DISCUSIÓN DE PREGUNTAS

para Instructor

Departamento de Capacitación - GeneXus

Junio 2023

CONTENIDO

PREPARACIÓN PARA EXAMEN DE INSTRUCTOR GENEXUS

Quien se presenta al examen de Instructor GeneXus es porque ha aprobado previamente la certificación de Analista Senior GeneXus, por lo que ya se lo ha evaluado técnicamente.

INTRODUCCIÓN

Se presentan a continuación preguntas de algunos temas importantes, con respuestas que pretenden ampliar sus conocimientos o su capacidad de integración y articulación de lo que ya sabe. En algunos casos se explican cuestiones que en el curso GeneXus no se exponen. No se le preguntarán en el examen, pero entendimos que ayudaban a comprender mejor la lógica de GeneXus, y por eso las explicamos aquí.

Los que encontrará serán tan solo algunos ejemplos de algunos de los temas importantes (no de todos), para que usted pueda hacerse una idea de qué tipo de razonamientos le pediremos en su examen.

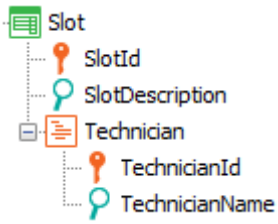
Nota: Las capturas de pantallas se realizaron en GeneXus 16, pueden sufrir variaciones en versiones posteriores.

PREGUNTA 1 - DISEÑO

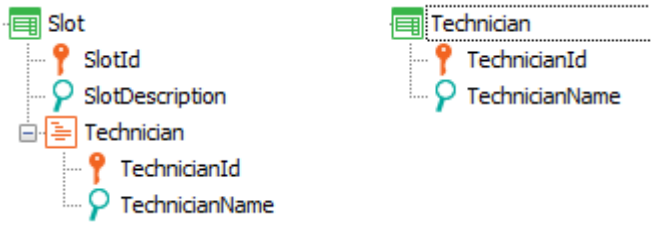
Se tiene una aplicación GeneXus para un casino. Cuenta con transacciones para registrar las máquinas de juegos (slots) así como los técnicos encargados de repararlos.

Sabiendo que una máquina de juegos (Slot) puede ser reparada por varios técnicos (Technician), y que un mismo técnico puede reparar varias máquinas, determina la opción correcta.

a. Una única transacción, Slot, con dos niveles:



b. Dos transacciones: Slot y Technician:



c. Dos transacciones: Slot y Technician:

d. Dos transacciones: Slot y Tecnhician:

e. Ninguna de las anteriores

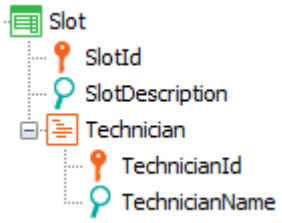
RESPUESTA

En el requerimiento nos pedían una relación N a N entre las entidades de la realidad Slot y Technician y es el diseño b) el único de los propuestos que la representa.

Obsérvese que las soluciones a) y c) son equivalentes y representan una relación diferente de la pedida entre las entidades. ¿Cuál? Una relación 1 a N, en la que Technician no existiría por sí solo, de manera independiente de los Slots; es decir, Technician sería una entidad débil, que para existir depende de la existencia del Slot al que se vincula. Esto se evidencia en el hecho de que si observamos la tabla que se crea para almacenar su información, ésta tiene clave primaria compuesta por SlotId y TechnicianId. Entonces, ¿cómo se ingresaría al sistema, para estas alternativas, tanto a) como c), un técnico sin asociarlo en la misma operación a un slot? Recordemos que es imposible dejar una clave primaria sin valor (porque en ese caso, ¿cómo identificaríamos al registro de la tabla?). SlotId es parte de la clave primaria de la tabla que registra a los técnicos, por lo que sería imposible ingresar un técnico sin dar valor al atributo SlotiId.

Y si esto sucede, es decir, si no es posible ingresar un técnico sin asociarlo a un slot, ¿cómo haríamos para que ese técnico esté asociado, además, a otros slots?

Por ejemplo, para la solución a):



editando en ejecución la información del Slot de Id 3 y descripción “Super Wizard” ingresamos en su grid al técnico de Id 1, David Roberts y confirmamos.

¿Cómo hacemos para asociarle a ese mismo técnico, David Roberts, otro slot, por ejemplo el 6, si el técnico existe únicamente como vinculado al slot 3?

Podría uno verse inclinado a decir que es muy fácil: simplemente editando ahora la información del Slot 6, y agregando en su grid una línea con TechnicianId = 1 y TechnicianName = David Roberts.

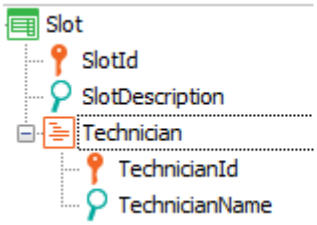
Pero observemos que nada garantiza que ya no haya un TechnicianId = 1 en ese grid con otro nombre de técnico, o incluso, sin existir previamente, que podamos asignarle otro nombre al que ingresamos para ese id 1. Porque TechnicianName es un atributo físico de esa tabla, no es uno inferido, como en el caso de la solución b).

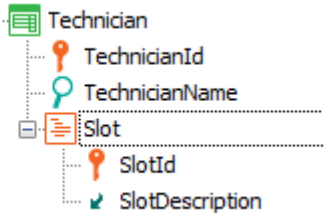
Podríamos, de todos modos, utilizar este diseño (que es 1 a N) asegurando “a mano” que no se utilice el número 1 de técnico más que para David Roberts. Pero lo que aquí se demuestra es que

no estamos diseñando correctamente nuestra realidad, porque de hacerlo bien, el sistema mismo no nos permitirá equivocarnos, como es el caso de la solución correcta, la b), donde el técnico se identifica únicamente por su Id y luego a los Slots se les indica qué técnicos (por su id) pueden reparar la máquina.

Por otro lado, la solución d) no puede ser, porque ambas transacciones determinan la misma tabla física (el orden en que aparecen los atributos no altera la constitución de la tabla). Aquí los slots y técnicos solo existen vinculados. Ninguno existe independientemente del otro.

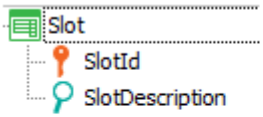
En cambio, qué pasaría si en su lugar se hubieran creado estas otras transacciones:

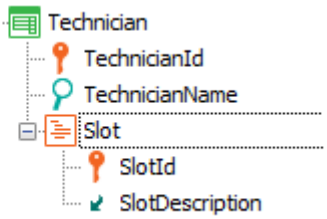




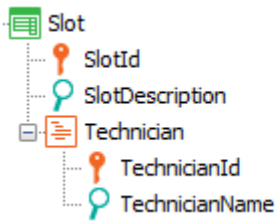
¿Qué tablas creará GeneXus a partir de ellas? Aunque a primera vista pueda parecer una solución errónea, y en parte no es la aconsejable, las tablas que se crearán serán las correctas, idénticas a la de la solución b). Observemos que la tabla correspondiente al segundo nivel de Slot será exactamente la misma que la del segundo nivel de Technician, puesto que la clave primaria de ambas es la misma, compuesta por ambos atributos, SlotId y TechnicianId. El problema de este diseño será solamente operativo: para poder ingresar los técnicos para un slot, antes habrá que haber creado esos técnicos mediante la transacción Technician, dejando los slots sin completar.

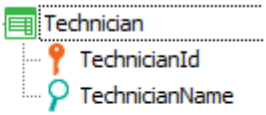
Por supuesto, una solución como:





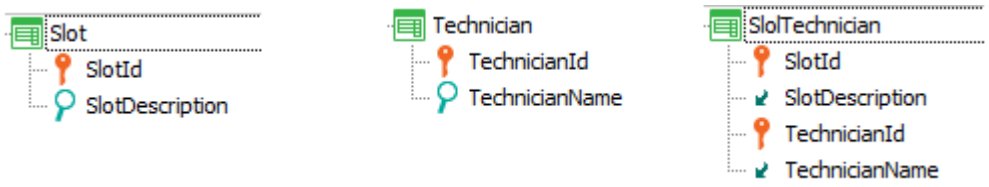
En cuanto a la representación N a N es completamente equivalente a la b) propuesta:





La única diferencia viene dada por la operativa: ¿qué es más conveniente? ¿Ingresar todos los técnicos sin vincularlos primeramente a los Slots y luego para cada slot ingresado asignar sus técnicos, o viceversa?

Las tablas generadas son exactamente las mismas en ambas soluciones. O incluso, se podría hasta llegar a tener esta otra solución, completamente equivalente en lo que refiere a la relación N a N:

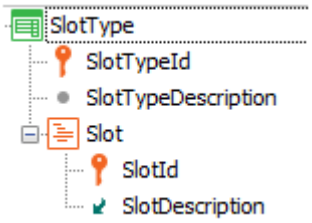


PREGUNTA 2 - DISEÑO

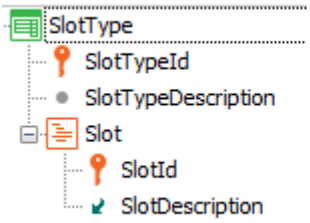
Se tiene una aplicación GeneXus para un casino. Ésta cuenta con transacciones para registrar las máquinas de juegos (slots) así como los tipos de máquinas existentes.

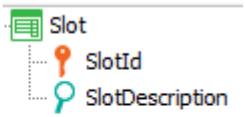
Sabiendo que cada máquina (Slot) corresponde a un tipo determinado (SlotType) y solo uno, y que puede haber muchos slots del mismo tipo, determine la opción correcta para el diseño de estas transacciones.

a. Una única transacción:

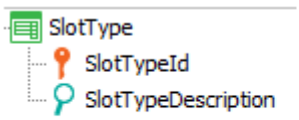


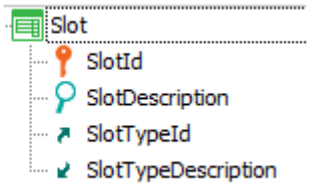
b. Dos transacciones:



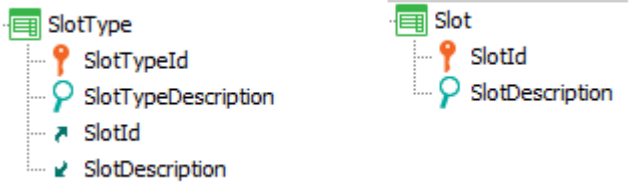


c. Dos transacciones:





d. Dos transacciones:



e. Ninguna de las anteriores.

RESPUESTA

Se nos está pidiendo implementar una relación 1-N fuerte (este dato lo deducimos de nuestro conocimiento de la realidad, pues no se ha explicitado en la letra del ejercicio), por lo que la respuesta correcta es la c).

Aquí, en la solución c), cada "slot" tendrá un único "tipo" debido a que en la transacción Slot el atributo SlotTypeId se encuentra en el mismo nivel en el que se encuentra el identificador, SlotId. De esta manera, así como un slot dado, identificado por SlotId, tendrá una única descripción, SlotDescription, también tendrá un único SlotTypeId (distinto sería si este atributo también tuviera el símbolo de llave). Por otro lado, nada impide que otro slot tenga el mismo valor para SlotTypeId (pues el atributo no es clave primaria ni candidata en esta transacción). Se cumple, entonces, el requerimiento de la letra: un slot solamente tiene un tipo y varios slots pueden tener ese mismo tipo.

¿Por qué la opción a) no es correcta, dado que también representa una relación 1-N? Porque en esta representación la entidad Slot sería débil, es decir, no puede identificarse solamente con su Id, sino que requiere del Id del tipo. Estaríamos diciendo que los Slots sólo tienen existencia en la medida en que existe un "tipo de slot" del cual dependen. No será posible ingresar un Slot sin haber especificado previamente su SlotTypeId. Dicho de otro modo, bajo esta representación, deduciríamos que la entidad del mundo real Slot, depende absolutamente para existir del "tipo de Slot", lo que contradice la realidad que queremos modelar. En ella, el "Tipo de Slot" y el "Slot" son entidades relacionadas pero de existencia propia. Cada una se identifica más allá de la otra. Eventualmente incluso, si se admitieran nulos para SlotTypeId en Slot (en la solución correcta, la c), podrían ingresarse "slots" sin ingresar su "tipo".

La opción b) claramente representa una relación N-N (en este caso para cada "tipo de slot", pueden ingresarse muchos "slots" asociados, y a la vez, cada "slot", representado por la transacción Slot, e identificado con el atributo SlotId, puede encontrarse repetido para muchos "tipos de slot" (el slot 1 lo podrá mostrar en su grid, así como el slot 2, etc.). Esto claramente no corresponde con la realidad planteada.

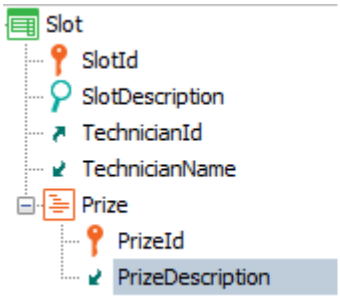
La opción d) está representando una relación 1-N pero opuesta a la que se pide. Obsérvese que aquí un “tipo de slot” tiene asociado un único “slot” y otro “tipo de slot” podrá tener asociado el mismo “slot” que el anterior. Por esta razón, aquí estamos diciendo que cada “slot” podrá tener muchos “tipos”, y que a un “tipo” solo le corresponde un “slot”.

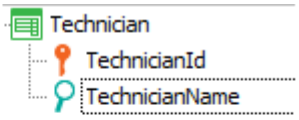
Resumen de entidad fuerte vs débil: una entidad es fuerte, cuando se puede identificar independientemente de las otras, aunque mantenga relación con ellas. En cambio, será débil, cuando no existe con independencia de otra entidad. Para existir requiere de la existencia de la otra. El caso típico es el de los teléfonos de clientes. No tiene sentido tener un teléfono independiente del cliente al que pertenece.

PREGUNTA 3 - DISEÑO

Se tiene una aplicación GeneXus para un casino.

Dadas las siguientes transacciones determine la relación entre los actores de la realidad Slot y SlotPrize (premio del Slot).





a. Relación 1 a 1 (por cada slot hay un solo premio y por cada premio un slot que lo brinda)
b. Relación 1 a N (por cada slot hay varios premios, pero donde cada premio corresponde únicamente a ese slot y no a otro) siendo ambas entidades fuertes.
c. Relación 1 a N (por cada slot hay varios premios, pero donde cada premio corresponde únicamente a ese slot y no a otro) siendo la entidad Slot fuerte pero SlotPrize débil.
d. Relación N a N (por cada slot hay muchos premios y cada premio puede ser brindado por muchos slots)

RESPUESTA

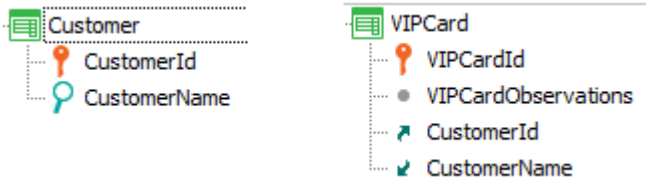
La correcta es la c). ¿Qué significa decir que SlotPrize es una entidad débil respecto a Slot? Que no existirá como entidad independiente. Su relación con Slot es de dependencia absoluta. Un premio determinado solo existe en la medida en que se indica el Slot al que está asociado. Es el premio "tal" del slot "tal". Nunca podrá decirse "es el premio tal" a secas. Siempre deberá indicarse de qué slot se trata. Si en cambio la relación fuera 1-N fuerte, el premio existiría como entidad independiente, identificable por sí mismo, sin tener que indicar el slot para lograr saber de qué premio estamos hablando.

PREGUNTA 4 - DISEÑO

Se tiene una aplicación GeneXus para un casino. Cuenta con transacciones para registrar los clientes así como las tarjetas VIP que se les emiten.

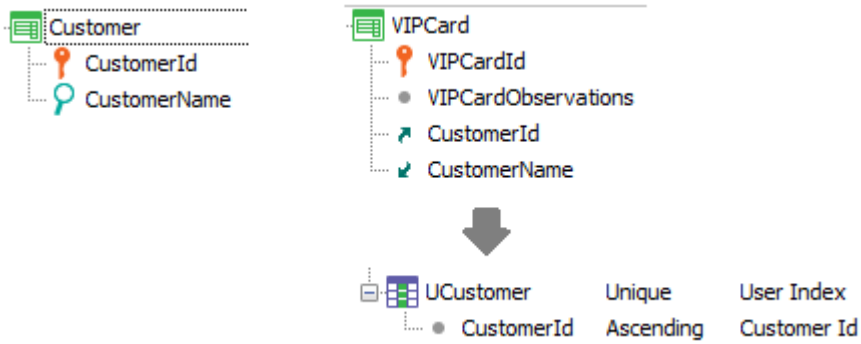
Sabiendo que cada cliente (Customer) puede tener una única tarjeta VIP (VIPCard) y que cada tarjeta VIP solo puede pertenecer a un cliente, determine la opción correcta para el diseño de estas transacciones.

a. Dos transacciones:

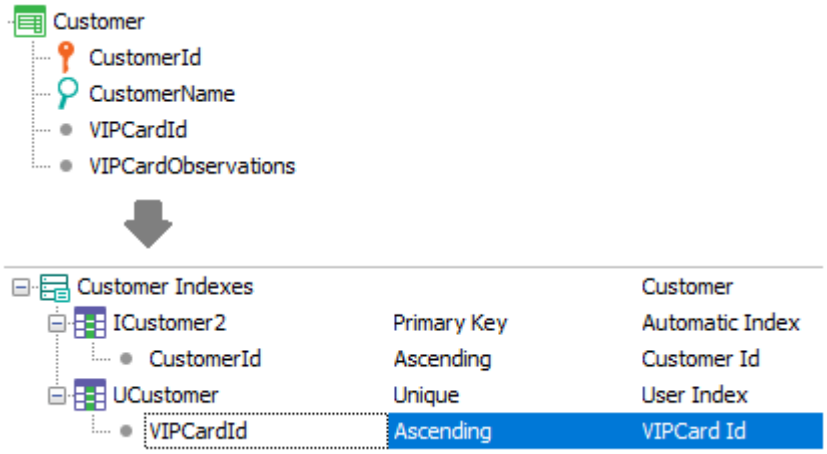


b. Dos transacciones:

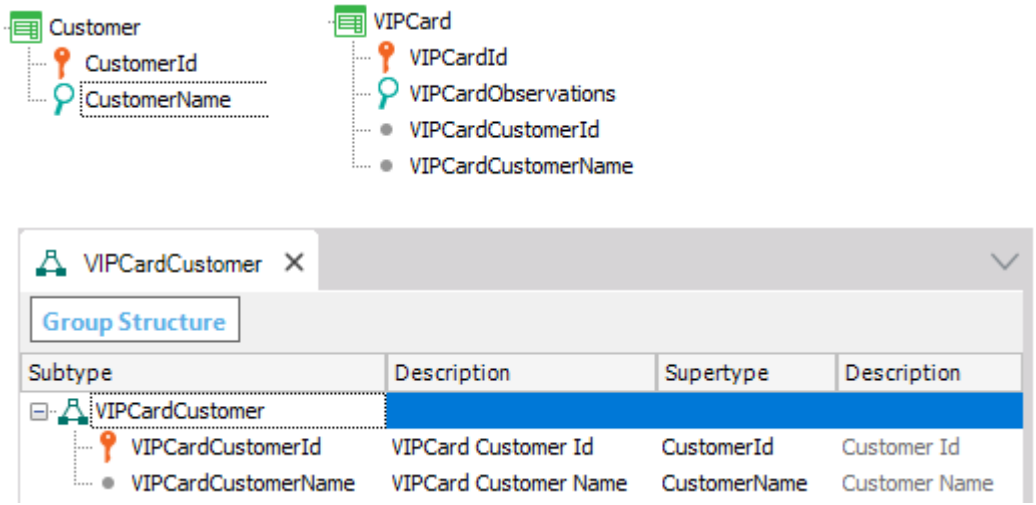
c. Dos transacciones y un índice unique:



d. Una transacción y un índice unique:



e. Dos transacciones y un grupo de subtipos:



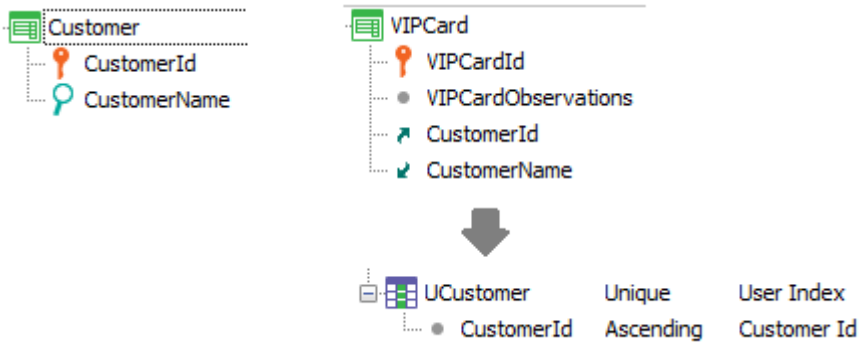
f. Ninguna de las anteriores

RESPUESTA

Se nos pide una relación 1 a 1 entre ambas entidades. Hay un supuesto implícito, y es que se trata de dos entidades separadas, cada una con su identificador. Cada tarjeta corresponderá a un cliente y solo a uno, pero es suficientemente fuerte como para tener una existencia propia. Es decir, el sistema trabajará con las tarjetas más allá del cliente. Habrá operativas donde deba ingresarse la tarjeta y no el cliente (por ejemplo para pagar). Por tanto la opción válida será la **c**) y no la d).

Discutamos las diferencias entre ambas soluciones y luego veremos por qué las demás están lejos de ser correctas.

Observemos primero la solución correcta, la c):

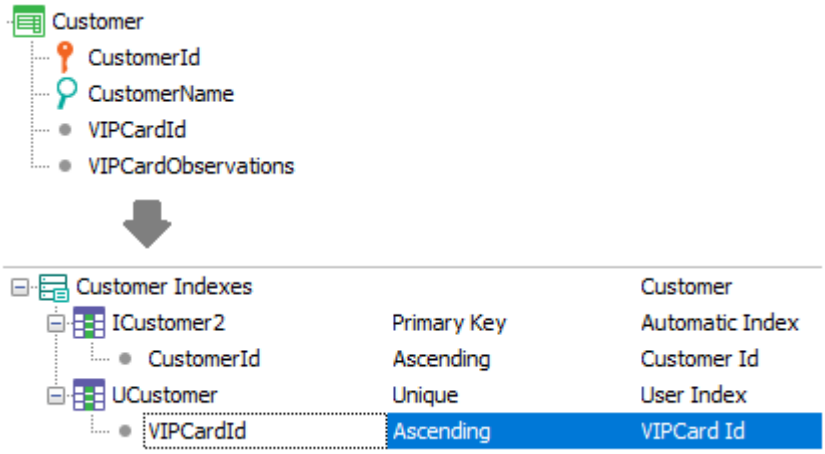


Aquí tenemos un identificador para cliente, CustomerId, y también tenemos un identificador para tarjeta VIPCardId, lo que hace que podamos utilizarlos en cualquier otra transacción como claves foráneas. Por ejemplo, en una factura podemos colocar al atributo VIPCardId, y se allí se inferirá el cliente, sin necesidad de especificarlo también. Como observación, no podemos hacer lo inverso, es decir, colocar en la estructura de la factura al atributo CustomerId y esperar que de allí se infiera la tarjeta. Sí podremos buscarla con una fórmula. ¿Por qué esto es así?

Porque en realidad GeneXus nos está permitiendo modelar una relación 1 a N a la que hacemos 1 a 1 a través de la definición del índice unique sobre lo que sería la clave foránea. Es por ello que a nivel de la tabla VIPCard definimos un índice de usuario Unique sobre el atributo CustomerId, que ya tenía definido un índice por Foreign Key automático.

De esta manera, cada vez que se ingrese una nueva tarjeta al sistema, si se llegara a intentar asignarle para el atributo CustomerId un número ya existente para otra tarjeta, la operación fallará porque utilizando ese índice unique encontrará inmediatamente que ese valor ya existe y nos lo informará.

En la solución d) hemos definido una única transacción, con clave primaria el id de cliente, y con clave candidata el id de la tarjeta. Con esto también conseguimos que un id de tarjeta no pueda repetirse entre clientes (y por supuesto, que un cliente tenga un único id de tarjeta), pero lo que no estamos pudiendo hacer es utilizar la tarjeta en otras entidades. No podrá ser clave foránea en ningún lado. Por ejemplo, no podremos ingresarla en la factura. Tendremos, necesariamente, que colocar el cliente (e inferir la tarjeta).



Yendo a lo más fino, en la solución c) podrían eventualmente existir clientes sin tarjetas (lo que no podrá suceder es que una tarjeta no tenga especificado un cliente). Una buena pregunta a contestarse es cómo se haría para evitar esto, es decir, para que cada cliente tenga necesariamente una tarjeta. Este tema se enlaza con el de UTL. Una posibilidad sería escribir la siguiente regla de invocación a procedimiento en la transacción Customer:

```
CreateVIPCard(CustomerId) on AfterInsert;
```

De este modo, una vez que se ha insertado en la tabla CUSTOMER el cliente (y ANTES DEL COMMIT), se invoca al procedimiento CreateVIPCard, que creará la tarjeta para ese cliente. Suponiendo que VIPCardId es autonumerado, podría estar programado así:

Rules:

```
Parm( in: &CustomerId );
```

Source:

```
&VIPCard.VIPCardObservations = “Card created automatically for the Client”
&VIPCard.CustomerId = &CustomerId
&VIPCard.Insert()
```

Observe que si bien el procedimiento no realizará commit (puesto que las operaciones de BCs no hacen que el procedimiento interprete que accederá a la base de datos y por tanto no coloca el commit implícito al final del fuente), dado que se lo ha invocado desde la transacción Customer antes de su Commit, al devolver el control a la transacción, esta realizará el commit y por tanto quedarán commiteados los dos registros insertados en la base de datos: el cliente en el tabla CUSTOMER insertado por la transacción antes de llamar al procedimiento y la tarjeta en la tabla VIPCARD insertada por el propio procedimiento al utilizar el método Insert.

Recuerde que en aplicaciones web no puede extenderse la UTL conformada por los registros manipulados por una instancia de iteración de transacción para abarcar también los registros manipulados por otra transacción. Pero sí entre transacción y objetos batch como el proc.

Discutamos brevemente por qué son incorrectas las demás opciones de la pregunta.

La opción a) claramente representa una relación 1 a N. No es lo que nos piden.

La opción b) convendría estudiarla con detenimiento:

En un primer golpe de vista, mirando la transacción Customer estamos diciendo que un cliente tiene una única VIPCard y mirando la transacción VIPCard decimos que una tarjeta tiene un único cliente. Así podría parecer que se está satisfaciendo el requerimiento. Sin embargo, las transacciones operan en conjunto y no solas. Si miramos el conjunto, podríamos decir entonces que VIPCardId sería una clave foránea en la tabla CUSTOMER, y que simétricamente, CustomerId sería una clave foránea en la tabla VIPCARD. Y podríamos aventurar que el error de este diseño es que si bien se cumple que un cliente tiene una sola tarjeta y que una tarjeta tiene un solo cliente, no se cumple que coincidan clientes y tarjeta en ambas tablas.

Es decir, el cliente 1 puede tener VIPCardId 5, pero cuando vamos a VIPCardId 5, resulta que esta podría tener el CustomerId 3, por ejemplo.

Sin embargo no nos dimos cuenta de que ni siquiera podrán ingresarse estos registros, debido a los controles de integridad referencial. ¿Cómo hago para ingresar el cliente 1 con tarjeta 5 si antes no la ingresé? Pero para ingresarla, ¿cómo hago si todavía no existe el cliente? Bueno, podría solucionarse permitiendo que VIPCardId en CUSTOMER, así como CustomerId en VIPCard admitan nulos. Así, la primera vez ingreso el cliente sin especificarle tarjeta, y luego ingreso la tarjeta, especificando el cliente y vuelvo al cliente en update para ahora sí especificarle la tarjeta.

Sin embargo, todo el análisis anterior partió de una premisa falsa. ¿Estamos seguros de que GeneXus creará dos tablas?

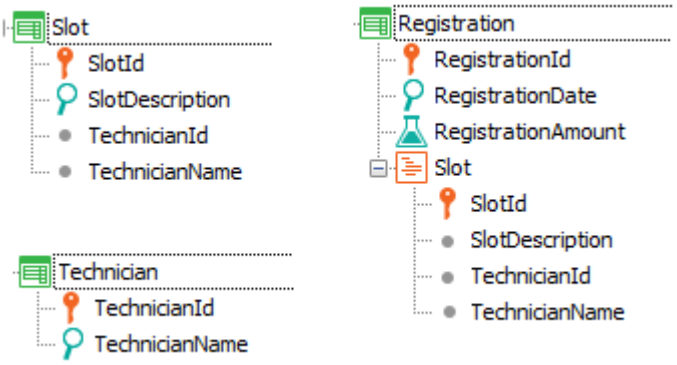
Recordemos que antes que nada, GeneXus normaliza las tablas, para evitar inconsistencias. Si observamos, al tener las referencias cruzadas estamos diciendo por un lado que CustomerId determina a VIPCardId, y por otro que a su vez VIPCardId determina a CustomerId. Así que estas dos

transacciones darán lugar a una única tabla que tendrá la composición de cualquiera de las dos transacciones (puede ser de la primera o de la última). Y eso, si lo pensamos, es completamente lógico. En este caso, si toma la última, la tabla coincidirá con la estructura de VIPCard, pero donde además creará un índice primario también por CustomerId (es decir, no permitirá ingresar para dos tarjetas distintas el mismo cliente). Se transformaría en un caso análogo al de la solución d), con todo lo que allí discutimos. Entonces esta solución es más próxima a la requerida, salvo por el hecho de que necesitamos que exista el cliente y la tarjeta por separado, como entidades.

La última solución, la e), es una variación de la a). Sigue representando una relación 1 a N (donde parece completamente innecesaria la definición del grupo de subtipos).

PREGUNTA 5 – DISEÑO/NORMALIZACIÓN

Se tiene una aplicación para un casino. Dado el siguiente diseño de transacciones, determine la estructura física de las TABLAS que GeneXus diseñará y creará.



a)

TECHNICIAN	SLOT	REGISTRATION	REGISTRATIONSLOT
TechnicianId*	SlotId*	RegistrationId*	RegistrationId*
TechnicianName	SlotDescription	RegistrationDate	SlotId*
	TechnicianId		TechnicianId

b)

TECHNICIAN	SLOT	REGISTRATION	REGISTRATIONSLOT
TechnicianId*	SlotId*	RegistrationId*	RegistrationId*
TechnicianName	SlotDescription	RegistrationDate	SlotId*
	TechnicianId	RegistrationAmount	TechnicianId

c)

TECHNICIAN	SLOT	REGISTRATION	REGISTRATIONSLOT
TechnicianId*	SlotId*	RegistrationId*	RegistrationId*
TechnicianName	SlotDescription	RegistrationDate	SlotId*
	TechnicianId		

d)

TECHNICIAN	SLOT	REGISTRATION	REGISTRATIONSLOT
TechnicianId*	SlotId*	RegistrationId*	RegistrationId*
TechnicianName	SlotDescription	RegistrationDate	SlotId*
	TechnicianId	RegistrationAmount	

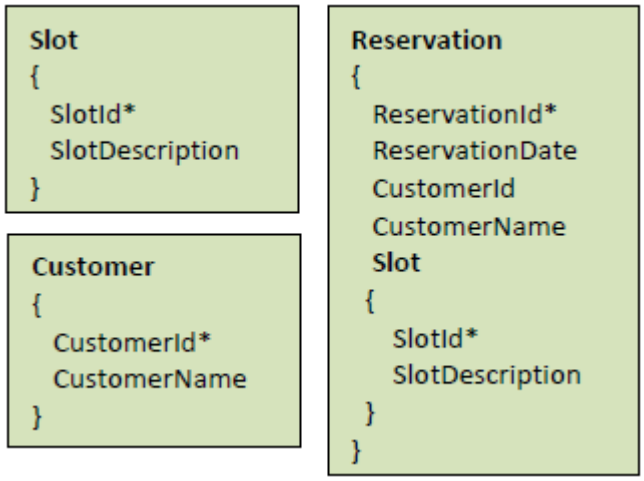
RESPUESTA

La correcta es la c)

- En transacción Slot el atributo TechnicianName es inferido a partir de la clave foránea TechnicianId.
- En el nivel Slot de la transacción Registration el atributo TechnicianId es inferido a través de la clave foránea SlotId (¡cuidado con creer que porque un atributo es clave primaria en una tabla eso automáticamente lo convierte en atributo almacenado y clave foránea en toda otra tabla asociada a una transacción en la que aparezca!) Sabiendo esto, ¿cómo haría para que el segundo nivel de Registration permitiera registrar un técnico independiente del del slot? Tendrá que utilizar subitpos. Piense la solución y luego vea la respuesta d) a la pregunta 10.
- RegistrationAmount es fórmula, por lo que no estará presente en ninguna tabla (a menos que explícitamente se la defina como redundante).

PREGUNTA 6 – TABLA EXTENDIDA

Se cuenta con una aplicación GeneXus para un casino. Teniendo las siguientes transacciones, determine la tabla extendida de la tabla RESERVATIONSLOT



a. RESERVATIONSLOT + RESERVATION
b. RESERVATIONSLOT + SLOT
c. RESERVATIONSLOT + RESERVATION + SLOT
d. RESERVATIONSLOT + RESERVATION + SLOT + CUSTOMER

RESPUESTA

La correcta es la d).

El concepto de tabla extendida es bien importante en GeneXus, porque toda su lógica se basa en abstraer la visión de las tablas físicas, y pasar a una visión más conceptual (de información unívocamente relacionada, que está desperdigada en distintas tablas físicas solo a los efectos de la normalización, pero que no obstante conforma lógicamente una unidad). De ahí que el for each permita trabajar casi por igual con todos los atributos de la tabla extendida (y no solo de la tabla base), así como los grids, grupos de Data Providers, reglas de transacciones, etc.

¿Para qué se normaliza una base de datos? Para evitar la duplicación de datos, que traería el consabido riesgo de tener datos inconsistentes.

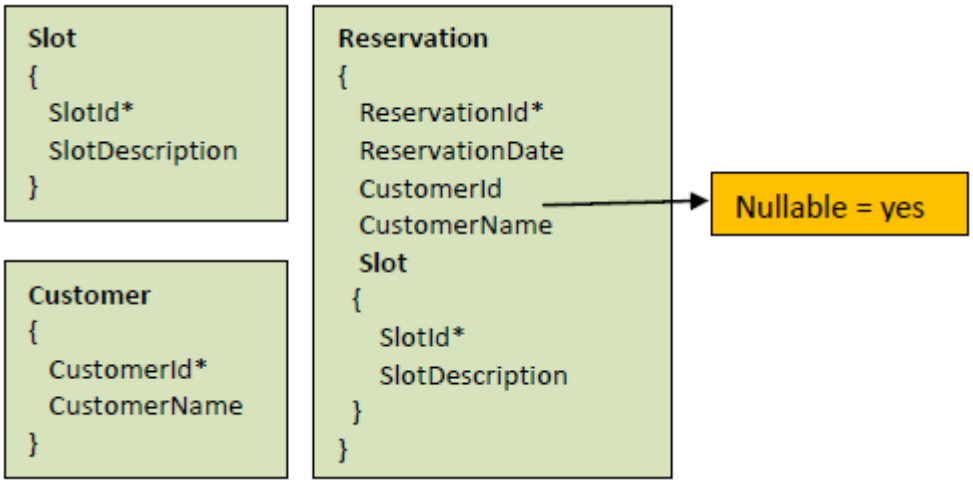
De no tener la necesidad de evitar la duplicación, tendríamos una única tabla para los slots de las reservas, con clave primaria {ReservationId, SlotId}. ¿Qué atributos tendría esa tabla? Tendría además de la clave primaria, la descripción del slot (SlotDescription), la fecha de la reserva (ReservationDate), el cliente de la reserva (CustomerId) con su nombre (CustomerName). Toda esa

información estaría junta en una tabla, a la que llamamos “tabla extendida”. Físicamente no existe, pero sí como herramienta conceptual.

PREGUNTA 7 – CÓMO DEJAR SIN VALOR UNA CLAVE FORÁNEA

Se tiene una aplicación GeneXus para un Casino, que cuenta con un conjunto de transacciones para registrar los slots (Slot), y la reserva de slots (Reservation) por parte de los clientes (Customer) según se muestra.

En algunas ocasiones se realizan reservas sin la necesidad de especificar el cliente (CustomerId). A partir del diseño propuesto, indique la afirmación que considere correcta:



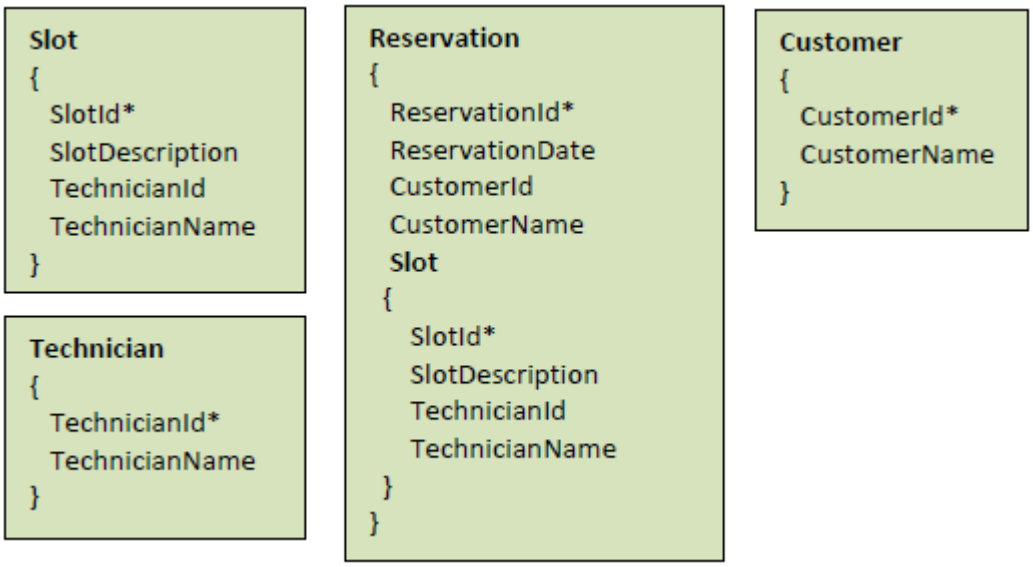
a. A pesar de declarar que el atributo CustomerId (clave foránea) en la tabla RESERVATION admite nulos (o sea, admite un valor no especificado a nivel de la base de datos), GeneXus siempre disparará los correspondientes controles de integridad referencial contra la tabla CUSTOMER, y no admitirá que no se ingrese un valor válido para esa clave foránea CustomerId.
b. Al momento de declarar que el atributo CustomerId (clave foránea) en la tabla RESERVATION admite nulos (o sea, admite un valor no especificado a nivel de la base de datos), entonces si no se ingresa un valor en esa clave foránea, GeneXus no dispara los controles de integridad referencial contra la tabla CUSTOMER, pero si se ingresa un valor en esa clave foránea, GeneXus sí disparará los controles de integridad referencial contra la tabla CUSTOMER.
c. Ninguna de las anteriores

RESPUESTA

La correcta es la b)

PREGUNTA 8 – UTILIZACIÓN DE ÍNDICES POR CLAVES PRIMARIA Y FORÁNEA

Dado el siguiente diseño de transacciones, determine qué índice es utilizado para validar con eficiencia al momento de intentar eliminar un técnico (Technician) a través de la transacción, que no existan slots que lo tengan asignado.



a. Índice por TechnicianId en la tabla TECHNICIAN
b. Índice por TecnnicianId en la tabla SLOT
c. Índice por SlotId en la tabla SLOT
d. Ninguna de las anteriores es correcta

RESPUESTA

La correcta es la b). Esta pregunta no es tan importante para el alumno, cuanto para el instructor (difícilmente la encuentre en preguntas de examen reales). Al momento de pretender eliminar un registro de la tabla TECHNICIAN a través de la transacción, como sabemos, se incluye lógica en ella para acceder a todas las tablas que tengan a TechnicianId (clave primaria) como clave foránea, es decir, a todas las tablas subordinadas, que la referencian. Debe accederse a cada una de estas tablas, para corroborar que no exista ningún registro que referencie al que quiere ser borrado. En este ejemplo hay sólo una subordinada: la tabla SLOT. ¿Cómo acceder a esta tabla para saber si

existe un registro con el valor para el atributo clave foránea TechnicianId igual al del registro que quiere borrarse de TECHNICIAN? Los índices son los mecanismos eficientes para realizar estas búsquedas, por lo que GeneXus utilizará el índice por clave foránea definido en la tabla SLOT, sobre el atributo clave foránea TechnicianId. De hecho, es para hacer eficientes los controles de integridad referencial en la eliminación que los índices por clave foránea son creados por GeneXus automáticamente en la base de datos.

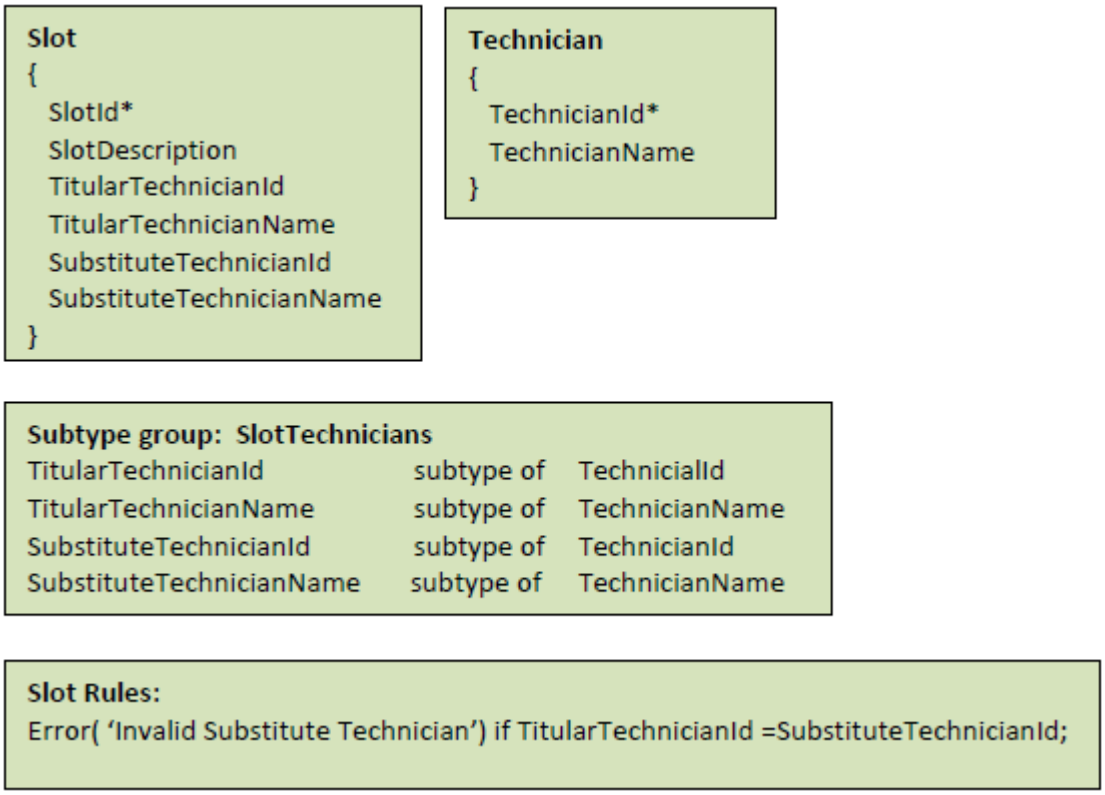
Es común confundirse y pensar que la respuesta correcta hubiera sido la a), pero razonando se entiende que el índice por TechnicianId en la tabla TECHNICIAN lo que hace es acceder a la tabla TECHNICIAN, donde no hay nada que buscar. La búsqueda debe efectuarse sobre SLOT. Este es un tema complejo, porque en la realidad depende un poco del DBMS la estrategia de acceso a las tablas utilizadas. Pero la lógica conceptual de GeneXus sería la indicada.

PREGUNTA 9 – DISEÑO CON SUBTIPOS AGRUPACIONES

Se tiene una aplicación GeneXus para un casino. Esta cuenta con un conjunto de transacciones para registrar los slots (Slot) y los técnicos encargados de las reparaciones (Technician).

Cada vez que se ingresa un nuevo slot se le debe asociar un técnico responsable y uno suplente. El sistema deberá controlar que no sea el mismo.

Determine si es verdadero o falso que la siguiente alternativa resuelve correctamente el requerimiento anterior.



RESPUESTA

Falso.

De definirse subtipos para el técnico titular y para el suplente, deben definirse en dos grupos distintos, para indicar el conjunto de información que se maneja conjuntamente. En la solución presentada se definió un solo grupo de subtipos, donde se colocaron tanto los subtipos que corresponden al titular como al suplente, mezclados. Esto es claramente incorrecto. ¿Por qué?

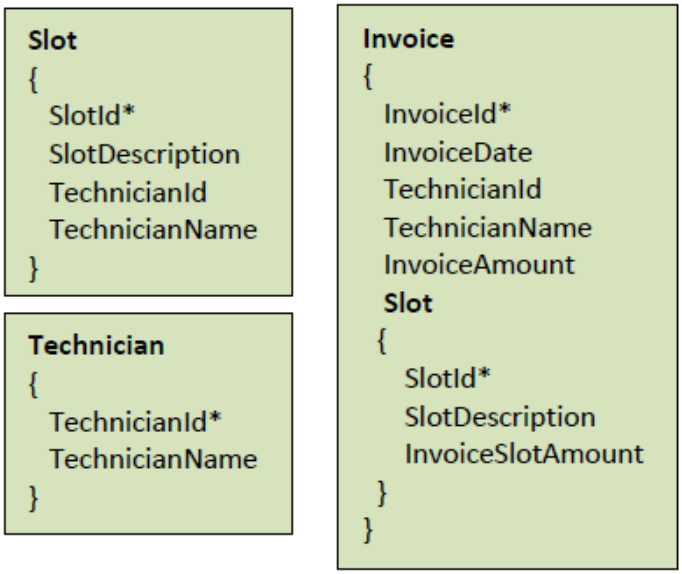
PREGUNTA 10 – DISEÑO CON SUBTIPOS (DOBLE REFERENCIA EN TABLA EXTENDIDA)

Se tiene una aplicación GeneXus para un casino, con un conjunto de transacciones para registrar los slots (Slot) y los técnicos encargados de las reparaciones (Technician).

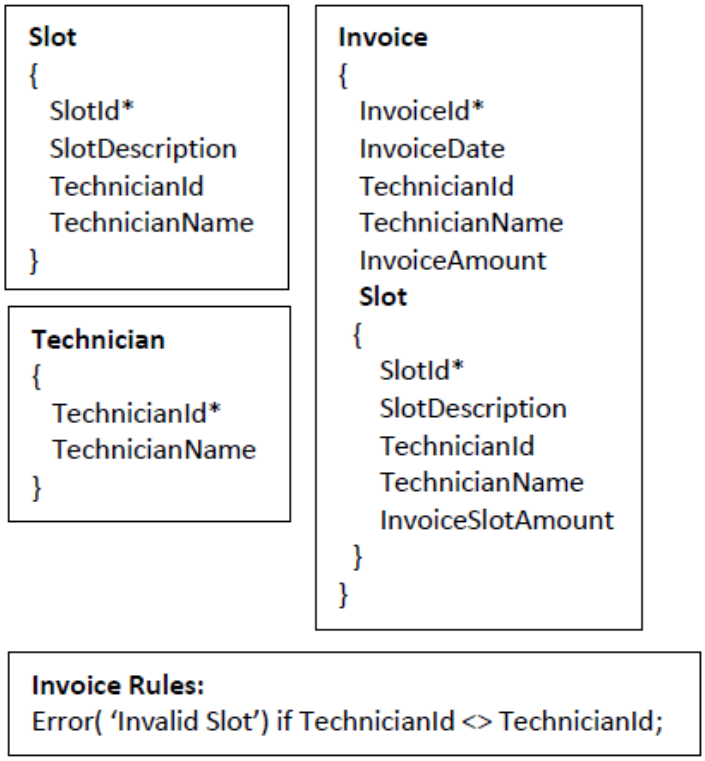
Un slot puede ser reparado por un solo técnico y cada técnico tiene asignados varios slots para reparar en caso de necesidad. Es así que a la hora de facturar los servicios de un técnico (Invoice) se debe verificar que los slots detallados efectivamente estén a cargo del técnico de la factura.

Determine de las siguientes opciones la que implementa este requerimiento.

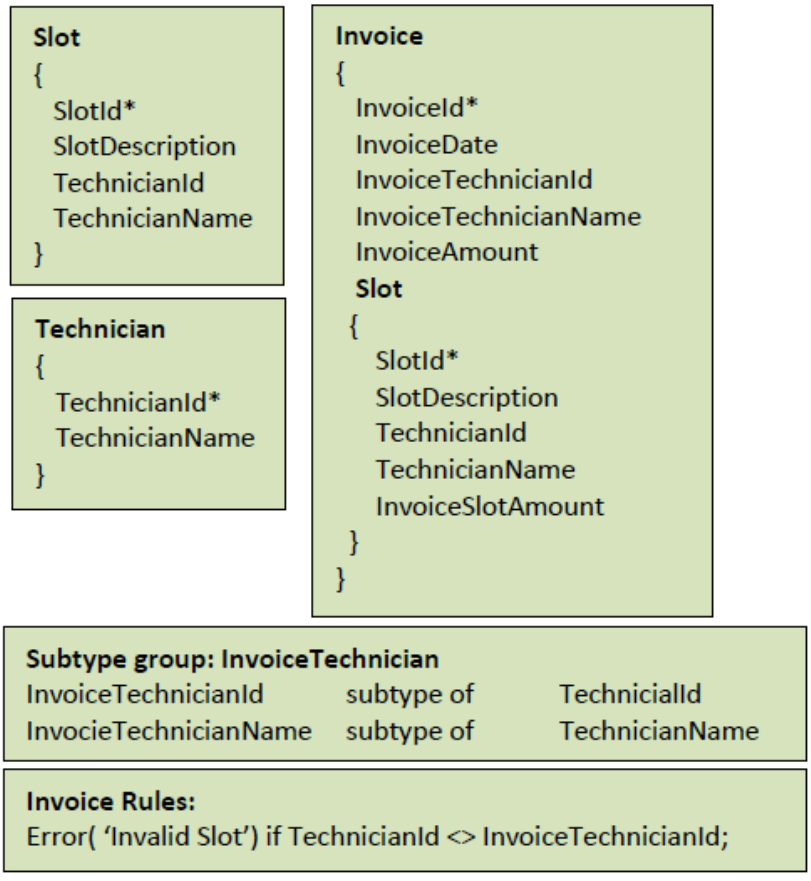
a.



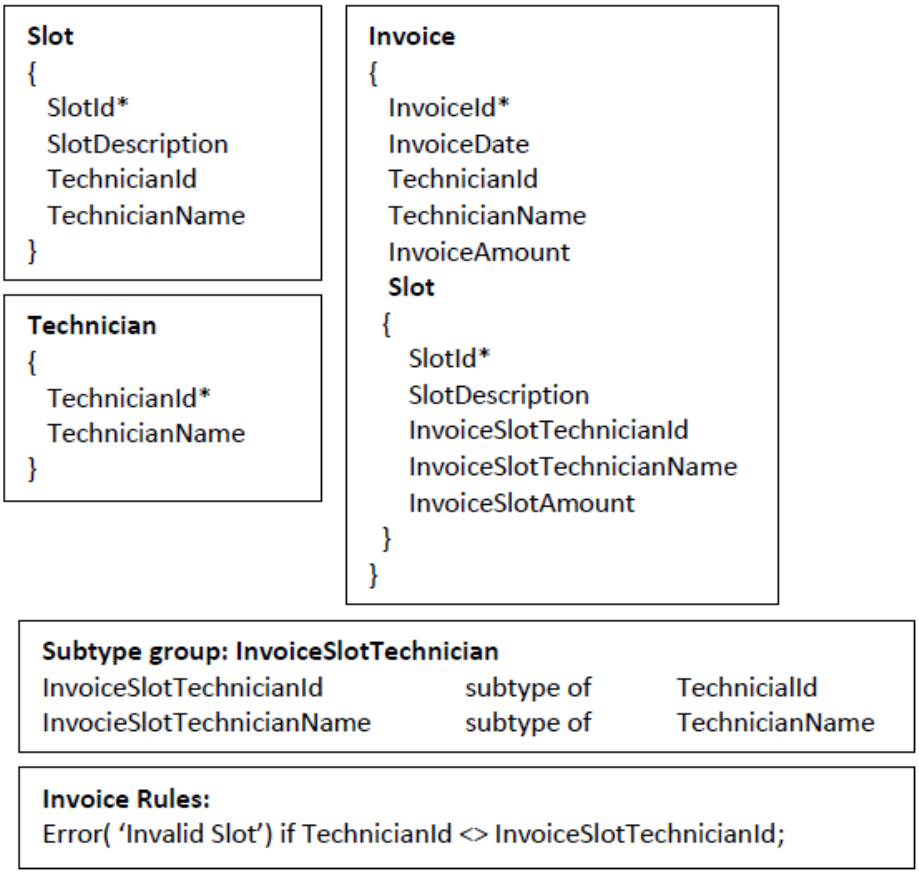
b.


Slot
{
SlotId*
SlotDescription
TechnicianId
TechnicianName
}
Technician
{
TechnicianId*
TechnicianName
}
Invoice
{
InvoiceId*
InvoiceDate
TechnicianId
TechnicianName
InvoiceAmount
Slot
{
SlotId*
SlotDescription
TechnicianId
TechnicianName
InvoiceSlotAmount
}
}
Invoice Rules:
Error('Invalid Slot') if TechnicianId <> TechnicianId;

c.


Slot
{
SlotId*
SlotDescription
TechnicianId
TechnicianName
}
Technician
{
TechnicianId*
TechnicianName
}
Invoice
{
InvoiceId*
InvoiceDate
InvoiceTechnicianId
InvoiceTechnicianName
InvoiceAmount
Slot
{
SlotId*
SlotDescription
TechnicianId
TechnicianName
InvoiceSlotAmount
}
}
Subtype group: InvoiceTechnician
InvoiceTechnicianId subtype of TechnicialId
InvocieTechnicianName subtype of TechnicianName
Invoice Rules:
Error('Invalid Slot') if TechnicianId <> InvoiceTechnicianId;

d.



RESPUESTA

Si bien hay otras formas de implementarlo, la única de las opciones presentadas que lo logra es la c). Obsérvese que en el caso de la d), como el subtipo InvoiceSlotTechnicianId no está en un grupo con un subtipo de SlotId, será un técnico independiente del que viene inferido del Slot.

Si bien la solución c) es correcta, en general no es la que solemos aconsejar. ¿Por qué?

No es en el cabezal de Invoice en que surge la necesidad de cambiarle el nombre al técnico, puesto que allí no existe ambigüedad alguna. Tampoco es en Slot donde se encuentra la ambigüedad.

Es en el segundo nivel de Invoice donde esa ambigüedad aparece. Es que en ese nivel tenemos dos técnicos: el de la factura y el que viene inferido del slot.

Entonces la solución que recomendamos es resolverla aquí. Pero para resolverla aquí tenemos que cambiar el nombre del TechnicianId y también el nombre de SlotId y agrupar todo eso. De este modo, estamos cambiando el nombre tanto a la clave primaria como a la foránea.

Subtype group – InvoiceSlot

InvoiceSlotId	suptype of	SlotId
InvoiceSlotDescription	subtype of	SlotDescription

InvoiceSlotTechnicianId subtype of TechnicianId
InvoiceSlotTechnicianName subtype of TechnicianName

Donde la transacción Invoice quedará:

```
Invoice
{
  InvoiceId*
  InvoiceDate
  TechnicianId
  TechnicianName
  InvoiceAmount
  Slot
  {
    InvoiceSlotId*
    InvoiceSlotDescription
    InvoiceSlotTechnicianId
    InvoiceSlotTechnicianName
    InvoiceSlotAmount
  }
}
```

Y las demás transacciones inmodificadas.

Con esta solución InvoiceSlotTechnicianId será inferido a través de InvoiceSlotId.

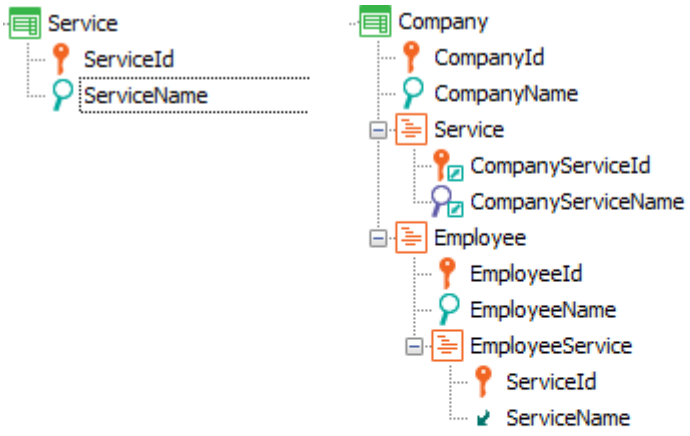
PREGUNTA 11 – DISEÑO CON SUBTIPOS – EVITAR RELACIÓN REFERENCIAL

Se necesitan modelar las transacciones para una realidad en la que se tienen empresas y servicios que éstas pueden contratar (como, por ejemplo, un servicio de emergencia médica). A su vez las empresas tienen empleados que también pueden tener contratados servicios que no tienen por qué coincidir con los de la empresa para la que trabajan. Interesa registrar esos servicios de los empleados porque así, si, por ejemplo, muchos empleados tienen contratado el servicio X de emergencia médica, puede intentarse un convenio con ese servicio para obtener algún descuento.

Los empleados solamente pueden trabajar en una empresa, pero no se los quiere representar como una entidad fuerte, sino dependiente de la empresa.

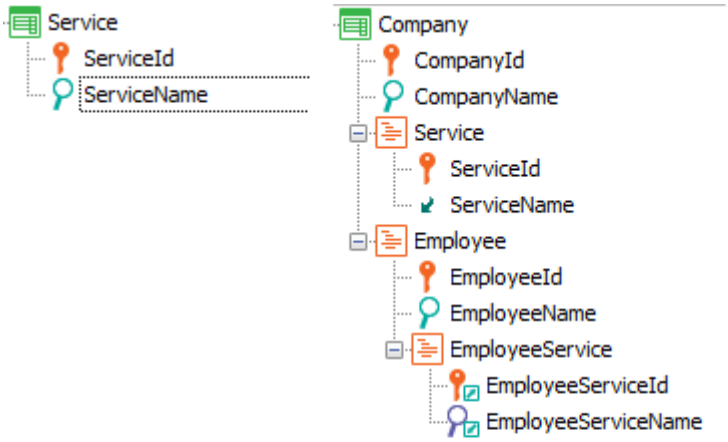
Explique cuál de las dos soluciones es la correcta y por qué la otra no lo es:

a. Dos transacciones y el siguiente grupo de subtipos:



Subtype	Description	Supertype	Description
CompanyService			
CompanyServiceId	Company Service Id	ServiceId	Service Id
CompanyServiceName	Company Service Name	ServiceName	Service Name

b. Dos transacciones y un grupo de subtipos:



Subtype	Description	Supertype	Description
EmployeeService			
EmployeeServiceId	Employee Service Id	ServiceId	Service Id
EmployeeServiceName	Employee Service Name	ServiceName	Service Name

RESPUESTA

La solución correcta es la a) y no la b).

Si observamos con detenimiento tenemos dos niveles paralelos: Service y Employee. Esto significa que todo lo que se infiera de cualquiera de ellos corresponderá a la misma compañía. Sin embargo, no queremos que el servicio del empleado exista como servicio de la empresa, puesto que en nuestra realidad el empleado podría tener contratados servicios diferentes a los de la compañía para la que trabaja. Dicho de otro modo: no queremos que se chequee cuando el usuario ingresa en el grid de servicios del empleado que el servicio ingresado exista como registro en la tabla correspondiente a Company.Service.

Es claro que necesitamos definir un grupo de subtipos puesto que en la misma transacción GeneXus no nos permitirá repetir el mismo nombre de atributo.

La pregunta que surge es: ¿da lo mismo definirlo en un nivel que en el otro? La respuesta: no.

Podríamos definir dos grupos de subtipos y se acabó el problema. Pero no es buena práctica definir más subtipos que los estrictamente necesarios, pues nunca es exactamente lo mismo tener el subtipo que tener el supertipo, como quedará claro con este ejemplo.

Por tanto, para resolver el problema nos alcanza con un único grupo. ¿Por qué la solución correcta es la a) y no la b)?

Es que, si GeneXus nos permitiera repetir el mismo nombre de atributo, claramente encontraría que en la tabla asociada al nivel Company.Employee.EmployeeService, de clave primaria {CompanyId,

EmployeeId, ServiceId} los atributos {CompanyId, ServiceId} confromarían una clave foránea a la tabla correspondiente al nivel Company.Service (ya que su clave primaria sería {CompanyId, ServiceId}).

Pero si lo que hacemos es cambiar el nombre (con un subtipo) de ServiceId en la tabla en la que este atributo es parte de una clave foránea, esto no elimina para GeneXus su función referencial.

En cambio, si el atributo al que le cambiamos el nombre (usando un subtipo) es el que cumple la función de clave primaria, entonces en la tabla en la que el atributo supertipo aparece, no establece la relación referencial.

Esto no es explica en el curso GeneXus, por lo que no tenía por qué saberlo.

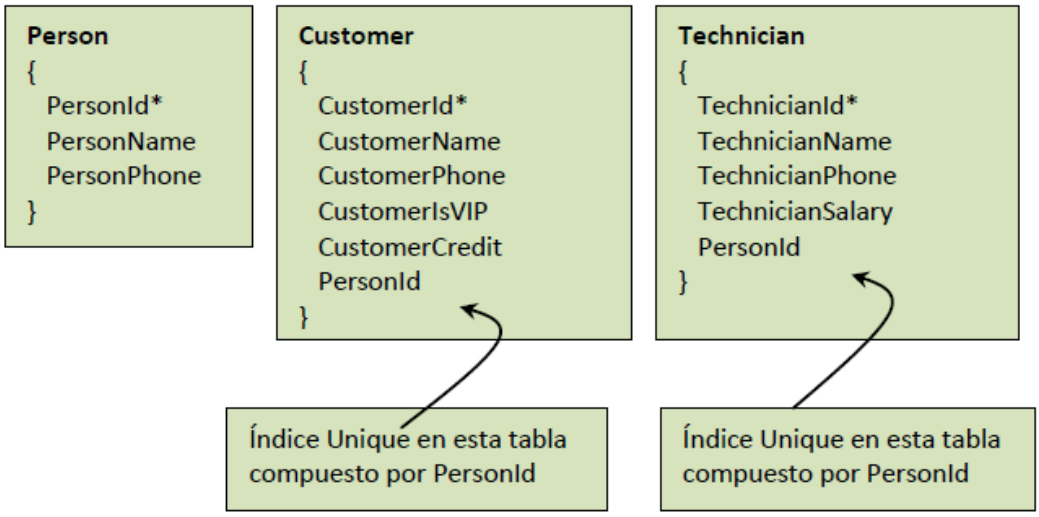
Pusimos el ejemplo para ayudarlo a razonar sobre casos un poco más complejos que, sin embargo, son comunes en apps de la vida real.

PREGUNTA 12 – DISEÑO - ESPECIALIZACIÓN

Se tiene una aplicación GeneXus para un casino.

Entre la información que se necesita registrar, está la de los técnicos que reparan los slots, así como la de los clientes del casino. Como tanto los técnicos como los clientes son personas, de las que se registra un conjunto de información común (nombre y teléfono), se desea registrar la info general una sola vez, y luego sólo registrar la información particular (por ejemplo, si la persona es un cliente, entonces se desea registrar si es un cliente VIP y el crédito que le proporciona el casino, y si es un técnico, su salario).

Determine si es verdadero o falso que la siguiente solución resuelve este caso de especialización adecuadamente en GeneXus.



RESPUESTA

Falso.

Si bien al definir el índice Unique sobre PersonId en la tabla CUSTOMER se establece una relación 1-1 con PERSON, en verdad no estamos representando que se trata de la misma entidad que debió separarse en dos tablas para que en la especializada sólo aparezcan los atributos que no son comunes a los técnicos. Por el contrario, lo que se está diciendo es que son dos entidades diferentes, con una relación 1 a 1, como podría ser la de un cliente con su tarjeta VIP (ver pregunta 4).

Aquí no estamos diciendo que la relación 1 a 1 existente entre Person y Customer es una especial, donde el cliente “es” una persona. Para ello, obsérvese que la persona tendrá un atributo para almacenar el nombre (PersonName) y que el cliente que tenga asociada esa persona, podrá tener otro nombre (CustomerName).

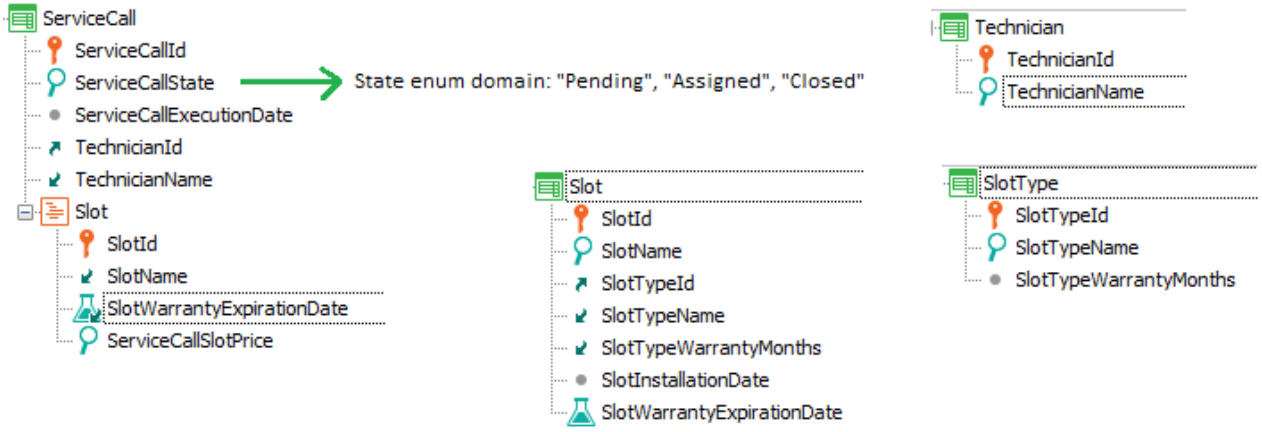
El mismo análisis puede hacerse entre Person y Technician.

La solución a este problema de especialización lo dan los grupos de subtipos. ¿Cómo sería? ¿Por qué definir a CustomerId como subtipo de PersonId resuelve el problema para Customer? Piense que decir que es un subtipo lo convierte en una clave foránea. Pero además, el decir que es identificador de la transacción Customer, lo convierte en clave primaria. Por tanto, CustomerId termina siendo clave primaria que además es foránea. Por lo que finalmente establece una relación 1-1, por clave, y esto último es lo que provoca la relación “es”. Análogo análisis para Technician.

PREGUNTA 13 – REGLAS Y EVENTOS EN TRANSACCIONES

Se está desarrollando un sistema para una empresa que provee, instala y repara Slots para casinos, bares, etc. El sistema debe registrar los clientes (casinos, bares, etc.), los Slots instalados en cada cliente con la fecha de instalación y de fin de la garantía, y los llamados de los clientes para que se les realicen reparaciones de Slots instalados. Los llamados se crean con estado: “pendiente”. Cuando se les asigna un técnico pasan a “asignado” y cuando el técnico presenta la boleta de atención correspondiente al llamado, con la fecha de cumplido y los precios que hay que cobrarle al cliente por los Slots reparados que no estaban en garantía, pasan a “cerrado”.

Para dar por cerrado, entonces, un llamado, hay que cambiarle el estado, ingresar la fecha de cumplido e ingresar los precios de las reparaciones.



No se puede permitir dar por cerrado un llamado si no se ingresaron los precios de todos los slots reparados que no estaban en garantía. Para ello, el desarrollador escribe en la sección de reglas de la transacción ServiceCall:

```
ServiceCallState = State.Closed if update and ServiceCallState = State.Assigned and not
ServiceCallExecutionDate.IsEmpty();
```

```
Error( 'Service call could not be closed') if update and ServiceCallState =
State.Closed and SlotWarrantyExpirationDate < ServiceCallExecutionDate and
ServiceCallSlotPrice.IsEmpty();
```

Razonando de la siguiente manera:

Si el estado del llamado era ‘Assigned’, y estamos queriendo actualizarlo, es porque lo queremos dar por cerrado. Si se ingresa fecha de cierre, entonces se le cambia el estado a ‘Closed’ sabiendo que de todas maneras, si a continuación, a nivel de las líneas o inmediatamente después, ocurre una regla de error (antes del commit), se desharán los cambios y tanto el estado como la fecha quedarán como estaban antes.

Analice esta solución y discuta otras variaciones.

RESPUESTA

Esta declaración de reglas no producirá el efecto deseado.

Cuando se trabaja en una transacción de dos niveles en modo Update, las únicas reglas a nivel de las líneas que se dispararán serán para aquellas líneas sobre las que el usuario haya realizado alguna acción. Si el usuario no edita en absoluto una línea determinada, así esa línea claramente viole una condición especificada en una regla de error, no se disparará la regla.

Por tanto si se ingresa a la transacción en modo update, se introduce una fecha y se confirma, teniendo slots en las líneas que no están en garantía, la regla de error nunca se disparará y se permitirá la grabación.

Por tanto tenemos que chequear, luego de trabajar con las líneas (es decir, luego de que se hayan modificado efectivamente en la tabla física todas aquellas con las que el usuario haya trabajado), que no haya quedado ninguna de las que no están en garantía, sin precio. Una posibilidad sería invocar a un procedimiento que recorra los registros correspondientes a las líneas y cuente a cuántos les falta el precio, y si a uno o más le falta, entonces se dispare la regla de error. ¿Sería así?

```
&Missing = MissingPrices( ServiceCallId ) if Update on AfterLevel Level
ServiceCallSlotPrice;

Error( 'Service call could not be closed') if &Missing <> 0;
```

Si bien la regla de error depende de la variable &Missing que es cargada por el procedimiento, como no está condicionada con el mismo evento de disparo que la invocación al proc, esta regla se evaluará al abrir la transacción y no en el momento deseado, que es después del disparo del proc.

Y si entonces hubiéramos escrito:

```
&Missing = MissingPrices( ServiceCallId ) if Update;

Error( 'Service call could not be closed') if Update and &Missing <> 0 on AfterLevel
Level ServiceCallSlotPrice;
```

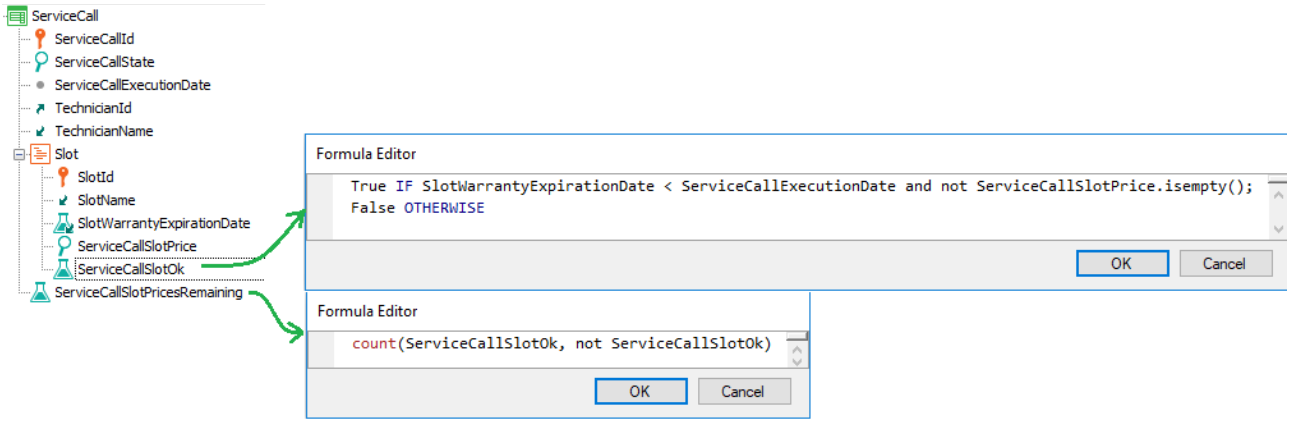
Aquí tendremos el problema de que estaremos disparando el procedimiento ni bien la transacción reconoce que está en modo Update, que es ni bien se abandona el campo identificador. No le hemos dado tiempo al usuario de hacer nada, ni siquiera de ingresar la fecha de cumplido del

llamado. Claramente tenemos que disparar el proc después de que se hayan actualizado todas las líneas que el usuario haya deseado actualizar.

Por tanto, tendríamos que haber escrito:

```
&Missing = MissingPrices( ServiceCallId ) if Update on AfterLevel Level
ServiceCallSlotPrice;

Error( 'Service call could not be closed') if Update and &Missing <> 0 on AfterLevel
Level ServiceCallSlotPrice;
```

Por supuesto que podríamos habernos evitado utilizar un procedimiento, definiendo las dos fórmulas que indicamos:



Y en las reglas:

```
Error( 'Service call could not be closed') if Update and ServiceCallSlotPricesRemaining
<> 0 on AfterLevel Level ServiceCallSlotPrice;
```

¿Y no podríamos aprovechar a también cambiar el estado a Closed después, cuando estemos seguros de que todo se realizará correctamente?

```
Error( 'Service call could not be closed') if Update and ServiceCallSlotPricesRemaining
<> 0 on AfterLevel Level ServiceCallSlotPrice;

ServiceCallState = State.Closed if update and ServiceCallState = State.Assigned and not
ServiceCallExecutionDate.IsEmpty()on AfterLevel Level ServiceCallSlotPrice;
```

El último momento posible para asignar un valor a un atributo del cabezal es inmediatamente antes de que el cabezal se grabe, y esto es mucho antes de on AfterLevel level 2do nivel.

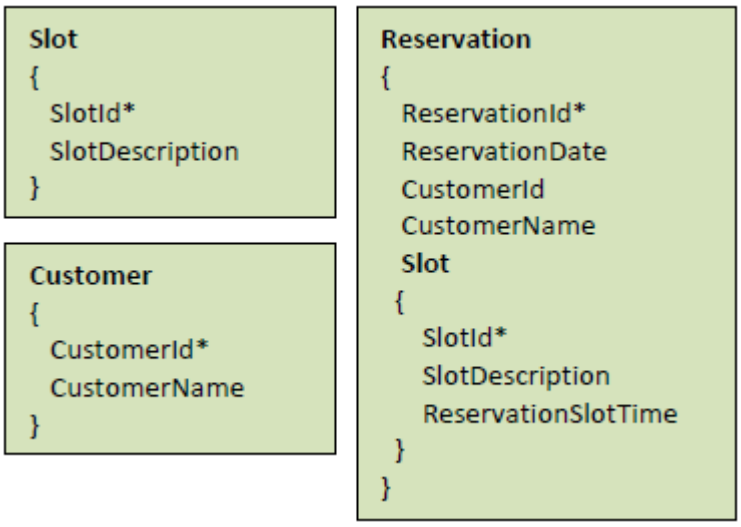
De hecho el último momento posible es on BeforeUpdate.

Es decir que a lo sumo podríamos escribir:

```
Error( 'Service call could not be closed') if Update and ServiceCallSlotPricesRemaining
<> 0 on AfterLevel Level ServiceCallSlotPrice;

ServiceCallState = State.Closed if ServiceCallState = State.Assigned and not
ServiceCallExecutionDate.IsEmpty()on AfterUpdate;
```

PREGUNTA 14 – FÓRMULAS INLINE

Se tiene una aplicación GeneXus para un casino. Ésta cuenta con transacciones para registrar los slots (Slot), y la reserva de los slots (Reservation) por parte de los clientes (Customer)



Dado el siguiente source de un procedimiento:

```
For each Reservation
&Slots = count(ReservationSlotTime)
print Info //ReservationDate, CustomerName, &Slots
Endfor
```

Determine si hay una tabla de partida (al momento de dispararse la fórmula) cuál es, y determine la tabla a ser navegada por la fórmula para obtener su resultado.

a. Tabla de partida: RESERVATION – Tabla navegada: RESERVATIONSLOT
b. Tabla de partida: RESERVATION – Tabla navegada: RESERVATION
c. Tabla de partida: CUSTOMER – Tabla navegada: RESERVATION
d. Tabla de partida: RESERVATIONSLOT – Tabla navegada: RESERVATIONSLOT

RESPUESTA

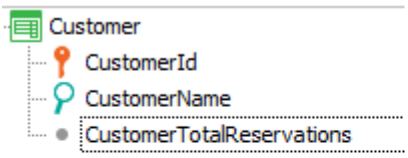
La respuesta correcta es la a).

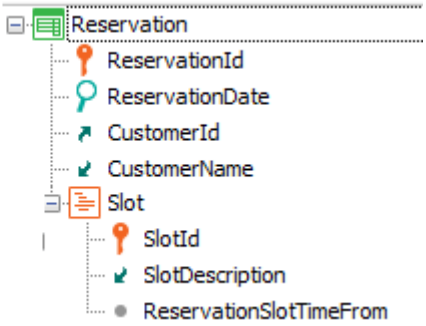
Cualquier cosa que se encuentre dentro de un for each, por el sólo hecho de estar allí dentro, refiere a una iteración de éste, por lo que tiene como contexto el registro de la iteración en el que se encuentra trabajando la ejecución en un momento dado. La tabla base del for each, al

especificarse transacción base sabemos que es RESERVATION. Si no se hubiese especificado transacción base, la tabla base del for each se calcula con los atributos que se encuentren dentro del for each “sueltos” (no los que estén dentro de una fórmula aggregate, como la Count, o los que estén dentro de un for each anidado a éste, u otro comando que tenga su propia resolución, como un new, por ejemplo). En este caso serán los del print block, por lo que la tabla base será RESERVATION. Por tanto el for each iterará sobre esa tabla, y por cada reserva, se dispara la fórmula count. Entonces existirá tabla de partida de la fórmula, que es el contexto que tiene al dispararse (estamos posicionados en una reserva). ¿Qué se cuenta? Los registros donde se encuentra ReservationSlotTime. ¿En qué tabla está ese atributo? En RESERVATIONSLOT. Pero al dispararse la fórmula, en una iteración del for each, tenemos un ReservationId instanciado (el del registro en el que nos encontramos trabajando). Por lo que contará los registros relacionados, los que corresponden a esta reserva. Siempre que la fórmula no tenga un contexto (por ejemplo, si está “suelta” dentro del Source) no tendrá tabla de partida. Por ese motivo, en ese caso, contaría todos los registros de la tabla navegada (RESERVATIONSLOT), justamente porque no hay un contexto que permita especializar el cálculo. El contexto es fundamental en GeneXus y lo que lo hace inteligente. Porque puede inferir cosas en base a ese contexto, sin tener que explicitarlo, ahorrándonos trabajo de programación.

PREGUNTA 15 – ACTUALIZACIÓN CON FOR EACH

En la aplicación para un Casino que estamos desarrollando, habíamos definido las transacciones Customer, Slot y Reservation (esta última para permitir a los clientes realizar reservas de slots para ser utilizados un día dado). Supongamos que la app ya estaba en producción, por lo que las tablas ya están cargadas con datos, cuando nos surge la necesidad de agregar un atributo, CustomerTotalReservations a la transacción Customer, cuyo valor será el total histórico de días en que el cliente ha realizado reservas. Además de agregar el atributo a la transacción Customer, ¿qué más debe hacer?





Suponga que un alumno elabora la siguiente solución:

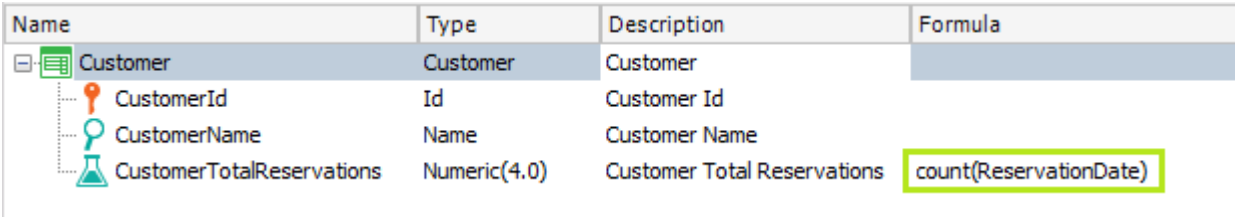
Crea un procedimiento con el siguiente Source y lo ejecuta luego de la reorganización:

```
For each Customer
&totalReservations = 0
For each Reservation
        &totalReservations += 1
endfor
        CustomerTotalReservation = &count
endfor
```

Realice un análisis completo de los problemas de esta solución.

RESPUESTA

Claramente el atributo CustomerTotalReservations responde a un cálculo conocido: corresponde a sumar todos los registros de la tabla Reservation correspondientes al cliente CustomerId. Por tanto, la primera pregunta que nos surge es por qué no definirlo como atributo fórmula en la propia transacción Customer. Sería una fórmula global, claro.

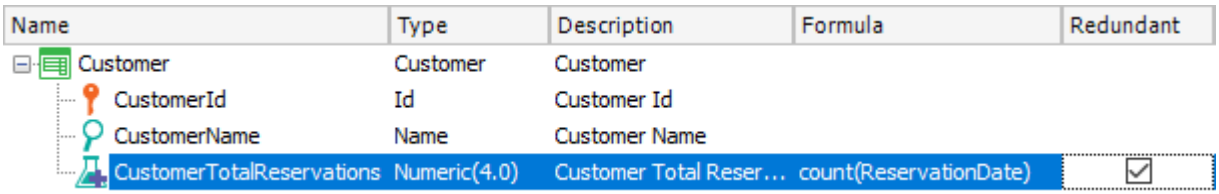


Name	Type	Description	Formula
Customer	Customer	Customer	
CustomerId	Id	Customer Id	
CustomerName	Name	Customer Name	
CustomerTotalReservations	Numeric(4.0)	Customer Total Reservations	count(ReservationDate)

De esta manera no tendríamos que hacer nada más, puesto que cada vez que se necesite el valor, se disparará la fórmula y se realizará el cálculo. Observemos que no necesitamos filtrar los registros de Reservation que correspondan al cliente, puesto que esa es una condición implícita debido al contexto (cuando se dispare la fórmula, CustomerTotalReservations, será porque estamos trabajando con un CustomerId determinado).

Pero ¿qué pasa si el sistema informático está en funcionamiento desde hace mucho tiempo y la cantidad de Reservas de cada cliente es enorme? Supongamos que se necesita con cierta frecuencia hacer reportes de los clientes ordenados de acuerdo a la cantidad de reservas realizadas. Cada vez tendrá que dispararse el cálculo, para cada cliente, y esto puede implicar un costo alto de performance.

Entonces puede surgir la necesidad de que este atributo se almacene físicamente, y simplemente se actualice cuando se agregue o elimine cada reserva. Para ello alcanza con definirlo como atributo redundante (y esto se realiza colocando visible la columna Redundant en la estructura de la transacción y marcando el check box para este atributo). A partir de esta alternativa GeneXus hará por nosotros todo lo necesario. A los efectos del desarrollador, es como si la fórmula siguiera siendo virtual, solo que no lo es, por lo que la performance mejorará notablemente a la hora de realizar el reporte pedido frecuentemente (porque se tomará directamente el valor almacenado y no volverá a dispararse la fórmula cada vez). De hecho, cuando se agregue una reserva para el cliente, GeneXus habrá generado código que se ejecutará actualizando el atributo fórmula sin que nos debamos preocupar (y no será volviendo a disparar la fórmula, sino simplemente sumando o restando 1 de acuerdo a si se está insertando o eliminando una reserva).



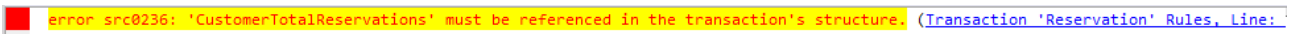
Name	Type	Description	Formula	Redundant
Customer	Customer	Customer		
CustomerId	Id	Customer Id		
CustomerName	Name	Customer Name		
CustomerTotalReservations	Numeric(4.0)	Customer Total Reser…	count(ReservationDate)	☑

El alumno podría querer realizar esto mismo, pero a mano con el procedimiento, que desde ya sabemos realizará lo esperado, aunque su programación es defectuosa, como veremos y esto hay que marcárselo al alumno enfáticamente.

De todos modos, con su solución lo único que conseguirá será inicializar el atributo CustomerTotalReservations, que no es fórmula, con el total de reservas de cada cliente al momento

de ejecutar el proc., pero no está resolviendo qué sucede cada vez que se ingresa o elimina una reserva nueva.

Por tanto a su solución le estaría faltando agregar en la transacción Reservation el atributo CustomerTotalReservations en la estructura –no importa que sea inferido: como lo vamos a utilizar en las reglas lo necesitamos declarar allí; de lo contrario no nos dejará grabar, advirtiéndonos:



y la regla:

```
Add( 1, CustomerTotalAmount);
```

Este es un buen momento para evaluar si se entiende lo que realiza la regla Add en todos los modos (Insert, Update, Delete).

Resumiendo: se agrega el atributo CustomerTotalAmount a la estructura de Customer, se reorganiza para que se agregue el atributo a la tabla, se ejecuta por única vez el procedimiento para inicializar su valor, y se agregar atributo CustomerTotalAmount a estructura de transacción Reservation y se agrega regla Add.

Todo esto es lo que hace automáticamente GeneXus al definir la fórmula como redundante.

Pregunta: en vez del procedimiento, ¿podríamos haber dado a la propiedad Initial Value del atributo CustomerTotalAmount en la transacción Customer el valor: count(ReservationDate), de modo que al reorganizar y agregar el atributo se cargue con el valor inicial que resulta de calcular esa fórmula para cada CustomerId?

Ahora vayamos a analizar el procedimiento del alumno, abstrayendo el resto de la solución:

```
For each Customer
&totalReservations = 0
For each Reservation
        &totalReservations += 1
endfor
        CustomerTotalReservation = &count
endfor
```

Ese procedimiento intenta asignar el valor al atributo CustomerTotalAmount que se ha agregado a la tabla y está vacío para cada uno de los clientes.

Por eso el alumno decidió escribir un for each que recorrerá la tabla CUSTOMER y para cada registro de la tabla, va a querer asignarle el valor a CustomerTotalReservation, y para ello lo que hace es codificar otro for each que recorra la tabla RESERVATION, pero solamente para aquellos registros que correspondan a ese CustomerId. Y lo que hace es sumar uno a una variable

previamente inicializada en cero, de manera tal que cuando termina de iterar por todos las reservas de ese cliente, tendrá su cantidad.

Una solución un poco mejor, habría sido.

```
For each Customer
        CustomerTotalReservation = count( ReservationDate)
endfor
```

Y otra posibilidad hubiese sido:

```
For each Reservation
        CustomerTotalReservation += 1
endfor
```

Es interesante discutir esta última. ¿Por qué no se le ocurrió al alumno? Por cada reserva se accede a la tabla extendida, en este caso podemos acceder al atribtuo CustomerTotalReservation y actualizarlo. La desventaja podría ser que estamos actualizando el atributo por cada reserva que tenga el cliente, en vez de contar todas las reservas y recién allí actualizar el atributo. Por otro lado, con esta solución si muchos clientes no han efectuado ninguna reserva, no serán recorridos inútilmente. Pero esto mismo podríamos también haberlo conseguido del siguiente modo:

```
For each Reservation
unique CustomerId
        CustomerTotalReservation = count( ReservationDate)
endfor
```

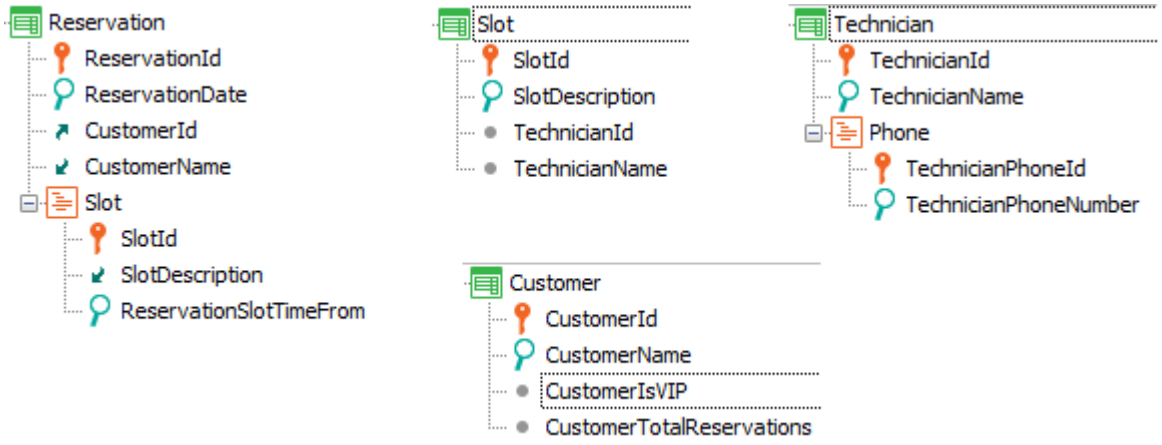
Que es como implementar un corte de control por cliente. Es decir, se recorre la tabla de reservas, agupando por CustomerId (es lo que significa “unique CustomerId”, es decir, que no nos vamos a quedar con los valores repetidos de CustomerId), y para cada CustomerId único encontrado, se cuentan los registros que tienen el atributo ReservationDate, para ese cliente (condición implícita). Y esa cuenta se la asigna al atributo CustomerTotalReservation, que es de la tabla extendida de Reservation.

Es importante entender las diferencias entre las soluciones, de modo de elegir la que más nos convenga. ¿Cuál es la solución más performante?

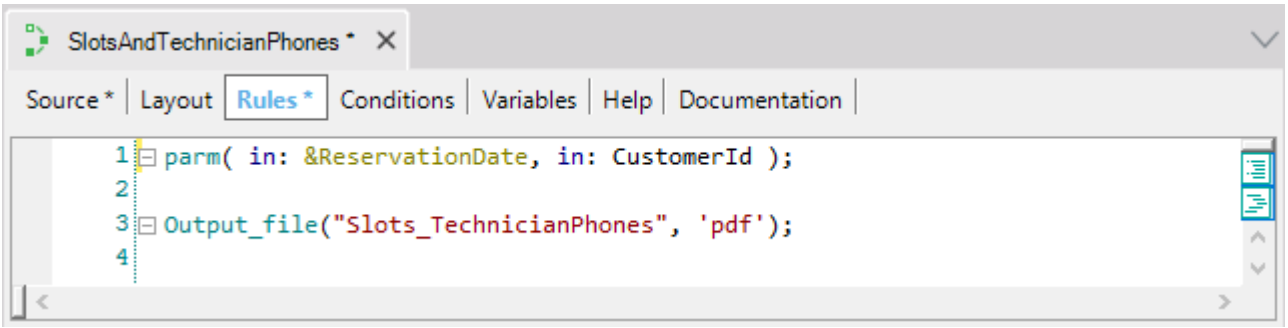
PREGUNTA 16 – CASOS DE FOR EACHS ANIDADOS

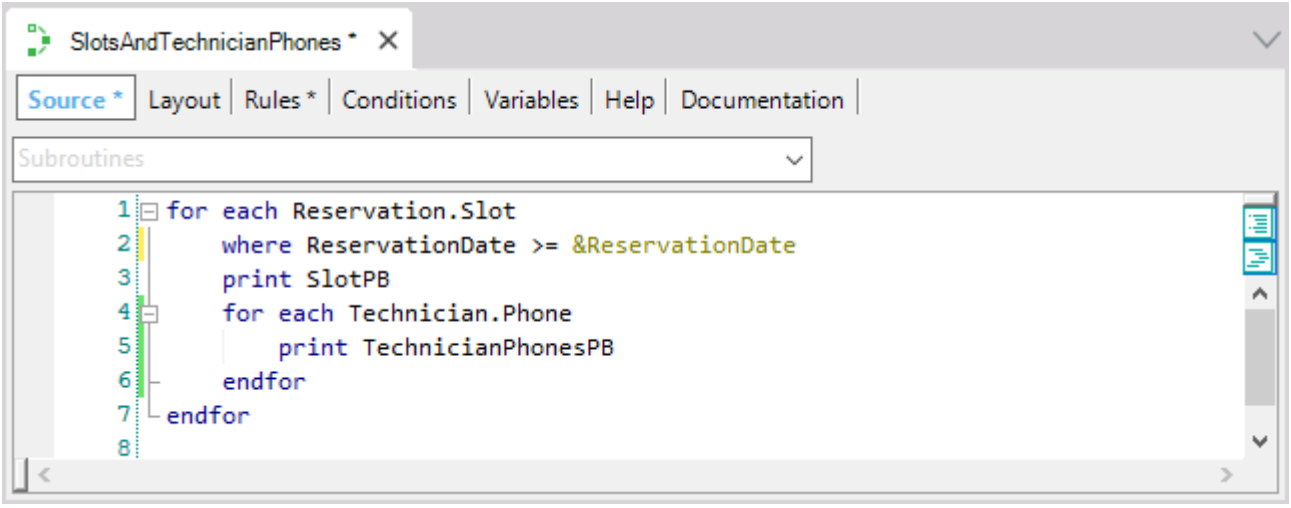
En la aplicación para un Casino que estamos desarrollando, habíamos definido las transacciones Customer, Slot y Reservation (esta última para permitir a los clientes realizar reservas de slots para ser utilizados un día dado). Además cada slot tiene un técnico asignado para realizar las reparaciones que requiera:

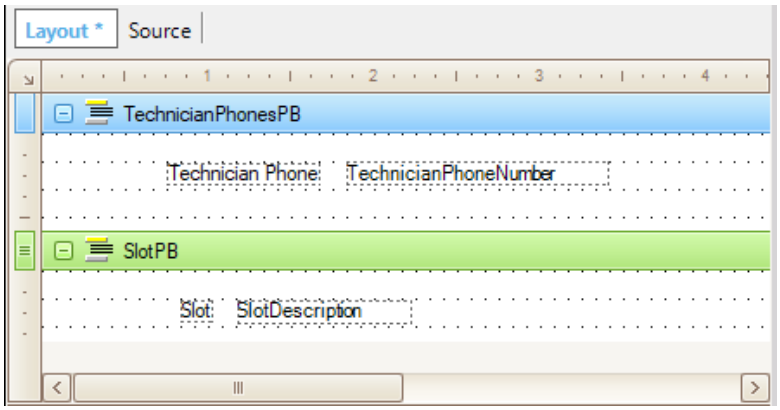
Se quiere realizar un listado pdf que muestre para un cliente VIP dado, desde una fecha dada, los slots que tiene reservados y para cada uno, los teléfonos del técnico del slot, por si hay que llamarlo de urgencia para reparar el slot.



Para resolver el requerimiento, se ha implementado un procedimiento como el que se muestra. ¿Está bien programado? Analícelo.







Y en las properties, Call Protocol = HTTP

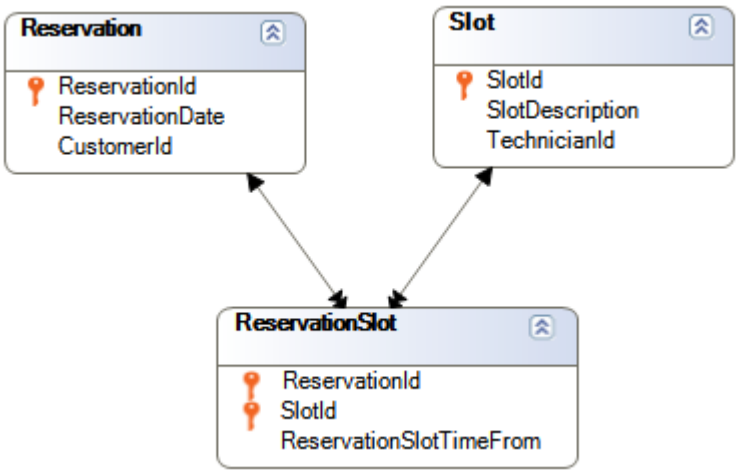
En la siguiente pregunta deberá pensar cómo implementar esto mismo pero como una pantalla web y no como un pdf.

RESPUESTA

Está bien programado. A primera vista podría pensarse que el Layout está invertido. Pero recuerde que el orden de los printblocks no dice nada acerca de cómo serán listados en la salida. Aquí solamente se declaran y diseñan. Es en el Source donde se envían a la salida con el comando print.

Analicemos si la información listada es la que se nos pide. Tenemos un par de for eachs anidados. En el primero decimos explícitamente que la tabla base será la correspondiente al nivel Slot de la transacción Reservation, es decir, la de nombre ReservationSlot. Deberíamos corroborar que dentro de ese for each no se esté utilizando ningún atributo que no pertenezca a la tabla extendida de RESERVATIONSLOT, pues de ser así, el listado de navegación arrojará una advertencia indicando que ese atributo no es accesible. Los atributos que debemos corroborar son los que, dentro del for each, no están dentro del segundo for each. En nuestro caso solamente tenemos dos atributos en

esta situación: el del where, y el del printblock de nombre SlotPB. Estos atributos son SlotDescription, presente en la tabla SLOT, y ReservationDate, presente en la tabla RESERVATION. Si observamos el diagrama de tablas:



vemos claramente que desde RESERVATIONSLOT accedemos a un único registro de la tabla SLOT y a un único registro de la tabla RESERVATION. GeneXus deberá, entonces, acceder a esas dos tablas cada vez que itere en el for each.

¿Con qué registros de la tabla base va a trabajar el for each? Claramente con aquellos que cumplan que, al ir a la tabla RESERVATION para evaluar el valor de ReservationDate, éste sea mayor o igual al valor de la variable &ReservationDate recibida por parámetro. ¿Solo eso? No, también deberá cumplir que el valor de CustomerId coincida con el recibido por parámetro directamente en ese atributo. Recordemos que "recibir en atributo" es establecer que ese atributo estará instanciado, es decir, formará parte del contexto del objeto. Dicho de otro modo, toda vez que ese atributo participe en algún lado, lo hará con el valor recibido cuando se invocó al objeto.

Como en el for each que estamos analizando ya se accede a la tabla RESERVATION que lo contiene, entonces se aplicará un filtro automático por ese valor de CustomerId.

Una observación importante: que el atributo recibido en la regla parm pertenezca a la tabla extendida del for each NO HACE que éste lo aplique automáticamente como filtro. Para que eso suceda, el for each tiene que estar accediendo particularmente a esa tabla de la extendida para hacer algo.

Para entenderlo mejor, pensemos qué hubiera pasado si no quisiéramos listar los slots de un cliente dado *a partir* de una fecha dada, sino *en* una fecha dada. Como en ese caso los dos filtros serán por igualdad, podríamos querer recibir ambos parámetros directamente en los atributos correspondientes. Es decir, habríamos especificado:

```
Parm(in: ReservationDate, in: CustomerId);
```

Y en el Source:

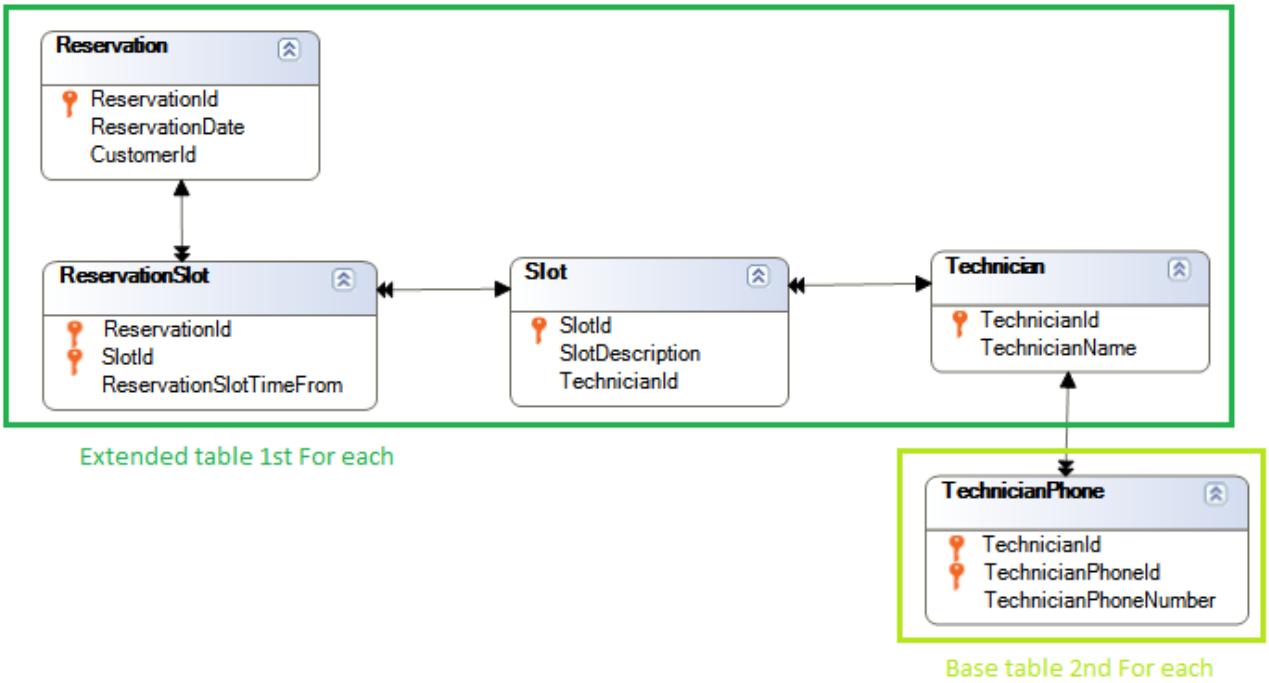
```
For each Reservation.Slot
print SlotPB
…
endfor
```

Pero al hacer esto, el primer for each ya no tendrá que acceder a la tabla Reservation, sino solo a Slot (para buscar el valor de SlotName). Y en este caso, entonces, no se aplicarán los filtros por igualdad, ya que son sobre la tabla Reservation. Puede constatarlo observando el listado de navegación. Se recorrerá toda la tabla ReservationSlot sin ningún filtro.

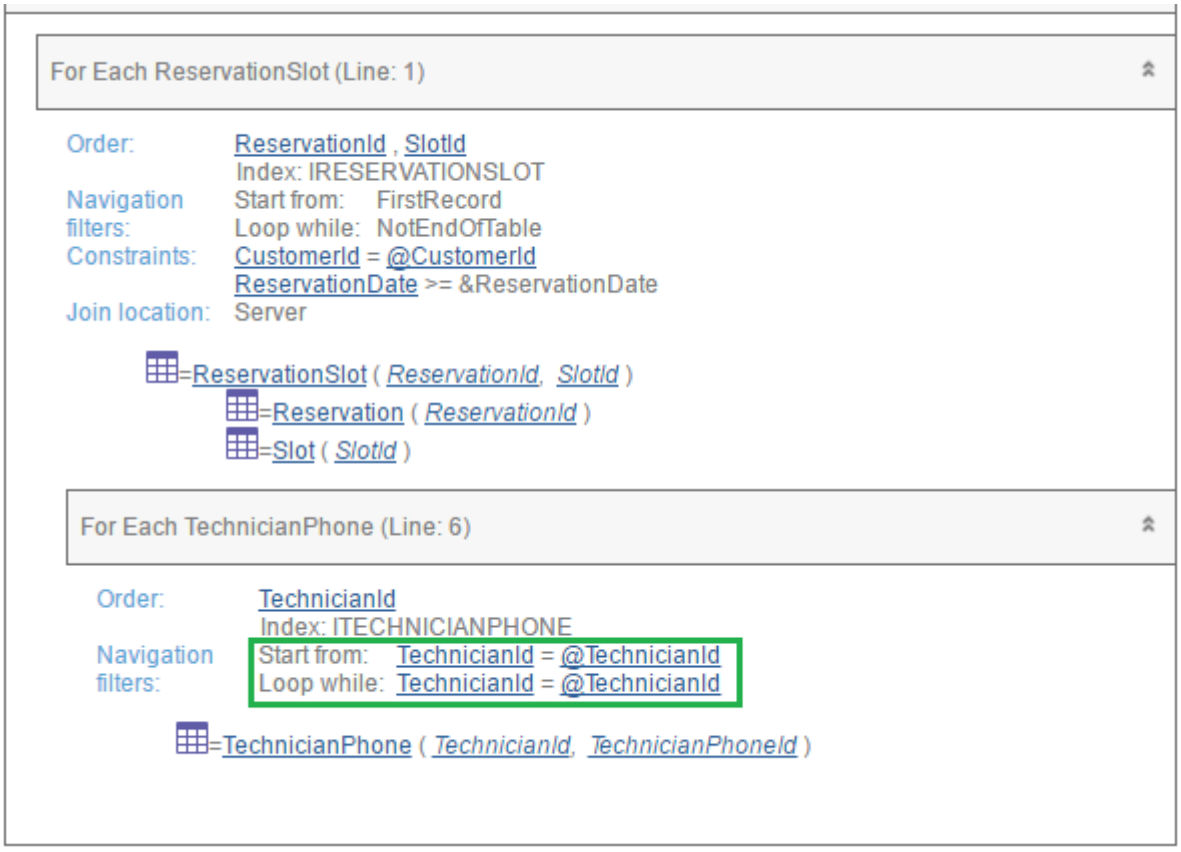
Por tanto deberá, al menos, declarar uno de los parámetros como variable, y utilizarla en un where filtrando por igualdad. Por ejemplo “where ReservationDate = &ReservationDate”. Como CustomerId está en esa misma tabla, ahora sí aplicará este filtro también.

Ahora pensemos qué pasa con el for each anidado. Su tabla base será claramente la asociada al nivel Phone de la transacción Technician (es decir, TechnicianPhone). La siguiente pregunta es: ¿establece filtros implícitos para la información que utilizará? Sabemos que sí, que mostrará los teléfonos del técnico de cada slot. ¿Por qué?

GeneXus busca si hay relación entre la tabla extendida del for each externo, y la tabla base del for each anidado:

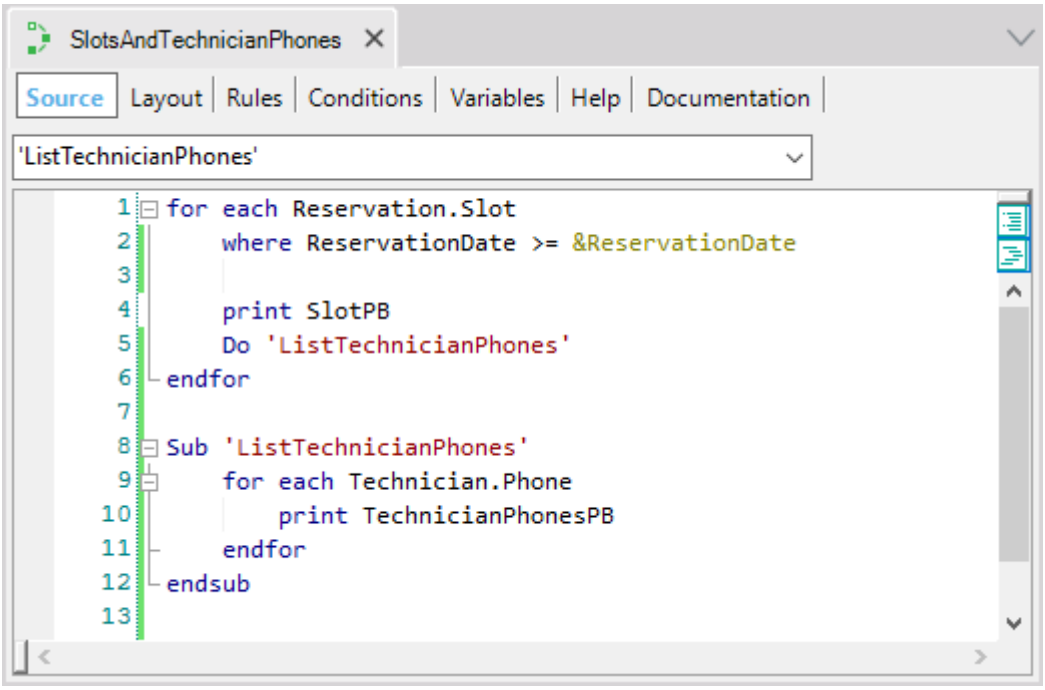


Es otra manera de buscar una relación 1 a N, aunque en este caso indirecta. Si cada reservation slot tiene un TechnicianId, y en la tabla a ser navegada también hay un TechnicianId, entonces GeneXus entiende que por la relación entre la información, se trataría del mismo. Por eso se realiza el join. Lo vemos claramente en el listado de navegación:

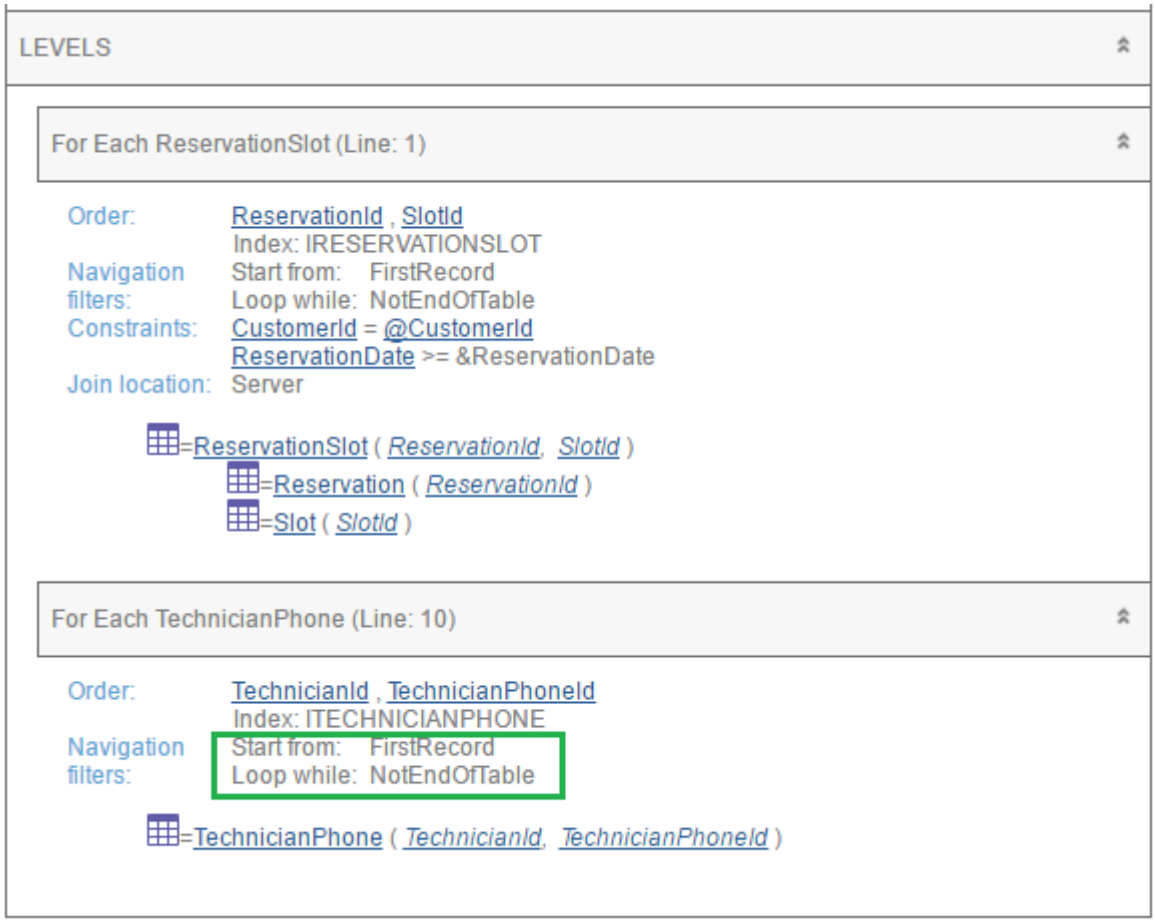


Donde siempre el @ está indicando información contextual (observe el @CustomerId, que refiere al valor recibido en el atributo CustomerId en la regla parm). Este @TecnhicianId refiere al valor del atributo de ese nombre de la tabla Slot accedida por el primer for each.

Si esto no fuera lo deseado por el programador, ¿hay manera de evitar esos filtros automáticos? La respuesta es sí, utilizando una subrutina para encapsular el código de este segundo for each (descuide, no tenía por qué saberlo; no se da en el curso GeneXus):



Si hace esto, en el listado de navegación ahora verá:

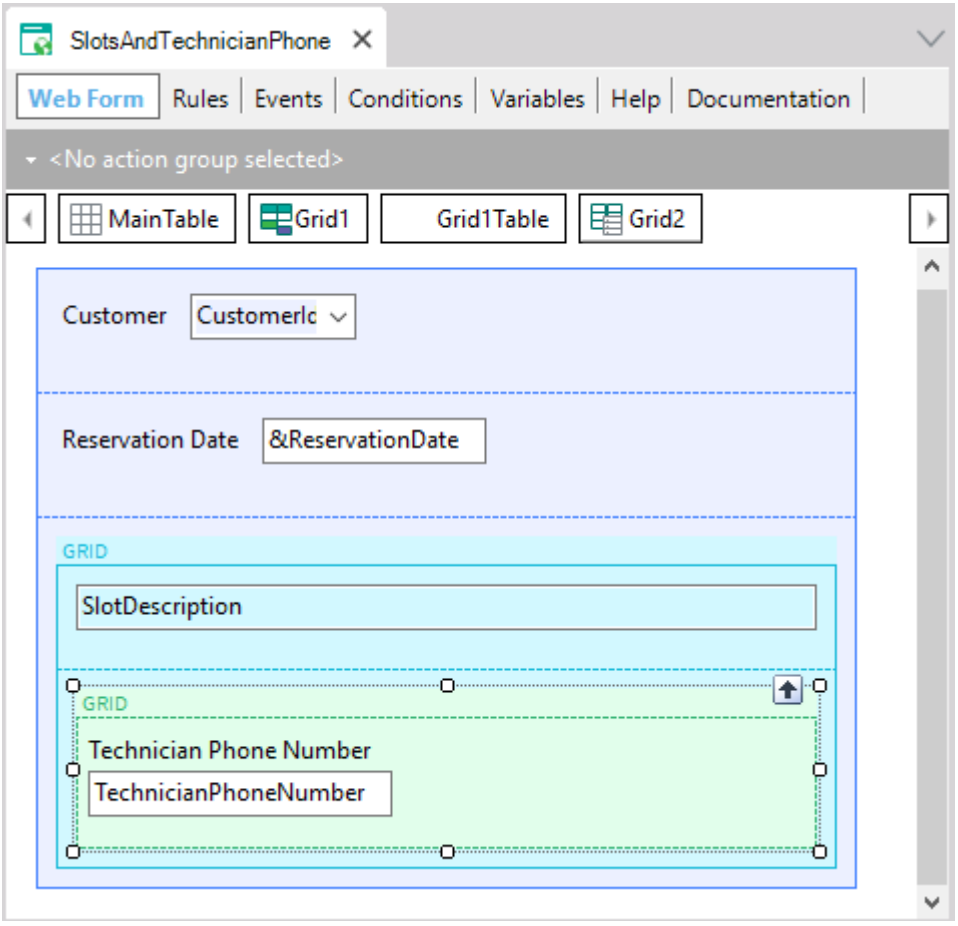


PREGUNTA 17 – GRIDS ANIDADOS

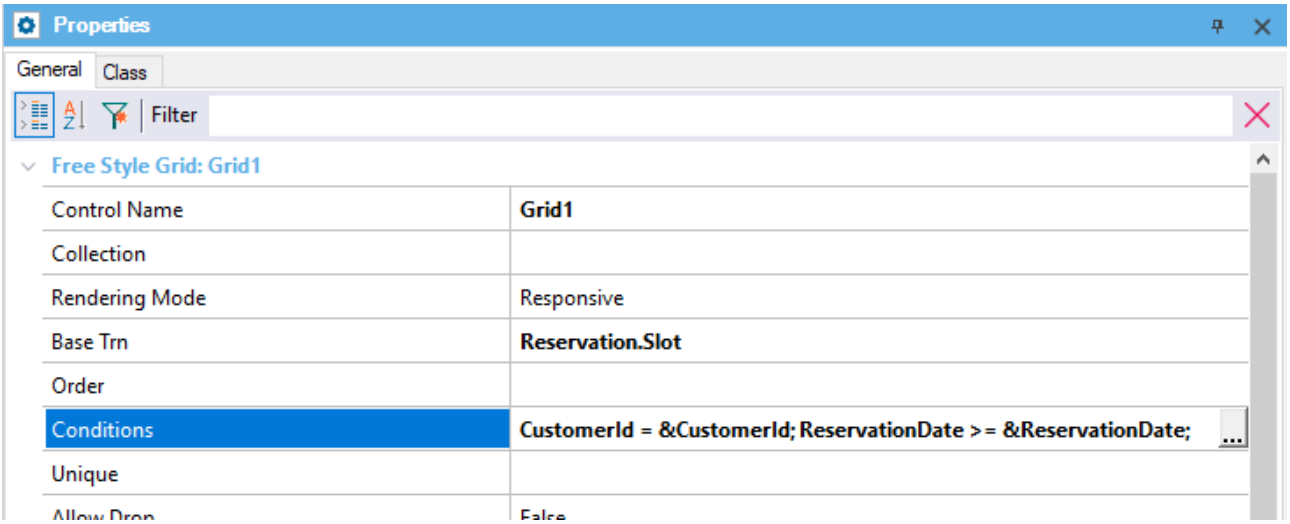
Ahora queremos implementar el mismo requerimiento que en la pregunta anterior, pero en vez de hacerlo en un listado pdf, queremos realizarlo en una pantalla web.

RESPUESTA

Para implementar esto mismo como una pantalla web y no un pdf, podríamos colocar dos variables para que el usuario ingrese allí los valores que desee de cliente y fecha en cada oportunidad, y luego la información se muestre con un par de grids anidados, donde el primero NECESARIAMENTE debe ser freesltyle. NO puede ser un grid estándar. ¿Por qué? Porque un grid estándar solo admite columnas fijas, de tipo atributo/variable. Y aquí lo que queremos colocar dentro del grid es, además del atributo SlotDescription, ¡OTRO GRID!

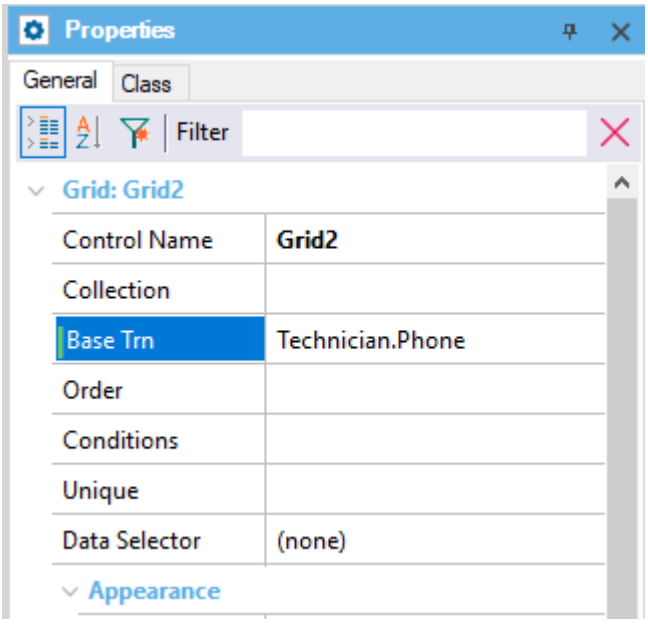


Al grid Freestyle le indicamos también su transacción base, igual que como lo hicimos con el For each externo en el procedimiento, y los filtros los escribimos a través de la propiedad Conditions del grid (que equivale a las cláusulas where del for each):



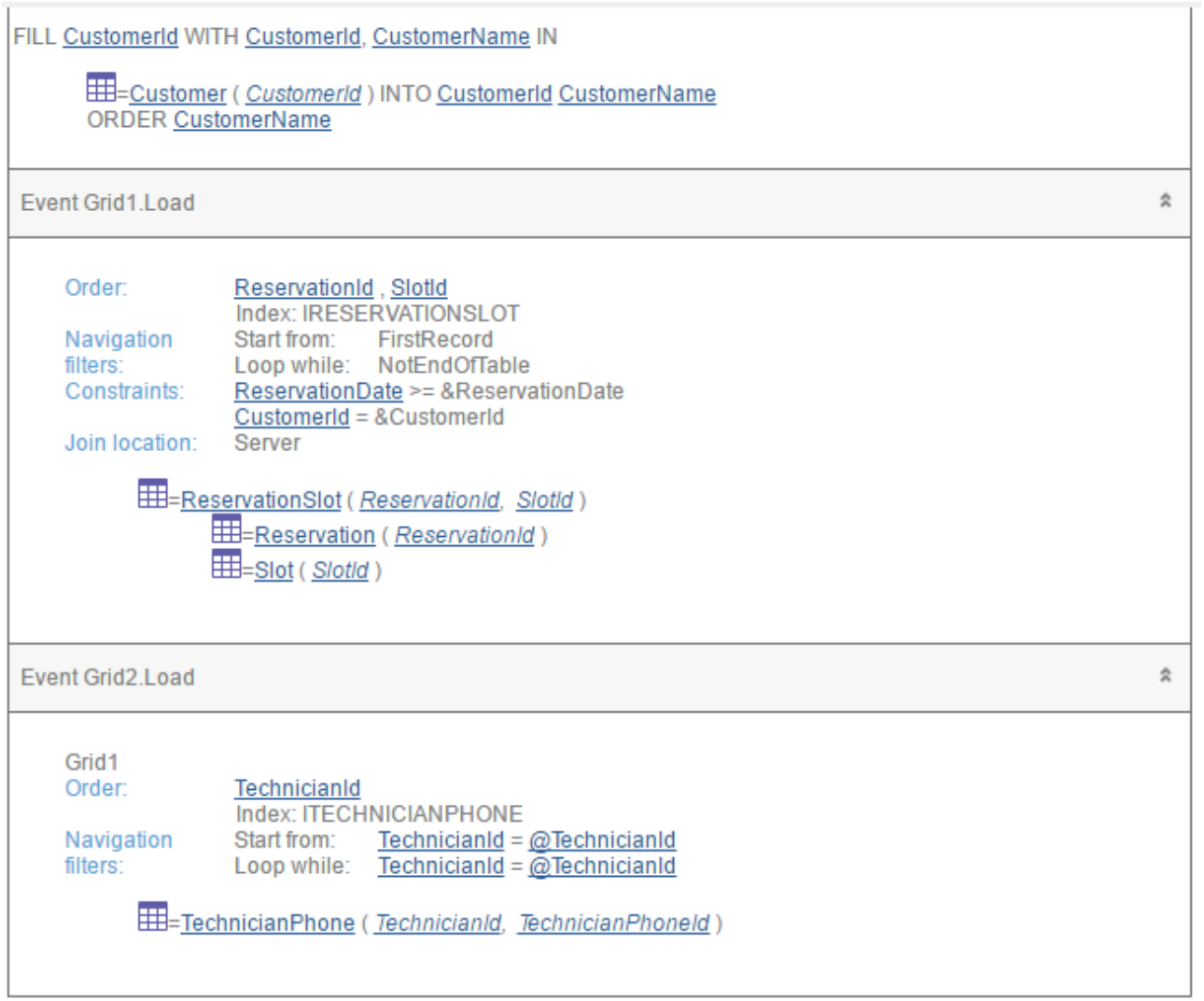
Si la condición por CustomerId no la hubiéramos escrito utilizando una variable en el propio form, sino que hubiésemos recibido al CustomerId por parámetro, valdría exactamente lo mismo que vimos en el caso del procedimiento.

¿Qué pasa con la información del grid anidado? Aquí hemos colocado un grid estándar, pero bien podría ser un freestyle también. Observemos que lo único que hemos hecho es colocar en él el atributo TechnicianPhoneNumber. Para asegurarnos la navegación que queremos, podríamos especificar también su transacción base, lo que sugerimos siempre como buena práctica:



Nos podemos preguntar si en el caso de grids anidados valdrá lo mismo que para el caso que ya estudiamos de for eachs anidados. Y la respuesta es sí.

Es decir, aplicará el filtro por TechnicianId del slot del que se trate, exactamente igual que para el caso de for eachs anidados del procedimiento.



Obsérvese, también, que hemos modificado el Control Type de la variable &CustomerId, para que muestre en ejecución un combo box con los valores de CustomerName de la tabla Customer, de manera tal que cuando el usuario selecciona el nombre de cliente deseado, internamente se asigna en la variable su CustomerId. Esto es utilizando la propiedad Control Type = “Dynamic Combo Box” del control &CustomerId. El listado de navegación nos está indicando al principio la navegación de la tabla Customer para resolver esto.

Si en lugar de haber especificado los grids como anidados los hubiésemos presentado como paralelos, allí sí tendríamos que haber programado explícitamente para el grid de teléfonos de los técnicos el filtro por TechnicianId en sus condiciones. Porque para grids paralelos, al igual que lo que sucede para for eachs paralelos, las navegaciones no se relacionan. Y es lógico, ¿por qué habrían de relacionarse automáticamente, si no hay nada que haga entender que se quieren vincular? En el caso de grids anidados, uno claramente pensaría que lo más probable es que si la info está vinculada en la realidad, también se la quiera presentar vinculada.

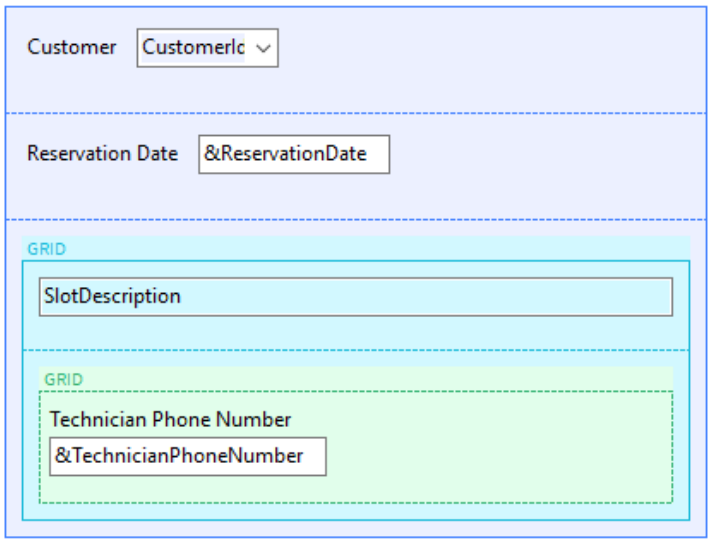
En el caso del listado pdf que vimos, cuando no queríamos que GeneXus aplicara los filtros implícitos al for each anidado creábamos una subrutina, cuyo efecto era no tener en cuenta el contexto. ¿Cómo haríamos aquí si por cada SlotDescription de la reserva quisiéramos ver en el grid anidado todos los teléfonos de los técnicos, y no los de los del slot? Es decir, ¿cómo hacemos para que no aplique el filtro implícito, que es el que realiza el join?

La forma más fácil será que el grid anidado sea sin tabla base. Cuando el grid es sin tabla base, toda su carga queda en manos del desarrollador, a través del evento Load del grid.

Viendo el listado de navegación anterior (el de los grids con tablas base), se indica el evento Load para cada grid y bajo éste se indica una navegación de tabla. El alumno podría pensar entonces que el evento Load se dispara una única vez, donde se ejecuta esa suerte de for each implícito. No hay que confundirse: sabemos que cuando los grids tienen tabla base, se dispara un evento Load por cada registro accedido en la navegación, inmediatamente antes de cargar su información en el grid.

Solamente ocurre que el evento Load de un grid se dispara una única vez cuando no tiene tabla base.

Entonces lo que haríamos sería modificar en el grid anidado el atributo TechnicianPhoneNumber por una variable:



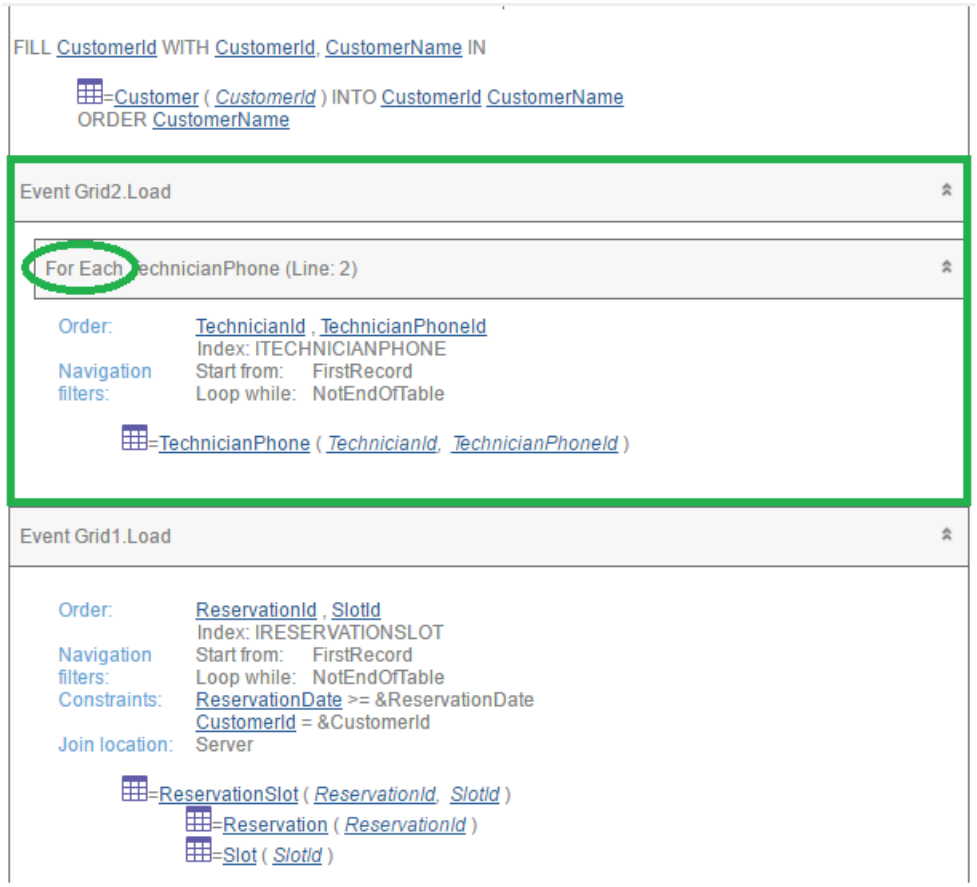
Pero además debemos tener cuidado de quitar toda referencia a tabla o atributo en cualquier otro lado relacionado con el grid. Obviamente debemos vaciar la propiedad Base Trn.

Y en el evento Load del grid:

```
Event Grid2.Load()
    for each Technician.Phone
        &TechnicianPhoneNumber = TechnicianPhoneNumber
        Load
    endfor
endevent
```

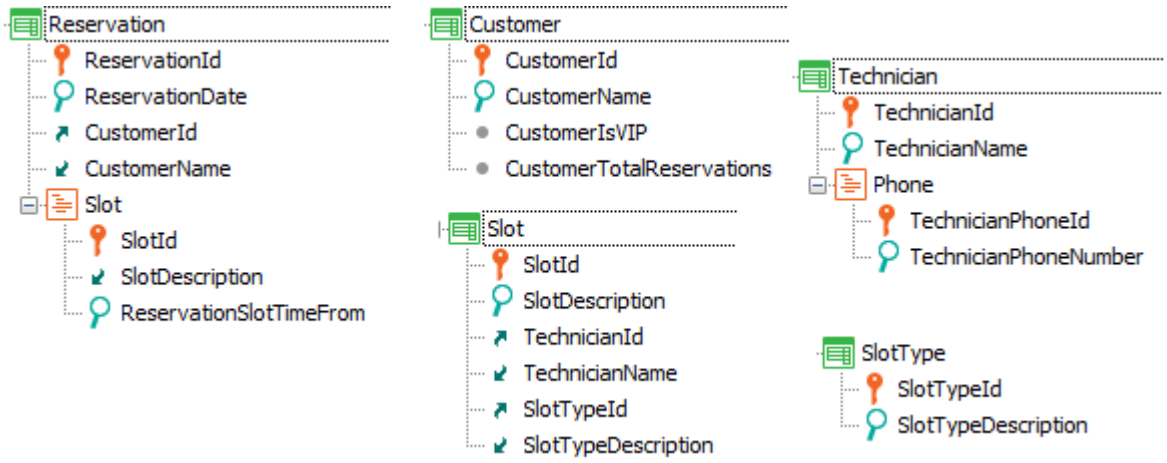
Si no colocamos el comando Load dentro del evento ¿qué pasará? No se agregará ninguna línea al grid. Es decir, no se cargará ningún teléfono, aunque sí se navegue su tabla.

Viendo el listado de navegación, ahora, vemos claramente la diferencia entre el grid con tabla base y el grid sin tabla base:



PREGUNTA 18 – GRIDS ANIDADOS (CONTINUACIÓN)

Extendiendo la pregunta anterior, hemos agregado ahora para cada Slot un tipo, que se registra en una transacción aparte:

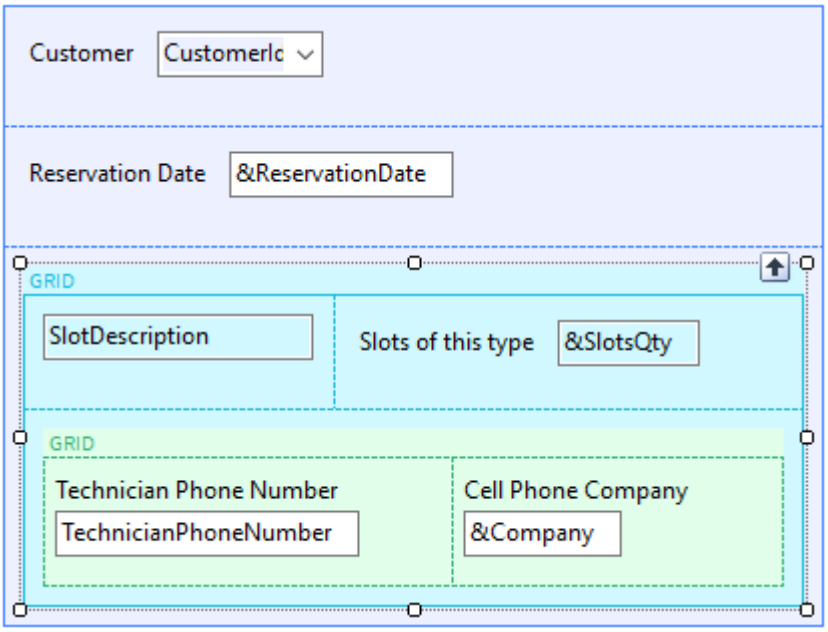


Se desea, ahora, que el web panel que muestra los slots reservados a un cliente a partir de una fecha dada muestre además:

1. para cada slot listado la cantidad de slots del mismo tipo existentes en el Casino; y
2. para cada teléfono del técnico asociado, la compañía telefónica correspondiente (que supongamos no se registra en la base de datos, sino que se encuentra con un proc a partir del análisis de los primeros dígitos del número –eventualmente podría ser un servicio externo-).

Analice por qué la solución encontrada a continuación no funciona (suponiendo que el procedimiento CellPhoneCompany está bien programado) y plantee una solución completa, y bien desarrollada. Pruebe discutir otras posibilidades.

Web Form:



Events:

```
Event Grid1.Load()
      &SlotsQty = count( SlotDescription, SlotTypeId = SlotTypeId)
Endevent

Event Grid2.Load()
      &Company = CellPhoneCompany(TechnicianPhoneNumber)
      Load
endevent
```

Como ayuda: uno de los problemas detectados por el alumno es que siempre se muestra valor 0 para los Slots of this type, cuando en verdad siempre hay por lo menos un slot de ese tipo: el que se está cargando en el grid.

Entonces intenta la siguiente modificación, y le sigue mostrando 0:

```
Event Grid1.Load()
&SlotTypeId = SlotTypeId
&SlotsQty = count( SlotDescription, SlotTypeId = &SlotTypeId)
Endevent
```

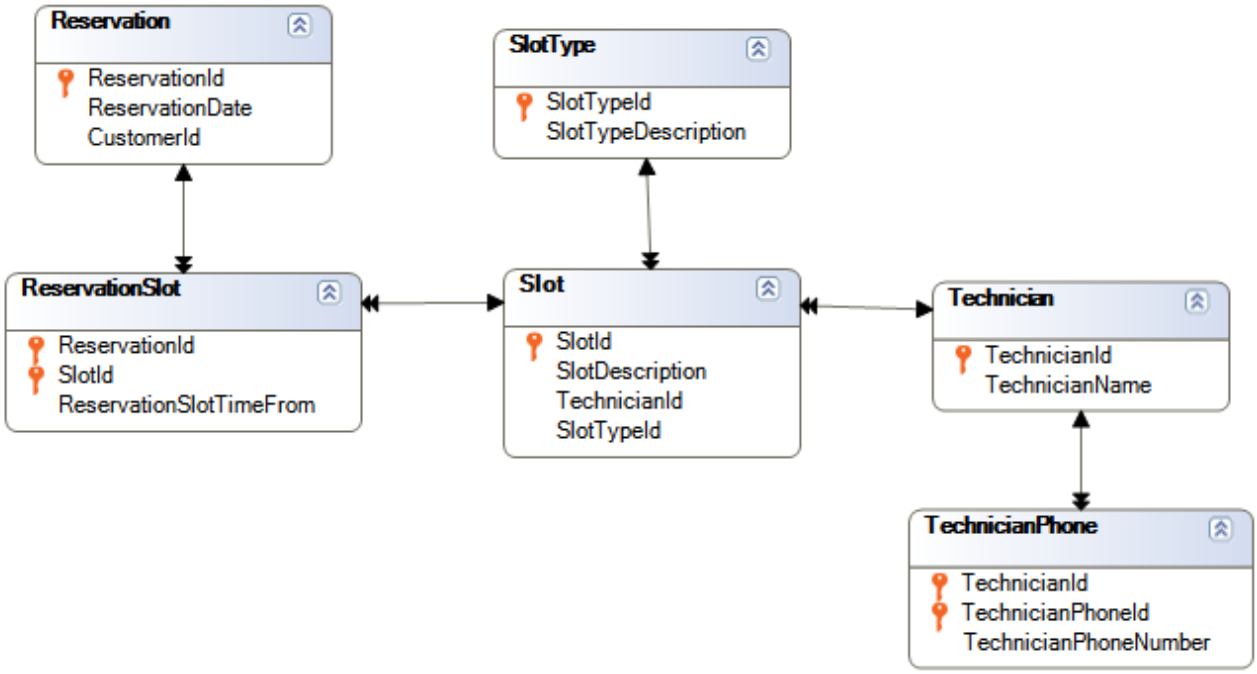
RESPUESTA

Se han agregado dos variables, &SlotsQty para el grid externo (el de nombre Grid1) y &Company para el grid interno, &Company.

Ambos grids son con tabla base, por lo que los atributos se cargarán automáticamente en ambos grids, de acuerdo a sus navegaciones.

Pero para cargar las variables, ¿cómo hacemos? Si un grid tiene tabla base, sabemos que se disparará n veces el evento Load del grid: una vez por cada línea a ser cargada, inmediatamente antes de cargarla, y el desarrollador no tiene necesidad de escribir el comando Load. Por tanto es en el evento Load donde se tiene que asignar valor a las variables que se hayan incluido en el grid.

Empecemos por el grid más externo. Es en el evento Load en el que tenemos que realizar el cálculo de la cantidad existente de slots del mismo tipo que el que se está por cargar en esa línea que disparó el Load. Por tanto hay un contexto para todo lo que se escriba dentro del Load: estamos posicionados en un registro de la tabla base, ReservationSlot, y en cada uno de los registros asociados por tabla extendida que necesitemos. Dicho de otro modo, estarán instanciados los atributos de la tabla ReservationSlot para ese SlotId, de la tabla Slot para ese slot si los necesitamos, de la tabla SlotType para ese slot si los necesitamos (así como de Technician para ese slot, y de Reservation).



Para calcular la cantidad de slots de la tabla Slot que tienen el mismo SlotTypeId que el del slot instanciado ya dentro del Load, ¿cómo hacemos? El alumno escribió inicialmente:

```
Event Grid1.Load()
      &SlotsQty = count( SlotDescription, SlotTypeId = SlotTypeId)

Endevent
```

¿Qué intentó hacer? ¿Por qué condicionó de esa manera los registros a ser contados con la fórmula count?

Seguramente haya probado antes de agregar esa condición con la fórmula sin condiciones:

```
Event Grid1.Load()
      &SlotsQty = count( SlotDescription)
Endevent
```

Tal vez pensando que por contexto ya iba a contar solamente los slots de ese tipo. Analicemos por qué no es así. Sabemos que las fórmulas inline determinan la tabla a ser navegada por los atributos especificados en ella, sin atender al contexto. Luego, atienden al contexto para agregar los filtros que correspondan. En nuestro caso, la tabla a ser navegada por el count será, claramente, Slot. Pero como la fórmula está dentro de un contexto en el que estamos posicionados en un registro de ReservationSlot que tiene un SlotId, aplicará ese filtro por igualdad. Entonces siempre contará un único registro, el que corresponda a ese SlotId.

Evidentemente no podemos utilizar la fórmula dentro del contexto del evento Load, porque necesitamos que no realice ese filtro por SlotId. Todas las soluciones que buscó el alumno son incorrectas puesto que no puede romper con ese contexto y, por tanto, de que se agregue automáticamente el filtro por SlotId.

No tendremos otra alternativa que hacer el cálculo en un procedimiento, para romper con esa contextualización.

```
Event Grid1.Load()
      &SlotsQty = SlotsQtyOfAType(SlotTypeId)
endevent
```

Y programaríamos el procedimiento SlotsQtyOfAType:

Rules:

```
      parm(in: &SlotTypeId, out:&SlotsQty);
```

Source:

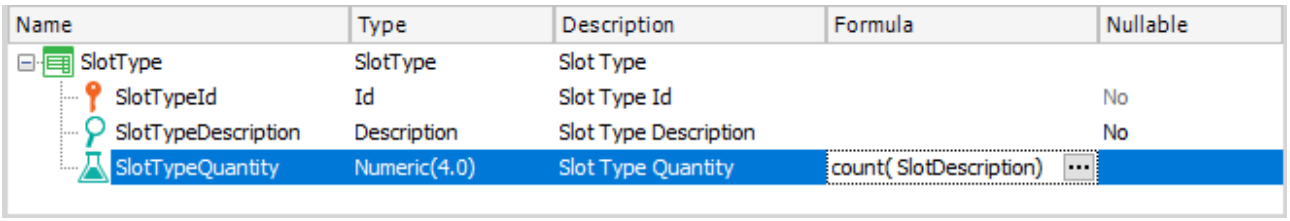
```
      &SlotsQty = count( SlotDescription, SlotTypeId = &SlotTypeId)
```

O utilizar una subrutina, para no salirnos del propio código, sobre todo si el proc no va a ser utilizado por ningún otro objeto:

```
Event Grid1.Load()
      &SlotTypeId = SlotTypeId
      Do 'SlotsQuantity'
endevent

Sub 'SlotsQuantity'
      &SlotsQty = count( SlotDescription, SlotTypeId = &SlotTypeId)
Endsub
```

En las subrutinas no se pueden pasar parámetros, y como su código se encuentra dentro del mismo programa que las invoca, ven el estado de las variables tal como están en el momento en que la subrutina fue invocada. Por eso cargamos el valor de la variable &SlotTypeId, para que la subtrutina pueda utilizarla.

¿Y si al alumno se le hubiera ocurrido definir en la transacción SlotType un atributo fórmula como mostramos a continuación?



Name	Type	Description	Formula	Nullable
SlotType	SlotType	Slot Type		
SlotTypeId	Id	Slot Type Id		No
SlotTypeDescription	Description	Slot Type Description		No
SlotTypeQuantity	Numeric(4.0)	Slot Type Quantity	count(SlotDescription)	

Y en el web form colocar ese atributo en lugar de la variable &SlotsQty, o, si no quiere tocar el web form, escribir en el Load:

```
Event Grid1.Load()
&SlotsQty = SlotTypeQuantity
endevent
```

¡Funcionará correctamente! Observe que no es lo mismo, entonces, un atributo fórmula (global) que una fórmula inline (local). Este ejemplo deja más claro el efecto de un ATRIBUTO fórmula.

Evidentemente no será una buena práctica llenar de atributos fórmula una transacción solamente porque para el caso particular de la implementación de un panel nos sirve. Hay que encontrar el balance costo/beneficio. Si este cálculo va a ser requerido en otros lugares, entonces allí tiene pleno sentido la definición de la fórmula a nivel de un atributo en la transacción.

Otra opción habría sido implementar el grid sin tabla base. Es decir, sin transacción base y sin atributos “sueltos” en los eventos relacionados. Y, por supuesto, con variables en el grid (así, en vez del atributo SlotDescription insertaremos una variable &SlotDescription). Pero tendremos el mismo problema que antes con la anidación. El evento Load nos quedará:

```
Event Grid1.Load()
      for each Reservation.Slot
            &SlotDescription = SlotDescription
            &SlotTypeId = SlotTypeId
```

```
        Do 'SlotsQuantity'
              Load
endfor
endevent

Sub 'SlotsQuantity'
&SlotsQty = count( SlotDescription, SlotTypeId = &SlotTypeId)

Endsub
```

No podemos olvidar escribir el comando Load para cargar, efectivamente, la línea en el grid.

Comando Load en un evento Load de un grid con tabla base

Ahora nos queda analizar la carga del grid anidado. Claramente el comando Load es innecesario.

```
Event Grid2.Load()
       &Company = CellPhoneCompany(TechnicianPhoneNumber)
       Load
endevent
```

Pero, ¿cuál es el efecto? No tiene efecto en este caso. Lo que veremos en ejecución es exactamente lo mismo. Sin embargo, escribir explícitamente el comando Load hace que GeneXus no coloque implícitamente al final de su código a este mismo comando. Esto es útil si quiero, como desarrollador, que solo algunos registros que "pasan" por el evento Load sean efectivamente cargados en el grid, y no todos.

Por ejemplo, si quiero que si la compañía es "X" entonces no se cargue el teléfono en el grid:

```
Event Grid2.Load()
&Company = CellPhoneCompany(TechnicianPhoneNumber)
If &Company <> !'X'
       Load
endif

endevent
```

Incluso, si quiero cargar más de una línea del grid en el mismo evento Load para un registro, entonces tendré que escribir dos veces el comando Load dentro del evento.

Con estos ejemplos mostramos que puede que se necesite en algunos casos escribir explícitamente el comando Load en un evento Load de un grid CON tabla base.

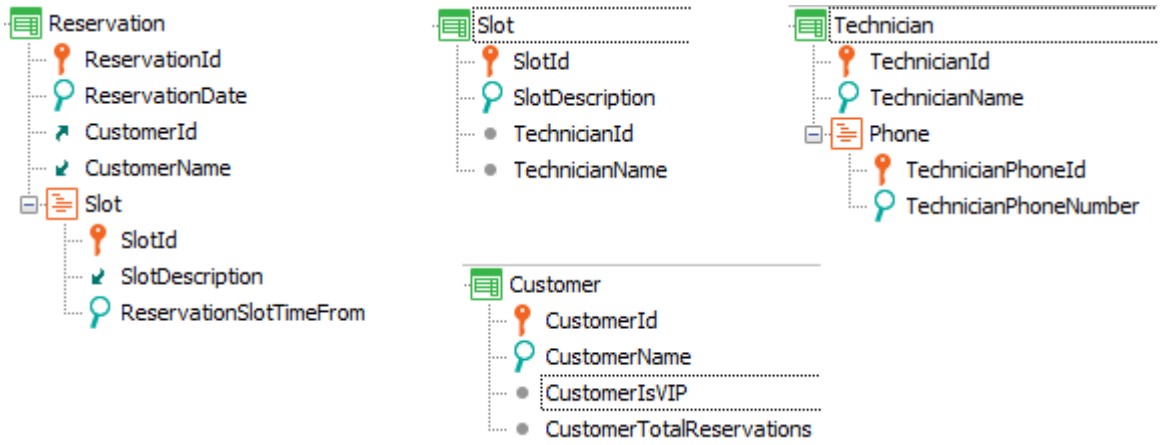
Cuándo colocar atributos ocultos en un grid

Un error frecuente que encontrará en implementaciones de alumnos: colocar el atributo TechnicianId en el grid anidado, pero oculto. ¿Por qué es innecesario? Nada de lo que necesite en

el evento Load requiere que el atributo esté en el grid, debido a que el momento de dispararse el evento Load es cuando se está en la base de datos, con acceso a la tabla extendida. Se necesitaría dejar el TechnicianId cargado en el grid solamente si se lo necesita después, en un evento de usuario, una vez que la info ya fue cargada en el grid, y ya no estamos más conectados a la base de datos, sino que estamos trabajando con la información de pantalla, exclusivamente.

PREGUNTA 19 – ¿UNIQUE O CORTE DE CONTROL?

Siguiendo con la aplicación:



Suponga que se quiere hacer un listado pdf que muestre para cada slot que ha sido reservado alguna vez, la cantidad de veces que se lo ha reservado.

Un alumno A presenta la siguiente solución:

```
For each Slot
	&slotsInReservations = count( ReservationSlotTimeFrom)
	print SlotPB //con el atributo SlotDescription y la variable &slotsInReservations

endfor
```

Mientras que un alumno B presenta esta otra:

```
For each Slot
	&slotsInReservations = 0
	For each Reservation.Slot
		&slotsInReservations += 1
	endfor
	print SlotPB //con el atributo SlotDescription y la variable &slotsInReservations
endfor
```

Usted les indica que ambas soluciones son equivalentes, aunque el alumno B está implementando con el for each interno lo que ya la fórmula count del alumno A realiza y lo hace mejor, dado que está pensada para estar optimizada. Generalmente (aunque no siempre) es mejor utilizar una fórmula que hacer a mano lo que la fórmula ya hace.

Pero más allá de eso, usted les indica que ambas soluciones erran en una cosa: no se les está pidiendo un listado de todos los slots, incluyendo aquellos que nunca fueron reservados.

Entonces el alumno A plantea la siguiente solución:

```
for each Reservation.Slot
       unique SlotId
       &slotsInReservations = count( ReservationSlotTimeFrom )
       print SlotPB //con el atributo SlotDescription y la variable &slotsInReservations
endfor
```

Mientras que el B plantea esta otra:

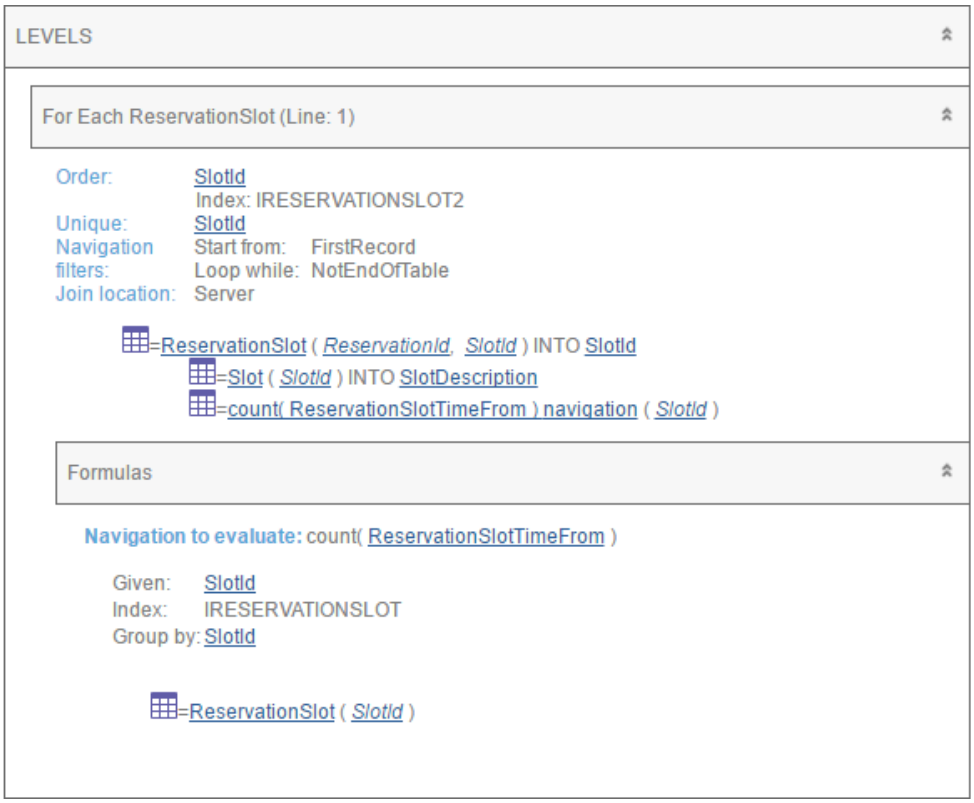
```
for each Reservation.Slot
       order SlotId
       &slotsInReservations = 0
       for each Reservation.Slot
              &slotsInReservations += 1
       endfor
       print SlotPB //con el atributo SlotDescription y la variable &slotsInReservations

endfor
```

¿Les dirá que ambas soluciones son equivalentes?

RESPUESTA

Sí, ambas serán equivalentes. De hecho, si observa los listados de navegación.

Este es el de A:



Donde puede ver que la fórmula aplica el filtro por SlotId (observe el “Given: SlotId”).

Y este el de B:

B implementó un corte de control, mientras que A utilizó la cláusula unique que fue pensada para este tipo de escenarios, donde se quiere navegar una tabla y realizar una agregación sobre esa misma tabla, de acuerdo a algún atributo o atributos que queremos utilizar una sola vez en la salida.

Es más probable que una solución que se ha implementado para resolver determinados escenarios y que contiene una fórmula tendrá más chances de ser resuelta eficientemente, antes que una solución en la que debemos implementar lo que la fórmula ya trae implementado. Pero esto no siempre es así.

A los alumnos podría entonces quedarles la idea de que siempre que necesiten implementar un corte de control podrían evitarlo utilizando la cláusula unique, pero esto no es cierto.

Por ejemplo, si lo que se pidiera no fuera una agregación, sino que para cada slot que ha sido reservado alguna vez se muestren las fechas de reserva, no tendremos otra alternativa que implementar un corte de control:

```
for each Reservation.Slot
       order SlotId
       print SlotPB   //con el atributo SlotDescription
       for each Reservation.Slot
              print ReservationPB   //con el atributo ReservationDate
       endfor

endfor
```

No hay manera de hacer esto con una cláusula unique.

Pregúntele al alumno si hay alguna diferencia entre colocar en la cláusula order el atributo SlotId y colocar el atributo SlotDescription.

Pero, por otro lado, para el problema original, ¿no es una mejor solución la que implementó el alumno A (join) pero simplemente agregando un if?:

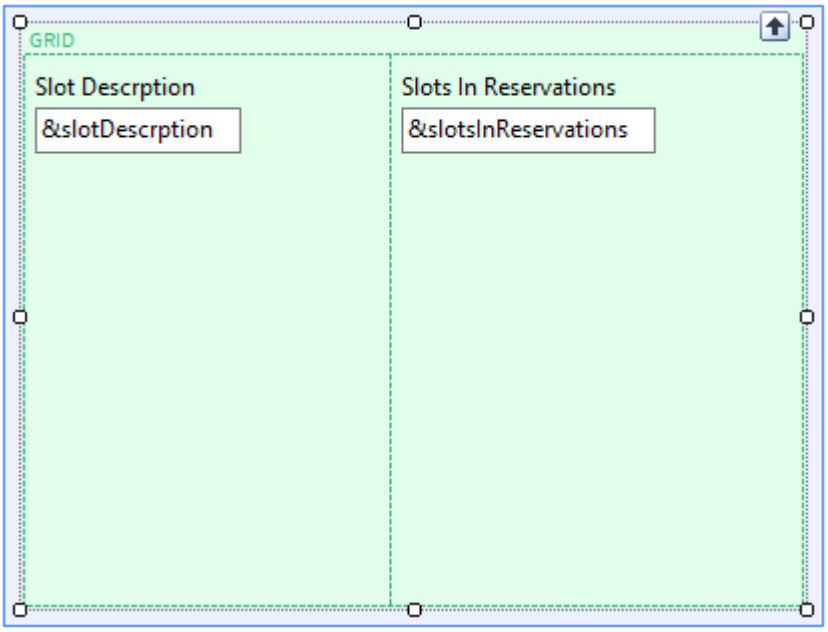
```
For each Slot
	&slotsInReservations = count(ReservationSlotTimeFrom)
	If &slotsInReservations > 0
		print SlotPB //con el att SlotDescription y la var &slotsInReservations
	endif

endfor
```

Es de esperar que haya una cantidad sensiblemente menor de registros en la tabla Slot que en la ReservationSlot. ¿Habrá diferencia entre esta solución y la del mismo alumno pero navegando solo ReservationSlot con unique?

PREGUNTA 20 – UNIQUE EN GRID (CONTINUACIÓN)

Ahora se pide que en vez de resolver lo anterior como pdf lo resuelva en un web panel.

El alumno entonces decide que tendrá que insertar un grid sin tabla base:



Para poder codificar en el evento Load su solución con la cláusula unique:

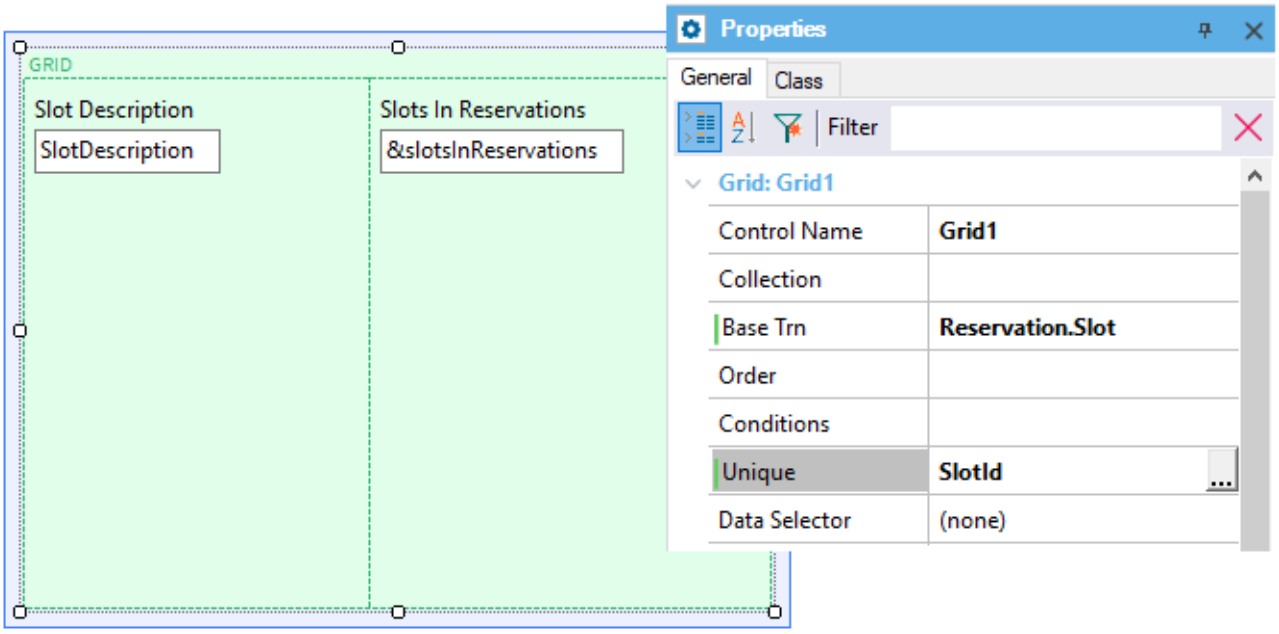
```
Event Grid1.Load()
	for each Reservation.Slot
		unique SlotId
		&slotDescrption = SlotDescription
		&slotsInReservations = count( ReservationSlotTimeFrom )
		Load
	endfor

endevent
```

Puesto que no tiene manera de hacer esto mismo con un grid con tabla base. Explíquele cómo se haría.

RESPUESTA

Es raro que lo que puede hacerse con for each no pueda hacerse con un grid con tabla base, dado que en ese caso por detrás hay una suerte de for each. Lo mismo podemos decir respecto a un grupo de data provider. Son elementos equivalentes en su lógica.

Por tanto alcanzará con que le pregunte al alumno si no existirá acaso una propiedad Unique a nivel del grid.




```
Event Grid1.Load()
      &slotsInReservations = count(ReservationSlotTimeFrom)
endevent
```

PREGUNTA 21 – UNIQUE EN DATA PROVIDER Y TRANSACCIÓN DINÁMICA (CONTINUACIÓN)

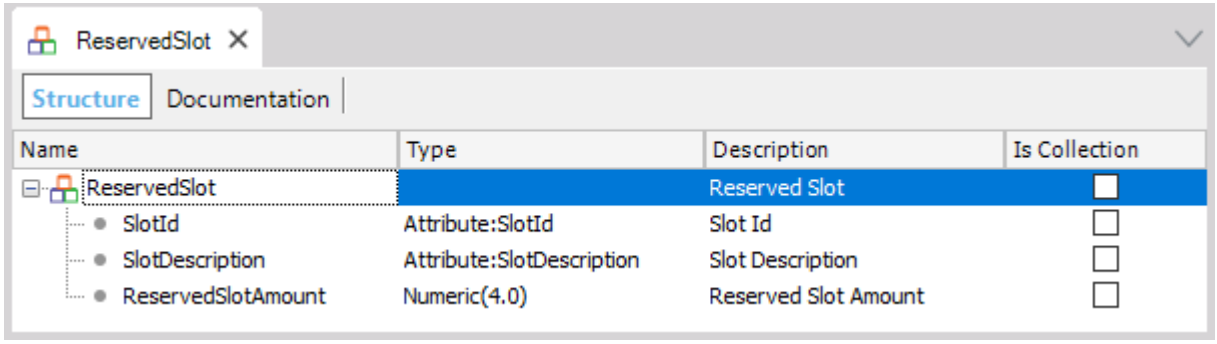
Usted desea que los alumnos puedan explorar las soluciones más interesantes para resolver el requerimiento, integrando los conceptos brindados en el curso. Por lo que ahora les plantea: "Si ahora deseamos mostrar en el grid (o en el pdf) los slots ordenados por cantidad de reservas, de mayor a menor, ¿cómo lo implementa?"

Imagine las posibles respuestas.

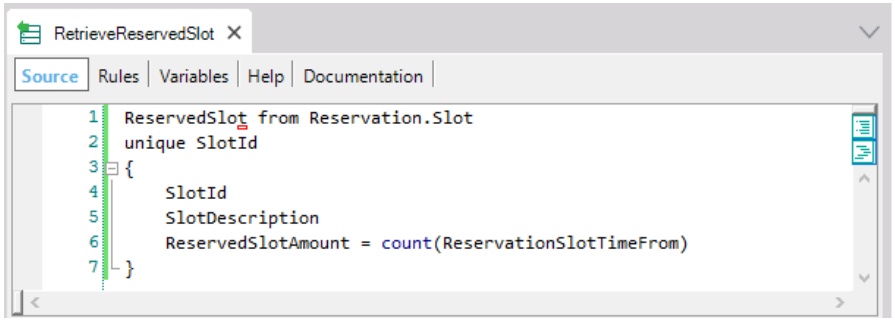
RESPUESTA

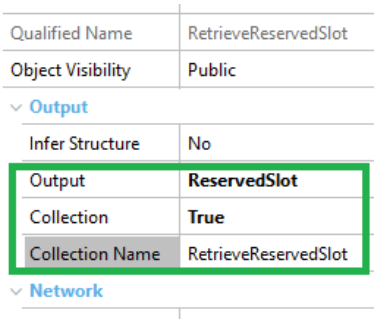
Algunos alumnos podrán pensar en cargar en un SDT colección los valores de SlotDescription y &slotsInReservations y luego en el web panel o procedimiento recorrer la colección ordenada.

Así, definir por ejemplo el SDT:



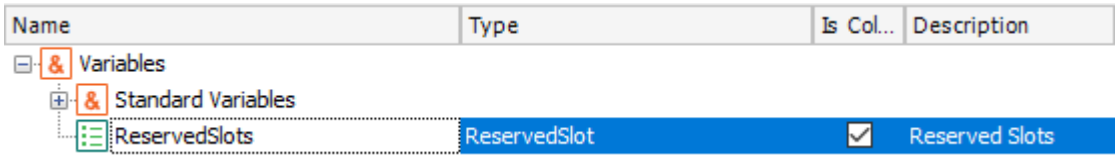
Y cargarlo con un Data Provider:



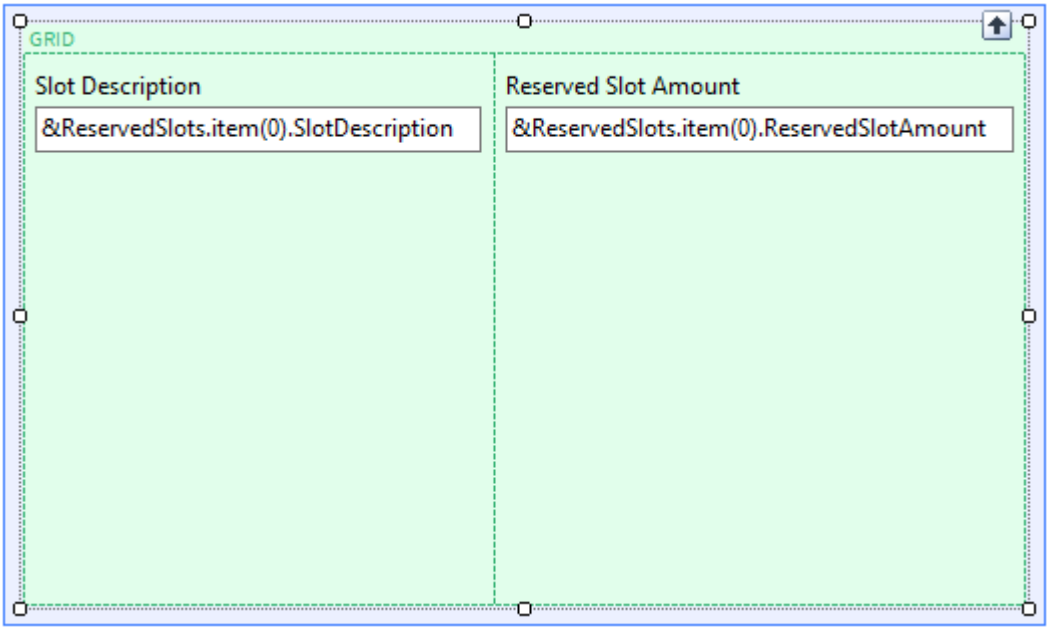


Observe que es análoga la forma de declarar el grupo del data provider que la de codificar el for each con la cláusula unique en el listado o en el web panel sin tabla base.

Entonces, luego en el web panel se podría definir una variable colección del SDT:



Insertarla en el web form, tras lo que GeneXus automáticamente la presentará en un grid:



Que también mostrará automáticamente con el contenido que tenga la variable colección antes de ejecutarse el Load.

Por lo que un buen momento para cargarla será el evento Refresh (si lo hiciéramos en el Start, se ejecutará una única vez. En cambio el Refresh se ejecutará toda vez que se refresque, y como estos datos podrían variar de un momento a otro (alcanzará con que se creen nuevas reservas), tal vez sea preferible el evento Refresh).

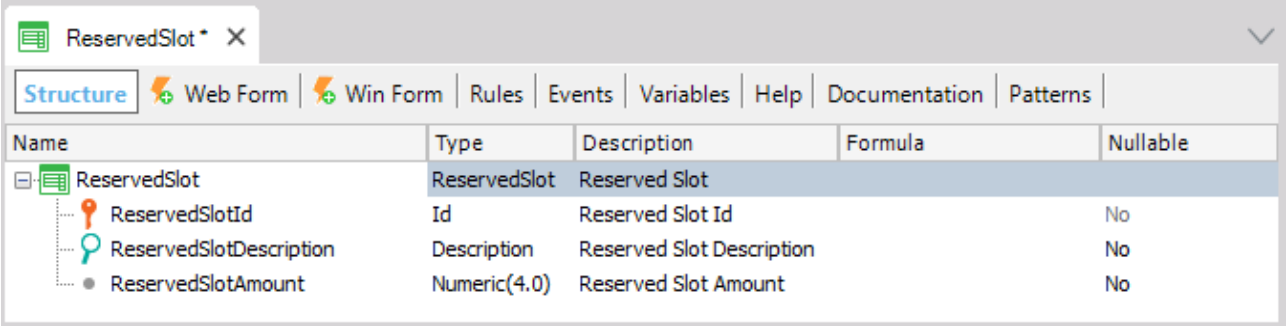
```
Event Grid1.Refresh()
       &ReservedSlots = RetrieveReservedSlot()
       &ReservedSlots.Sort("[ReservedSlotAmount]")
endevent
```

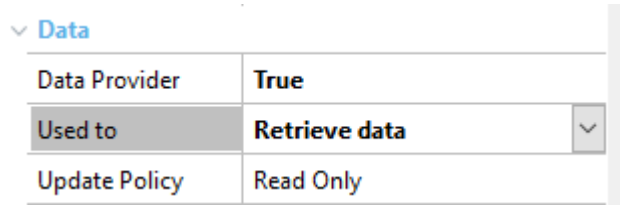
Observemos que aquí invocamos al Data Provider para que nos devuelva la colección cargada, y luego lo que hacemos es ordenar esa colección de manera descendente por ReservedSlotAmount.

Como a continuación del evento refresh se ejecutará la carga del grid, con esto tendremos el requerimiento resuelto.

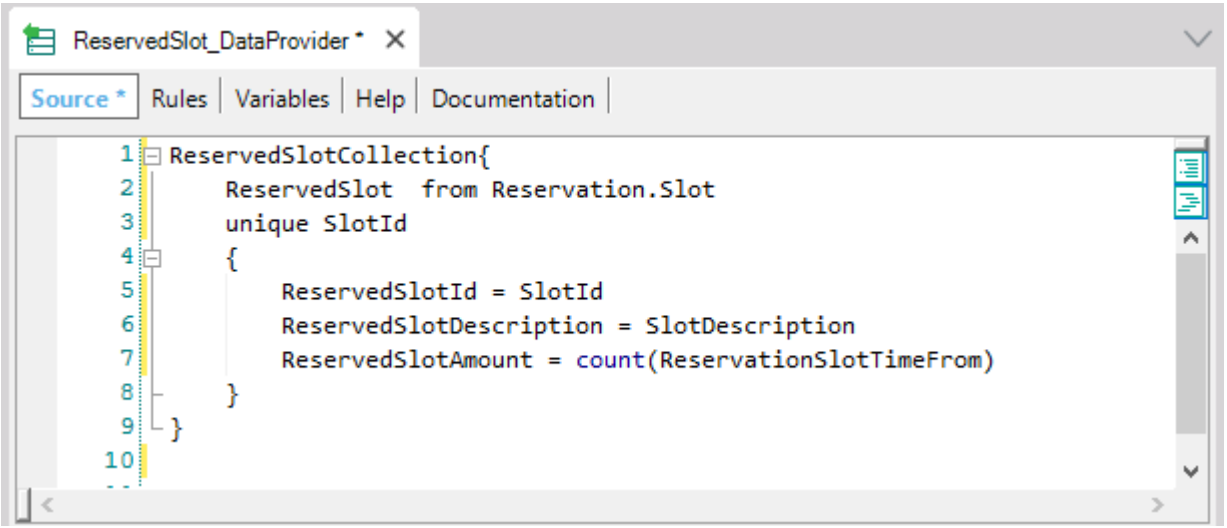
SOLUCIÓN CON TRANSACCIÓN DINÁMICA

Otros alumnos, sin embargo, pensarán en una solución equivalente, muy parecida, pero que puede resultar mejor: ¿por qué en vez de definir un SDT colección y un Data Provider que lo cargue, no definir una transacción dinámica?

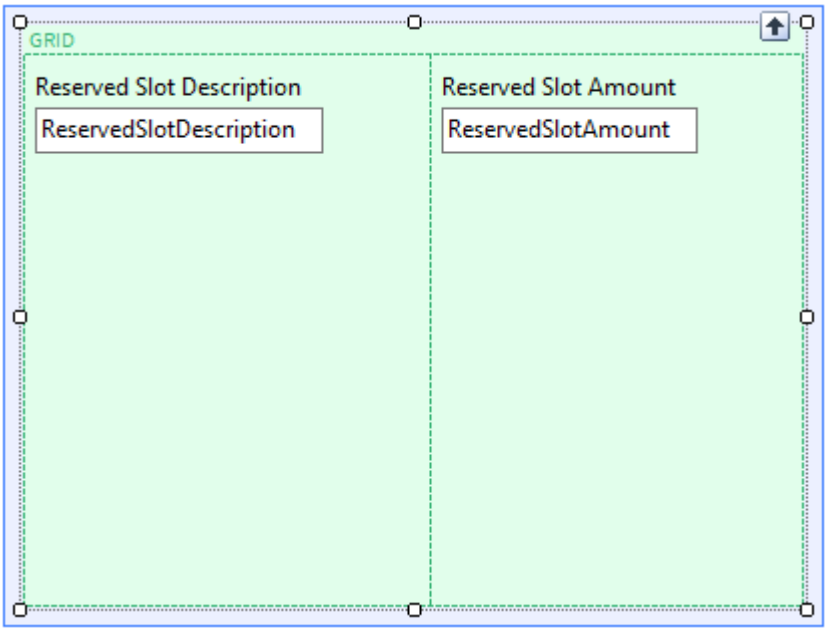




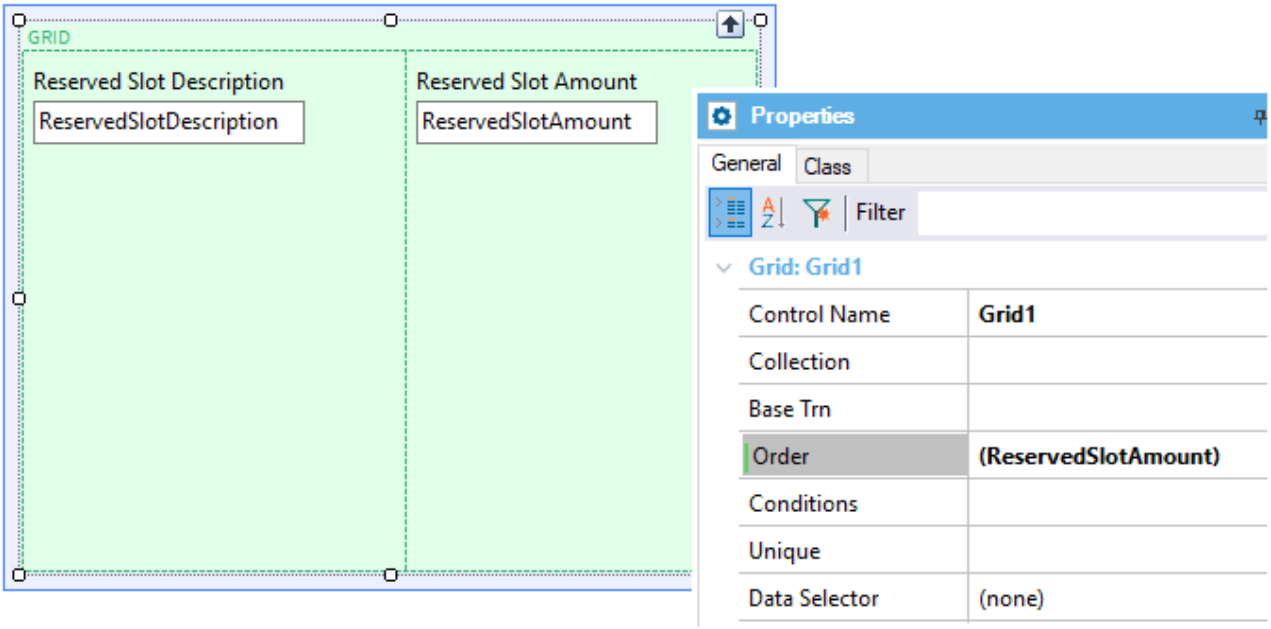
Deberá definir el Data Provider de manera análoga a como lo hicieron los otros alumnos con el SDT colección:



Y en el web panel cargar el grid como si fuera una transacción común y corriente y no una dinámica:

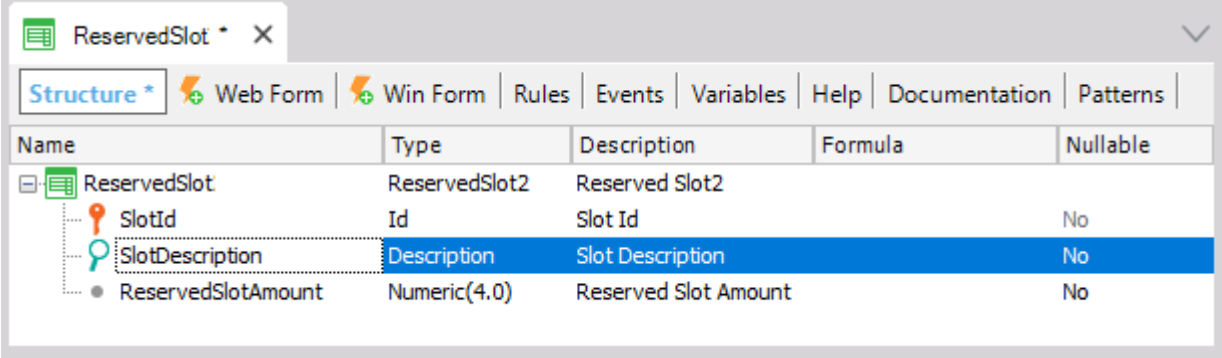


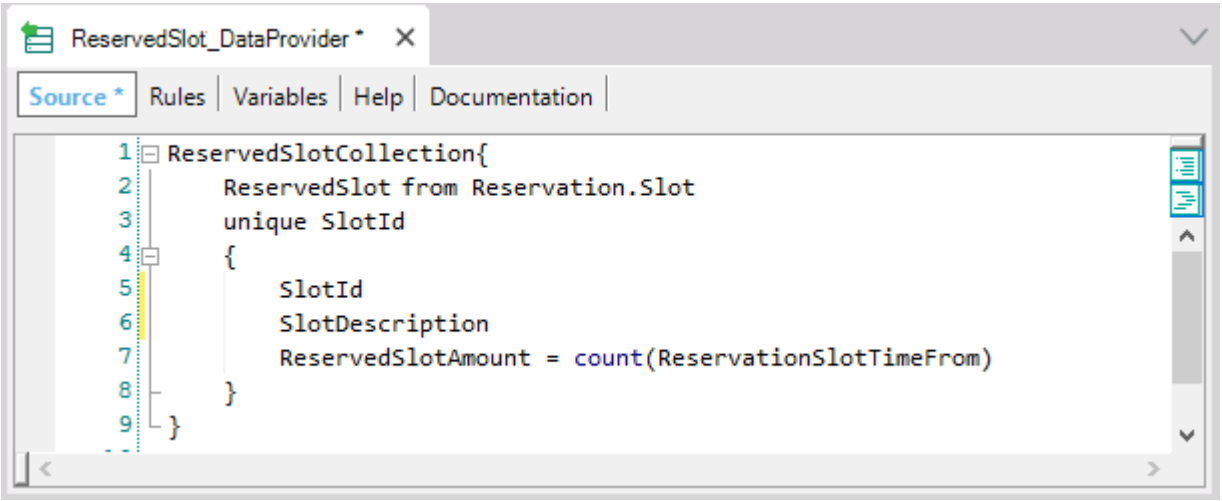
Con el orden definido en las propiedades del grid:



Observemos que aquí no hay que programar absolutamente nada más.

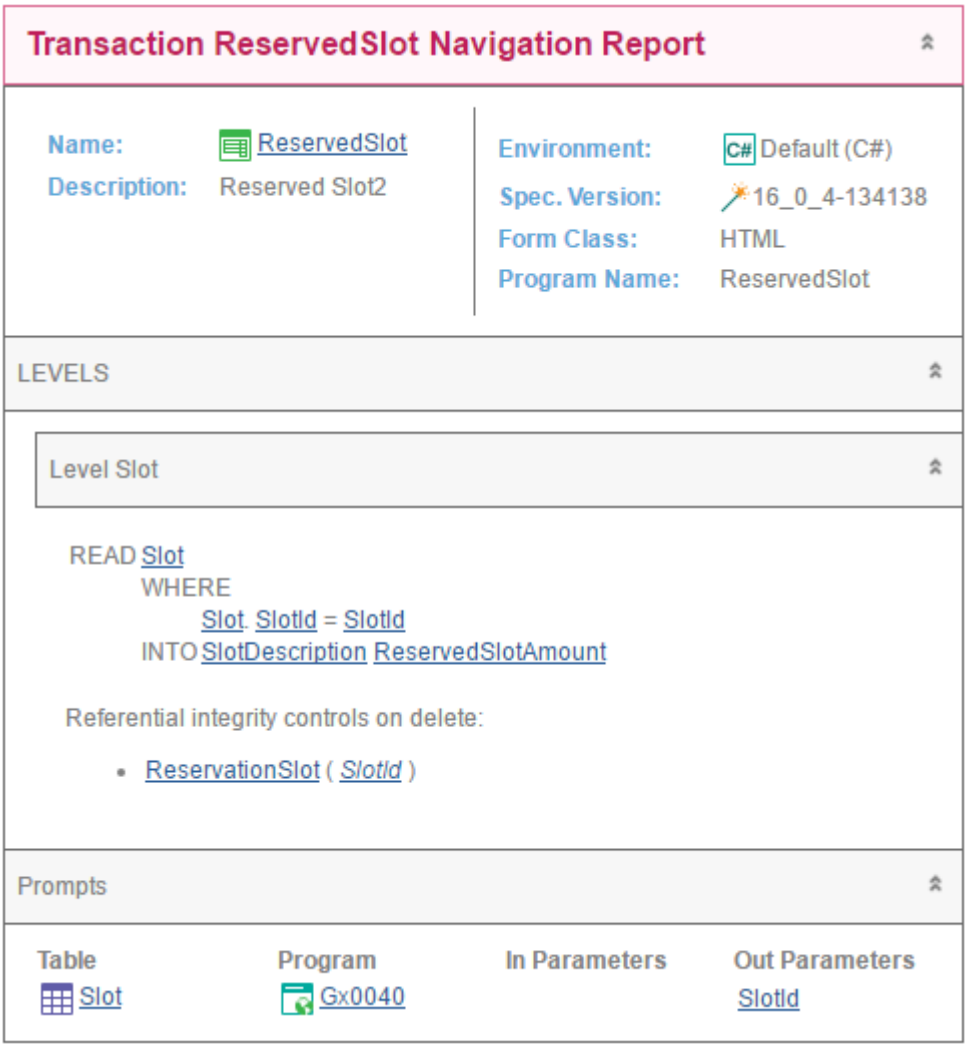
El alumno llegó a esta solución. Usted lo felicita. Y le realiza la siguiente pregunta: ¿por qué en la transacción dinámica en vez de utilizar los atributos ya existentes SlotId y SlotDescription, definiste dos nuevos atributos? Es decir, ¿por qué no definir la transacción de este otro modo?:



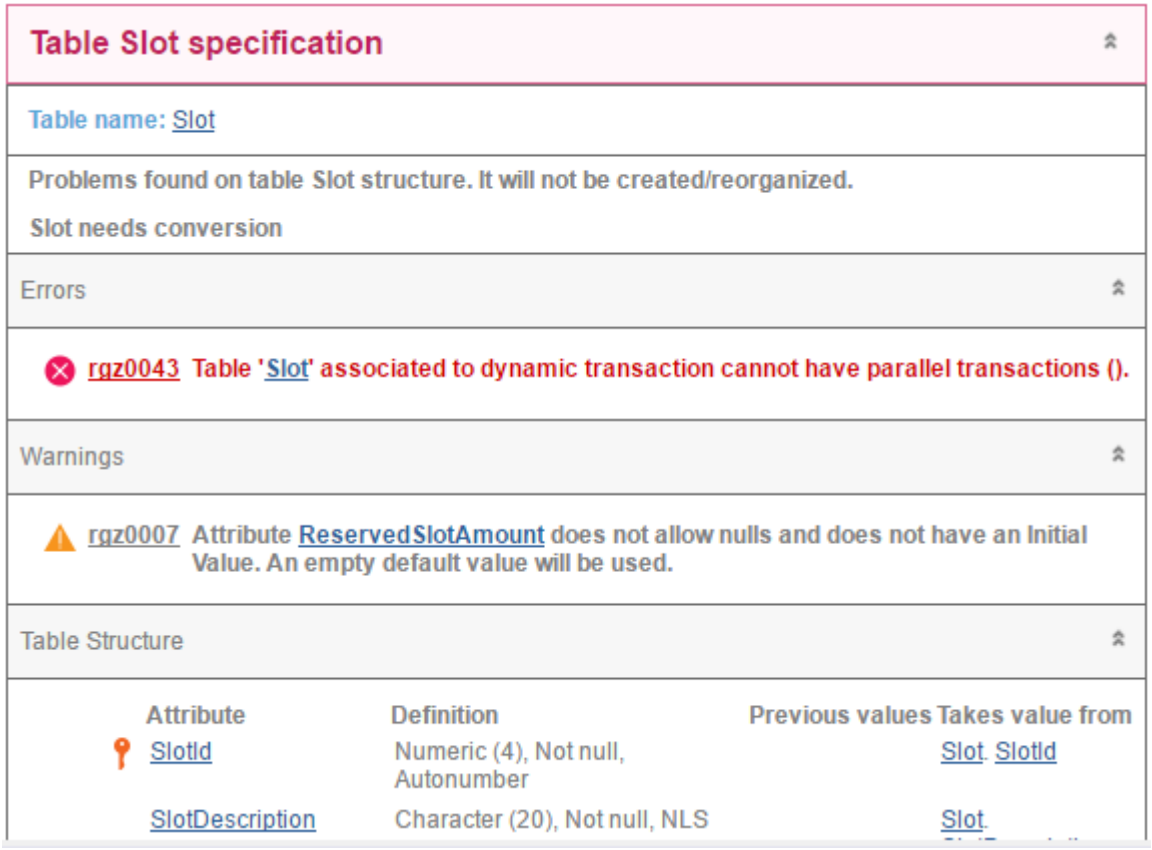


Para que de esta manera el slot de la transacción dinámica sea entendido como un slot efectivamente, en cualquier otro contexto.

Si le hace mirar el listado de navegación al alumno, entenderá por qué no es posible:



Es que se trataría de una transacción paralela a la transacción Slot, pues compartirían la misma clave primaria. Esto no es posible. GeneXus, al intentar especificar le mostrará el siguiente error:



¿Qué pasaría si SlotId no fuera la clave primaria de la transacción dinámica, sino tan solo una clave foránea? ¿Es posible utilizar el atributo en ese caso? Piénselo y busque confirmar su hipótesis.

Pero la respuesta es que sí, es posible. Piense todas las aristas del problema. Por ejemplo, ¿qué pasa si se quiere eliminar un slot de la tabla Slot que tiene algún relacionado por esta transacción dinámica? ¿Se realiza ese control de integridad referencial para evitar la eliminación?

PREGUNTA 22 – UNIQUE PARA AGEGACIÓN

Suponga que ahora necesita listar para cada cliente con reservas de slots posteriores a una fecha dada, su nombre y cantidad total de slots reservados a partir de esa fecha.

Un alumno plantea la siguiente solución:

```
for each Reservation unique CustomerId
where ReservationDate >= &fromDate
&amount = count(ReservationSlotTimeFrom)
print AmountByCustomer //con CustomerName y &amount
endfor
```

Usted menciona que hay algo que está mal programado. Entonces otro alumno plantea que la solución no es correcta porque la tabla navegada por la fórmula count no es Reservation.

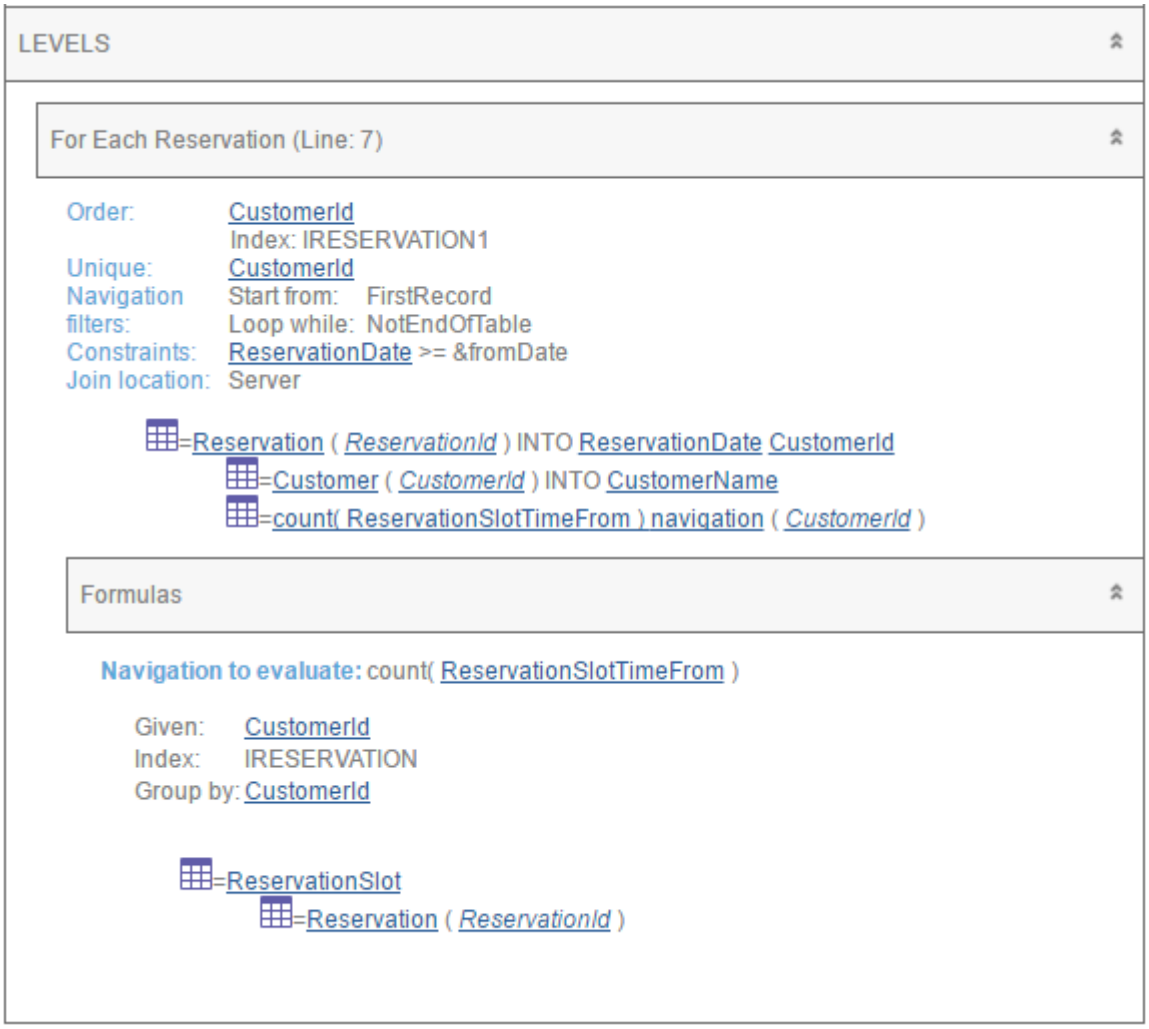
¿Qué contestará usted?

RESPUESTA

Es buena práctica recomendar a los alumnos analizar el listado de navegación, que ya nos da pistas para resolver los problemas.

El alumno que contestó que el problema era que la fórmula count no navegaba Reservation claramente quedó con la idea de que solamente en ese escenario se puede utilizar un for each con unique para hacer agregaciones. Este ejemplo muestra que eso no es así.

En este caso la fórmula count navegará ReservationSlot, pero filtrando por el CustomerId (unique) del for each.



Es decir, podemos tener el mismo CustomerId en muchas reservas (posteriores a la fecha &fromDate), pero se va a tomar una sola de esas reservas en el for each (por el unique) y luego, dentro de su cuerpo, se ejecutará la fórmula, que filtrará por ese CustomerId. Todo eso es correcto.

El error es bien simple: se están contando los registros de la tabla ReservationSlot que corresponden a una reserva del cliente, así sea de fechas anteriores a &fromDate.

Lo correcto sería entonces:

```
for each Reservation unique CustomerId
where ReservationDate >= &fromDate
&amount = count(ReservationSlotTimeFrom, ReservationDate >= &fromDate)
print AmountByCustomer //con CustomerName y &amount

endfor
```

Observe que la cláusula where filtra los registros sobre los que se operará en el cuerpo del for each, mientras que la condición de la fórmula filtra los registros que serán contados. Ambas son necesarias.

www.genexus.com