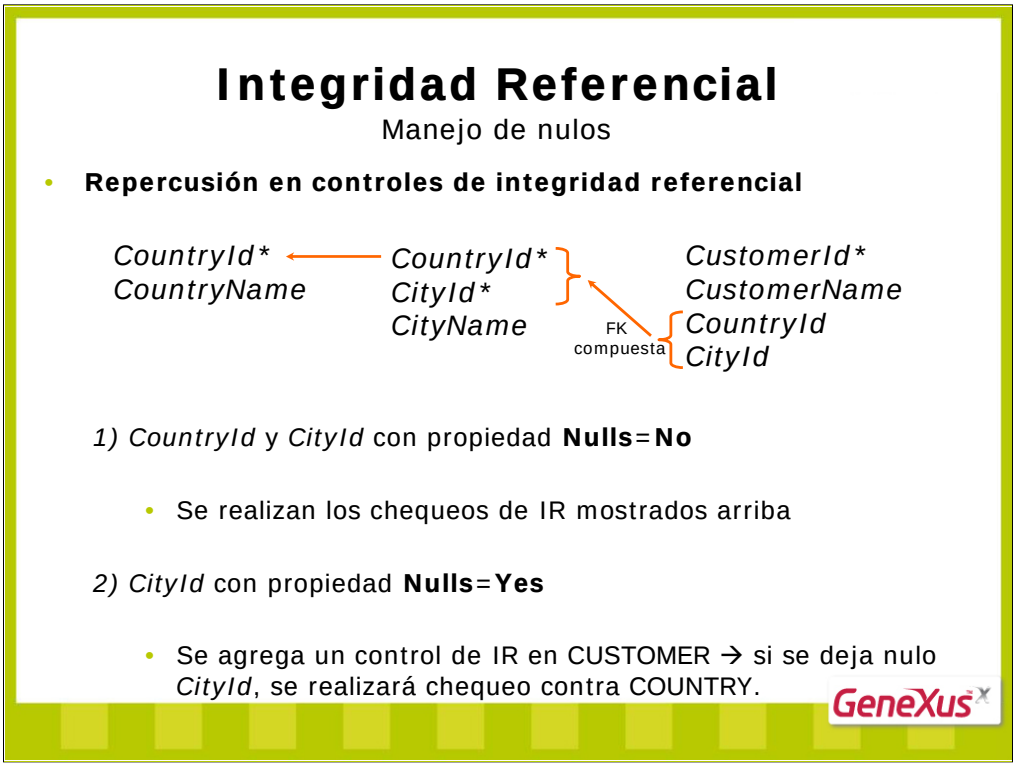
Los valores posibles de configurar para la propiedad **Nulls** son:

No: significa que el atributo no permitirá el valor null en la tabla asociada (valor por defecto)

Yes: significa que el atributo sí admitirá el valor null en la tabla asociada

La definición de nulls es utilizada por GeneXus al momento de crear / reorganizar las tablas de la base de datos, **ya que el soporte o no soporte de nulabilidad de los atributos en su tabla, se define a nivel de la base de datos**.

O sea que modificar el valor de la propiedad **Nulls** para un atributo implicará ejecutar una reorganización (para redefinir a nivel de la base de datos el soporte de nulabilidad del atributo en su tabla).



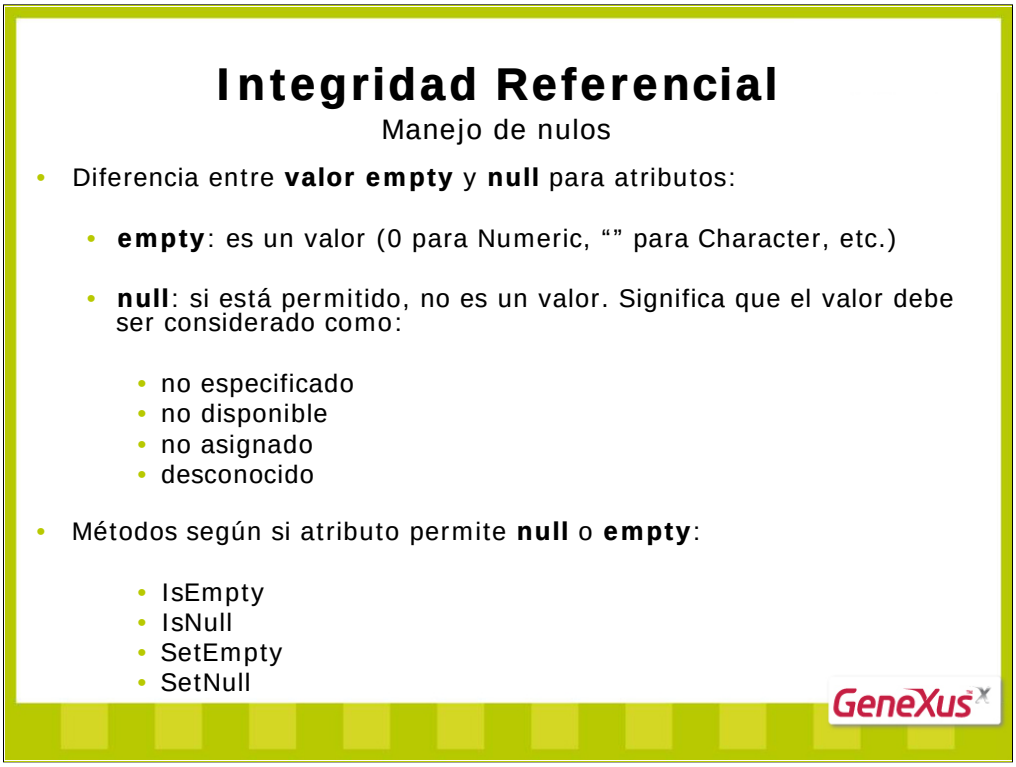
Repercusión en controles de integridad referencial

La definición de nulls para atributos que conforman una clave foránea le dice a GeneXus **cuán fuerte** es la referencia a la otra tabla.

Si ninguno de los atributos que componen una clave foránea permiten valores nulos (caso 1), se tratará de una referencia fuerte (también conocida como “not null reference”), ya que establece que la FK deberá siempre apuntar a un registro existente de la tabla referenciada.

Por el contrario, una clave foránea que tenga al menos un atributo que soporte nulos (caso 2), establece una referencia débil (también conocida como “null reference”), ya que si alguno de los atributos que conforman la clave foránea son nulls, entonces la referencia no será chequeada.

Cuando una clave foránea es compuesta y están permitidos los nulos para algunos de sus atributos, pueden aparecer nuevas referencias (no chequeadas en caso de tratarse de referencias fuertes) si los atributos que restan componen también una clave foránea. Un ejemplo es el que presentamos arriba, donde en caso de que el usuario deje nulo el valor de *CityId* para un cliente, se realizará el chequeo de IR contra COUNTRY, para asegurarse que el país ingresado sea correcto.



Métodos para trabajar con nulos y vacíos

IsEmpty, IsNull: devuelven true en caso de que el atributo contenga valor empty o null respectivamente.

SetEmpty, SetNull: configuran en el atributo el valor empty o null, respectivamente.

Ejemplos:

* error('You must specify a name') if *CustomerName*.IsEmpty();

* msg('Warning: You didn't specify a Country') if *ContryId*.IsNull();

Tabla base
y
Tabla extendida

Los criterios de normalización del diseño de la base de datos apuntan a minimizar la posibilidad de inconsistencia en los datos. Una base de datos diseñada de esta manera tiene una serie de ventajas importantes (tal es así que actualmente la normalización de datos es un estándar de diseño), pero se deben tener en cuenta también algunos inconvenientes.

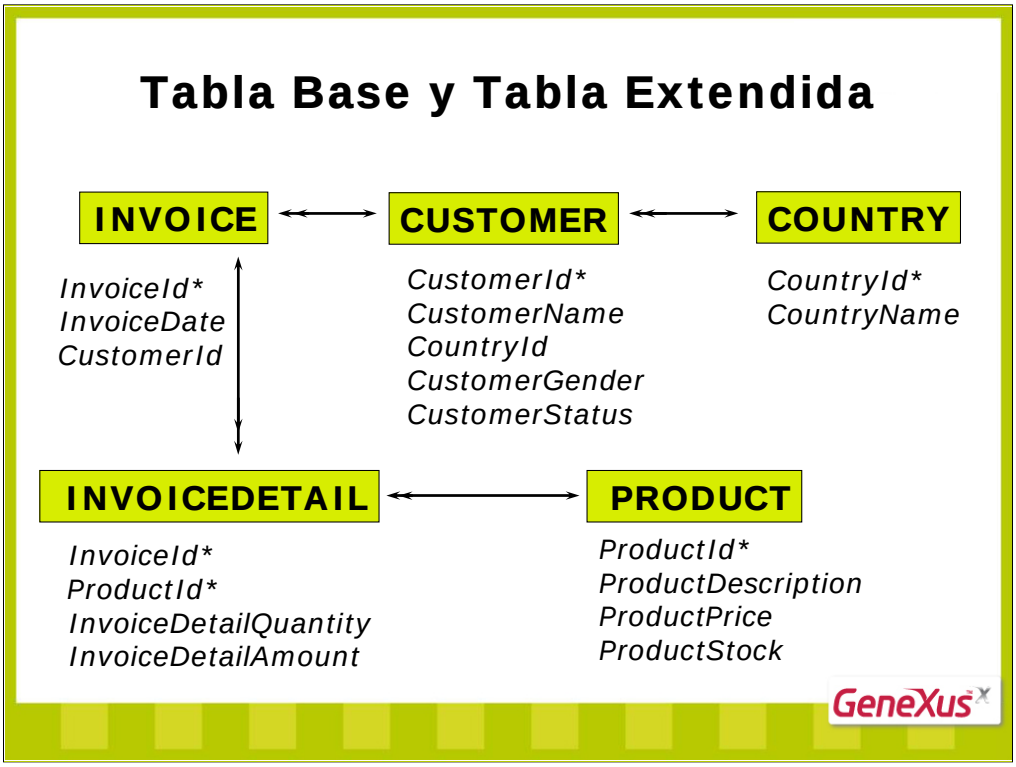
El inconveniente más notorio es que los datos se encuentran dispersos en muchas tablas, y por lo tanto cuando se quieren hacer consultas más o menos complejas a la base de datos, se debe consultar una cantidad importante de tablas.

Así, por ejemplo, si el siguiente diagrama representa nuestro modelo de datos:

para listar las facturas sería necesario consultar las tablas: INVOICE e INVOICEDETAIL (líneas de Facturas), CUSTOMER, COUNTRY y PRODUCT.

Para simplificar esta tarea GeneXus utiliza **el concepto de tabla extendida**.

Llamamos **tabla base** a cualquier tabla de la base de datos en la cual estemos posicionados en determinado momento; y dada cierta tabla base, su **tabla extendida** comprenderá a todos los atributos de la propia tabla base, más todos los atributos de las tablas que tengan información relacionada unívocamente con la **tabla base** (relación N-1 desde la tabla base, directa e indirectamente).



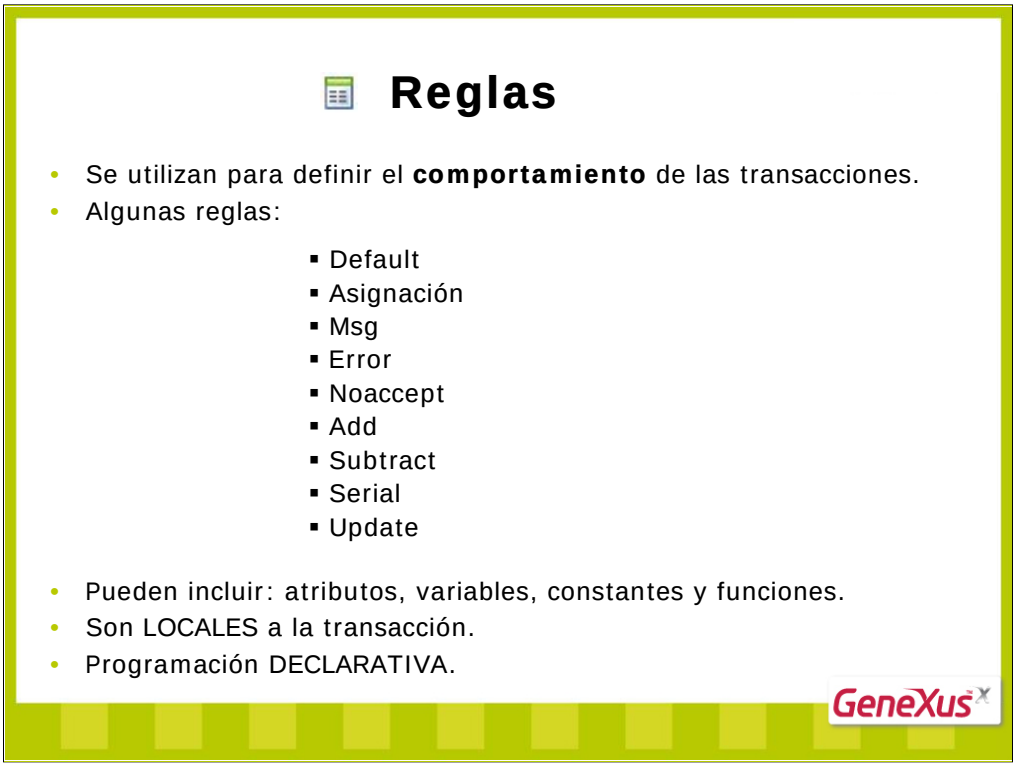
Utilizando el diagrama es fácil determinar cuál es la **tabla extendida** correspondiente a una **tabla base** cualquiera:

Partiendo de la tabla base, se deben seguir las relaciones N-1, (es decir, se deben seguir las flechas que tienen punta doble partiendo desde la tabla base, y punta simple en el otro extremo).

Todas las tablas a las cuales se pueda llegar siguiendo las flechas que representan relaciones N-1 desde la tabla base, formarán parte de su tabla extendida.

El siguiente cuadro muestra la tabla extendida correspondiente a cada una de las tablas de nuestro modelo de datos:

Base Table	Extended Table
COUNTRY	COUNTRY
CUSTOMER	CUSTOMER+COUNTRY
INVOICE	INVOICE+CUSTOMER+COUNTRY
INVOICEDETAIL	INVOICEDETAIL+INVOICE+CUSTOMER+COUNTRY+PRODUCT
PRODUCT	PRODUCT



Las reglas, en el objeto transacción, cumplen un rol muy importante ya que permiten **programar su comportamiento** (por ejemplo: asignar valores por defecto, definir controles sobre los datos, etc.).

Se escriben de forma **declarativa**, es decir que el orden en el que se escriben no significa que sea el orden en el que se ejecutarán.

Pueden involucrar a los **atributos definidos en la estructura de la transacción**[1], así como a variables definidas dentro del objeto, constantes y funciones.

Son solo válidas dentro de la transacción en la que están definidas, es decir, son **locales**.

---

[1] Todas las reglas de transacciones pueden involucrar atributos de las tablas base asociadas a la transacción; y la mayor parte de ellas puede también involucrar atributos de las tablas extendidas de dichas tablas base. Para poder referenciar un atributo en una regla, el mismo deberá estar incluido en la estructura de la transacción (ya sea que pertenezca a alguna de las tablas base asociadas a la transacción, o a sus tablas extendidas).

Algunas reglas válidas para transacciones son:

Default

OBJETIVO: Permite asignar un valor por defecto a un atributo o variable; el valor por defecto inicializará al atributo o variable si se está realizando una inserción por medio de la transacción (modo Insert), pero el usuario final podrá cambiarlo si ese valor no es el que desea.

SINTAXIS: **Default(** *att* | ***&****var*, *exp* **);**

DONDE:
• *att*: es un atributo perteneciente a alguna de las tablas base asociadas a la transacción.
• *var*: es el nombre de una variable.
• *ex*p: es una expresión que puede involucrar constantes, funciones, variables u otros atributos.

FUNCIONALIDAD: Esta regla asigna el valor de la expresión *exp* como valor por defecto del atributo *att* o variable *var,* cuando la transacción se ejecuta en modo Insert.

Esta regla no es válida para atributos que forman parte de la clave primaria de alguno de los niveles de la transacción, porque es disparada luego de que la clave es ingresada (ya que solo en ese momento el modo es conocido y esta regla se dispara solo en modo Insert).

EJEMPLOS:
Default(*InvoiceDate*, *&today*); /*Regla definida en la transacción “Invoice”*/

Cuando se está insertando una factura nueva, se sugiere como valor de *InvoiceDate* el valor contenido en la variable del sistema *today*.

Default(*InvoiceDate, Today()*); /*Regla definida en la transacción “Invoice”*/

Análoga a la anterior, solo que en lugar de utilizar la variable del sistema *today*, se utiliza la función Today que devuelve la fecha correspondiente al día.

Default(*InvoiceDetailAmount*, *ProductPrice***InvoiceDetailQuantity*); /* Regla definida en “Invoice” */

Cuando se está insertando una línea de factura, se sugiere que el atributo *InvoiceDetailAmount* (importe de la línea) tome el valor resultante de evaluar la expresión *ProductPrice***InvoiceDetailQuantity* (precio del producto por cantidad llevada del mismo).

Nota: El tipo de datos de la expresión debe coincidir con el tipo de datos del atributo o variable

Regla de asignación

OBJETIVO: Permite asignar a un atributo o a una variable, el valor de una expresión.

SINTAXIS: *att* | ***&****var* **=** *exp* [**if** *cond*] [**on** *evento/momento de disparo*]***;***

DONDE:
• *att:* es un atributo perteneciente a alguna de las tablas base asociadas a la transacción, o a sus tablas extendidas (debe estar declarado en la estructura).
• *var:* es el nombre de una variable.
• *exp:* es una expresión que puede involucrar constantes, funciones, variables u otros atributos, y debe ser del mismo tipo que *att* o *var.*
• *cond*: es una expresión booleana (puede contener los operadores lógicos and, or, not)
• *evento/momento de disparo:* es uno de los eventos predefinidos de GeneXus disponibles para reglas de transacciones, que permiten definir el momento específico de ejecución de una regla.

FUNCIONALIDAD: Una regla de este tipo permite asignar al atributo o variable de la izquierda, el valor resultante de evaluar la expresión. Como puede verse, la condición es opcional; de no ponerla, la asignación se realiza siempre.

La asignación a un atributo, implica su actualización en el registro que corresponda. Se pueden definir reglas de asignación a atributos de alguna de las tablas bases asociadas a la transacción, e incluso de sus tablas extendidas. Esto significa que pueden actualizarse atributos inferidos en una transacción (siendo necesario declararlos en la estructura).

EJEMPLO:

InvoiceDetailAmount **=** *ProductPrice*InvoiceDetailQuantity***;** /*Regla definida en la transacción "Invoice"*/

Si el importe de cada línea de una factura se calcula siempre multiplicando el precio del producto por la cantidad llevada del mismo, podemos utilizar esta regla de asignación para liberar al usuario de realizar el cálculo.

Error

OBJETIVO: Permite desplegar un mensaje de error si la condición se satisface. Sirve para definir los controles que deben cumplir los datos.

SINTAXIS: **Error('***msg***'** | **&***var* | *character expresion, msgId* **) if** *cond [**on** evento/momento de disparo]**;***

DONDE:
• *msg*: es un string con un mensaje de error a desplegar.
• *var*: es el nombre de una variable de tipo character, que contiene un string con un mensaje de error a desplegar.
• *character expression*: es una expresión cuyo tipo resultante es character y que será desplegada.
• *msgId:* es un string (sin espacios en blanco ni comillas) que será utilizado solo si la transacción es definida también como business component[1].
• *cond*: es una expresión booleana (que puede contener los operadores lógicos and, or, not)
• *evento/momento de disparo:* es uno de los eventos predefinidos de GeneXus disponibles para reglas de transacciones, que permiten definir el momento específico de ejecución de una regla.

FUNCIONALIDAD: Esta regla despliega el mensaje del parámetro *msg*, *var* o *character expresion*, si la condición *cond* que se evalúa resulta verdadera. El mensaje de error se despliega en una ventana popup cuando el ambiente de trabajo es Windows y en el control Error Viewer y/o un cuadro de texto cuando el ambiente de trabajo es Web, **deteniendo cualquier actualización a la base de datos.** Cuando la transacción se ejecuta como business component (estudiaremos este tema mas adelante en el curso), de dispararse el error, generará una entrada en el SDT messages, con identificador *msgId*.

EJEMPLOS:

Error('*No se permiten clientes sin nombre*') if *CustomerName*.isEmpty();
/*Regla definida en la transacción "Customer"*/

Se evalúa la condición *CustomerName*.isEmpty(), y si ésta se satisface, se despliega el mensaje 'No se permiten clientes sin nombre' en pantalla. No se permite continuar hasta que el usuario ingrese un nombre en el campo *CustomerName* o abandone la transacción (en cuyo caso no se hará en la base de datos actualización alguna).

Error('*No se permite eliminar facturas***'**) if *Delete*;
/* Regla definida en la transacción "Invoice" */

Se necesita prohibir la eliminación de facturas. Con esta regla, si el usuario intenta realizar una eliminación, la condición dará True y se disparará la regla, evitando la eliminación.

Msg

OBJETIVO: Permite desplegar un mensaje de advertencia si la condición se satisface.

SINTAXIS: **Msg('***msg***'** | **&***var* | *character expresion, msgId***) if** *cond [**on** evento/momento de disparo]**;***

DONDE:
msg, var, character expresion, msgId, cond, evento/momento de disparo: son los mismos que para la regla error.
Observar que la sintaxis es exactamente la misma.

FUNCIONALIDAD: Esta regla se utiliza para presentar mensajes de advertencia al usuario. Despliega el mensaje del primer parámetro, si la condición se satisface, análogamente a la regla Error; pero a diferencia de esta última, permite continuar con la ejecución si la condición sigue satisfaciéndose. Del mismo modo, si la transacción es business component[1], de dispararse la regla genera entrada en el SDT messages.

Los mensajes de advertencia se despliegan en el Error Viewer.

EJEMPLO:

Msg(**'***No se permiten clientes sin nombre*') if *CustomerName*.isEmpty();
/*Regla definida en la transacción "Customer"*/

Se evalúa la condición *CustomerName*.isEmpty(), y si ésta se satisface se despliega el mensaje **'***No se permiten clientes sin nombre***'**. A diferencia de lo que ocurre con la regla Error, aquí sí se permite continuar la ejecución, pues no se trata de un error sino de una advertencia.

Noaccept

OBJETIVO: Permite indicar que los atributos no aceptarán valores por parte del usuario (serán solo de salida).

SINTAXIS: **Noaccept(** *att1*[**,** *atti*]…**)** [**if** *cond*]**;**

DONDE:
atti: es un atributo perteneciente a alguna de las tablas bases asociadas a la transacción.
cond: es una expresión booleana (puede contener los operadores lógicos and, or, not).
evento/momento de disparo: es uno de los eventos predefinidos de GeneXus disponibles para reglas de transacciones, que permiten definir el momento específico de ejecución de una regla.

FUNCIONALIDAD: En una transacción, todos los atributos que pertenecen a las tablas base asociadas a la transacción, por defecto son aceptados. Si queremos que algunos atributos con estas características no sean aceptados, entonces contamos con la regla **Noaccept**.

EJEMPLO:
Noaccept(*IvoiceDate*) if Update; /* Regla definida en la transacción "Invoice" */

Si se está modificando una factura (modo Update), no se permite que se modifique su fecha.

Subtract

OBJETIVO: Sustrae el valor de un atributo al valor de otro atributo, si se satisface la condición especificada.

SINTAXIS: **subtract(***att1*, *att2***)** [**if** *cond*]**;**

DONDE:
att1, att2: son atributos pertenecientes a alguna de las tablas base asociadas a la transacción, o a sus tablas extendidas (y deben estar declarados en la estructura)
cond: es una expresión booleana (que puede contener los operadores lógicos and, or, not).

FUNCIONALIDAD: La sustracción se realiza teniendo en cuenta el modo en el que se esté trabajando en la transacción (Insert, Update o Delete).

En modo:
- Insert: se le sustrae al valor del atributo *att2,* el valor del atributo *att1*
- Delete: se le suma al valor de *att2,* el valor del atributo *att1*
- Update: se le sustrae al valor del atributo *att2,* la diferencia entre el valor nuevo y el viejo de *att1*

EJEMPLO:

En la transacción "Invoice", cada vez que se ingresa una línea con un producto que se está comprando, se debe disminuir el stock del mismo, según la cantidad llevada.

Esto podría hacerse en forma sencilla con la siguiente regla de asignación:

ProductStock = *ProductStock* – *InvoiceDetailQuantity*;

Si prestamos atención, sin embargo, vemos que esta regla no nos sirve, pues no está condicionada a un modo particular, razón por la cuál se disparará tanto cuando se está insertando una nueva línea en la factura, como cuando se está eliminando o modificando una ya existente. Y en estos últimos dos casos es incorrecto disparar la regla.

De hecho, cuando se esté eliminado una línea existente, debe realizarse la operación contraria, es decir, se debe “devolver” al stock lo que se había quitado cuando se insertó la línea.

Lo correcto entonces, teniendo en cuenta todos los casos posibles sería tener 3 reglas:

ProductStock = *ProductStock* – *InvoiceDetailQuantity* if Insert;
ProductStock = *ProductStock* + *InvoiceDetailQuantity* if Delete;
ProductStock = *ProductStock* + old(*InvoiceDetailQuantity*) – *InvoiceDetailQuantity* if Update;

Aquí estamos utilizando la función “old”, que devuelve el valor almacenado del atributo (es el valor antes de modificarlo). Para evitar tener que hacer todo esto, GeneXus provee la regla **subtract** que se encarga de hacer la asignación correcta de acuerdo al modo. Entonces podemos sustituir las 3 asignaciones anteriores, por:

subtract(*InvoiceDetailQuantity*, *ProductStock*);

Esta regla tiene la inteligencia para, dependiendo del modo, restar o sumar.

Add

OBJETIVO: Suma el valor de un atributo al valor de otro atributo, si se satisface la condición especificada

SINTAXIS: **add(** att_1, att_2**)** [**if** *cond*]**;**

DONDE:
att1, att2: son atributos pertenecientes a alguna de las tablas base asociadas a la transacción, o a sus tablas extendidas (y deben estar declarados en la estructura).
cond: es una expresión booleana.

FUNCIONALIDAD: La adición se realiza teniendo en cuenta el modo en el que se esté trabajando en la transacción (Insert, Update o Delete).

En modo:

- Insert: se le suma al valor del atributo *att2,* el valor del atributo *att1*
- Delete: se le sustrae al valor de *att2,* el valor del atributo *att1*
- Update: se le suma al valor del atributo *att2,* la diferencia entre el valor nuevo y el viejo de *att1*

EJEMPLO:
Definimos en la transacción “Customer”, un atributo de nombre *CustomerTotalPurchases,* para registrar el importe total de compras efectuadas por el mismo. El comportamiento que se desea es que cada vez que se cree una factura para un cliente dado, se le sume el total de la factura (*InvoiceAmount*) al total de compras efectuadas por el cliente (*CustomerTotalPurchases*).

Al igual que vimos en la regla subtract, no debemos olvidar que en la transacción “Invoice” podemos también eliminar y modificar facturas, y no solo crearlas; por lo tanto es importante tener en cuenta el modo de trabajo en la transacción (Insert, Update, Delete). GeneXus nos libera de tener que considerar nosotros a los modos, teniendo que escribir las siguientes tres reglas de asignación en la transacción “Invoice”:

CustomerTotalPurchases = CustomerTotalPurchases + InvoiceAmount if Insert;
CustomerTotalPurchases = CustomerTotalPurchases – InvoiceAmount if Delete;
CustomerTotalPurchases = CustomerTotalPurchases – old(InvoiceAmount) + InvoiceAmount if Update;

y en su lugar nos provee de la regla **add**, que se encarga de sumar o restar, dependiendo del modo.

Así es que si en la transacción “Customer” agregamos el atributo *CustomerTotalPurchases* (total de compras del cliente):

*CustomerId**
CustomerName
CustomerAddress
CustomerGender
...
CustomerTotalPurchases

y en la transacción “Invoice” inferimos al atributo *CustomerTotalPurchases,* ya que pertenece a la tabla extendida y podemos definir la regla:

add(*InvoiceAmount*, *CustomerTotalPurchases*);

y logramos el comportamiento deseado, que es:

• si se inserta una factura (Insert): se le suma al valor del atributo *CustomerTotalPurchases* el valor del atributo *InvoiceAmount*
• si se elimina una factura (Delete): se le sustrae al valor del atributo *CustomerTotalPurchases* el valor del atributo *InvoiceAmount*
• si se modifica una factura (Update): se le suma al valor del atributo *CustomerTotalPurchases,* la diferencia entre el valor nuevo y el viejo de *InvoiceAmount*

Serial

OBJETIVO: Permite numerar serialmente atributos numéricos.

SINTAXIS: **Serial(** att_1, att_2, *step***);**

DONDE:

• *att1*: es un atributo perteneciente a alguna de las tablas base asociadas a la transacción (es decir, no inferido), que desea autonumerarse (debiendo estar declarado en la estructura).
• *att2*: **Tiene que pertenecer a una tabla directamente superordinada a la del atributo** att_1.
• *step*: es el paso o incremento de la serialización.

FUNCIONALIDAD: El propósito de esta regla es asignar un número correlativo a att_1 cada vez que se inserta un registro en la tabla a la que pertenece att_1. Se toma el valor de att_2 (att_2 contiene el último número utilizado en la autonumeración), se le suma el valor del parámetro *step*, y el valor resultante se asigna tanto al atributo att_1 del nuevo registro, como al atributo att_2 para conservar el último número asignado.

Es decir, cuando se está insertando un registro por medio de una transacción en la cual se ha definido la regla: Serial(att_1, att_2, *step*);, se accede al att_2 (habrá un solo valor de este atributo relacionado, pues pertenece a una tabla directamente superordinada[1]), se le suma el valor *step,* y se asigna el valor obtenido tanto a att_1 del registro que va a ser insertado, como a att_2 perteneciente a una tabla directamente superordinada con respecto a la tabla que contiene a att_1.

Si se diseña a la transacción “Invoice” conteniendo un Número de Línea de Factura (atributo *InvoiceDetailId)* como identificador único del segundo nivel, la estructura de la transacción sería:

Invoice
{
*InvoiceId**
CustomerId
CustomerName
InvoiceDate
InvoiceLastLineId

Detail
{
InvoiceDetailId*
ProductId
ProductDescription
……
}
}

En este diseño el atributo *ProductId* no es identificador único del nivel, sino clave foránea únicamente.

Cada línea tiene un número de línea que la identifica en forma única, y es posible ingresar el mismo producto en distintas líneas.

Podría ser útil asignar por medio del sistema, números correlativos al campo ***InvoiceDetailId***, definiendo la regla:

serial(*InvoiceDetailId*, *InvoiceLastLineId*, 1);

El primer parámetro de la regla **serial** define cuál es el atributo a numerar automáticamente, en el segundo parámetro debe indicarse un atributo cuya función es guardar el último valor asignado hasta el momento, y por último el tercer parámetro es para indicar el incremento (en este caso se incrementa de uno en uno).

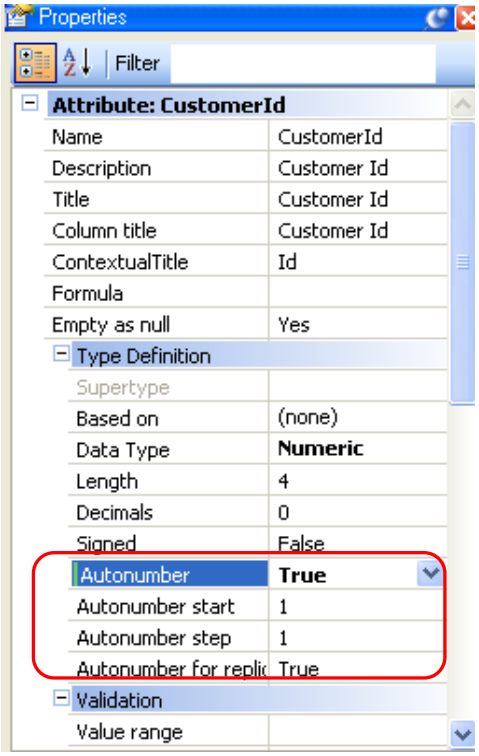
El segundo parámetro (en el ejemplo *InvoiceLastLineId*) debe pertenecer a una tabla directamente superordinada a la tabla que contiene el atributo que se desea numerar automáticamente (*InvoiceDetailId*). La regla serial lo requiere así. En el ejemplo, se puede observar que *InvoiceLastLineId* se encuentra en la tabla de clave *InvoiceId**, la cual es directamente superordinada respecto a la tabla que contiene el atributo a numerar *(InvoiceDetailId).*

Es decir, cada factura tendrá en el cabezal un atributo que almacenará el último número de línea asignado hasta el momento (*InvoiceLastLineId*). La regla serial está implementada de forma tal que necesita este atributo (para fijarse el último número utilizado, sumarle el incremento, y asignar el próximo número a la nueva línea).

Consideración:

La regla serial es útil a la hora de autonumerar líneas, no así cabezales (por ejemplo identificadores de facturas, de clientes, etc.). El motivo es el siguiente: para utilizar la regla serial, se requiere definir un atributo en una tabla directamente superordinada; esto resulta bien sencillo si se desean autonumerar líneas ya que alcanza con incluir este atributo en el nivel de la estructura inmediatamente superior al del atributo a autonumerar.

Sin embargo, contamos con una solución mucho más simple para autonumerar cabezales: cuando una tabla tiene una clave simple (es decir formada por un solo atributo) y el tipo de datos es numérico, puede numerarse automáticamente utilizando la funcionalidad que brindan los manejadores de base de datos para esto. La forma de indicarlo en GeneXus es configurando la propiedad Autonumber del atributo clave:



Si en la propiedad **Autonumber** de un atributo numérico clave, se selecciona el valor True, significa que se realizará la numeración automática del mismo. Se agregarán las siguientes propiedades en el diálogo:

Start: Mediante esta propiedad se configura a partir de qué número comienza la numeración automática.

Step: Mediante esta propiedad es posible configurar el incremento del campo (entre dos registros).

For replication: Esta propiedad es sólo válida para el motor de base de datos SQL Server; el valor Yes le indica a éste que no debe aplicar la propiedad en caso de que la tabla sea receptora de replicación (sino que debe mantener los números que le vienen por replicación).

Para profundizar en el manejo de esta propiedad, recomendamos acceder al Help de GeneXus.

Update

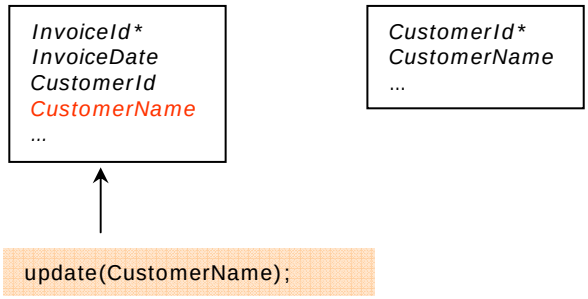
OBJETIVO: Posibilita actualizar en el form de una transacción (win/web) atributos de la tabla extendida (inferidos).

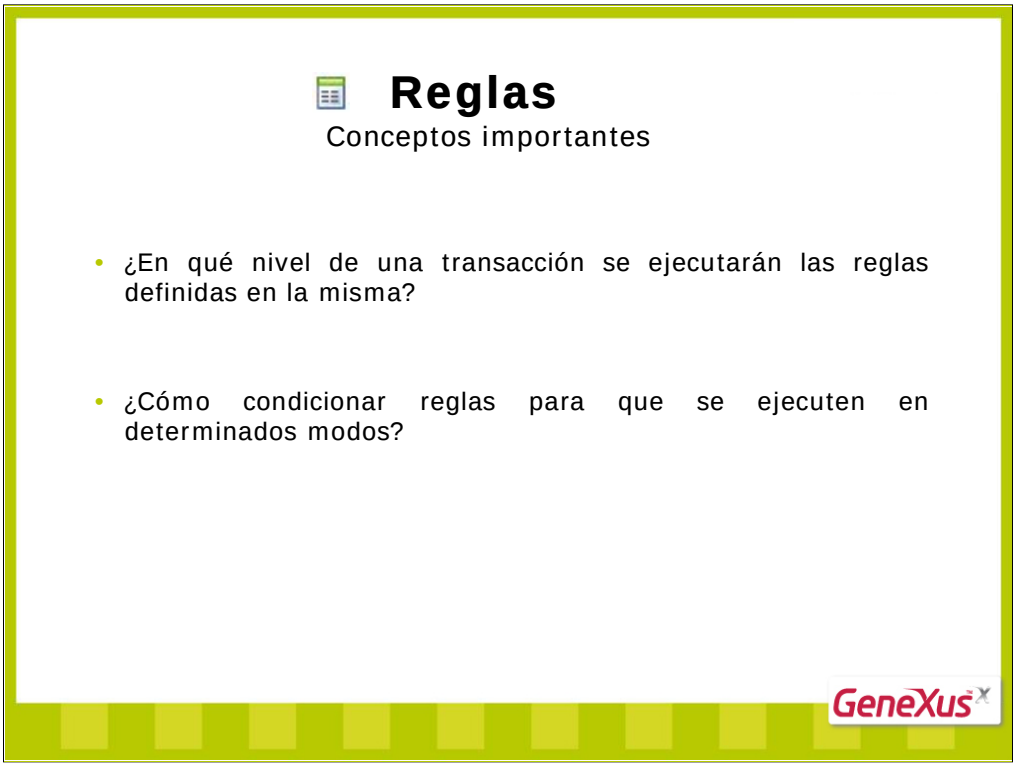
SINTAXIS: **Update(** *att1*[**,** *atti* …]**);**

DONDE:
atti: es un atributo perteneciente a la tabla extendida de alguna de las tablas bases asociadas a la transacción.

FUNCIONALIDAD: En una transacción, todos los atributos que pertenecen a las tablas base asociadas a la transacción, por defecto son aceptados y los que perteneciendo a la tabla extendida, no pertenecen a la base, son inferidos, y por tanto no aceptados.
Pero si queremos que algunos de estos atributos inferidos sean aceptados, para que el usuario pueda modificar desde el form su valor, entonces contamos con la regla **Update**.

EJEMPLO:





¿En qué nivel de una transacción se ejecutarán las reglas definidas en la misma?

La mayor parte de las veces no es necesario agregar explícitamente en la definición de las reglas el nivel de la transacción en el cual se desea que se disparen, ya que los atributos involucrados en las reglas le dan la pauta a GeneXus del nivel en el cual corresponde ejecutarlas.

Por ejemplo, si una regla referencia únicamente a atributos del primer nivel de la transacción en la cual se encuentra definida (ya sea en la propia regla o en la condición de disparo), GeneXus entenderá que la misma estará *asociada* al primer nivel de la transacción.

Análogamente, si una regla referencia solamente a atributos del segundo nivel de la transacción en la cual se encuentra definida (ya sea en la propia regla o en la condición de disparo), GeneXus entenderá que la misma estará *asociada* al segundo nivel de la transacción.

En el caso que una regla referencie atributos de varios niveles, GeneXus entenderá que la regla estará *asociada* al último de los niveles de los atributos involucrados, ya que será en el último nivel en el que contará con los valores de todos los atributos implicados.

A continuación presentamos ejemplos concretos:

1) Si se define la siguiente regla en la transacción “Invoice”:

Default(*InvoiceDate*, *&today*);

como el único atributo que se menciona en la regla es *InvoiceDate*, y es un atributo del primer nivel de la transacción, GeneXus determinará que se tratará de una regla *asociada* al primer nivel.

2) Si se define la siguiente regla en la transacción “Invoice”:

subtract(*InvoiceDetailQuantity*, *ProductStock*);

como los dos atributos que se mencionan en la misma se encuentran en el segundo nivel de la transacción, GeneXus determinará que se tratará de una regla *asociada* al segundo nivel.

3) Si se define la siguiente regla en la transacción “Invoice”:

*InvoiceDetailDiscount= InvoiceDetailAmount * CustomerDiscountPercentage/100***;**

siendo *InvoiceDetailDiscount* (descuento correspondiente a una línea) un atributo perteneciente al segundo nivel de la transacción “Invoice” y *CustomerDiscountPercentage* (porcentaje de descuento otorgado a un cliente) un atributo declarado en el primer nivel de la transacción “Invoice”, GeneXus determinará que se tratará de una regla *asociada* al segundo nivel de la transacción.

Cuando nos referimos a que una regla está *asociada* a determinado nivel, significa que la misma se ejecutará para cada instancia con la cual se trabaje a través de ese nivel (si se cumple la condición de disparo de la regla, claro está).

En el caso del primer ejemplo visto, la regla **Default**(*InvoiceDate*, *&today*) es una regla *asociada* al primer nivel de la transacción “Invoice”. Esto significa que se ejecutará para todo cabezal de factura que se inserte a través del primer nivel de la transacción “Invoice” (la regla **Default** tiene la particularidad de dispararse únicamente cuando el modo de ejecución es Insert).

En el caso del segundo ejemplo visto, la regla **Subtract**(*InvoiceDetailQuantity*, *ProductStock*) es una regla *asociada* al segundo nivel de la transacción “Invoice”. Esto significa que se ejecutará para toda línea de factura que se inserte, actualice o elimine a través del segundo nivel de la transacción “Invoice”.

En el caso del tercer ejemplo, la regla:
*InvoiceDetailDiscount = InvoiceDetailAmount * CustomerDiscountPercentage/100* es una regla *asociada* al segundo nivel de la transacción “Invoice”. De modo que esta regla se ejecutará para toda línea de factura que se inserte, actualice o elimine a través del segundo nivel de la transacción “Invoice”.

Concluyendo, tal como se desprende de todo lo explicado, *para cada factura con la cual se trabaje a través de la transacción “*Invoice*”*:

- para el cabezal: se ejecutarán las reglas *asociadas* al primer nivel
- para cada una de las líneas: se ejecutarán las reglas *asociadas* al segundo nivel

Es importante saber que como norma general GeneXus siempre **determina que una regla se dispare en el primer momento en que sea posible**, es decir, en aquel momento en el que cuente con todos los datos necesarios. Y solamente en algunos casos que así lo requieran, una misma regla se volverá a disparar más adelante.

¿Qué nivel de disparo por defecto se le asociará a una regla que no referencia atributos?

Cuando no hay atributos involucrados en una regla, el nivel *asociado* por defecto a la regla será el primero.

Por ejemplo, la siguiente regla definida en la transacción “Invoice”:

Error(‘No se permite eliminar facturas’) if Delete;

no tiene atributos involucrados, por lo tanto, el nivel *asociado* por defecto a la regla será el primero.

¿Cuándo hay que especificar explícitamente el nivel de disparo de una regla?

Existe una cláusula opcional de nombre **Level** que permite modificar el nivel por defecto de disparo de una regla, cambiándolo por un nivel posterior.

Es decir, si por ejemplo una regla se ejecuta por defecto para el primer nivel de una transacción y se desea que se ejecute para el segundo, se deberá agregar a la regla el componente **Level** seguido de un atributo o conjunto de atributos del segundo nivel. Esto hará que la regla se ejecute para cada una de las instancias correspondientes a las líneas, y no para la instancia correspondiente al cabezal como era el comportamiento por defecto.

Por ejemplo, si definimos la siguiente regla en la transacción “Invoice”:

Msg(‘La fecha de la factura es mayor a la fecha actual’) if *InvoiceDate > &Today*;

por defecto esta regla estará *asociada* al primer nivel de la transacción, ya que el único atributo referenciado en la regla se encuentra en el primer nivel de la transacción. Si por algún motivo deseamos que la regla se ejecute para cada una de las instancias correspondientes a las líneas en vez de ejecutarse para la instancia correspondiente al cabezal, tendremos que agregar en la definición de la regla, la cláusula **Level** seguida de uno o varios atributos del segundo nivel:

Msg(‘La fecha de la factura es mayor a la fecha actual’) if *InvoiceDate>&Today* **Level** *InvoiceDetailAmount*;

Agregar la cláusula **Level** a una regla solamente tiene sentido si a continuación de la misma se mencionan atributos que son de algún nivel posterior a los niveles de los atributos implicados en la definición de la regla en sí.

En el ejemplo que sigue, el hecho de haber agregado la cláusula **Level** a la regla no aporta más información de la ya aportada por el atributo implicado en la definición de la regla en sí:

Msg('La fecha de la factura es mayor a la fecha actual**') if** *InvoiceDate* > *&Today* **Level** *CustomerId*;

Es fácil comprender que el atributo *InvoiceDate* ya le da la pauta a GeneXus de que se trata de una regla *asociada* al primer nivel, así que lo especificado por la cláusula **Level** en el ejemplo no aporta más información y por lo tanto su agregado es innecesario.

Por último, en el ejemplo que sigue:

*InvoiceDetailDiscount= InvoiceDetailAmount * CustomerDiscountPercentage/100* **Level** *InvoiceDate*;

si bien se incluyó la cláusula **Level** en la definición de la regla, como el atributo que sigue a la cláusula es de un nivel superior al nivel de los atributos referenciados en la regla, la cláusula **Level** definida no aportará información útil en este caso tampoco. Es decir, no es posible que habiendo involucrados atributos de un segundo nivel en una regla, la misma se ejecute en el primer nivel, ya que en el primer nivel no se tiene la información del o de los niveles inferiores (además de que hay N instancias para el o los niveles inferiores). De modo que la regla seguirá estando *asociada* al segundo nivel de la transacción "Invoice", no habiendo aportado información útil la cláusula **Level** en este ejemplo.

Concluyendo, la cláusula **Level** solamente tiene sentido que sea agregada para modificar el nivel por defecto de disparo de una regla, a un nivel posterior.

¿Cómo condicionar reglas para que se ejecuten en determinados modos?

GeneXus provee las siguientes funciones booleanas para poder incluirlas en la **condición de disparo de las reglas** con el objetivo de limitarlas a que se ejecuten puntualmente en algunos de los modos posibles:

- Insert
- Update
- Delete

Ejemplos de uso (todas las reglas corresponderán a la transacción "Invoice")

1) Noaccept(*InvoiceDate*) if Update;

Si se está modificando una factura (modo Update), no se permite que se modifique su fecha. Si se definiera la regla sin la condición de disparo que hemos explicitado, el atributo *InvoiceDate* se deshabilitaría en todos los modos de ejecución.

2) Error('No se permite eliminar facturas') if Delete**;**

Si se intenta eliminar una factura, se disparará el error deteniendo la eliminación. Como no hay atributos involucrados en la regla, por defecto el nivel asociado a la regla será el primero.

3) Error('No se permite eliminar líneas de facturas') if Delete **Level** *InvoiceDetailQuantity***;**

Si se intenta eliminar una línea de una factura, se disparará el error deteniendo la eliminación. Observar que se ha explicitado en la regla la cláusula **Level** seguida de un atributo del segundo nivel de la transacción "Invoice", para indicar que se desea que la misma esté asociada al segundo nivel de la transacción.