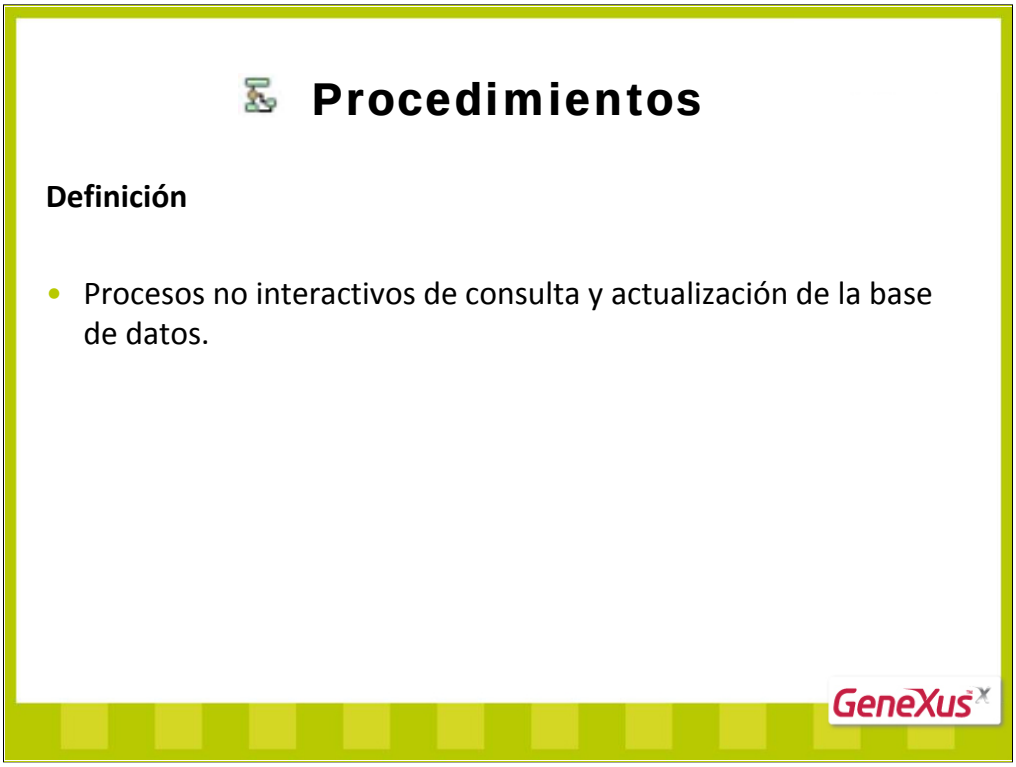
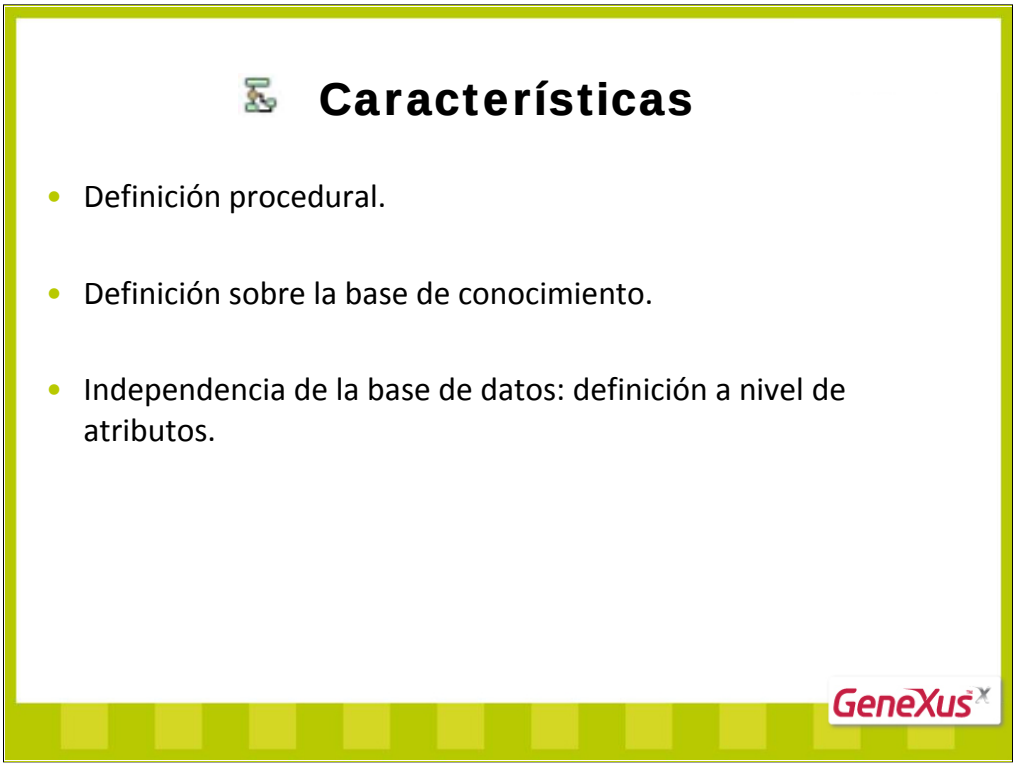
Objeto Procedimiento



Procedimientos:
Definen procesos no interactivos de consulta y actualización de la base de datos. Los procedimientos pueden generar un archivo formato PDF, mediante el cual es posible listar información por pantalla o impresora. Además, los procedimientos pueden actualizar la base de datos[1].

---

[1] Como veremos más adelante, existe un tipo de datos especial, que no es estrictamente un tipo de datos, sino algo un poco más complejo, el business component, por medio del cuál se podrán realizar actualizaciones a la base de datos en cualquier objeto GeneXus. Por tanto, utilizando variables de tipo de datos business component, podrán realizarse actualizaciones incluso en los objetos que por naturaleza no ofrecen esta posibilidad, como los web panels.



Definición procedural

A diferencia de las reglas de las transacciones donde las especificaciones se realizan en forma declarativa y GeneXus determina en el momento de generar el programa la secuencia de ejecución, en los los procedimientos las especificaciones se realizan en forma procedural. De esta forma, la secuencia de ejecución es determinada por el analista, utilizando para ello un lenguaje bastante simple que contiene comandos de control, de impresión, de acceso a la base de datos, etc.

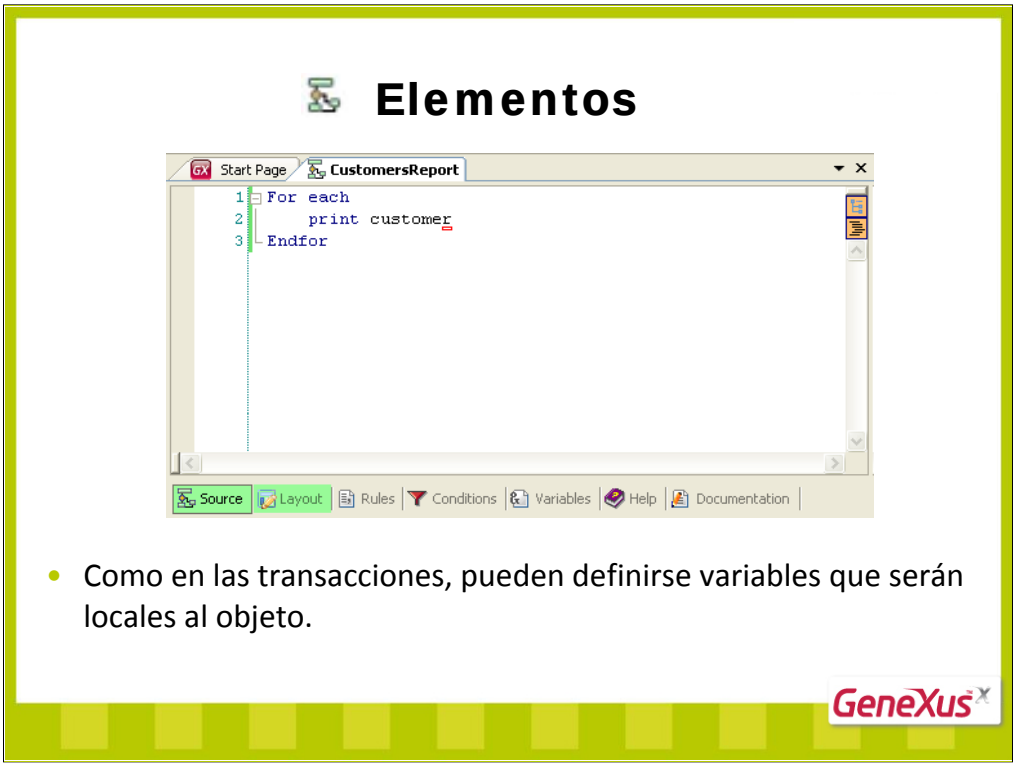
Definición sobre la base de conocimiento

La gran potencia del lenguaje de los procedimientos radica en que las definiciones se hacen sobre la base de conocimiento y no directamente sobre el modelo físico (tablas, índices, etc.). Esto nos permite utilizar automáticamente todo el conocimiento ya incorporado o generado por GeneXus a partir de las especificaciones realizadas.

Por ejemplo, si deseamos desplegar el resultado de una fórmula alcanza con nombrar al atributo fórmula en el lugar adecuado y GeneXus disparará su cálculo desplegando el resultado, sin necesidad de que el analista tenga que brindar ninguna otra información. La información de cómo se calcula un atributo fórmula está contenida en la base de conocimiento.
También podremos utilizar el concepto de tabla extendida, ya que GeneXus conoce las relaciones entre las tablas de la base de datos, por lo que el analista no necesita explicitar estas relaciones a la hora de recuperar datos.

Independencia de la base de datos: definición a nivel de atributos

La definición de los procedimientos se hace a nivel de atributos: no es necesario indicar explícitamente cuáles tablas serán recorridas ni mediante qué índices. Con solo mencionar los atributos a los que se desea acceder es suficiente para que GeneXus determine esta información. Esto es posible porque GeneXus tiene un completo conocimiento de la estructura de la base de datos.
De esta manera logramos una real independencia de la base de datos, ya que cualquier cambio en las tablas será manejado automáticamente por GeneXus y de esta forma, para actualizar los programas alcanzará, gran parte de las veces, con regenerar los objetos sin tener que modificar nada de lo programado en ellos.



Para cada procedimiento se puede definir:

• **Source**: Aquí se escribe el código correspondiente a la lógica del procedimiento. También pueden definirse al final del código subrutinas[1] que podrán ser invocadas desde el propio código mediante el comando adecuado.

• **Layout**: Así como las transacciones tienen una pantalla (form), los procedimientos tienen un "layout" de la salida. En esta sección se define la presentación del procedimiento: los datos que se quieren listar y el formato de la salida.

• **Rules**: permiten definir cierto comportamiento particular, como los parámetros que recibe el objeto, si el procedimiento implementará un listado pdf, etc.

• **Conditions**: Condiciones que deben cumplir los datos para ser recuperados (filtros).

• **Variables**: Variables locales al objeto (ídem que con el objeto transacción)

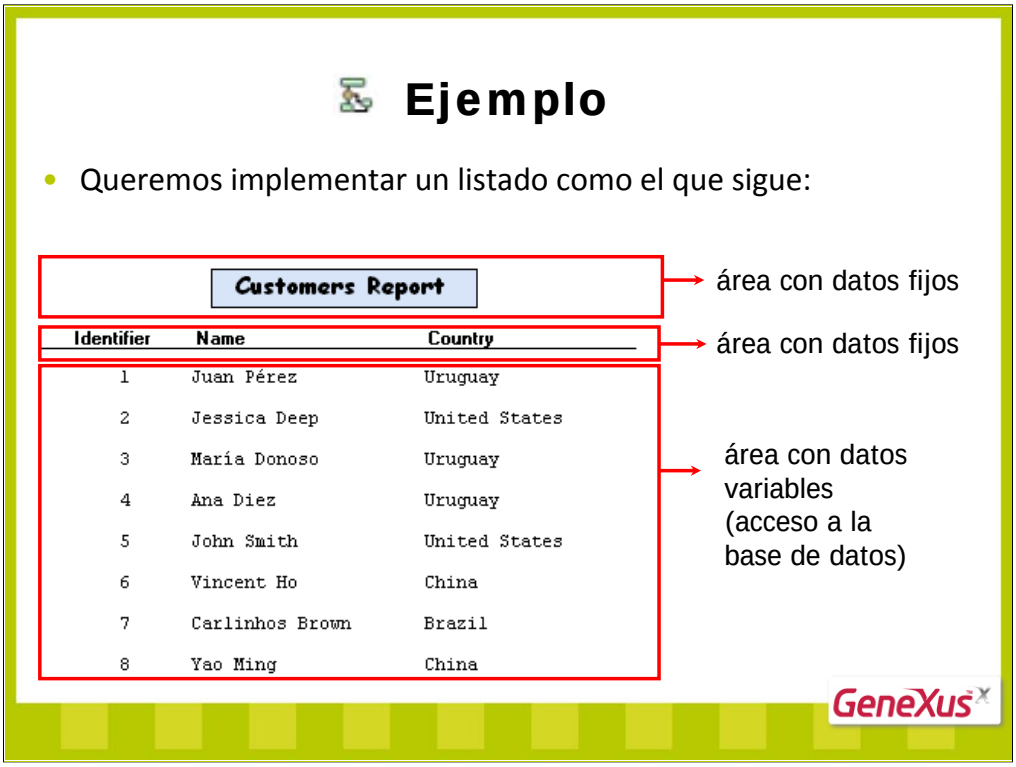
• **Help**: Permite la inclusión de texto de ayuda, para ser consultado por los usuarios en tiempo de ejecución, para el uso del procedimiento. Se puede redactar una ayuda para cada lenguaje.

• **Documentation**: Permite la inclusión de texto técnico, de tipo wiki, para ser utilizado como documentación del sistema.

• **Propiedades:** Definen aspectos generales del procedimiento, como su nombre, descripción, tipo de salida (impresora, archivo, pantalla). Recordemos que alcanza con presionar F4 para obtener las propiedades.

---

[1] No se tratarán en el presente curso.



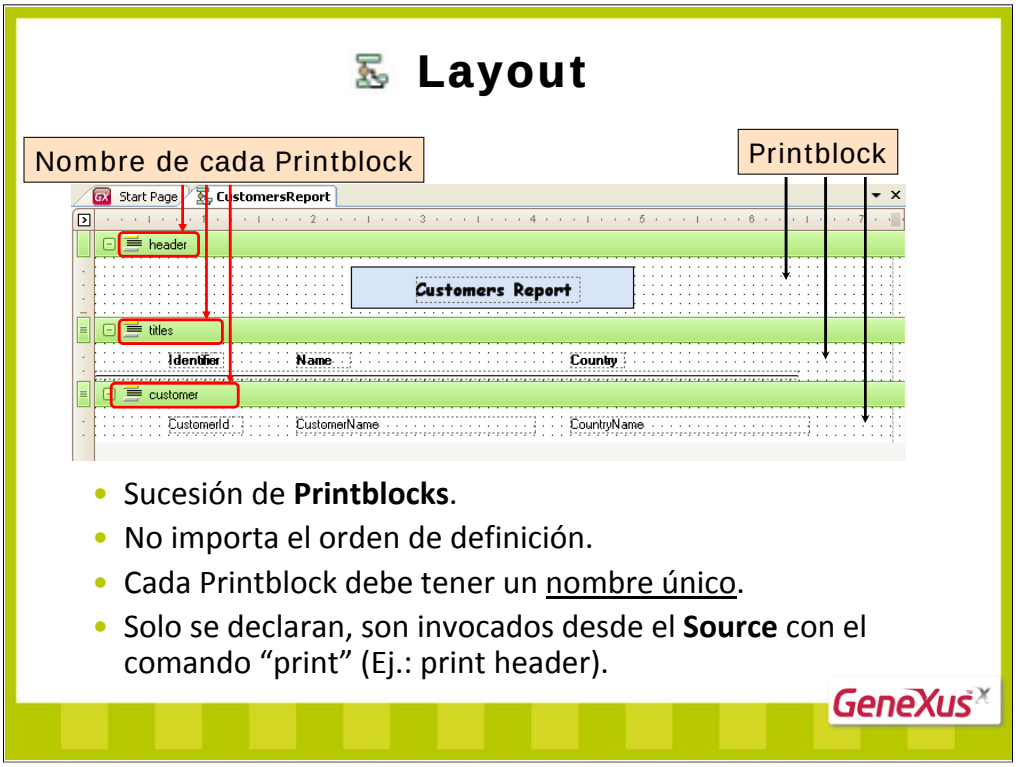
Por ejemplo, supongamos que queremos implementar un procedimiento para imprimir el identificador, nombre y país de todos nuestros clientes y queremos que el listado luzca como se muestra en la figura.

Para ello, debemos identificar en la salida del listado las distintas áreas que lo componen. A cada una de ellas la representaremos con un **Printblock**.

Los primeros dos Printblocks lucirán en GeneXus tal cuál las primeras dos áreas señaladas pues éstas contienen únicamente textos, líneas, recuadros. También podríamos haber fusionado estas dos áreas convirtiéndolas en una y utilizando por tanto un único Printblock.

El tercer Printblock será el correspondiente al área de datos variables de la figura anterior, que representa información que debe ser extraída de la base de datos.
Lo que queremos mostrar en este caso es el identificador y nombre de cada cliente, junto con el nombre del país al que pertenece. Esta información es la representada por los atributos *CustomerId*, *CustomerName* y *CountryName* de la base de conocimiento de la aplicación, por lo que el tercer Printblock contendrá los tres controles atributo *CustomerId*, *CustomerName* y *CountryName*.

Transformando las áreas en Printblocks, el Layout del procedimiento nos quedará como el que figura en la página siguiente.



El Layout de un procedimiento será una **sucesión de Printblocks** que no tienen por qué seguir el orden en el que se desea que aparezcan en la salida.

En el ejemplo anterior, el mismo procedimiento habría sido impreso si se hubieran especificado los Printblocks en el orden inverso (o en cualquier orden).

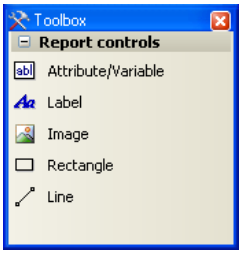
Aquí simplemente se declaran. El orden en el que se ejecutan queda determinado en la sección **Source** que es la que contiene la lógica del procedimiento. Desde allí serán invocados mediante un comando específico para tal fin (el comando print).

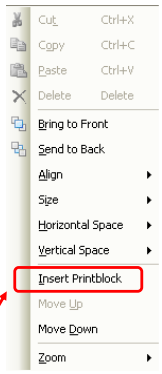
Por esta razón, cada Printblock deberá tener un nombre único para poder ser referenciado luego desde el Source.

En el ejemplo, para listar todos los clientes, el Printblock de nombre “customer” deberá ser invocado dentro de una estructura repetitiva en el Source. Esta estructura repetitiva es el comando **For each** que estudiaremos luego.

Layout: Printblock

- Para definir los Printblocks tenemos los siguientes controles disponibles:





- y para insertar un Printblock damos botón derecho sobre el layout y elegimos **Insert Printblock**.

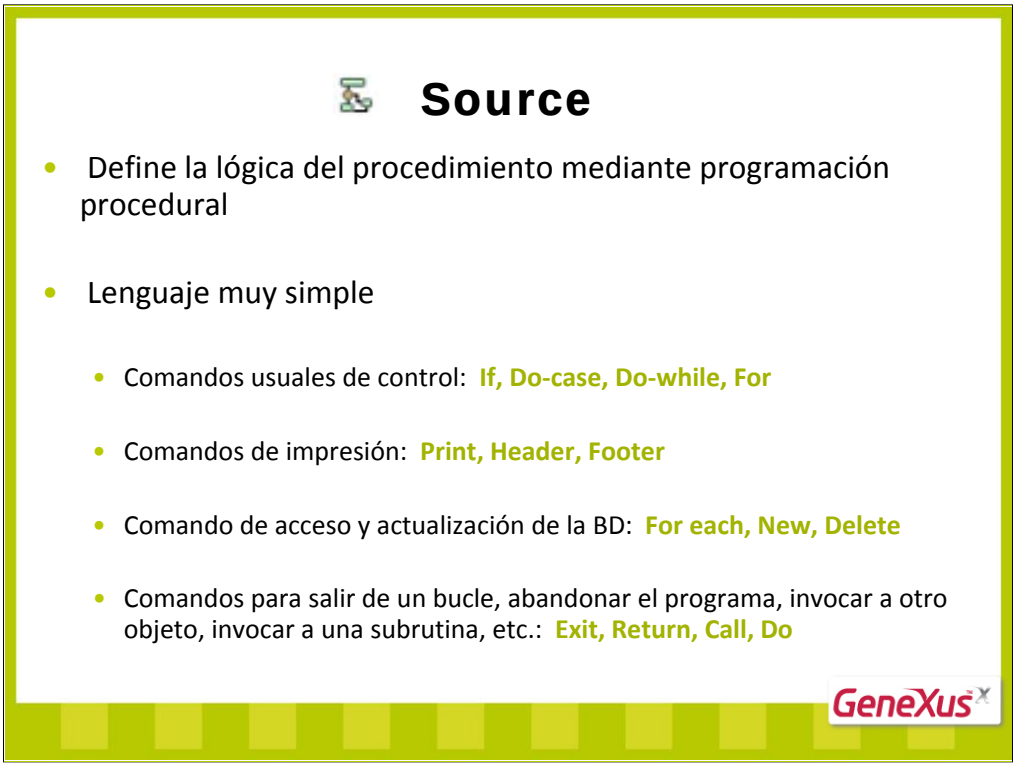
El Printblock es un tipo de control válido solamente en los procedimientos, que es insertado y eliminado del Layout por el analista, y que contendrá otros controles -atributos, textos, recuadros, líneas, etc.-, siendo estos últimos los que efectivamente especifican qué es lo que se quiere desplegar en la salida.

Para insertar los controles en el Form de una transacción contábamos con una toolbox. La misma toolbox se utiliza para insertar los controles en el Layout. De hecho esta toolbox está disponible para todos los objetos GeneXus que se creen, y en cada caso tendrá los controles disponibles según el tipo de objeto.

Para insertar un Printblock damos botón derecho en cualquier lugar del layout y seleccionamos Insert Printblock.

Como todo control, el Printblock tiene propiedades que pueden ser configuradas por el usuario. En particular, tiene la propiedad “Name”, muy importante dado que es el identificador del Printblock. Con este identificador es que el Printblock puede ser invocado desde el **Source** para ser impreso.

Para acceder a la propiedades de un Printblock, lo seleccionamos y presionamos F4 o View/Properties.



En esta sección se define la lógica del procedimiento.

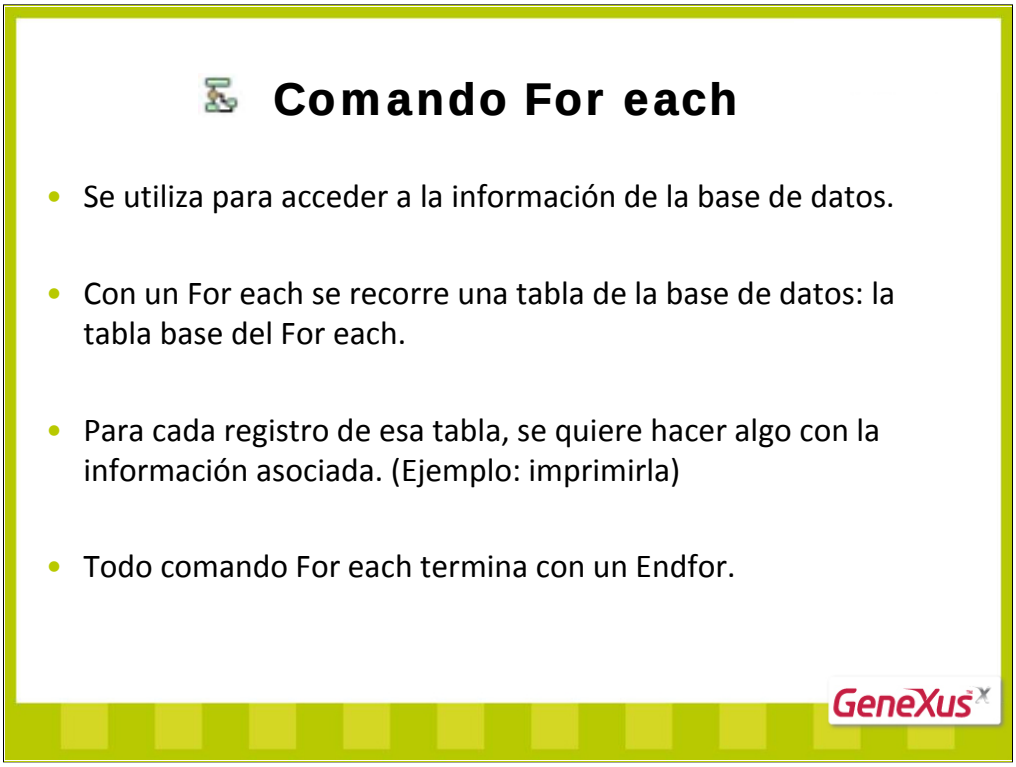
El lenguaje que se utiliza para programar el código fuente de los procedimientos es muy simple, y consta de algunos comandos que iremos viendo.

El estilo de programación es procedural –imperativo– por lo que el Source será una **sucesión de comandos** para los que el orden es fundamental: el orden en el que estén especificados corresponderá, salvo excepciones, al orden en el que serán ejecutados.

Existen, como en todo lenguaje imperativo, comandos de control para la ejecución condicional (if, do case), la repetitiva (do while, for), para invocar a otro objeto (call), para cortar las iteraciones dentro de un bucle (exit) o abandonar el programa (return), así como también comandos específicos de este lenguaje: para imprimir un Printblock del Layout (print), para acceder a la base de datos (For each), para insertar nuevos registros en una tabla (new), para invocar a una subrutina (do), etc.

Al final de la sucesión de comandos que constituye el código general o principal del procedimiento, pueden definirse subrutinas que podrán ser invocadas (mediante el comando do) desde el código general. No pueden ser invocadas desde otro objeto (son locales).

Por su importancia, empezaremos estudiando en profundidad el comando de acceso a la base de datos, fundamental a la hora de recuperar la información almacenada. Luego se tratarán brevemente los comandos de control, que son comunes a todos los lenguajes de programación imperativa, los comandos de asignación y los de impresión.



La definición del acceso a la base de datos para recuperación de información se realiza con un único comando: el comando **For each**[1].

Usando el For each se define la información a la que se va a acceder. La forma de hacerlo se basa en nombrar los atributos a utilizar.

Así, con este comando se definen qué atributos se necesitan y en qué orden se van a recuperar, y GeneXus se encarga de encontrar cómo hacerlo. No se especifica de qué tablas se deben obtener, ni qué índices se deben utilizar para acceder a esas tablas: eso lo inferirá GeneXus. Evidentemente esto no siempre es posible, y en tales casos GeneXus da una serie de mensajes de error indicando por qué no se pueden relacionar los atributos involucrados.

La razón por la cuál no se hace referencia al modelo físico de datos es porque de esta manera la especificación del procedimiento es del más alto nivel posible, de tal forma que ante cambios en la estructura de la base de datos la especificación del mismo se mantenga válida la mayor parte de las veces.

Cuando aparece un For each se está indicando que se quiere recuperar información de la base de datos. Concretamente GeneXus sabe que con un For each se quiere **recorrer (o navegar) una tabla**. Para **cada** registro de esa tabla, se quiere hacer algo con la información asociada (ej: imprimirla).

Por lo tanto, todo comando For each tendrá una tabla física asociada: la tabla que será recorrida o navegada. A esta tabla la llamaremos **tabla base del For each**.

---

[1] Cuando estudiemos los business components veremos que utilizando su método Load también se consigue consultar la base de datos.

Comando For each

- Ejemplo: Listado de clientes

CUSTOMER ⟷ COUNTRY

- Layout:

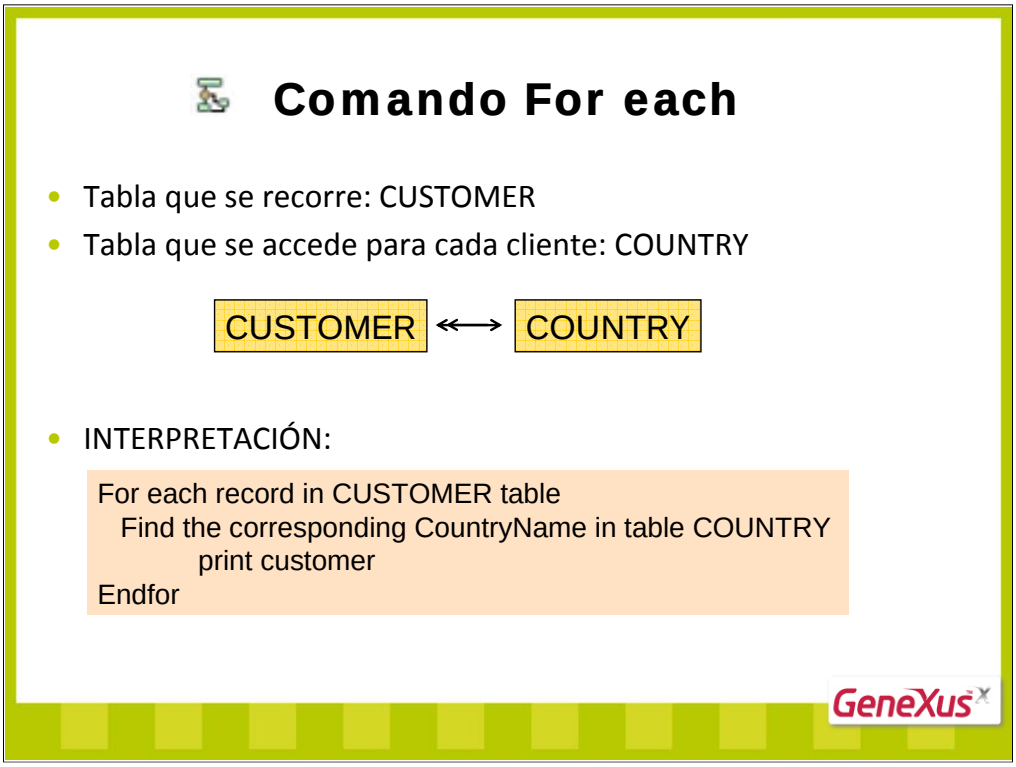
- Source:

```
For each
    print customer
Endfor
```

GeneXus

Intuitivamente resulta claro que con este comando estamos queriendo listar identificador, nombre y país de cada uno de los clientes de la base de datos. Es decir, queremos que se recorra la tabla CUSTOMER, y para cada cliente se recupere de la tabla COUNTRY el nombre del país al que pertenece, imprimiendo esta información, junto con el identificador y nombre del cliente. (Observar que la tabla COUNTRY pertenece a la extendida de CUSTOMER)

¿Cómo infiere esto GeneXus si todo lo que hicimos en el For each del ejemplo fue nombrar los atributos que nos interesaba mostrar?

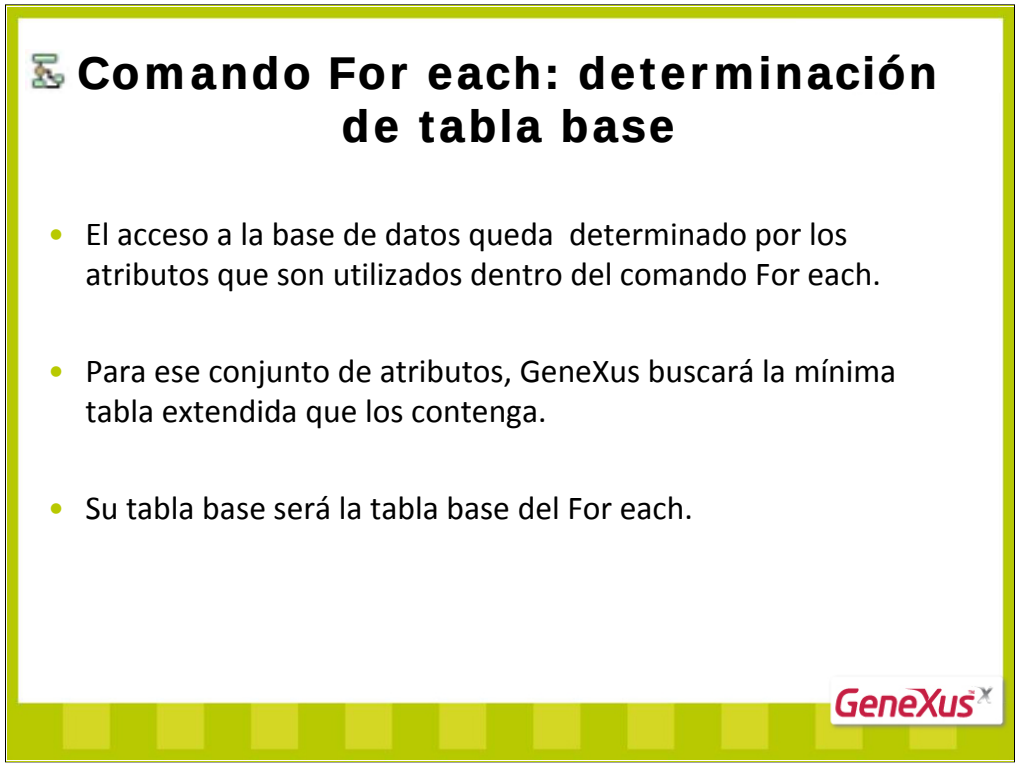


Dentro de todo For each **se navega** -recorre o itera- **la tabla base**, pero puede **accederse** a las tablas que constituyen **su tabla extendida** para recuperar información, que por pertenecer a la extendida estará unívocamente relacionada con cada registro de la tabla base con el que se esté trabajando en cada iteración (el concepto de tabla extendida es muy importante en este comando y sugerimos repasar su definición).

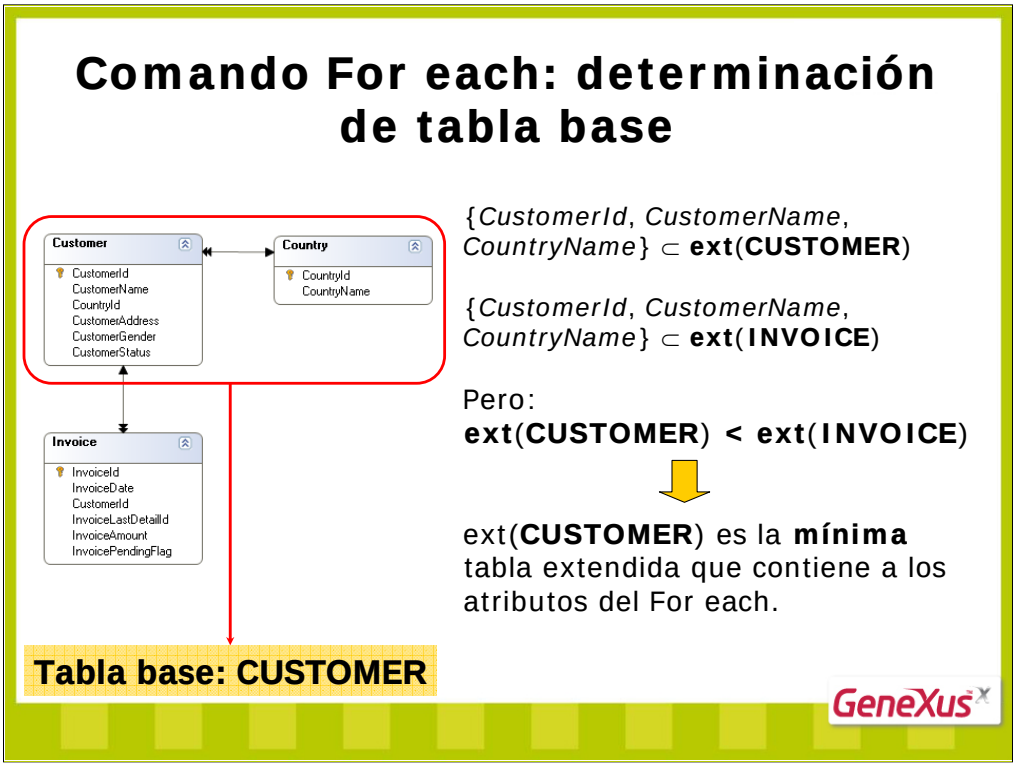
Es por ello que en el For each del ejemplo, la tabla base será CUSTOMER, y además se accederá "para cada" cliente, no solo a los datos de su registro, sino a los **del registro asociado** en la tabla COUNTRY (que está en la extendida de CUSTOMER). Decimos entonces que **se recorre** CUSTOMER y **se accede** además a COUNTRY para buscar el resto de la información requerida.

Como podemos ver claramente en el ejemplo presentado, no le damos explícitamente a GeneXus esta información. No es necesario, ya que GeneXus conoce las relaciones entre las tablas, y en base a los atributos mencionados **dentro** del For each, puede encontrar sin necesidad de más información una **tabla extendida que los contenga**.

La **tabla base** de esa **extendida** es la que elige como **tabla base del For each**.



A la tabla base correspondiente a esa tabla extendida la llamaremos **tabla base del For each** y será recorrida en forma secuencial, ejecutando para cada registro lo que se indique en los comandos internos al For each.



Para el ejemplo presentado en el que se quieren listar de cada uno de los clientes su identificador, nombre y nombre de país, si observamos los atributos utilizados dentro del For each, vemos que ellos son los contenidos en el Printblock de nombre "customer": *CustomerId*, *CustomerName* y *CountryName*.

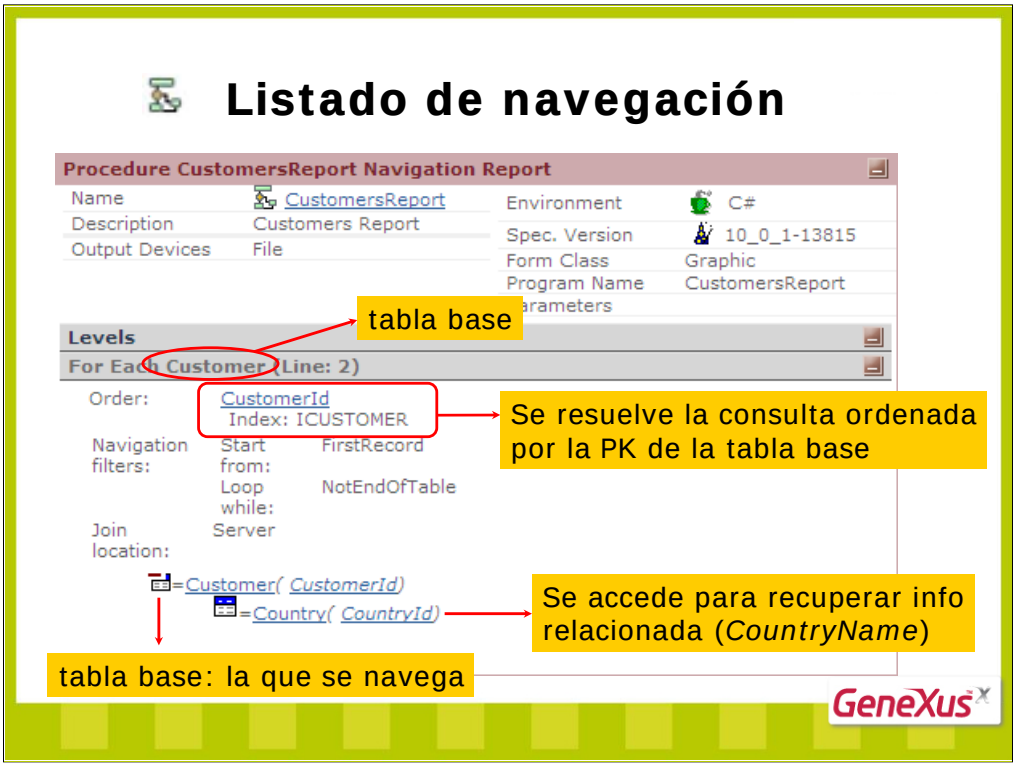
¿En qué tablas están estos atributos?

- *CustomerId* está en 2 tablas:
  - CUSTOMER como clave primaria (PK).
  - INVOICE como clave foránea (FK).
- *CustomerName* está solo en CUSTOMER (es un atributo secundario).
- *CountryName* está solo en COUNTRY (es un atributo secundario).

GeneXus conoce las relaciones entre las tablas. Podemos ver el diagrama correspondiente a las tablas en las cuales aparecen los atributos del For each (Tools/Diagrams).

Aquí puede verse claramente el por qué del requerimiento de que la tabla extendida sea la mínima (entendiendo por mínima aquella que involucra menor cantidad de tablas). La tabla extendida de INVOICE también contiene a todos los atributos del For each, pero no es mínima, pues la de CUSTOMER también los contiene.

Por lo tanto, se va a recorrer secuencialmente la tabla CUSTOMER, y para cada registro de esa tabla, se va a acceder a la tabla COUNTRY, para recuperar **el** registro de la misma que cumpla: COUNTRY.*CountryId* = CUSTOMER.*CountryId* y para el mismo se va a recuperar el valor del atributo *CountryName*, para poder imprimirlo, junto con el código y nombre del cliente.



Listado de navegación

GeneXus ofrece para todos sus objetos un listado conocido como **listado de navegación**, que es el resultado de la especificación del objeto. Este listado es muy útil para los procedimientos, ya que indica cuáles son las tablas que se están accediendo en cada For each del Source, si existe un índice para recuperar los datos de la tabla base, y en caso de que así sea cuál es ese índice (su nombre), si se aplican filtros sobre los datos o se van a listar todos, etc.

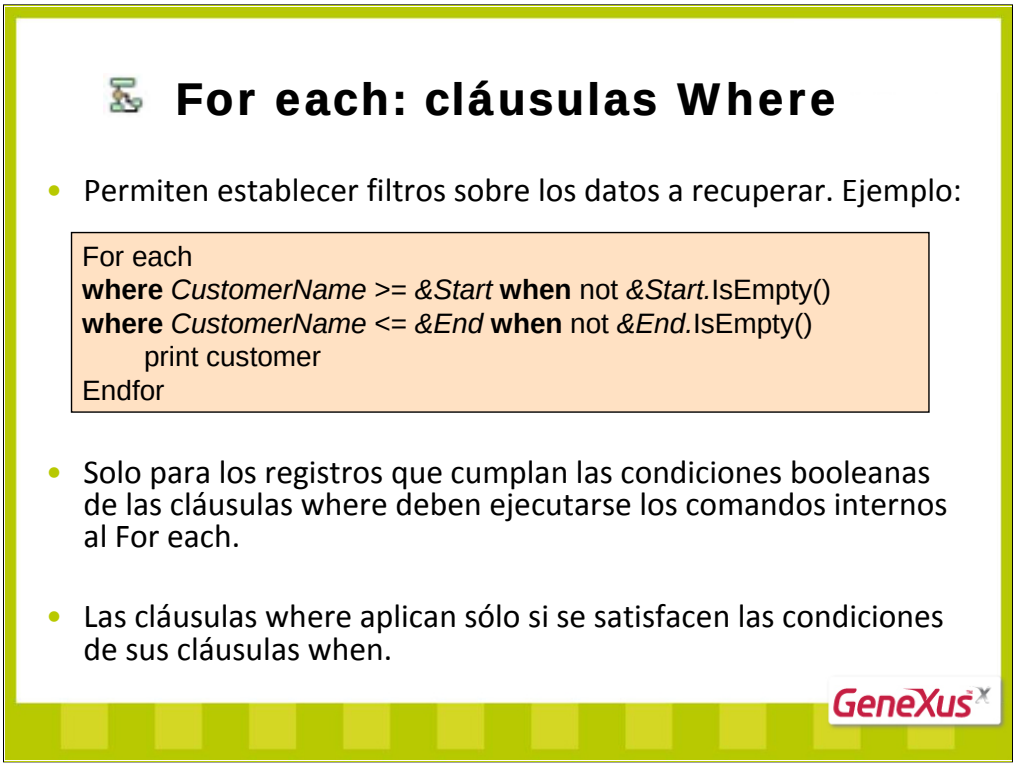
De esta manera, el analista no tiene que ejecutar el objeto para verificar que la lógica sea la esperada. Con estudiar el listado de navegación ya tiene la información necesaria para saber si se está recorriendo la tabla esperada, si se están aplicando correctamente los filtros deseados, etc.

Como puede verse en el listado correspondiente al procedimiento del ejemplo, muestra para el comando For each del Source, cuál es su **tabla base**, por qué **orden** se va a resolver esa consulta (será el orden en el que se imprimirán los resultados), si existe un índice que satisfaga ese orden cuál es su nombre, y además aparecen dos elementos más: los filtros de navegación y el diagrama de tablas.

Los **filtros de la navegación** indican qué **rango** de la tabla base se va a a recorrer. En el ejemplo se va a recorrer **toda** la tabla base del For each: empezando por el primer registro de CUSTOMER, y hasta que se alcance el fin de tabla (utilizando el índice ICUSTOMER).

También se muestra en un pequeño **diagrama de tablas**, la tabla base del For each con su clave primaria, e indentadas todas las tablas de la extendida que deban accederse para recuperar información asociada al registro de la tabla base con el que se esté trabajando en cada iteración del For each. En este caso se muestra solamente la tabla COUNTRY.

En el comando For each del ejemplo no aparece explícitamente ninguna información respecto al orden en el que queremos que se imprima la información. En este caso GeneXus elige como orden la **clave primaria de la tabla base del For each**. Es por esta razón que para el For each del ejemplo GeneXus determinó que el orden será el correspondiente al atributo *CustomerId*, clave primaria de la tabla CUSTOMER.



Para restringir los datos que se quieren listar en un For each se utilizan las **cláusulas where** del comando.

Si en el listado de clientes no queremos listar **todos** los clientes, sino solo aquellos cuyo nombre esté dentro de un rango ingresado por el usuario, entonces debemos agregar al For each que habíamos visto una **clálusula where**, para especificar los filtros deseados sobre los datos:

```
For each
where (CustomerName >= &Start) and (CustomerName <= &End)
    print customer
Endfor
```

donde las variables *&Start* y *&End* deben definirse en el procedimiento con el mismo tipo de datos que *CustomerName*, y cargarse con valores fijos ó recibidos por parámetro[1].

Con la **cláusula where** definida le estamos diciendo a GeneXus que no queremos quedarnos con todos los registros de la tabla base, sino solo con aquellos para los que se satisfaga la condición booleana de la cláusula.

En el ejemplo escribimos una sola cláusula where con una condición compuesta, pero podríamos haber programado lo mismo con dos cláusulas where, como se muestra arriba en la transparencia.

Es decir, cuando aparecen varios "where" la condición de filtro que se va a aplicar sobre los datos es la conjunción booleana de todas las condiciones de los "where" que aparezcan.

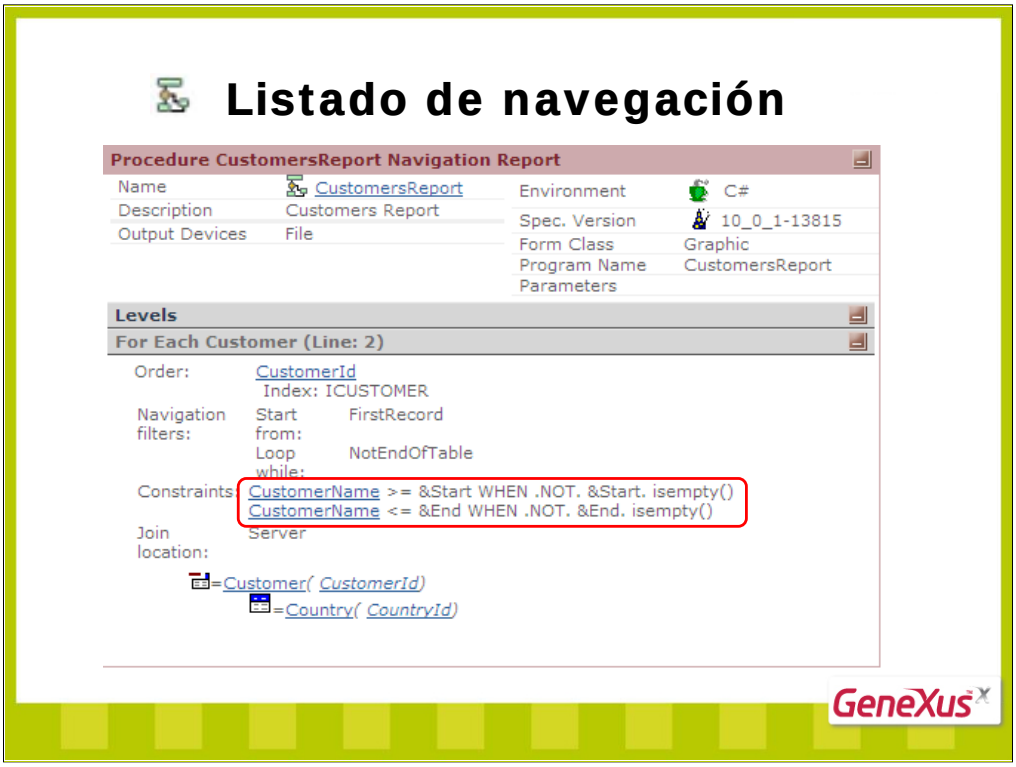
Observemos que en la transparencia las cláusulas where están a su vez condicionadas con **claúsulas when**. Esto se lee de la siguiente forma: se aplicará el filtro establecido por el where solo cuando se satisfaga la condición del when.

En el ejemplo, solo se aplicará el primer filtro: "*CustomerName >= &Start*" si la variable *&Start* no está vacía. Si está vacía, este filtro no se aplicará. Análogo es el caso de la segunda cláusula when. Observar que si *&Start* y *&End* están vacíos, no aplicará ninguna de las cláusulas where, y por tanto se listarán todos los clientes (como si las cláusulas where no hubiesen sido escritas).

Cada condición booleana de un "where" puede estar compuesta de varias expresiones booleanas concatenadas con los operadores lógicos and, or y not.

---

[1] A través de un objeto que los pide al usuario, por ejemplo un Web Panel.



Listado de navegación

Aparece un nuevo elemento en este listado que no estaba presente antes, cuando no teníamos cláusulas where: las **constraints** (restricciones).

¿Qué información nos brinda este listado de navegación?

- que la tabla base del For each seguirá siendo CUSTOMER

- que se seguirá ordenando la consulta por *CustomerId*, utilizando el índice ICUSTOMER correspondiente

- que seguirá recorriendo toda la tabla CUSTOMER en busca de la información

- pero que para cada cliente evaluará si cumple con las restricciones que aparecen enumeradas (*CustomerName*>= *&Start* si *&Start* no está vacía y *CustomerName*<=*&End* si *&End* no está vacía) y solo en caso de que las cumpla, ejecutará para ese cliente los comandos que aparecen dentro del For each. En este caso, imprimirá los valores de los atributos del Printblock “customer”: *CustomerId*, *CustomerName*, *CountryName*.

- que debe acceder a la tabla COUNTRY cuya clave primaria es *CountryId* para obtener algún dato (*CountryName*)

For each: cláusulas Where

- Atributos permitidos: los de la tabla extendida de la tabla base del For each

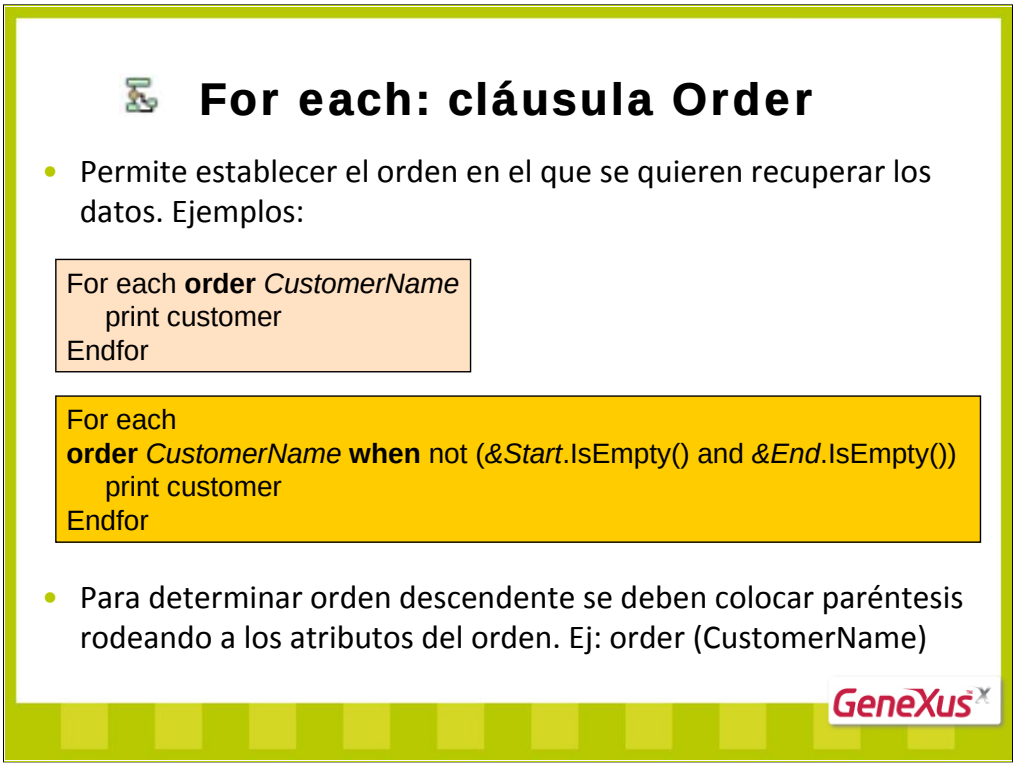
Ejemplo:

```
For each
where CountryName = 'Uruguay'
      print customer
Endfor
```

Los atributos utilizados en las condiciones de filtro pueden ser cualesquiera de la **tabla extendida del For each**.

En el ejemplo, si bien la tabla base del For each es CUSTOMER, estamos filtrando los datos a recuperar utilizando el atributo *CountryName*, que es de la tabla COUNTRY, perteneciente a la extendida de CUSTOMER.

En este ejemplo tampoco se explicita nada con respecto al orden, por lo cuál los datos aparecerán ordenados por la clave primaria de la tabla base, es decir, por identificador de cliente, *CustomerId*.



Si queremos realizar un listado de todos los clientes pero ordenado por nombre del cliente en lugar de por código, lo único que tenemos que hacer es modificar el comando For each agregando esta información del orden.

Esto se logra utilizando la **cláusula order** del For each, como se muestra en el primer ejemplo.

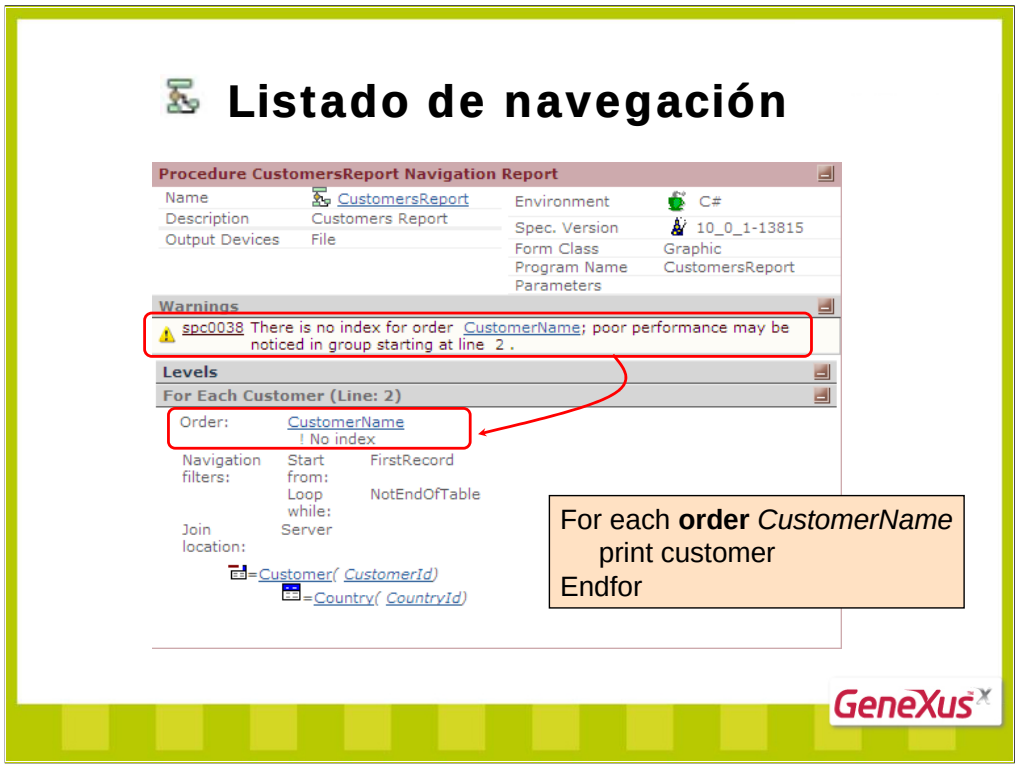
Como no existe un índice definido en la tabla CUSTOMER por el atributo *CustomerName*, GeneXus indicará en el listado de navegación mediante una advertencia ("warning") que **no existe un índice para satisfacer el orden**, lo que podría ocasionar problemas de performance, dependiendo de la plataforma de implementación elegida, de la cantidad de registros que deben ser leídos de la tabla, etc.

En ambientes cliente/servidor, si no existe índice para satisfacer el orden, el For each se traduce en una consulta SQL ("select") que es resuelta por el motor del DBMS de la plataforma.

Al igual que en el caso de las cláusulas where, la **cláusula order** puede condicionarse, como se muestra en el segundo ejemplo. En caso de no cumplirse la condición del when, no aplicará ese orden y de no existir orden incondicional (sin cláusula when) como en el ejemplo, el orden a utilizar será indefinido, significando esto que el orden resultante podrá variar de DBMS a DBMS e incluso entre ejecuciones sucesivas.

Pueden especificarse varias **cláusulas order** condicionadas (con **cláusula when**) consecutivas y una sin condición (la última de la lista). La primera cláusula order cuya condición del when se satisfaga, será la elegida y su orden el utilizado.

Cláusula Order None: cláusula que evita que GeneXus elija por defecto el orden de los atributos de la clave primaria de la tabla base y utilice un orden de navegación indefinido.
Si se utiliza la cláusula **Order None**, GeneXus entiende que no desea establecer ningún orden de recorrida en particular y delega esta tarea al DBMS.
La cláusula **order none** admite condición para aplicarse (**when**).



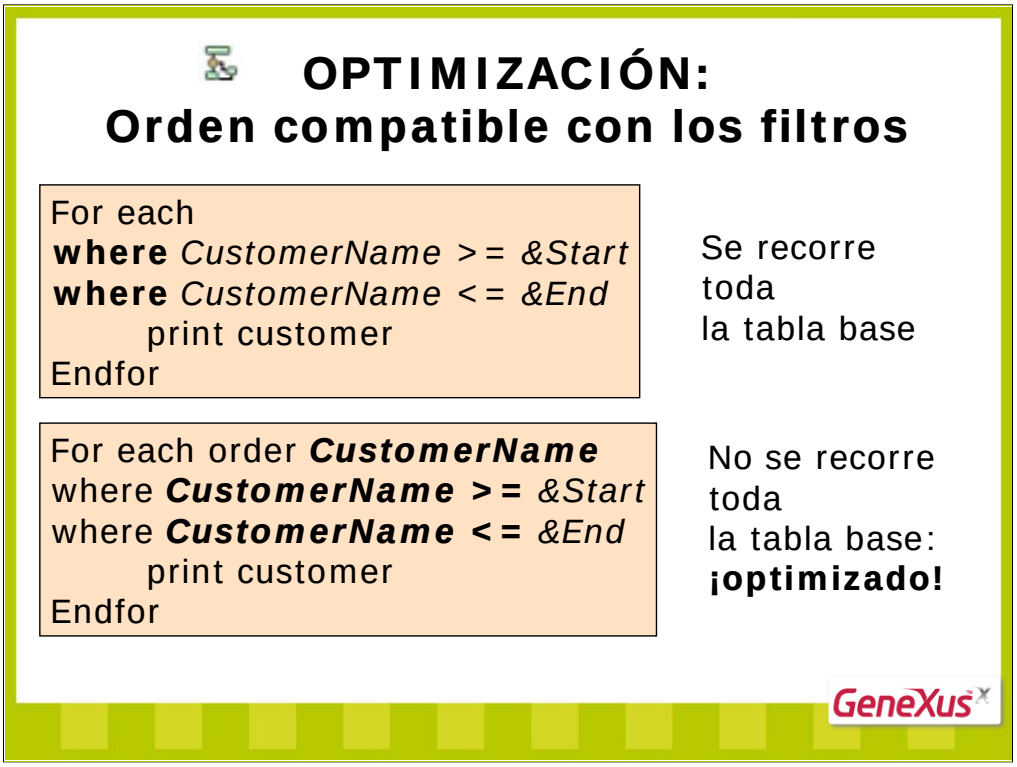
Listado de navegación

Cuando no existe índice que satisfaga el orden de un For each, como es el caso del ejemplo, el analista GeneXus puede resolver crearlo (**índice de usuario**). En la mayoría de los casos el procedimiento será bastante más eficiente de esta manera, pero se debe mantener un índice más (lo que implica mayor almacenamiento y mayor procesamiento para mantener actualizado el índice)

No es posible recomendar a priori cuál de las dos soluciones es la mejor (no índice vs índice de usuario), por lo tanto se debe estudiar, caso por caso, la solución particular teniendo en cuenta la plataforma de implementación y siendo fundamental la frecuencia con que se ejecutará el procedimiento. De cualquier manera, si al comienzo no se definió el índice de usuario y posteriormente se decide definirlo, alcanza con regenerar el procedimiento (sin modificar nada de lo programado en el mismo) y éste pasará a utilizarlo.

El listado de navegación anterior nos informa que:

• la tabla base del For each es CUSTOMER
• y se recorrerá ordenada por *CustomerName*
• no existe un índice definido para ese atributo
• se recorrerá toda la tabla CUSTOMER con el orden especificado
• no hay condiciones de filtro, por lo que para todos los registros de la tabla se ejecutarán los comandos dentro del For each (en nuestro caso, el comando print).



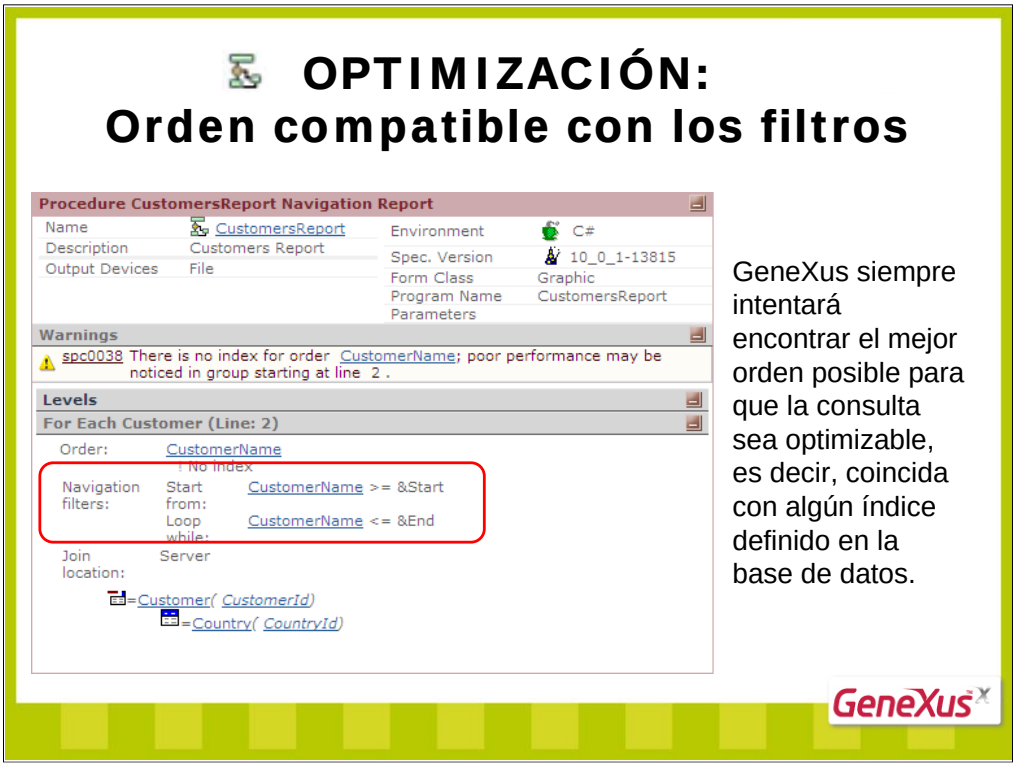
Si nos interesa filtrar los clientes de forma tal que sus nombres pertenezcan a un rango, en el primer ejemplo, como no especificamos **cláusula order** GeneXus ordena por clave primaria, es decir, por *CustomerId*.

En este caso, el listado de navegación nos va a informar que se debe recorrer toda la tabla CUSTOMER, y para cada registro de la misma se debe evaluar si el registro cumple o no con las condiciones (restricciones o "constraints"). En caso afirmativo, se imprimen para el mismo los datos correspondientes.

Si en lugar de ordenar los datos por *CustomerId* pedimos que se ordenen por *CustomerName*, como se presenta en el segundo ejemplo, la tabla base se recorre ordenada por *CustomerName* y como en los filtros establecemos que queremos quedarnos solo con aquellos clientes cuyo nombre, *CustomerName*, esté en el rango determinado, entonces ¡ya no será necesario recorrer toda la tabla base para obtener los datos que cumplen con las condiciones!

Diremos que esta segunda consulta está optimizada en ese sentido. Tener en cuenta, sin embargo, que GeneXus no tiene creado un índice en forma automática por *CustomerName*, y aquí habrá que evaluar si conviene crear un índice de usuario (que debe ser mantenido por Genexus luego) o no crear índice y que se cree un índice temporal en ejecución para resolver la consulta si el DBMS no puede resolverlo de otra forma.

El listado de navegación nos informará si una consulta está o no optimizada, de acuerdo a si recorrerá toda la tabla (desde "First Record" hasta "End of table") o si recorrerá por el contrario un rango más reducido.



El listado de navegación nos informa que la consulta está optimizada ya que recorre solamente los registros incluidos en el filtro (desde *CustomerName*>=&*Start* hasta *CustomerName* <=*&End*).

Para determinar el orden se tiene en cuenta:

•Los atributos de la cláusula Order especificada por el usuario
•Las restricciones que aplican al nivel (atributos mencionados en la regla Parm del procedimiento, condiciones explícitas tanto en el Where como en las Conditions)
•La existencia de índices sobre estos atributos.

Distinguimos 2 casos:

1) Se escribe una cláusula order

- El For each quedará ordenado por esos atributos, exista o no un índice por éstos.
- Si no existe un índice con los atributos del order, pero existen condiciones implícitas o condiciones explícitas **por igualdad**, se busca si existe un índice que contenga los atributos de las condiciones más los del Order. La condición explícita prevalece sobre la implícita para la determinación del Order, en caso que sean diferentes y exista índice por cada una de ellas.
- Si existe un índice, los atributos de las condiciones serán agregados en la lista del Order para que dicho índice sea considerado en su lugar.

2) No se escribe cláusula order

- En este caso, si existe algún índice por los atributos de la condición, el Order quedará determinado por los atributos del índice.
- **Si no existe un índice que corresponda con las condiciones**, o sea que no se puede optimizar la recorrida según las condiciones del nivel, entonces se ordenará por los atributos de la Primary Key.

Por ejemplo, si tenemos las transacciones:

```
COUNTRY
{
   CountryId*
}
```

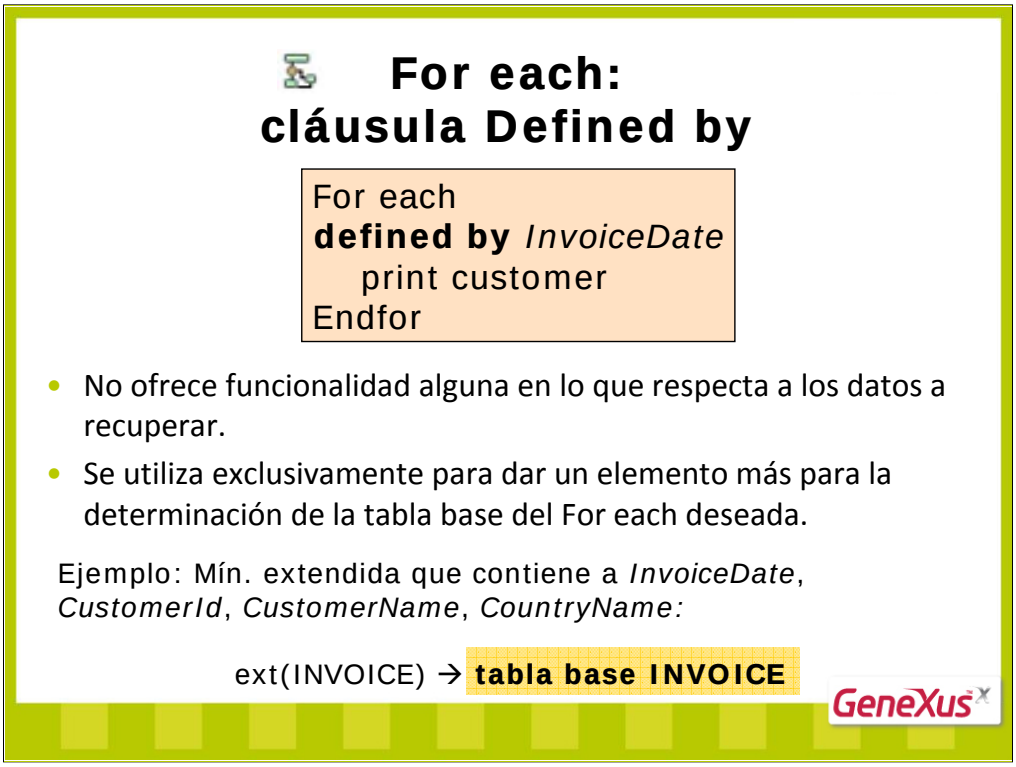
```
CITY
{
   CountryId*
   CityId*
}
```

El For each:

```
For Each order CityId
Where CountryId = 1
    ...
Endfor
```

Recorrerá la tabla CITY, ordenando por: CountryId, CityId y utilizando el índice ICITY (índice por clave primaria que contiene ambos atributos).

Aunque es el propio DBMS el que resuelve el plan de acceso más apropiado, la información antes mencionada influirá en su determinación.



Puede darse el caso de que para un For each haya más de una tabla base cuya extendida contenga a los atributos del For each, siendo mínima. Ante esta ambigüedad, GeneXus escoge la "primera" de estas tablas extendidas mínimas.

Para resolver este tipo de ambigüedad surge la cláusula **defined by**, que permite nombrar atributos de la tabla base deseada, que no se utilizarán para devolver la consulta ordenada por esos atributos, ni para filtrar la información a recuperar, ni para ser desplegados en el listado (es decir, no tienen funcionalidad alguna con respecto a los datos a recuperar), sino solo para **aportar más información que permita determinar la tabla base del For each**.

En la cláusula **Defined by** se debe hacer referencia a por lo menos un atributo de la **tabla base** deseada.

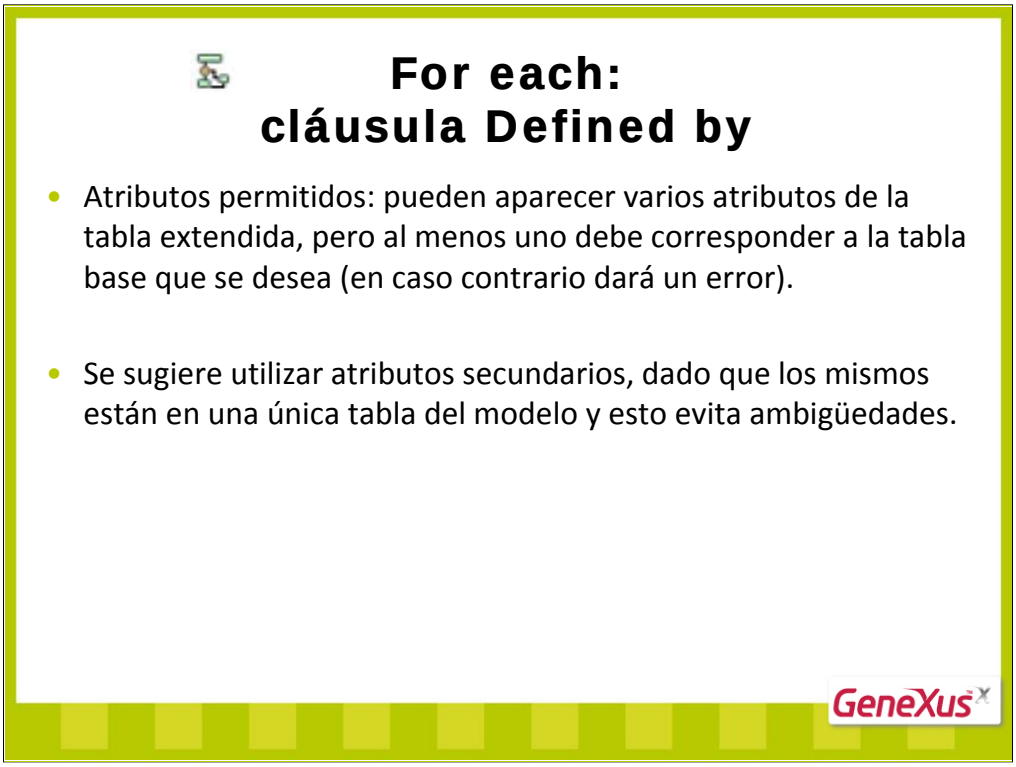
De la misma manera, se puede (y se suele) utilizar para modificar la que sería la tabla base en caso de no nombrarse ningún atributo más dentro del For each. Este es el caso del ejemplo presentado, en el que no queremos listar todos los clientes de la tabla CUSTOMER, sino por el contrario, queremos listar todos los clientes de las facturas. De no nombrarse dentro del For each algún atributo de INVOICE, la tabla base sería CUSTOMER.

En la mayoría de los casos no es necesario utilizar esta cláusula. Sin embargo, para procedimientos más o menos complejos, aún cuando no exista problema de ambigüedad, se recomienda el uso del **Defined by** porque mejora bastante el tiempo de especificación del procedimiento.
Sin embargo, no puede aconsejarse un uso indiscriminado. La contra de utilizar esta cláusula cuando no es necesaria es que ata un poco más el código al diseño de las tablas.

Supóngase por ejemplo que no se tiene creada una tabla COUNTRY, y se tiene de cada cliente el país al que pertenece, como atributo secundario. Si lo que queremos es listar los clientes y su país, serían equivalentes:

```
For each                          For each
                                  Defined by CountryName
   print customer                    print customer
Endfor                            Endfor
```

donde customer es un Printblock con los atributos *CustomerId*, *CustomerName* y *CountryName*.
Si ahora decide crearse la tabla COUNTRY y tener en la transacción "Customer" a *CountryId* como FK, si bien el primer For each continuará siendo válido y haciendo lo que queremos, el segundo dejará de funcionar, ya que en el Defined By no hay ningún atributo de la tabla base.



Pueden aparecer varios atributos, para el posible caso en el que no alcance uno solo para determinar completamente la tabla base deseada, donde al menos uno de ellos deberá estar asociado a la tabla base deseada.

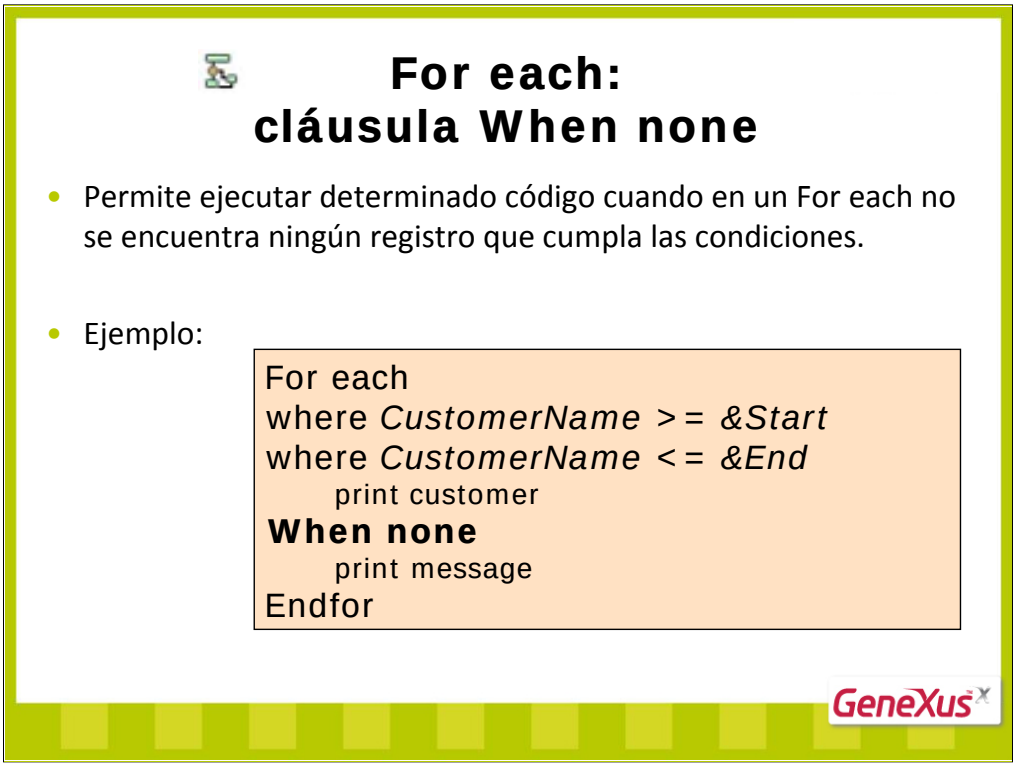
Se recomienda el uso en el Defined by de **atributos secundarios** de la tabla base que se desea navegar, ya que como sabemos, los atributos secundarios solo pueden estar en una tabla del modelo y de esta forma eliminamos por completo toda posible ambigüedad.

Esto sin embargo, no es obligatorio, es decir, pueden nombrarse en el Defined by atributos primarios cuando se sepa que no habrá ambigüedad en la elección de la tabla base.

Un error común es creer que cuando un For each tiene esta cláusula, la tabla base del mismo queda determinada exclusivamente a partir de los atributos mencionados en el defined by.

La realidad es que los atributos del defined by arrojan una o más tablas base **candidatas** a ser la tabla base del For each, pero luego hay que ver si tomando cada tabla base candidata, su extendida contiene a todos los demás atributos del For each, además de los del defined by.

Si ninguna de las posibles tablas base candidatas cumplen esta condición, entonces el procedimiento dará un error al ser especificado, y no podrá ser generado (recordemos que debe cumplirse que todos los atributos del For each estén contenidos en una misma tabla extendida).



El Printblock *message* (que podrá contener un texto advirtiendo al usuario de que no existen clientes que cumplan los filtros) se ejecuta sólo cuando no se entra en el For each, es decir, cuando no hay ningún registro correspondiente a la tabla base del For each para el que se cumplan las condiciones de filtro.

También se aplica a *For each [selected] line*, *XFor Each* y *XFor First*, comandos que veremos más adelante.

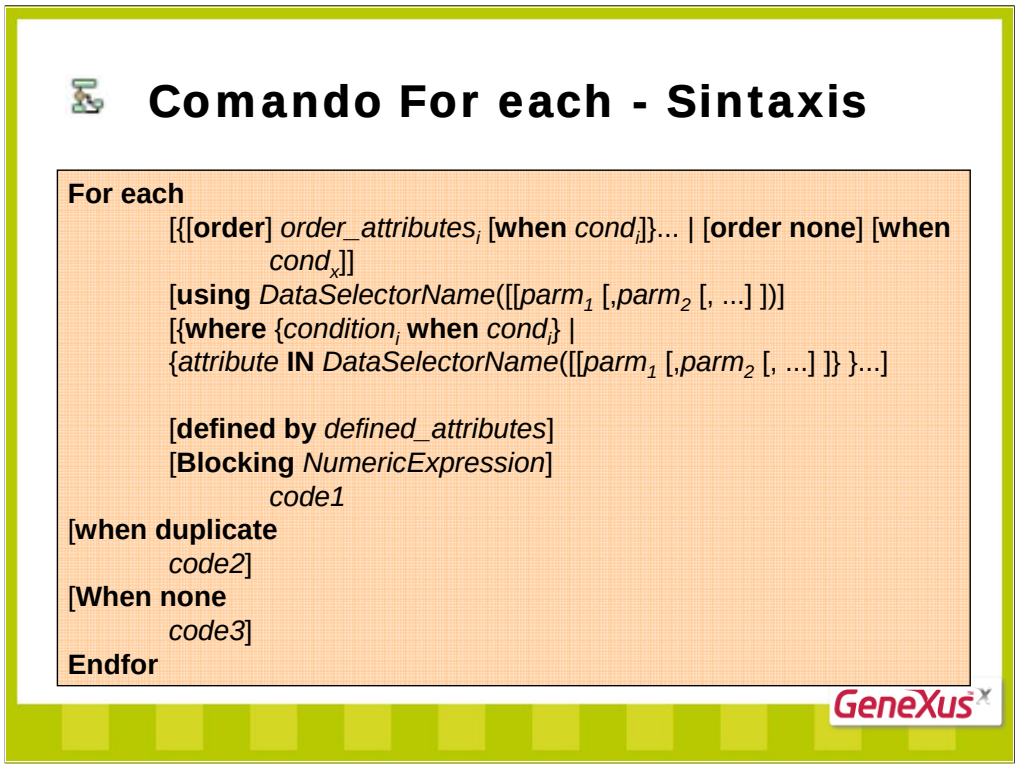
La **cláusula When none** debe ser la última dentro del For each. Las acciones a realizar cuando no existe ningún registro para el que se cumplan las condiciones, quedan determinadas por el *bloque de código* que hay entre la **cláusula When none** del For each y el **Endfor**.

Cuando un For each no tiene condiciones de filtro, los comandos del When none se ejecutarán solo en el caso en que se cumpla que la tabla base del For each esté vacía, porque solo en ese caso no habrá ningún registro que cumpla las condiciones de filtro.

Importante:

• Si aparecen atributos en el bloque de código correspondiente al When none, éstos **no** son tenidos en cuenta para determinar la tabla base del For each.

• Si se incluyen For eachs dentro del When none no se infieren Joins ni filtros de ningún tipo con respecto al For each que contiene el When none, ya que son considerados For eachs paralelos.



La sintaxis presentada generaliza el ejemplo con el que vinimos trabajando.

Order *order_attributes::= att$_1$, …, att$_n$*
Es una lista de atributos, que indican el orden en el que será devuelta la consulta, siendo *att$_i$* un atributo de la base de conocimiento escrito simple, o entre paréntesis curvos. Cuando un atributo del order aparece rodeado de paréntesis curvos se está indicando orden descendente para el mismo.

Pueden mencionarse atributos de la **tabla extendida**.

Es posible definir varias cláusulas order condicionales, y una incondicional, que debería ser la última listada. El por qué responde al hecho de que como solamente una de esas cláusulas order tomará efecto, se van evaluando sus condiciones (las del when) hasta la primera que de True, y con esa se queda. Si ninguna da true y existe una cláusula incondicional (es decir, sin when), se tomará ese orden. Si no existe tal cláusula, el orden será indefinido, queriendo esto significar que dependerá de la plataforma, e incluso podrá variar entre ejecuciones sucesivas. La justificación para escribir cláusulas order condicionales, deriva de si queremos aplicar cláusulas where condicionales. Es decir, por motivos de optimización de las consultas.

Por ejemplo, si queremos filtrar por *CustomerName* > *&Name* when not *&Name*.IsEmpty(), entonces para optimizar la consulta deberíamos ordenar por *CustomerName*, pero si no se va a aplicar el filtro, dado que &Name está vacía, entonces será mejor dejar un orden indefinido.
Para ello especificamos la cláusula order condicional:

order *CustomerName* **when** not *&Name*.IsEmpty()

En lugar de todo lo anterior, también puede especificarse una **cláusula order none** que se agrega para cuando no nos interesa un orden en particular y queremos que éste quede indefinido.

Elección del índice: GeneXus elige automáticamente el índice a utilizar para satisfacer el orden. GeneXus siempre intentará encontrar el mejor orden posible para que la consulta sea optimizable, es decir, coincida con algún índice definido en la base de datos. Para determinar el orden se tiene en cuenta:

• Los atributos de la cláusula Order especificada por el usuario
• Las restricciones que aplican al nivel (atributos mencionados en la regla Parm del procedimiento, condiciones explícitas tanto en el Where como en las Conditions)
• La existencia de índices sobre estos atributos.

Aunque es el propio DBMS el que resuelve el plan de acceso más apropiado, la información antes mencionada influirá en su determinación.

Los atributos del **order** son tenidos en cuenta a la hora de determinar la **tabla base del For each**. Pero ellos por sí solos no la determinan. Deben examinarse también otras partes del For each.

Using *DataSelectorName*
Permite definir filtros acorde al criterio definido en el DataSelector[1] definido en *DataSelectorName.*

Where *Condition*
Condición booleana que deberán cumplir los datos para ser procesados dentro del For each, pudiendo ser una condición compuesta, utilizando los operadores lógicos and, or y not.

Los atributos que aparezcan en la condición booleana pueden ser tanto de la **tabla base del For each como de la extendida**.

Como se desprende de la sintaxis, para un mismo For each pueden especificarse n cláusulas where sucesivas, cada una con una condición:
where $cond_1$
where $cond_2$
...
where $cond_n$

La ocurrencia de n cláusulas where es equivalente a la ocurrencia de una sola cláusula, con la conjunción booleana de las condiciones:

where $cond_1$ **and** $cond_2$ **and** ... **and** $cond_n$

Los datos de la tabla extendida del For each que cumplan con <u>todas</u> las condiciones de los “where” serán los procesados en los comandos internos al For each (los del bloque de código $code_1$).

Al igual que ocurre con la cláusula order, podrán condicionarse los filtros (con cláusulas when). De esta manera, primero se evalúa la cláusula when de cada cláusula where, y de cumplirse su condición, aplicarán el filtro especificado en el where.

<u>Nota:</u> Existe también la cláusula **Option Distinct** del For Each que permite retornar los registros que cumplan unicidad de valores de los atributos referenciados. No veremos esta cláusula. El lector interesado puede recurrir a las distintas fuentes de documentación para estudiarla (Help, Wiki)

---

[1] El objeto DataSelector se verá más adelante en el curso.

Defined by
defined_attributes::= att_1, att_2,…,att_p
Es un conjunto de atributos que serán utilizados a los solos efectos de determinar la tabla base del For each.

Al mencionar aquí algunos atributos de la tabla que se desea recorrer, éstos participarán en la determinación de la tabla base del For each.

La cláusula **defined by** aparece para solucionar algunos problemas de ambigüedad en la determinación de la tabla base (cuando existen varias tablas extendidas mínimas que contienen a los atributos del For each) o cuando se desea que la tabla base sea otra, distinta de la que sería determinada por los atributos que aparecen en el resto del For each (este caso cobrará sentido cuando se estudie "corte de control"). También se utiliza para mejorar el tiempo de especificación en procedimientos complejos.

Los atributos de esta cláusula no determinan por sí solos la tabla base del For each. Podría darse el caso de que de estos atributos surja determinada tabla como la **candidata a tabla base**, pero si luego el resto de los atributos del For each no están contenidos en la extendida de esa tabla, el For each dará un error y el objeto que lo contiene no podrá ser generado.

$code_1$
Es una sucesión de comandos en los que pueden utilizarse atributos de la **tabla extendida** del For each. A este bloque de código le llamaremos **cuerpo del For each**.

Los atributos que figuren en este bloque de código participarán en la determinación de la tabla base del For each.

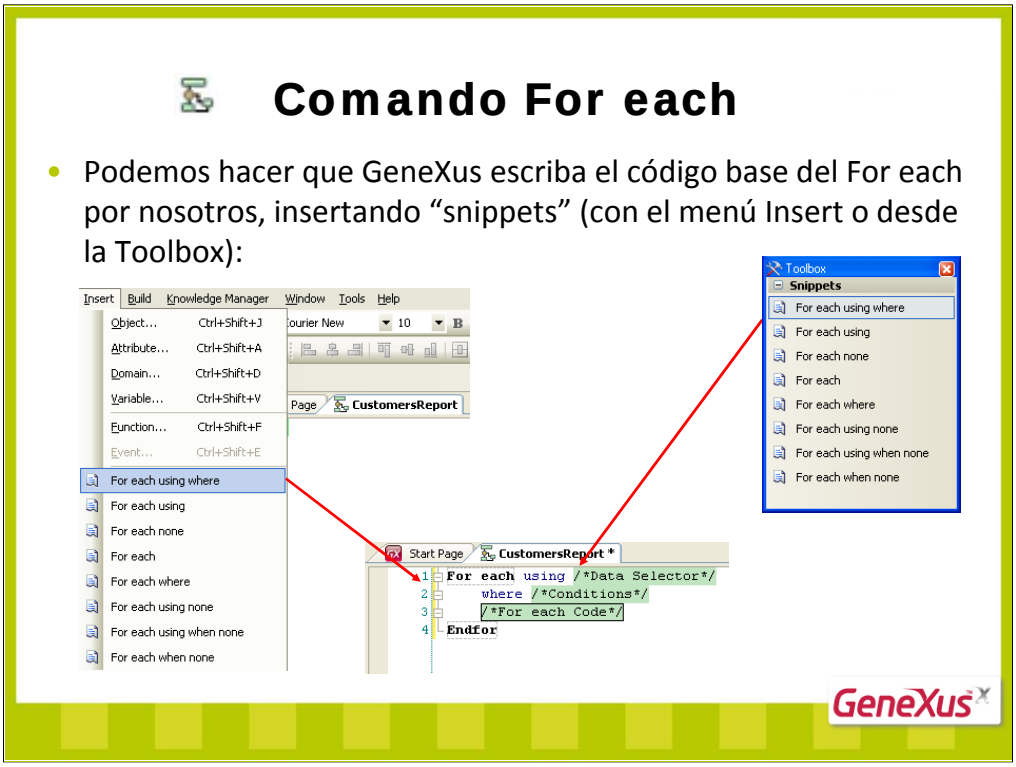
Los comandos especificados se ejecutarán secuencialmente para los datos de la tabla extendida que cumplan las condiciones de filtro, considerándose los datos en el orden especificado.

Blocking
Si bien se abordará este tema más adelante en el curso, a modo de idea general diremos que la especificación de esta cláusula permite realizar actualizaciones y eliminaciones en bloque, reduciendo así el número de accesos a la base de datos.

When Duplicate
Esta cláusula solo tiene sentido en procedimientos (dado que tiene que ver con la actualización) y se verá más adelante.
Se ejecutará esta cláusula si dentro del cuerpo del For each *$code_1$*, se intenta actualizar un atributo que es clave candidata (tiene índice unique) y ya existe un registro con ese valor. GeneXus utiliza el índice unique para asegurar la unicidad de esa clave candidata y en caso de encontrar duplicados, si el For each tiene programada esta cláusula, ejecutará su código: *$code_2$* .
De no existir la cláusula no se ejecutará código alguno.

When none
En caso de que no existan datos que cumplan las condiciones de filtro no se ejecutarán los comandos del *$code_1$* sino que se ejecutarán los del bloque de código *$code_3$*.

➢ Tanto para When Duplicate como para When none: Si se incluye dentro de alguno de los dos un comando For each, no se infieren joins ni filtros con respecto al For each que los contiene (el del When none | when duplicate). Son consideradas navegaciones independientes (la del *$code_1$*, *$code_2$* y *$code_3$*).



Los “code snippets” son plantillas de código que GeneXus tiene predefinidas, que nos ayudan a escribir el código fuente. Cuando estamos trabajando en el Source de un procedimiento, la Toolbox nos muestra varios snippets que nos facilitan la escritura del comando For each.

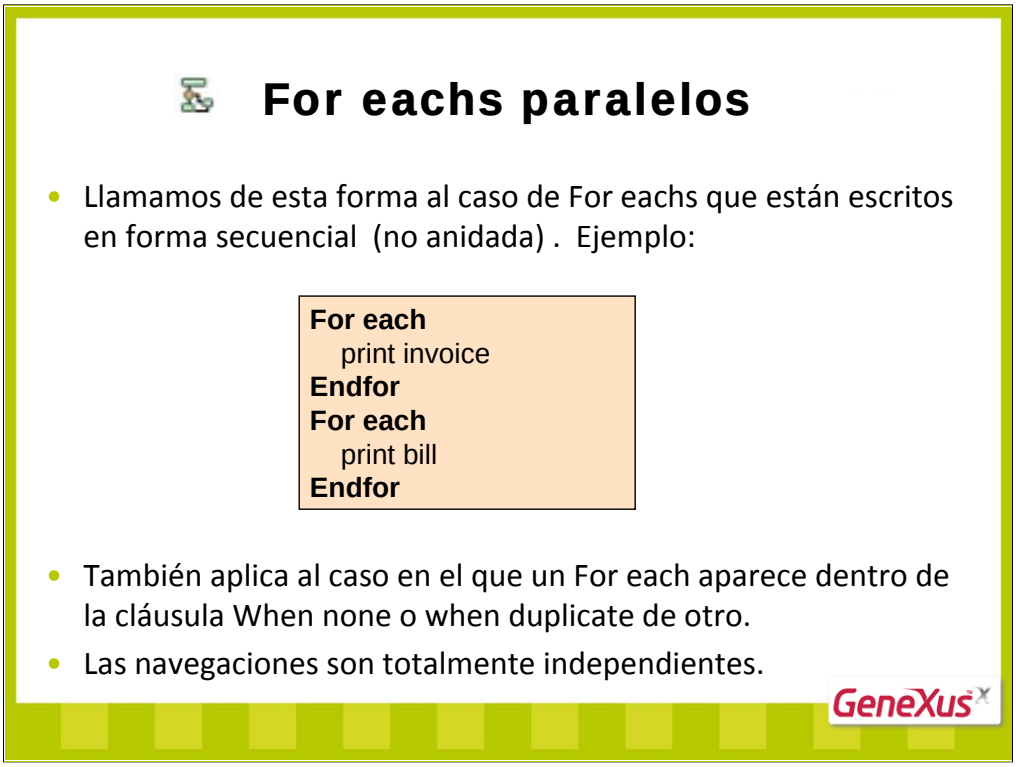
Los snippets tienen a su vez un “atajo” (shorcut), que hacen que la escritura de código sea más rápida todavía. Por ejemplo, digitando simplemente “fe” se escribe automáticamente el siguiente código:

```
For each
    /*For each Code*/
Endfor
```

Luego el usuario sustituye la línea de comentarios con el código necesario.

La lista de los atajos de cada snippet, es la siguiente:

- fe (For each)
- feu (For each using)
- fen (For each When none)
- feun (For each using When none)
- few (For each where)
- feuw (For each using where)
- fewn (For each where When none)
- feuwn (For each using where When none)



El For each es un comando como otros, y por tanto puede aparecer varias veces dentro del Source, tanto en forma paralela (independiente), como anidado a otro For each.

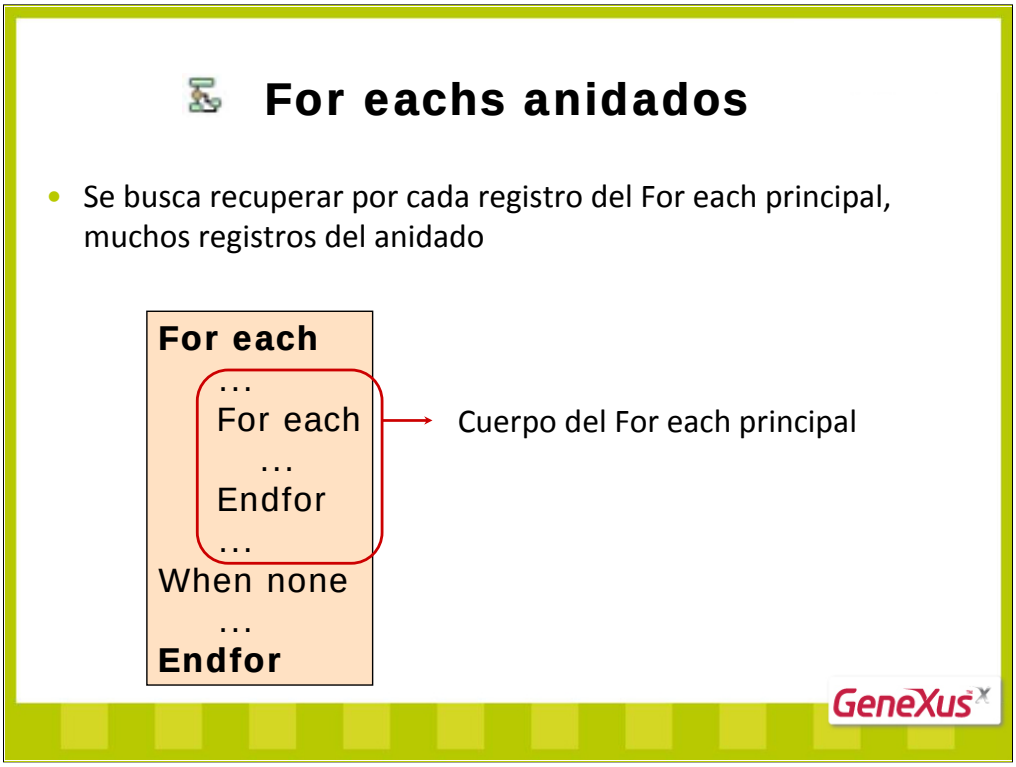
Cuando dentro del **cuerpo del For each** (*code*$_1$) aparece otro For each, decimos que se trata de For eachs anidados. GeneXus soporta varios niveles de anidamiento para los For eachs.

El caso en el que un For each aparece en el bloque de código *code*$_3$, si bien puede verse como un anidamiento de For eachs porque uno aparece dentro de otro, el comportamiento en cambio, es como el del caso de For eachs paralelos.

Un For each “anidado en el When none” de otro, solo se ejecutará si no existe ningún registro de la tabla base del For each que lo contiene que cumpla las condiciones de filtro.

En el ejemplo, “invoice” y “bill” son dos Printblocks del Layout que continen atributos de las tablas INVOICE y BILL (recibo) respectivamente.

Hemos definido dos For eachs paralelos. El primero recorrerá todas las facturas y el segundo todos los recibos.



El For each es una estructura repetitiva que permite recuperar muchos registros de una tabla. Cuando uno piensa en For eachs anidados, es evidente que lo que se busca recuperar es, por cada registro del principal, muchos registros del anidado.

¿Qué cosas podría GeneXus detectar que se quiere hacer con este tipo de estructuras, de forma tal de poder inferir el comportamiento automáticamente con la menor codificación posible?

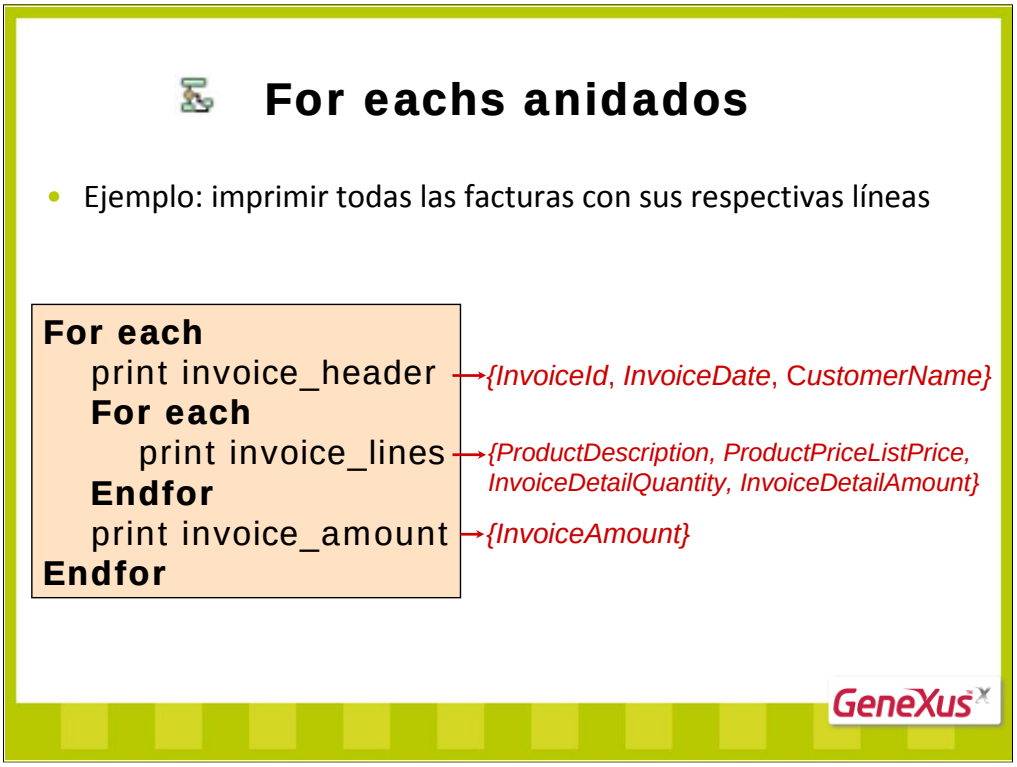
• para cada registro de una tabla recuperar algunos de otra: los relacionados.
• para cada registro de una tabla recuperar todos los de otra.
• procesar información por grupos, esto es, agrupar los registros de una tabla según el valor de un atributo o conjunto de atributos y para cada grupo, recuperar algunos registros: los correspondientes al grupo.

Siempre la relación es uno a muchos: por cada registro de una tabla recuperar muchos de la otra (pudiéndose tratar de la misma tabla).

Utilizando esta lógica es que GeneXus infiere las **tablas base** y el **comportamiento** de los For eachs anidados, sabiendo que lo que se desea es implementar alguna de las tres opciones anteriores.

Por ejemplo si queremos realizar un listado de todas las facturas del sistema, donde para cada una queremos imprimir también el detalle de la misma, debemos recorrer dos tablas: INVOICE (que almacena los cabezales) e INVOICEDETAIL (que almacena las líneas), y lo haremos de una forma bien simple, sin nombrar las tablas, y sin tener que explicitar todo, como veremos.

Otro ejemplo es el de un listado de todos los clientes, donde para cada uno se quieren imprimir, además de sus datos personales, los datos de todas sus facturas. Este comportamiento se logra con un par de For eachs anidados, donde el primero recorrerá la tabla CUSTOMER y el segundo recorrerá la tabla INVOICE, recuperando solo las facturas de ese cliente, como veremos en breve.

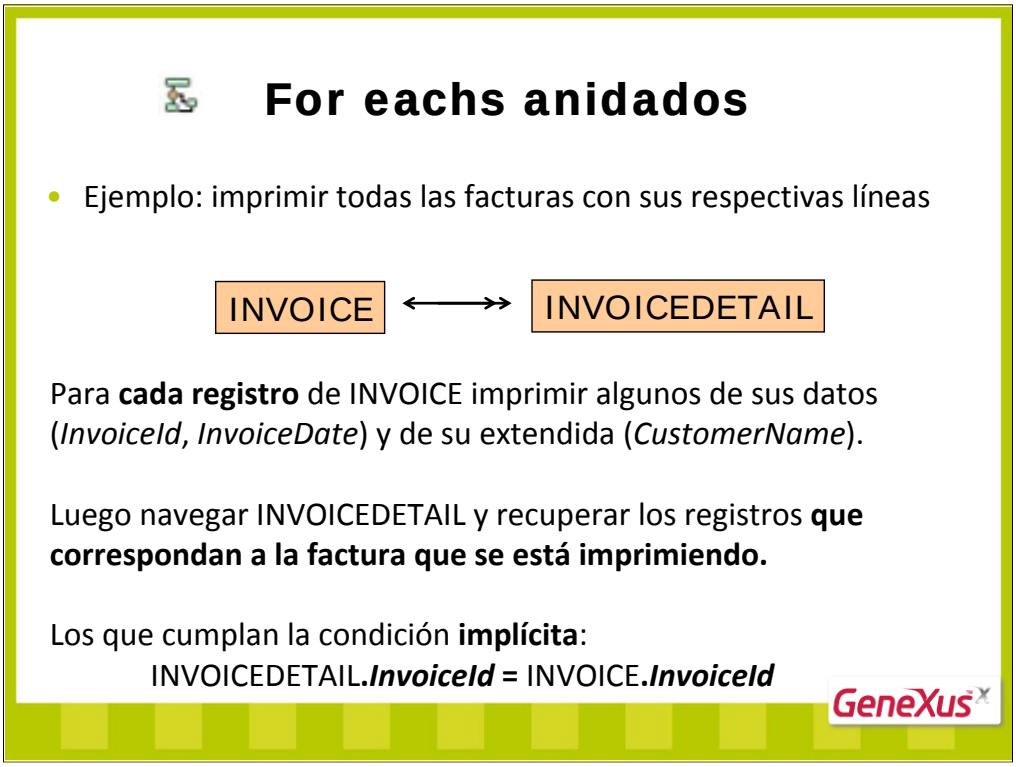


Queremos realizar un listado de todas las facturas del sistema, donde para cada una se muestre tanto información de su cabezal como de sus líneas.

En el Source programado, claramente recorremos la tabla INVOICE, imprimiendo los atributos que nos interesan del cabezal y para cada registro de esa tabla, recorremos la tabla INVOICEDETAIL, para imprimir los atributos que nos interesan de sus líneas.

¡Tan simple como eso! Otra vez, nos alcanza con nombrar los atributos que queremos utilizar y del resto se encarga GeneXus.

Observemos que ni siquiera tuvimos que especificar condición de filtro sobre los registros de INVOICEDETAIL a recuperar. GeneXus se da cuenta de la relación existente entre las tablas, y aplica automáticamente la condición de filtro sobre los datos, de manera tal de solo imprimir las líneas de "esa" factura (recordar que algo idéntico ocurría con las fórmulas verticales, como InvoiceAmount, donde la condición de filtro sobre los registros a ser sumados o contados quedaba implícita, y no había que especificarla).

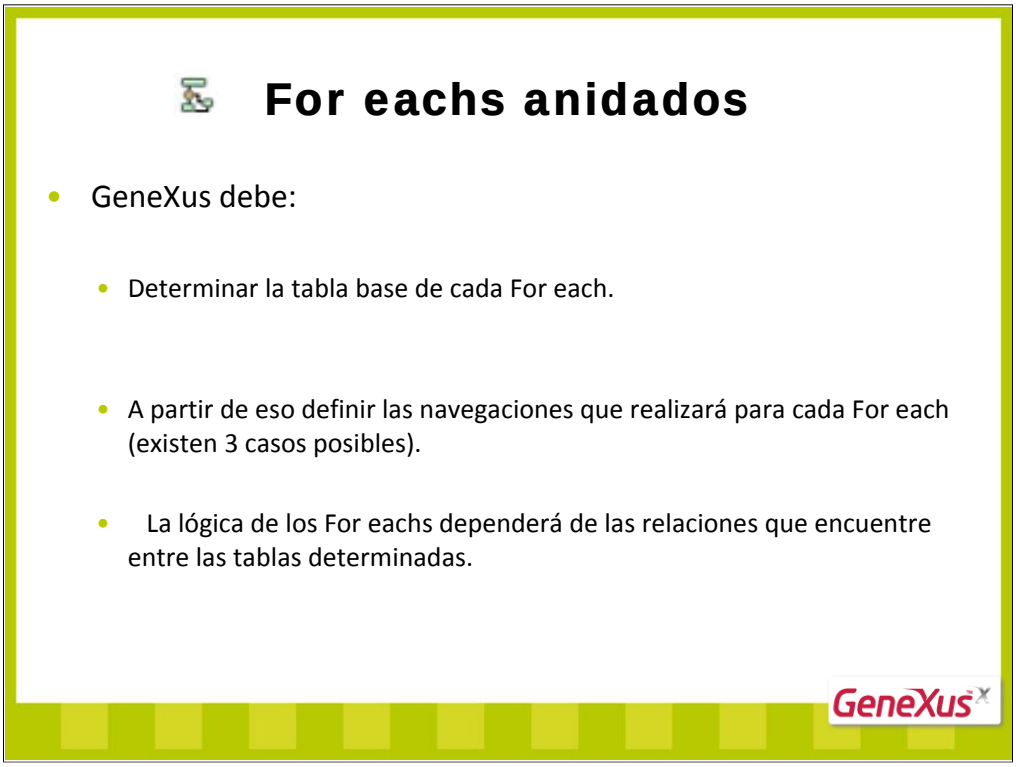


Como para un For each simple, GeneXus debe determinar para cada For each (principal y anidado) cuál es su tabla base. Y luego, a partir de esa determinación, inferirá la lógica correspondiente.

Más adelante veremos con exactitud cómo es que GeneXus determina cada tabla base. Aquí nos quedaremos con la idea intuitiva de que lo hace en forma similar a como lo hacía para el caso de un For each simple.

Encuentra, pues, que debe recorrer las tablas INVOICE e INVOICEDETAIL.

Como esas tablas están relacionadas de acuerdo a una relación 1-N infiere más que eso: infiere además que debe aplicar la condición sobre los datos de INVOICEDETAIL que se indica arriba.



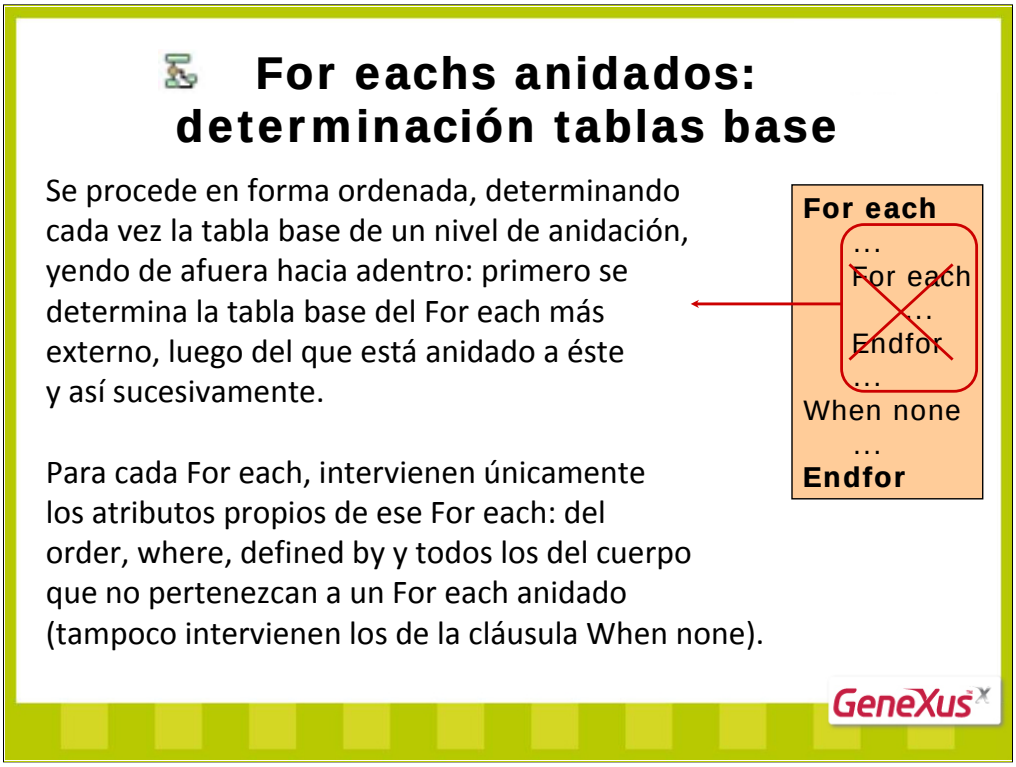
Cuando tenemos For eachs anidados, GeneXus debe determinar la **tabla base** de cada uno, y esas serán las tablas que se navegarán.

Para cada registro de la tabla base del For each principal, se ejecutarán los comandos del cuerpo del mismo. Entre esos comandos, se encuentra el For each interno, que se ejecutará, como cualquier otro comando, en el lugar donde se encuentre, realizando una navegación sobre su tabla base.

Pero para la determinación de la tabla base del For each anidado, podrá influir la tabla base del For each principal, por lo que las determinaciones no son por completo independientes, como podría pensarse.

A partir de la **determinación de las tablas base** de cada For each primero y de las **relaciones** que encuentre GeneXus entre las tablas involucradas luego, surgen tres posibles casos de For eachs anidados: join, producto cartesiano y corte de control, respectivamente.

Estudiaremos en lo que sigue cada uno de estos casos con ejemplos, para luego generalizar todo lo visto.



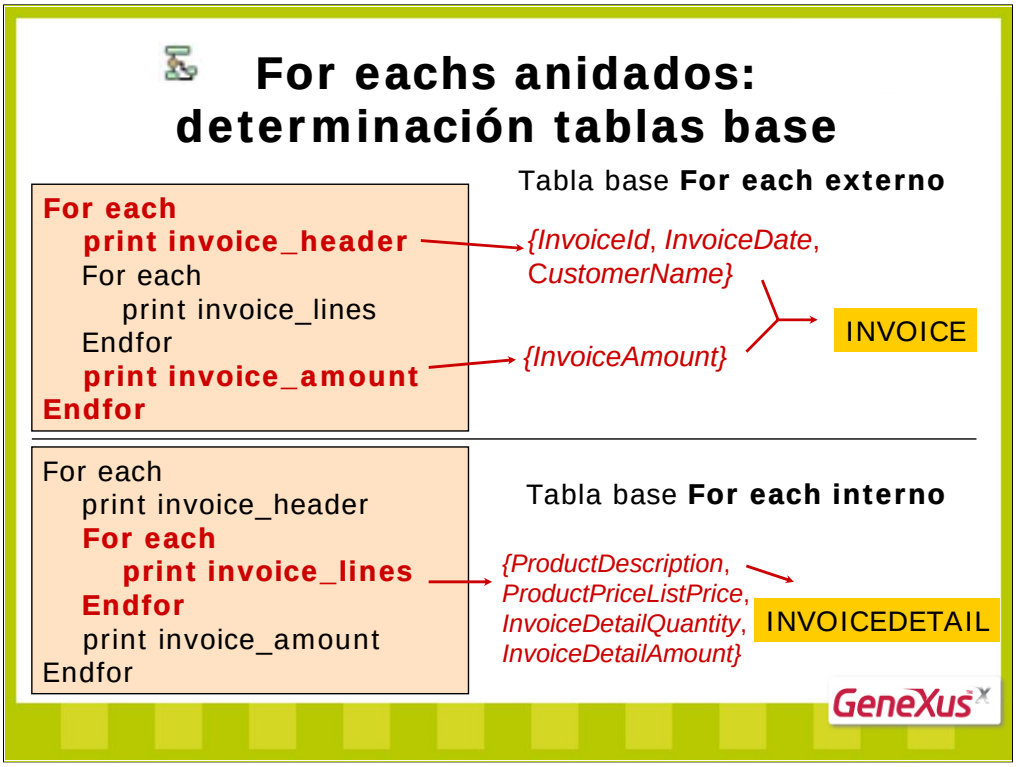
Consideremos el caso más simple, de un par de For eachs anidados.

• La **determinación de la tabla base del For each principal**, es análoga al caso de For each simple (sin anidamientos). En este caso se consideran todos los atributos del For each principal, descartando los For eachs anidados que éste contenga (y todos sus atributos). GeneXus encuentra la mínima tabla extendida que contenga a los atributos referidos y define así la tabla base a través de la cual llega a todas las demás.

• **Para determinar la tabla base del For each anidado**, GeneXus se fija si los atributos utilizados dentro del cuerpo del mismo están incluidos o no dentro de la tabla extendida previamente determinada (la del For each principal). En caso afirmativo, GeneXus determina que la tabla base del For each anidado será la misma que la del For each principal.
En caso contrario, se busca la mínima tabla extendida que cumpla que contiene a todos los atributos del For each anidado y que tenga alguna relación con la tabla base dell For each principal. La tabla base de dicha tabla extendida, será la tabla base del For each.

Si no puede encontrar una tabla extendida mínima que cumpla ambas condiciones, pero cumple que contenga a los atributos, finalmente la elige, pero siempre busca la forma de encontrar relaciones entre ambas tablas bases. Sólo en este último caso de que no se encuentran relaciones, se procede a determinar la tabla base como si se tratase de For eachs independientes.

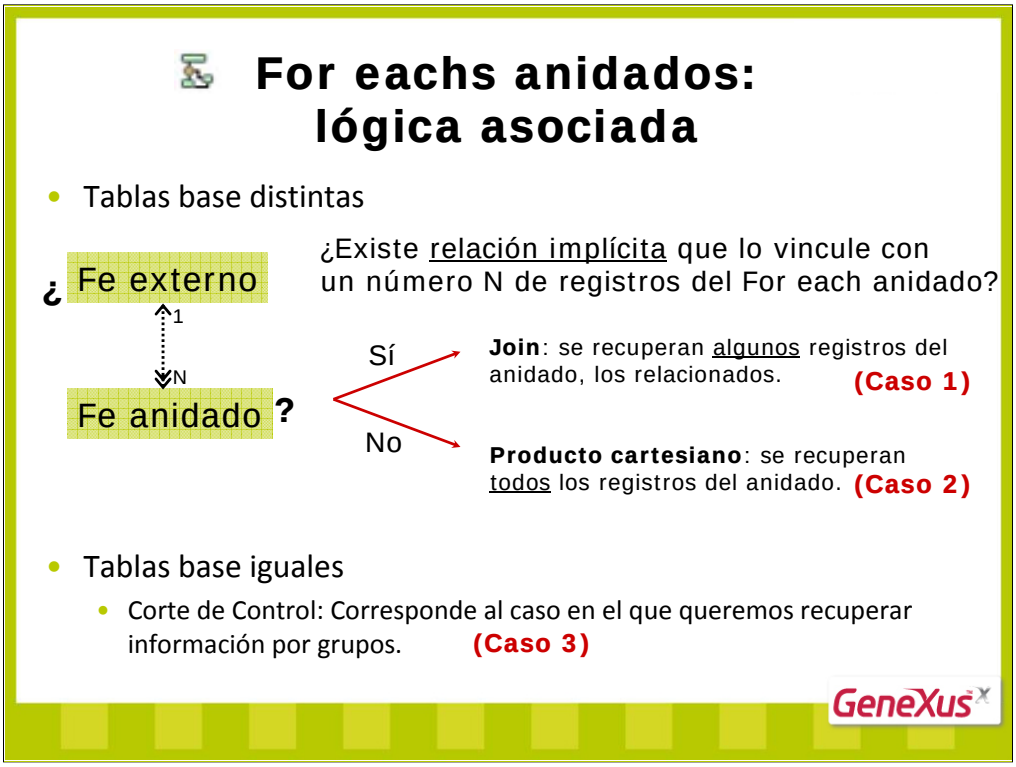
Veremos un esquema resumiendo lo anterior más adelante.



Para el ejemplo que presentamos antes, mostramos arriba cómo se determina cada tabla base.

Para la del anidado, como los atributos que figuran no están contenidos en la tabla extendida de INVOICE (que es la tabla base del principal), entonces se pasa a determinar su tabla base como se explicó anteriormente: se busca la tabla extendida mínima que contenga a *ProductDescription*, *ProductPriceListPrice*, *InvoiceDetailQuantity* e *InvoiceDetailAmount*, y que de ser posible esté relacionada con INVOICE.

La tabla que cumple ambos requisitos en INVOICEDETAIL.



De la determinación de las tablas base, surgen los tres casos de For eachs anidados que se mencionan y que estudiaremos uno a uno en lo sucesivo.

Este tema es de vital importancia, ya que la mayoría de las aplicaciones requieren navegaciones complejas sobre las tablas, donde se requiere una mixtura de todos estos casos.

Este es el caso en el que GeneXus determina que las tablas base de cada For each son distintas y hay una especie de relación 1-N (pudiendo ser ésta indirecta, como veremos luego) entre las tablas que se recorren.

Es decir, por cada registro de la tabla base del For each principal, GeneXus encuentra que hay N relacionados con él, directa o indirectamente, en la tabla base del For each anidado.

Al encontrar esta relación, aplicará condiciones de filtro automáticas en el For each anidado, de forma tal de solo quedarse con esos registros relacionados.

El ejemplo del procedimiento en el que se imprimen todas las facturas del sistema, con sus detalles, cae dentro de esta categoría. En ese caso hay una relación 1-N directa entre las tablas que se recorren: para cada cabezal de factura, se lista el mismo, junto con todas sus líneas de factura, es decir, todos los registros de INVOICEDETAIL que están relacionados con el registro de INVOICE en el que se está posicionado en cada iteración.

Este es uno de los casos más comunes de For eachs anidados, donde se quiere recorrer una tabla, y para cada registro de la misma, recorrer otra tabla, relacionada con la primera por una relación N-1. GeneXus encuentra esta relación, y en la navegación interna solo recupera los registros asociados, y de ahí el nombre de "join" para este caso.

En el ejemplo presentado arriba ocurre lo mismo. La tabla base del primer For each es CUSTOMER, y la del segundo, INVOICE. Como encuentra atributo en común entre la **tabla extendida del For each principal** y la **tabla base del anidado**[1], *CustomerId*, determina que ese atributo actuará como condición de filtro en la recorrida de la tabla del For each anidado.

---

[1] Esta es una forma de expresar formalmente lo que habíamos dicho en términos informales: relación directa o indirecta 1-N entre las tablas base. La relación será directa cuando la tabla base del principal tenga relación 1-N con la tabla base del anidado, es decir, sea superordinada de esta última. La relación será indirecta cuando esto no exista una relación directa entre las tablas base, pero sí entre la tabla extendida del primero y la tabla base del segundo. También será indirecta cuando la tabla extendida del anidado incluya a la tabla base del principal. De esto veremos ejemplos luego.

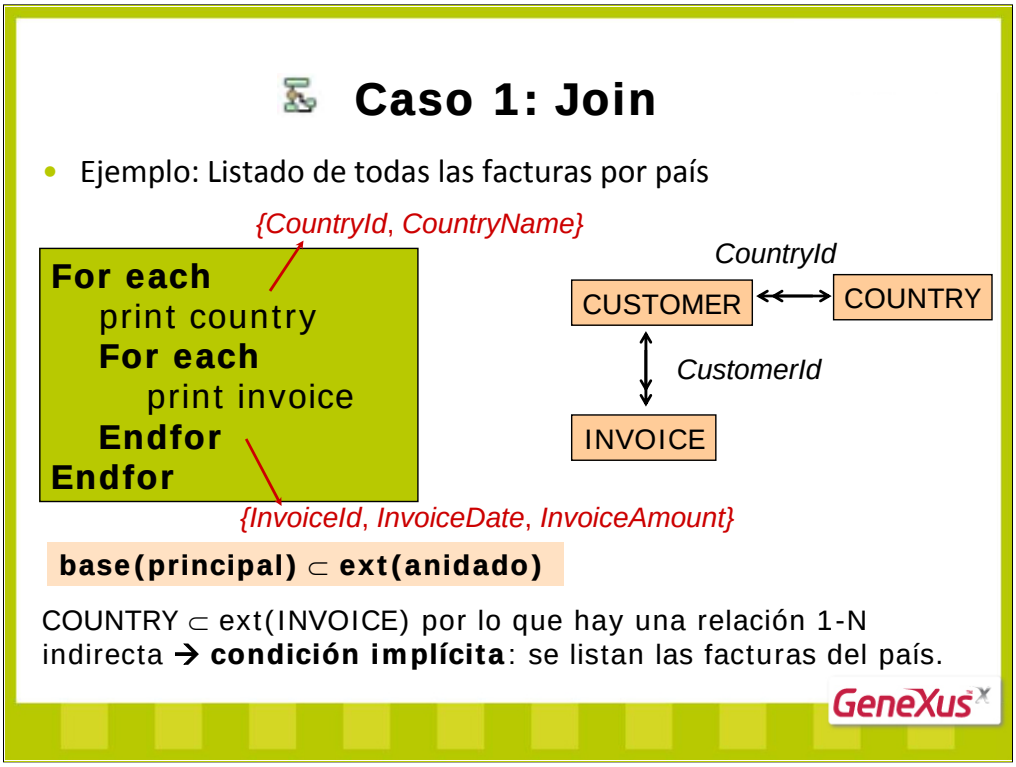
Veamos, por ahora intuitivamente, cómo hace GeneXus para determinar las tablas base. Los atributos utilizados en el For each externo son *CustomerId* y *CustomerName*, por lo que la tabla base de este For each será claramente CUSTOMER. Observemos que solo participan en la determinación de esta tabla base los atributos del For each principal, no los del anidado.

Luego GeneXus debe encontrar la tabla base del For each anidado. Los atributos que participan son *InvoiceId*, *InvoiceDate* y *InvoiceAmount*, es decir, los atributos internos a este For each. Observemos que estos atributos no pertenecen a la tabla extendida del principal, que era CUSTOMER. Por tanto se pasa a determinar su tabla base como la de cualquier For each simple: la tabla extendida de INVOICE contiene a todos los atributos del For each anidado, y es la mínima tabla extendida que los contiene. Por lo tanto, INVOICE será elegida como tabla base del segundo For each.

Observemos, nuevamente, que no explicitamos cláusula where en el For each anidado para filtrar las facturas del cliente del For each principal. Justamente, por tratarse de tablas relacionadas por una relación 1-N directa, esta condición es aplicada implícitamente por GeneXus y puede verse claramente en el listado de navegación que se muestra arriba.

Hay otro hecho interesante que podemos observar en este listado: si bien en el For each anidado no especificamos cláusula order, GeneXus no eligió el orden por clave primaria de la tabla base sino por el atributo relación, *CustomerId*. De esta manera, está optimizando automáticamente la consulta.

Acá se presenta, pues, una excepción a la regla que enunciaba que cuando no se especifica cláusula order en un For each, GeneXus determina como orden el de la clave primaria de la tabla base de dicho For each.



Si queremos realizar un listado de las facturas emitidas por país, tenemos otro caso de For eachs anidados con distintas tablas base, donde la información que se quiere listar está relacionada.

Aquí queremos recorrer las tablas COUNTRY e INVOICE.

Observemos que si bien no están relacionadas directamente, sí lo están en forma indirecta. De hecho, COUNTRY pertenece a la tabla extendida de INVOICE. Por lo tanto, por cada factura se puede encontrar un solo país relacionado con la misma.

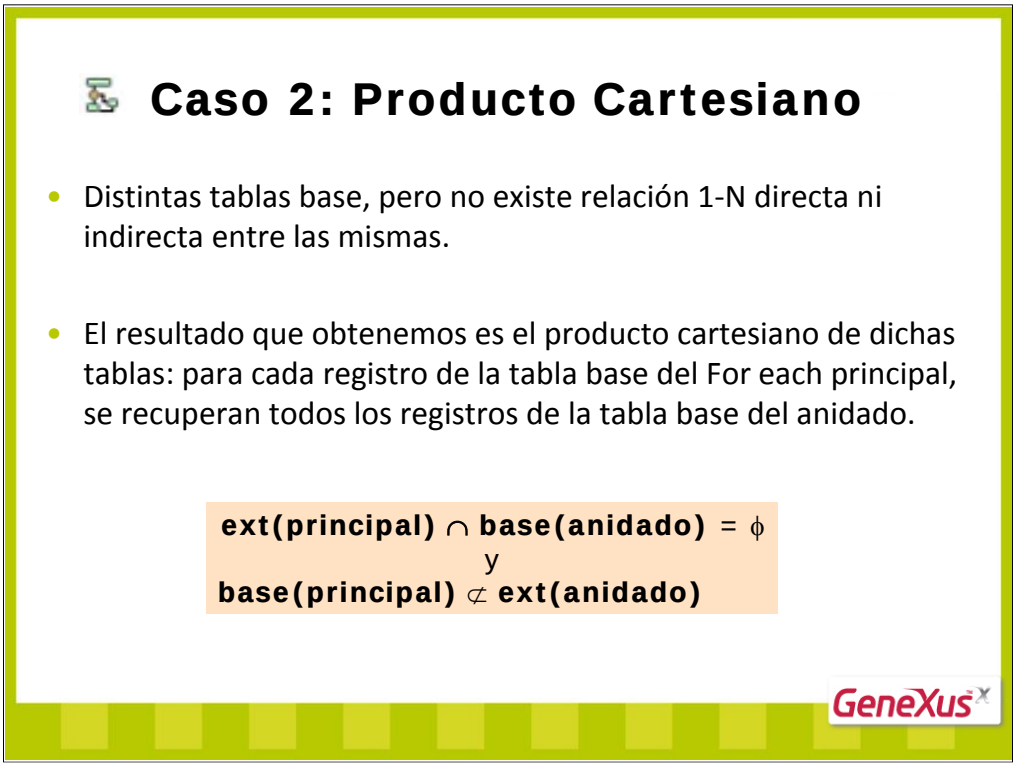
Hilando fino, este es un caso un poco más complejo que el anterior, porque si bien la tabla extendida del For each principal no tiene intersección con la tabla base del anidado (ext(COUNTRY) ∩ INVOICE = ϕ), sin embargo sí existe una relación 1-N indirecta, y GeneXus la encuentra.

En este caso, la **tabla base del For each principal está incluida en la extendida del anidado** (COUNTRY ⊂ ext(INVOICE)), por lo que hay una relación 1-N indirecta.

Por este motivo, no necesitamos especificar cláusula where en el For each interno para filtrar las facturas del país del For each principal.

Esto puede verse claramente en el listado de navegación, que mostrará el filtro:
“CountryId = CountryId” para el segundo For each, pero esta vez como ‘Constraint’ dado que no puede optimizar la recorrida.

Puede probar el lector este caso en GeneXus y estudiar detenidamente el listado de navegación resultante.



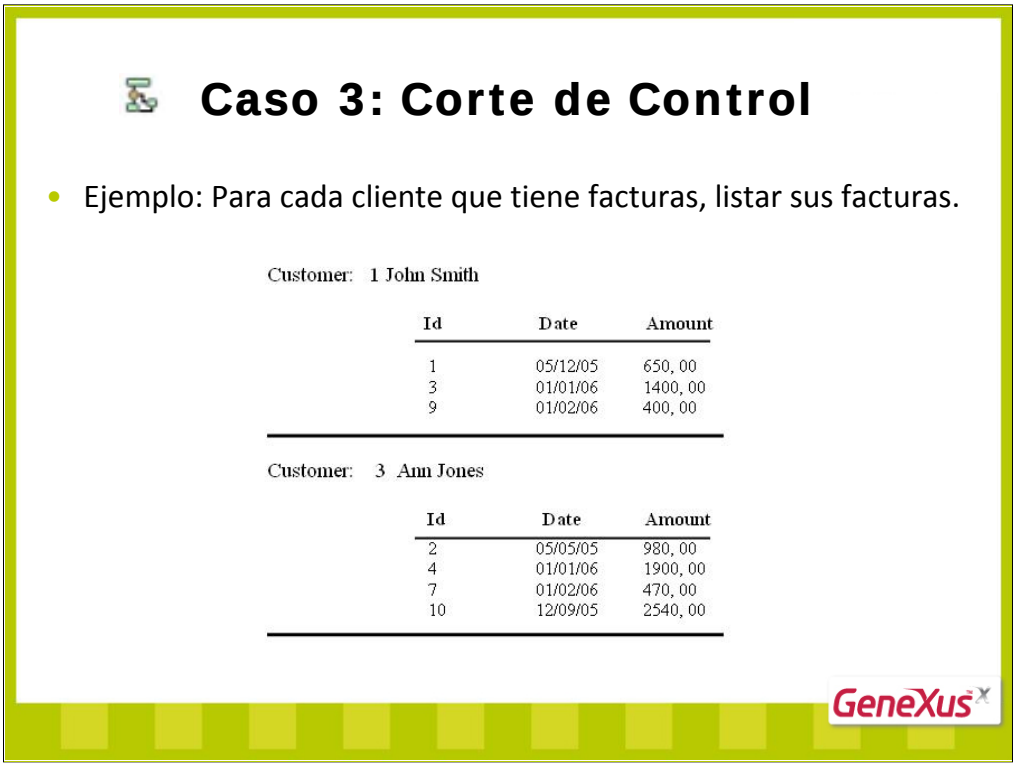
En este caso GeneXus no logra encontrar una relación 1-N directa o indirecta entre las tablas y por lo tanto no aplica filtros implícitos a los registros del For each anidado, vale decir, realiza un producto cartesiano entre las tablas.

El caso se da cuando:
• **ext**(For each principal) ∩ **base**(For each anidado) = ϕ y
• **base**(For each principal) ⊄ **ext**(For each anidado)

Para cada registro de la tabla base del For each principal se recorre toda la tabla base del For each anidado.

Por ejemplo, si la tabla base de un For each fuera COUNTRY y la del anidado PRODUCT, evidentemente no existirá relación y se hará un producto cartesiano y se recorrerá para cada país, todos los productos.

Por supuesto que el programador podrá establecer filtros sobre los datos a recuperar, pero éstos ya no serán condiciones implícitas inferidas por GeneXus, sino especificadas explícitamente por el programador.



En el ejemplo que habíamos visto antes, del listado de clientes y sus facturas, ¿qué ocurre si un cliente no tiene facturas?

Como la tabla base del For each principal es CUSTOMER, el cliente sale impreso antes de saberse si tiene o no facturas.

Si no deseamos que esto ocurra, es decir, que salgan listados clientes que no tengan facturas, entonces la solución es acceder únicamente a las facturas, pues si un cliente está en esta tabla, ¡es porque está en una factura!.

Pero para poder agrupar las facturas por cliente, de forma tal de poder desplegarlas de ese modo, debemos recorrer la tabla INVOICE ordenada por *CustomerId*. De esta forma procesaremos la información de un cliente, y luego pasaremos al siguiente, para procesar su información, y así sucesivamente.

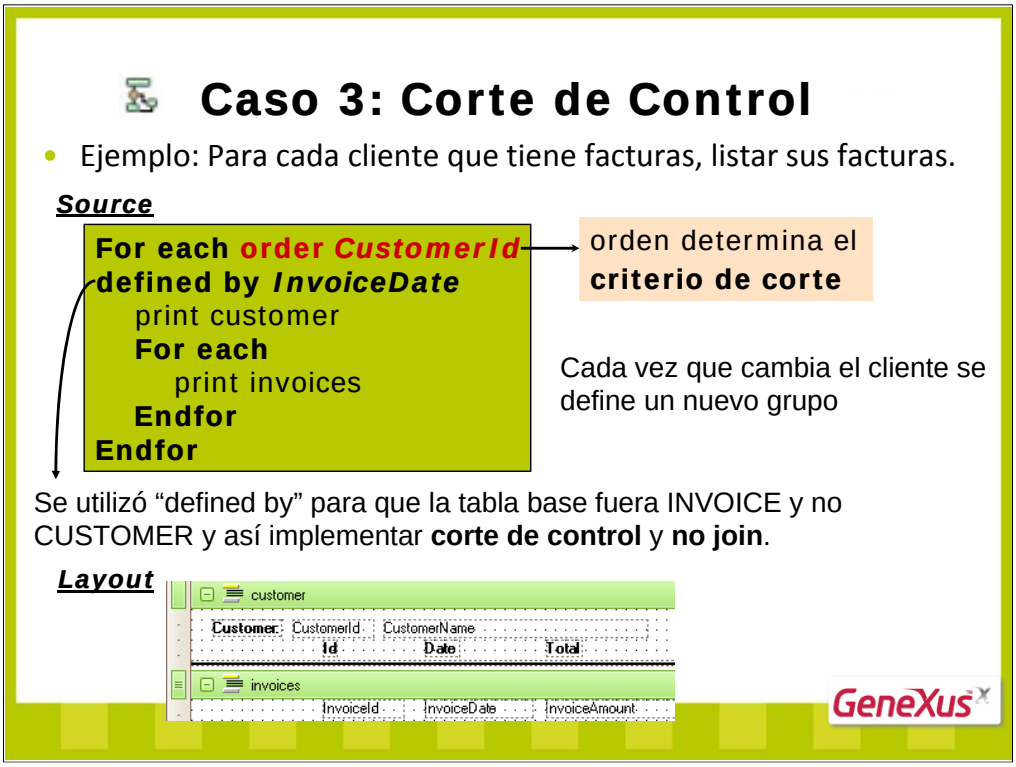
Si imaginamos un puntero que se va desplazando secuencialmente por la tabla INVOICE, podemos escribir el pseudocódigo de nuestro procedimiento como sigue:

1. Para el registro apuntado, **retener** el valor del atributo de corte o agrupamiento, *CustomerId.*

2. Acceder a la tabla CUSTOMER (que está en la extendida de INVOICE) para recuperar el *CustomerName* e imprimirlo junto con el *CustomerId* (“print customer”)

3. Mientras el valor de *CustomerId* del **registro apuntado** coincida con el valor **retenido** en el paso 1 (aquí se procesan todas las facturas del cliente)
    a. Imprimir *InvoiceId*, *InvoiceDate* e *InvoiceAmount* del registro apuntado. (“print invoice”)
    b. Avanzar el puntero al siguiente registro y volver al paso 3.

4. Volver al paso 1. (cuando se llegó a este punto, es o bien porque se llegó al fin de tabla o bien se cambió de cliente).



GeneXus brinda una forma de implementar lo anterior de una forma absolutamente sencilla.

El pseudocódigo visto en la página anterior se implementa en GeneXus con el par de For eachs anidados que se muestran arriba.

Si se compara este código con el que vimos unas páginas atrás para el caso de join, vemos que existen solamente dos diferencias: la cláusula order que aparece en este código, en conjunción con el defined by. Con solo esos dos cambios al listado original, cambiamos radicalmente el comportamiento, puesto que en este caso solamente se listarán los clientes si tienen facturas.

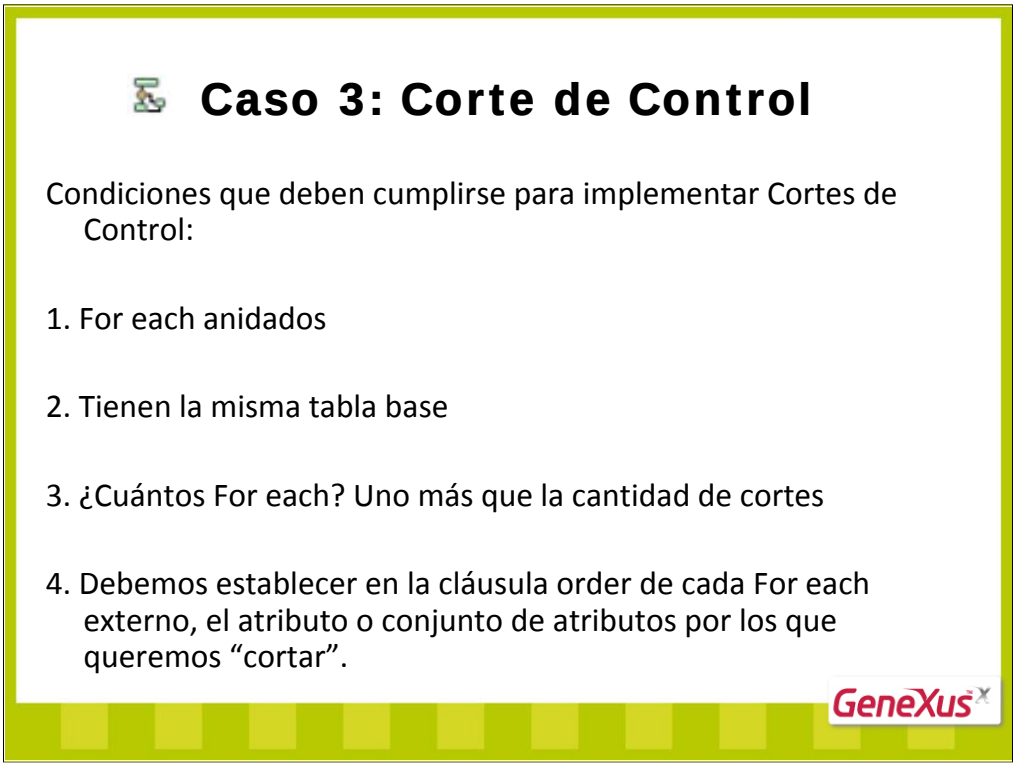
En este caso, ambas cláusulas (order y defined by) son indispensables para que este procedimiento funcione del modo que queremos. Si agregamos solo una de ellas, pero no la otra, el resultado será otro.

No en toda implementación de un corte de control deberá especificarse una cláusula defined by en el For each principal, mas sí una cláusula order.

La **cláusula order** es indispensable, porque es la que especifica por qué atributo o conjunto de atributos se realizará el corte (o agrupamiento). Es decir, especifica esa información común al grupo, que se procesará una sola vez (dentro el código del For each externo).

La **cláusula defined by** no es indispensable en todos lo casos. En este sí lo fue, porque de no especificarla, GeneXus determinaría como tabla base del For each principal CUSTOMER, que no es lo que queremos (pues no queremos implementar un join, cosa que ya hicimos antes, sino un corte de control, para solo recorrer la tabla INVOICE).

Pero también podríamos haber utilizado otra solución para modificar la tabla base del For each principal: utilizar en vez del defined by el comando **print if detail** dentro del cuerpo del primer For each (este comando le dice a GeneXus que tome como tabla base del For each, la que determine para el anidado).



Un corte de control es bien simple de implementar y puede hacerse siguiendo las consideraciones anteriores.
En el order del For each más externo, debemos mencionar el “primer” atributo de corte, en el order del segundo For each debemos mencionar el “segundo” atributo de corte, y así sucesivamente. No es obligación mencionar atributo/s en el order del For each más interno (en todos los demás For each sí lo es).

Corresponde al caso en el que nos interesa trabajar con la información de una tabla, pero agrupada por algún atributo o conjunto de atributos.

Los cortes de control pueden ser simples, dobles, triples, etc.

A continuación veremos un ejemplo de un corte de control doble.

Ejemplo: Corte de control doble

Supongamos que queremos como antes listar los clientes y sus facturas, pero queremos agrupar las facturas de cada cliente por fecha. Es decir, queremos mostrar, para cada cliente, para cada fecha, las facturas existentes.

Ejemplo:

Customer: 1 Juan Pérez

Date: 12/05/05

Invoice	Total
1	15

Date: 01/01/06

Invoice	Total
9	35
3	30

Customer: 3 María Donoso

Date: 06/06/05

Invoice	Total
2	20

Date: 12/08/05

Invoice	Total
4	40
8	15

Date: 02/02/06

Invoice	Total
7	20

…

Como ahora queremos agrupar por cliente, y dentro de ese grupo por fecha de factura, necesitamos tres For eachs anidados:

```
For each order CustomerId
defined by InvoiceDate
    print customer
    For each order InvoiceDate
        print date
        For each
            print invoice
        Endfor
    Endfor
Endfor
```

Como ejercicio, sigamos todos los pasos que realiza GeneXus para inferir el comportamiento del procedimiento.

1. Determinación de la tabla base de cada For each

Como siempre, para determinar las tablas base de For eachs anidados, se empieza de afuera hacia adentro, determinando la de cada For each, sin tomar en cuenta los atributos de los For eachs internos al que se está considerando.

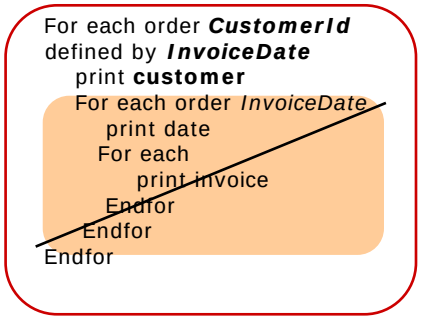


Tabla base 1er. For each
Solo intervienen los atributos de los lugares señalados en negrita.

Mínima tabla extendida que los contiene: ext(**INVOICE**)

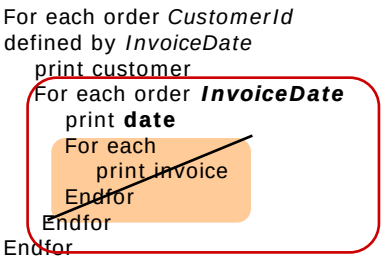


Tabla base 2do. For each
Intervienen los atributos de los lugares señalados en negrita y como todos ellos están incluidos en la extendida del 1er. For each, entonces se determina la misma tabla base: **INVOICE**

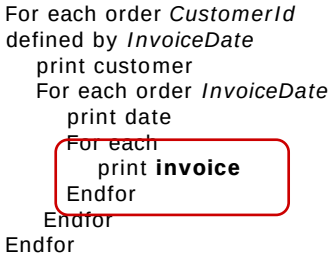


Tabla base 3er. For each
Intervienen los atributos de los lugares señalados en negrita y como todos ellos están incluidos en la extendida del 2do. For each, entonces se determina la misma tabla base: **INVOICE**

2. Determinación de la navegación

Luego de determinadas las tablas base, GeneXus determina la navegación. Como en este caso son tres For eachs sobre la misma tabla base, se trata de un doble corte de control.

Podemos pensar que cuando hablamos de corte de control, ya sea simple, doble, triple, cuádruple, etc., tenemos un solo puntero, que se utiliza para avanzar en los registros de la tabla base.

Recordemos que la cláusula order es fundamental para establecer el criterio de corte en cada par de For eachs.

Como en nuestro caso queremos agrupar por *CustomerId* y luego, para todas las facturas con ese cliente, agrupar por *InvoiceDate*, entonces tendremos que ordenar el primer For each por *CustomerId* y el inmediatamente anidado por *InvoiceDate*.

En el ejemplo estamos diciendo que:

Mientras no se alcance el fin de tabla
Imprimir los datos del cliente de la factura actual
Mientras no cambie el cliente
Imprimir la fecha de la factura actual
Mientras no cambie la fecha
Imprimir los datos de la factura actual (nro y total)
Avanzar el puntero al siguiente registro

Recomendamos al lector implementar en GeneXus este procedimiento y observar detenidamente el listado de navegación.

Verá que GeneXus elige un único orden, para el que no tiene un índice creado: el compuesto por la concatenación de los órdenes de cada For each con cláusula order. Esto resulta evidente si pensamos en términos de un único puntero que se va desplazando por la tabla base.

Una vez determindo el Order del Corte de Control se optimizará, buscando el mejor índice teniendo en cuenta las condiciones explícitas o implícitas del nivel.

Resumen: Determinación general de las tablas base

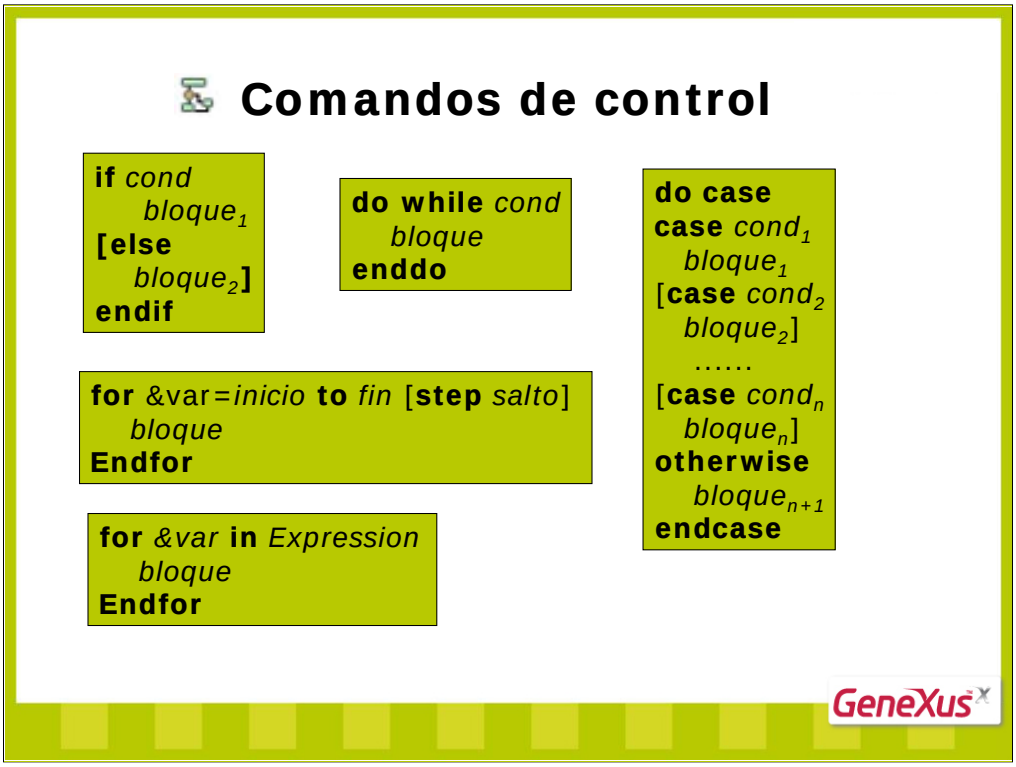
Se procede en forma ordenada, determinando cada vez la tabla base de un nivel de anidación, yendo de afuera hacia adentro: primero se determina la tabla base del For each más externo, luego del que está anidado a éste y así sucesivamente.

Determinación tabla base del For each externo
Se determina a partir de los atributos que aparecen dentro de ese For each: cláusulas **order**, **where**, **defined by** y **cuerpo del For each**, exceptuando los atributos que estén dentro del For each anidado. No participan los atributos que estén dentro del When none, en caso de que el For each principal tenga esta cláusula. Al igual que en el caso de un For each simple, se encuentra la mínima tabla extendida que contenga los atributos mencionados.

Determinación tabla base del For each anidado
Podemos vernos tentados a pensar que por analogía lo que se debería hacer es extraer los atributos del For each anidado, y hacer lo mismo que antes, es decir, encontrar la mínima tabla extendida que contenga a esos atributos, como si se tratase de un For each independiente. ¡Pero no son For eachs independientes!

Para el For each anidado, GeneXus se fija primeramente si los atributos utilizados dentro del cuerpo del mismo están incluidos o no dentro de la tabla extendida previamente determinada. En caso afirmativo, GeneXus determina que la tabla base del For each anidado será la misma que la del For each principal (y será un caso de corte de control).

En caso contrario, se determina de la siguiente manera: busca la mínima tabla extendida que contenga a todos los atributos del For each anidado, tratando de encontrar aquella que tenga alguna relación con la tabla base del for each principal.
Si no encuentra una tabla extendida que contenga a todos los atributos del For each anidado y que además esté relacionada, se queda con la tabla extendida mínima que contenga a los atributos aunque no tenga relación.



Los comandos introducidos son similares a los existentes en los lenguajes de programación imperativa conocidos, por lo que no incluimos documentación de este tema. Puede encontrarla en el curso no presencial o en el Help de GeneXus. Los dos últimos, no obstante, incorporan algunos elementos interesantes sobre el manejo de arrays y de colecciones[1] en GeneXus, por lo que mostraremos algunos ejemplos.

For to step:
• *inicio*, *fin* son expresiones numéricas
• *salto* es una constante numérica
• *var* es alguna variable numérica
• *bloque* es una sucesión de comandos válidos del lenguaje

Permite iterar una cierta cantidad de veces: desde el valor ***inicio*** que toma la variable ***&var*** cuando se ingresa al bucle, hasta el valor ***fin*** que tomará la misma luego de cierta cantidad de iteraciones. De iteración en iteración la variable ***&var*** se va incrementando automáticamente en una cantidad igual a ***salto***. El valor por defecto de ***salto*** es 1, por lo que si no se especifica la **cláusula step** el incremento de la variable será de uno en uno. El valor de ***salto*** puede ser negativo y en ese caso se irá decrementando la variable de iteración en iteración.

Ejemplo

```
For &i = 1 to 5
   &ok = PInvoicing.udp( &month )
Endfor
```

---

[1] Las colecciones representan listas de largo variable. Se verán cuando estudiemos el tipo de datos estructurado (SDT).

For in Expression:
• *Expression* es cualquier expresión cuyo valor sea una colección, vector o matriz.
• *var* es una variable que debe tener el mismo tipo de datos que los elementos de Expression.

Esta estructura de programación permite recorrer con menos código una colección o una variable array de una o más dimensiones. Se almacena en la variable *&var* los valores de cada elemento de la colección, array o matriz.

Para el caso de **arrays de una dimensión:**

```
for &var in &varArray()
   … // code
Endfor
```

el código se expande (es equivalente) a:

```
    &x = 1
        do while &x <= rows(&array())
                &var = &Array(&x)
                bloque
                &x += 1
        enddo
```

En el caso de **dos dimensiones:**

```
for &var in &varMatrix()
   … // code
Endfor
```

el comando se expande (es equivalente) a:

```
        &x = 1
        do while &x <= rows( &array() )
           &y = 1
           do while &y <= cols( &array() )
              &var = &array( &x, &y )
              bloque
              &y += 1
           enddo
           &x += 1
        enddo
```

También puede ser una variable colección de cualquier tipo, incluyendo una variable con la propiedad Collection en 'False' pero de un tipo de datos 'SDT collection' .

```
for &var in &collection
...
endfor
```

La expresión también puede no ser una variable, por ejemplo en el caso del resultado de un DataProvider o de un Procedure:

```
for &var in DataProvider(parms)
...
Endfor

for &var in Procedure(parms)
...
endfor
```

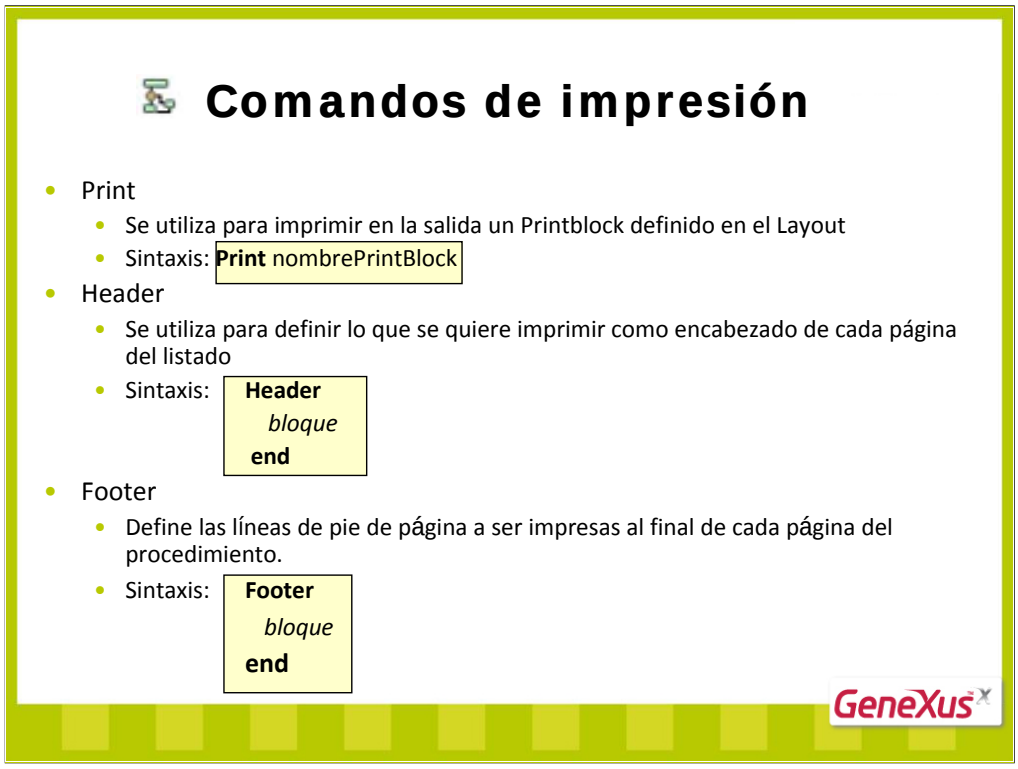
Consideraciones:

• No es posible modificar los valores de la colección, array o matriz en la recorrida. Esto significa que cambios en el valor de *&var* en el alcance de la estructura, no afectan al correspondiente valor de la &collection o del *&array(&x),* o de *&array(&x, &y)*).

• No es posible obtener la posición de la colección/array/matriz durante la recorrida, para esto es necesario definir una variable que actúe como contador.

•Estas estructuras pueden anidarse para recorrer varios arrays, matrices o colecciones. Esto incluye el caso de que se invoque a una Subrutina que también tiene un For In Expression.

•Al igual que en un For Each o un Do While, es posible incluir comando que “corten” la recorrida, como Exit o Return.



Aquí veremos algunos comandos de impresión, que permiten diseñar la salida del procedimiento.

Print

donde *nombrePrintBlock* es el identificador de un Printblock del Layout. Si no existe en el Layout un Printblock con ese nombre, al intentar salvar el objeto se desplegará un mensaje de error informando sobre esta situación.

De esta forma se implementa la impresión en la salida de los Printblocks del Layout.

Cuando el Printblock que se quiere imprimir no contiene atributos, entonces este comando se puede utilizar en cualquier lugar del Source.

Los atributos indican acceso a la base de datos y este acceso no puede realizarse en cualquier lado, sino únicamente dentro del comando específico para ello, esto es, el comando For each.

Por tanto no es correcto escribir el comando print fuera de un “For each” si el Printblock que se está queriendo imprimir contiene atributos.

La única excepción a esta regla se produce cuando los atributos del Printblock están incluidos entre los parámetros recibidos por el procedimiento, pues en este caso no es necesario acceder a la base de datos para recuperar sus valores, dado que ya vienen instanciados.

Header

Aquí se define lo que se quiere imprimir como encabezado de cada página del listado.
Este encabezado es opcional. Si no se especifica, entonces las páginas del listado no tendrán encabezado.

Ejemplo: En el procedimiento en el que queríamos imprimir un listado con el código, nombre y país de cada uno de los clientes de nuestro sistema, si queremos que en cada página del listado aparezca el encabezado: “CUSTOMERS REPORT”, entonces alcanza con escribir en el Source:

```
Header
   Print title
End
```

Donde “title” es el nombre de un Printblock del Layout que contiene este texto.

También podríamos haber escrito directamente: Print title al comienzo del Source, pero en este caso, si el listado tiene varias páginas, solo saldrá impreso este texto en la primera. En el otro caso, saldrá en cada una de las páginas, como encabezado.

Footer
Define las líneas de pie de página a ser impresas al final de cada página del procedimiento.

Los comandos del bloque de código son ejecutados cuando se llega al final de una página.

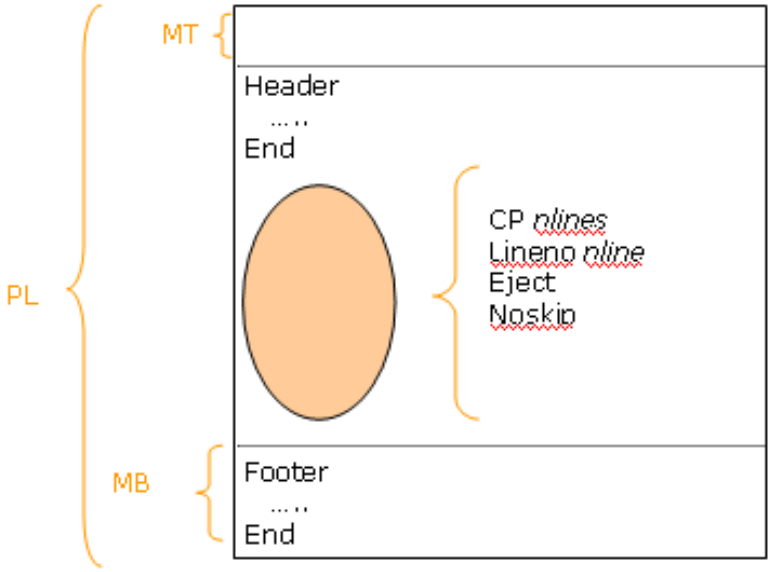
Ejemplo:

```
Footer
   print endOfPageText
end
```

donde endOfPageText es el nombre de un Printblock del Layout

Diseño de la salida

Existen algunos comandos para diseñar la salida del procedimiento. Presentamos aquí algunos a los efectos de la documentación.



MT *nlines:* nlines es el número de línea en el que se quiere empezar a imprimir el listado. En caso de no especificarse un valor se asume el valor por defecto que es 0.

MB *nlines:* nlínes es el número de líneas que se desea dejar como margen inferior.
En caso de no especificarse un valor se asume el valor por defecto que es 6.

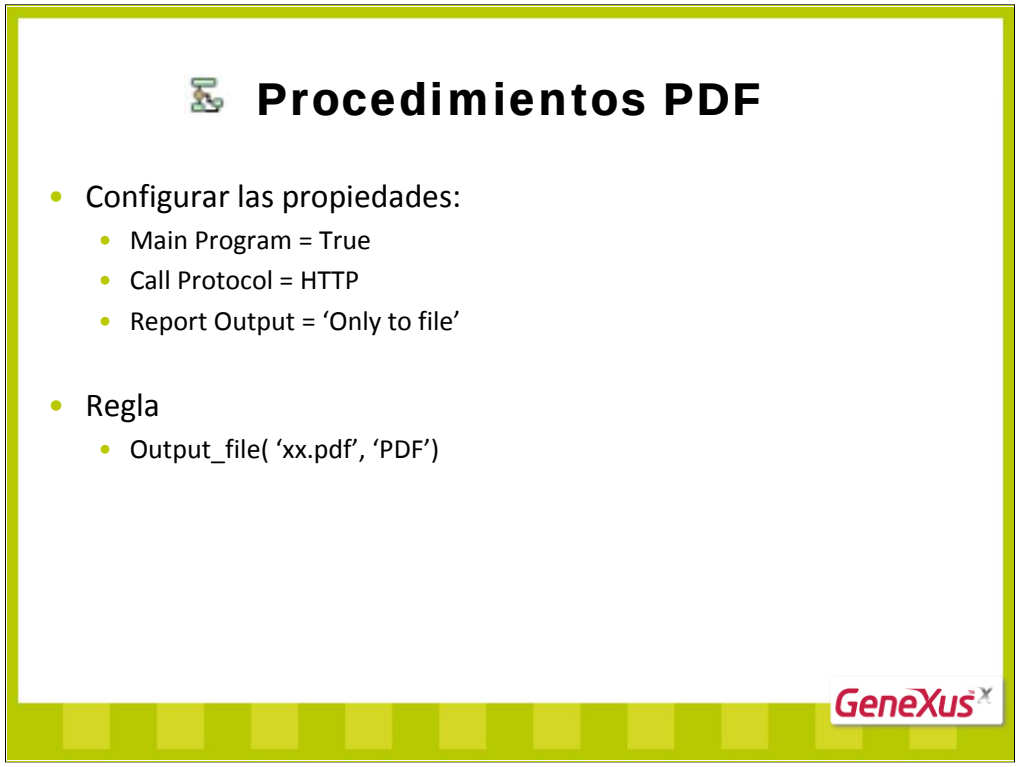
PL *nlines:* Setea el largo de página. El número de líneas que será impreso es el número especificado menos el margen de abajo (valor por defecto es 6). Ej: PL 66
Setea el largo de página a 66 líneas, aunque sólo 60 líneas serán impresas en el form, con un margen inferior de 6 líneas.

CP *nlines:* Si queda en la página actual un número de líneas mayor o igual al número especificado, continúa imprimiendo en la misma página. De lo contrario, pasa a imprimir en la próxima página (fuerza a un salto de página).

Lineno *nline*: Define el número de línea donde va a ser impresa la siguiente línea. Si el número de línea actual es mayor al número especificado, entonces, la línea será impresa en la próxima página. El conteo de líneas comienza en la línea 0.

Eject: Fuerza a un salto de página.

Noskip: Tiene que estar inmediatamente después de un Printblock. Si el comando se encuentra entre dos líneas, este comando las imprimirá en la misma línea.



En Web los listados solamente pueden ser PDF y se deben configurar las propiedades y regla anteriores para que funcionen.

Definición de un objeto como main

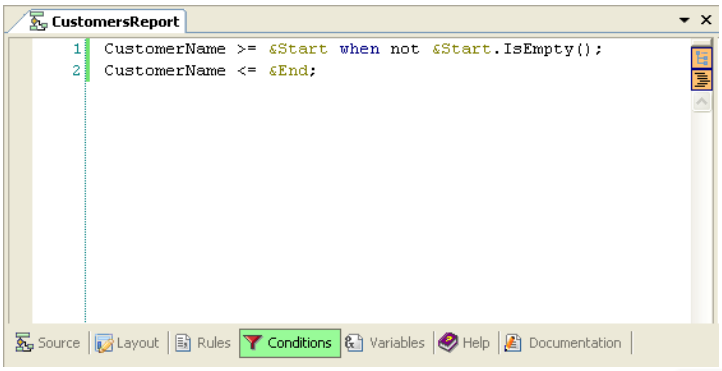
Al definir que un objeto **es main** (en este caso un procedimiento, pero podría ser una transacción, web panel, etc.), GeneXus genera un programa ejecutable con la lógica del objeto mismo y la de todos los objetos invocados directa o indirectamente por él.

El programa ejecutable generado se podrá compilar y ejecutar de forma independiente, es decir, al seleccionar **Build / Run** se verá como programa independiente del Developer Menu y podrá compilarse y ejecutarse.

La definición de un objeto como main se realiza editando las propiedades del objeto, y configurando la propiedad *Main program* del mismo con valor *True*.

Condiciones

- Permiten especificar condiciones globales que deberán cumplir los datos a ser recuperados.
- Ejemplo:



GeneXus

En esta sección se permiten establecer condiciones que deben cumplir los datos para ser recuperados.

Una “condición” es equivalente a la cláusula “where” del comando For each (incluso tiene la misma sintaxis) con una salvedad: mientras que la cláusula “where” está ligada a **un** For each específico: aquel al que pertenece, las “condiciones” están ligadas a **todos** los For eachs del Source en los que tenga sentido aplicarlas.

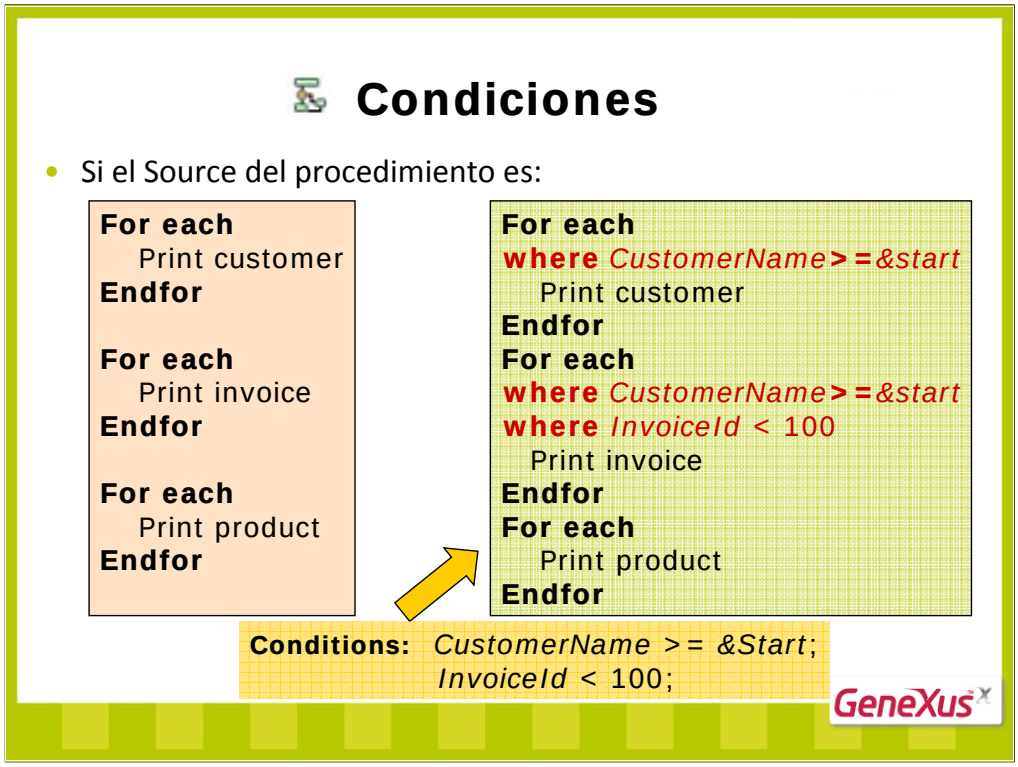
¿Y para qué For eachs tiene sentido aplicarlas?
Las condiciones generalmente involucran atributos. Si en la tabla extendida de un For each se encuentran los atributos que intervienen en una “condición”, entonces la misma se aplicará a este For each para filtrar los datos quedándose únicamente con aquellos que satisfagan tal condición.

En el listado de navegación del procedimiento se indican los For eachs a los que se aplica cada condición de las especificadas en la sección “Conditions”.

Si en el Source del procedimiento “PrintCustomers” tenemos:

```
For each
   Print customer
Endfor
```

Entonces al especificar las condiciones que se muestran en la transparencia, estaremos filtrando los clientes, de acuerdo a las condiciones (es equivalente a tener las condiciones como “where”).



donde:
• "customer" es un Printblock que contiene los atributos *CustomerId*, *CustomerName*, *CountryName*
• "invoice" es un Printblock que contiene los atributos *InvoiceId*, *CustomerId*, *CustomerName*, *InvoiceDate*, *InvoiceAmount*
• "product" es un Printblock que contiene los atributos *ProductId*, *ProductDescription*, *ProductStock*

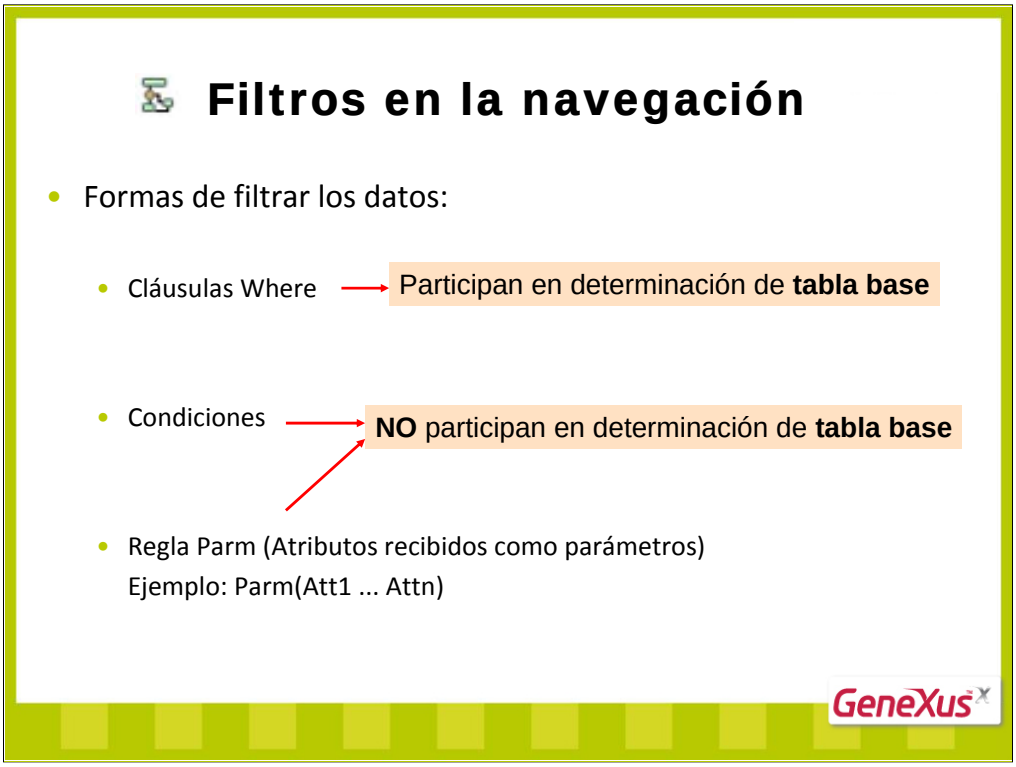
Si el procedimiento anterior tiene definidas las condiciones mostradas arriba, el procedimiento será equivalente a uno sin "condiciones" y con el Source que se muestra a la derecha.

Observemos que en este caso las condiciones se traducen en cláusulas where pero que se aplican solo a los For eachs para los que tiene sentido aplicarlas. En la tabla extendida del último For each no se trabaja con nombre de cliente, ni con identificador de factura. No tiene sentido aplicar las condiciones en este For each ya que no existe ninguna relación.
En el primero, solo tiene sentido aplicar la que involucra a *CustomerName* y no la otra.

Observación
Los atributos involucrados en las "condiciones" no participarán en la determinación de las tablas base de los For eachs del Source (a diferencia de los filtros que se especifican mediante cláusulas where).

Es decir, las tablas base de los For eachs que aparezcan en el Source se determinan sin mirar las "condiciones". Una vez determinadas, recién en ese momento las condiciones son examinadas para determinar a cuáles For eachs se aplicarán y a cuáles no.

Lo mismo ocurre con los atributos que se reciban como parámetro. Aplicarán como filtro global por igualdad para los For eachs en los que tenga sentido, pero no participarán en la determinación de las tablas base.



Resumimos aquí las distintas formas de filtrar en un procedimiento la información a recuperar de la base de datos:

a. cláusulas where
b. Condiciones
c. parm(*att*, ..., *att*)

Estudiemos las diferencias y similitudes entre ellas.

1. Las **cláusulas where** aplican exclusivamente al For each en el que se encuentran, mientras que los filtros especificados como **condiciones** o los que quedan determinados por los atributos en la **regla parm** son globales, es decir, aplicarán a todos los For eachs del Source en los que tenga sentido aplicarlos.

2. Los filtros que quedan determinados al recibir atributos en la **regla parm** son filtros por igualdad, es decir, para los For eachs en los que tenga sentido aplicarlos, se instanciarán únicamente los registros que tengan el mismo valor que el recibido por parámetro. En cambio, los filtros especificados en las **cláusulas where** de un For each o en las **condiciones** pueden ser expresiones booleanas cualesquiera, incluso compuestas.

3. Mientras que los atributos que aparecen en las cláusulas where participan en la determinación de la tabla base del For each donde se encuentran, los que aparecen en las condiciones o en la regla parm no lo hacen. Recién entran en juego LUEGO de determinadas las tablas base de los For eachs del Source.