DEBATE DE PERGUNTAS

para Instrutor

Departamento de Treinamento - GeneXus

Junho 2023

CONTEÚDO

PREPARAÇÃO PARA EXAME DE INSTRUTOR GENEXUS

Quem se apresenta para o exame de Instrutor GeneXus é porque foi aprovado anteriormente na certificação de Analista Sênior GeneXus, por isso já foi avaliado tecnicamente.

INTRODUÇÃO

São apresentadas a seguir, perguntas sobre alguns temas importantes, com respostas que buscam expandir seus conhecimentos ou sua capacidade de integração e articulação do que já sabe. Em alguns casos, são explicadas questões que no curso GeneXus não são expostas. Não será solicitado no exame, mas entendemos que ajudarão a entender melhor a lógica do GeneXus, e é por isso que as explicamos aqui.

O que encontrará serão apenas alguns exemplos de alguns dos temas importantes (não de todos), para que você possa ter uma ideia de que tipo de raciocínio solicitaremos em seu exame.

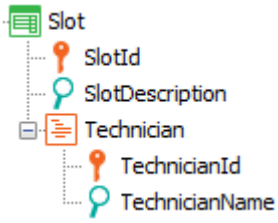
Nota: As capturas de tela foram feitas no GeneXus 16, podem sofrer variações nas versões posteriores.

PERGUNTA 1 - DESENHO

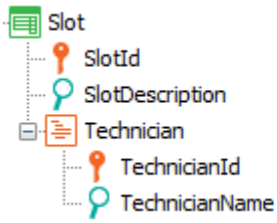
Existe uma aplicação GeneXus para um cassino. Possui transações para registrar as máquinas de jogos (slots), bem como os técnicos encarregados de repará-las.

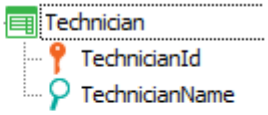
Sabendo que uma máquina de jogos (Slot) pode ser reparada por vários técnicos (Technician) e que um mesmo técnico pode consertar várias máquinas, determine a opção correta.

a. Uma única transação, Slot, com dois níveis:

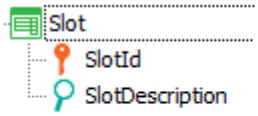


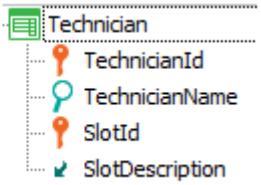
b. Duas transações: Slot e Technician:



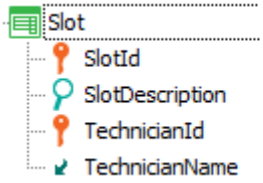


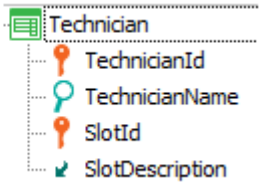
c. Duas transações: Slot e Technician:





d. Duas transações: Slot e Technician:





e. Nenhuma das anteriores

RESPOSTA

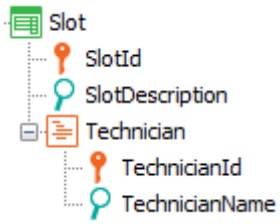
No requisito, foi solicitado um relacionamento N para N entre as entidades da realidade Slot e Technician e é o desenho b) o único dos propostos que o representa.

Observe que as soluções a) e c) são equivalentes e representam um relacionamento diferente daquele solicitado entre as entidades. Qual? Um relacionamento 1 para N, no qual Technician não

existiria por si só, de maneira independente dos Slots; isto é, Technician seria uma entidade fraca, que para existir depende da existência do Slot ao qual está vinculado. Isto é evidenciado no fato de que, se observamos a tabela criada para armazenar suas informações, esta possui chave primária composta por SlotId e TechnicianId. Então, como seria inserido no sistema, para estas alternativas, tanto a) como c), um técnico sem associá-lo na mesma operação a um slot? Lembremos que é impossível deixar uma chave primária sem valor (porque, nesse caso, como identificaríamos o registro da tabela?). SlotId faz parte da chave primária da tabela que registra os técnicos; portanto, seria impossível inserir um técnico sem atribuir valor ao atributo SlotId.

E se isso acontece, ou seja, se não é possível inserir um técnico sem associá-lo a um slot, como faríamos para que esse técnico esteja associado também a outros slots?

Por exemplo, para a solução a):



editando em execução as informações do Slot de Id 3 e descrição “Super Wizard”, inserimos em seu grid ao técnico de Id 1, David Roberts e confirmamos.

Como associamos a esse mesmo técnico, David Roberts, outro slot, por exemplo 6, se o técnico existe unicamente como vinculado ao slot 3?

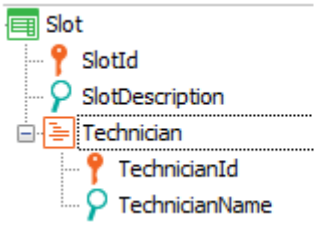
Alguém poderia estar inclinado a dizer que é muito fácil: simplesmente editando agora as informações do Slot 6 e adicionando em seu grid uma linha com TechnicianId = 1 e TechnicianName = David Roberts.

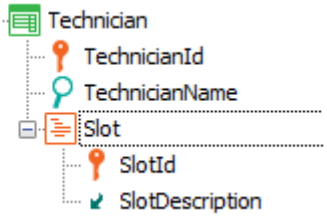
Mas, observemos que nada garante que já não exista um TechnicianId = 1 nesse grid com outro nome de técnico, ou mesmo, sem existir previamente, que possamos atribuir outro nome ao qual inserimos para esse id 1. Como TechnicianName é um atributo físico dessa tabela, não é um inferido, como no caso da solução b).

No entanto, poderíamos utilizar este desenho (que é 1 para N), garantindo “manualmente” que não seja utilizado o número 1 de técnico, exceto para David Roberts. Mas o que é mostrado aqui é que não estamos desenhando corretamente nossa realidade, porque se fizermos o que é certo, o próprio sistema não nos permitirá cometer erros, como é o caso da solução correta, a b), onde o técnico é identificado apenas por seu Id e então para os Slots são indicados quais técnicos (por seu id) podem reparar a máquina.

Por outro lado, a solução d) é impossível, porque ambas as transações determinam a mesma tabela física (a ordem na qual aparecem os atributos não altera a constituição da tabela). Aqui, os slots e os técnicos existem apenas vinculados. Nenhum existe independentemente do outro.

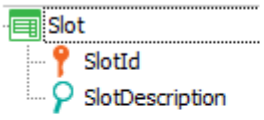
Em vez disso, o que aconteceria se em seu lugar tivessem sido criadas estas outras transações:

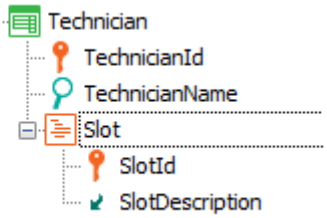




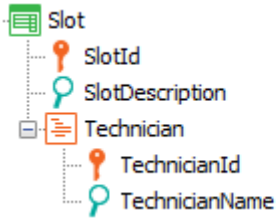
Quais tabelas GeneXus criará a partir delas? Embora à primeira vista possa parecer uma solução errada e, em parte, não seja a aconselhável, as tabelas que serão criadas serão as corretas, idênticas às da solução b). Observemos que a tabela correspondente ao segundo nível de Slot será exatamente igual à do segundo nível de Technician, pois a chave primária de ambas é a mesma, composta pelos dois atributos, SlotId e TechnicianId. O problema desse desenho será apenas operacional: para poder inserir os técnicos para um slot, antes devem ter sido criados esses técnicos por meio da transação Technician, deixando os slots não preenchidos.

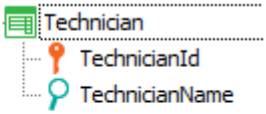
Claro, uma solução como:





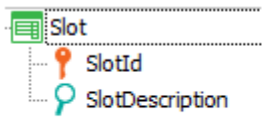
Quanto à representação N para N é completamente equivalente à b) proposta:

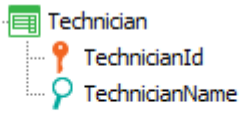


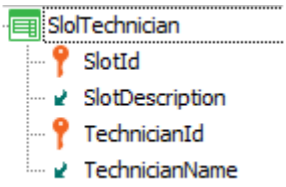


A única diferença é dada pela operação: qual é mais conveniente? Inserir todos os técnicos sem vinculá-los primeiramente aos Slots e, em seguida, para cada slot inserido, atribuir seus técnicos ou vice-versa?

As tabelas geradas são exatamente as mesmas nas duas soluções. Ou ainda, poderia chegar a ter esta outra solução, completamente equivalente em relação ao relacionamento N para N:





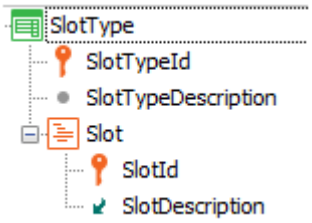


PERGUNTA 2 - DESENHO

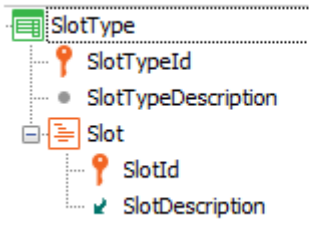
Existe uma aplicação GeneXus para um cassino. Esta, possui transações para registrar as máquinas de jogos (slots), bem como os tipos de máquinas existentes.

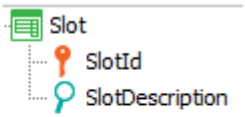
Sabendo que cada máquina (Slot) corresponde a um tipo determinado (SlotType) e apenas um, e que pode haver muitos slots do mesmo tipo, determine a opção correta para o desenho destas transações.

a. Uma única transação:

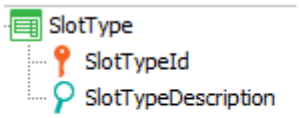


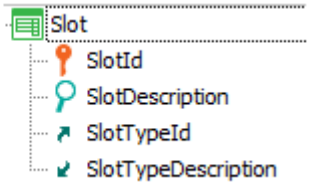
b. Duas transações:



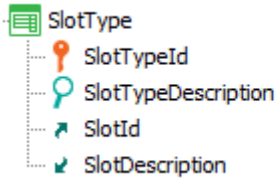


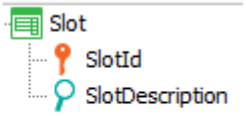
c. Duas transações:





d. Duas transações:





e. Nenhuma das anteriores.

RESPOSTA

Está sendo solicitado implementar um relacionamento 1-N forte (este dado deduzimos do nosso conhecimento da realidade, pois não foi explicado na descrição do exercício), portanto a resposta correta é a c).

Aqui, na solução c), cada "slot" terá um único "tipo", porque na transação Slot, o atributo SlotTypeId está no mesmo nível em que se encontra o identificador, SlotId. Dessa forma, assim como um determinado slot, identificado pelo SlotId, terá uma única descrição, SlotDescription, também terá um único SlotTypeId (seria diferente se esse atributo também tivesse o símbolo de chave). Por outro lado, nada impede que outro slot tenha o mesmo valor para SlotTypeId (já que o atributo não é chave primária nem candidata nesta transação). Cumpre-se então o requisito da descrição: um slot tem apenas um tipo e vários slots podem ter o mesmo tipo.

Por que a opção a) não está correta, dado que também representa um relacionamento 1-N? Porque nesta representação, a entidade Slot seria fraca, ou seja, não pode ser identificada apenas com seu Id, mas requer o Id do tipo. Estaríamos dizendo que os Slots só existem na medida em que existe um "tipo de slot" do qual dependem. Não será possível inserir um Slot sem antes ter especificado seu SlotTypeId. Em outras palavras, sob essa representação, deduziríamos que a entidade do mundo real Slot depende absolutamente para existir do "tipo de Slot", o que contradiz a realidade que queremos modelar. Nela, o "Tipo de Slot" e o "Slot" são entidades relacionadas, mas de existência própria. Cada uma se identifica além da outra. Eventualmente, mesmo se nulos forem permitidos para SlotTypeId em Slot (na solução correta, a c), poderiam ser inseridos "slots" sem inserir seu "tipo".

A opção b) representa claramente um relacionamento N-N (neste caso, para cada “tipo de slot”, podem ser inseridos muitos “slots” associados e, ao mesmo tempo, cada “slot”, representado pela transação Slot e identificado com o atributo SlotId, pode ser encontrado repetido para muitos “tipos de slot” (o slot 1 poderá mostrá-lo em seu grid, bem como o slot 2, etc.) Isto claramente não corresponde à realidade proposta.

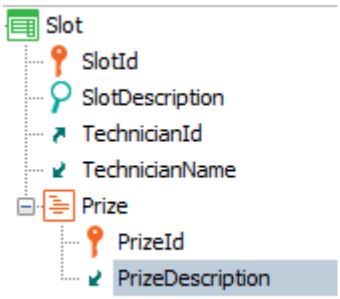
A opção d) representa um relacionamento 1-N, mas oposto ao solicitado. Observe que aqui um “tipo de slot” possui um único “slot” associado e outro “tipo de slot” pode ter associado o mesmo “slot” que o anterior. Por esse motivo, aqui estamos dizendo que cada “slot” poderá ter muitos “tipos” e que para um “tipo” apenas lhe corresponde “slot”.

Resumo de entidade forte vs. fraca: uma entidade é forte, quando pode ser identificada independentemente das outras, mesmo que mantenha relação com elas. Em vez disso, será fraca quando não existe com independência de outra entidade. Para existir requer a existência de outra. O caso típico é o dos telefones de clientes. Não faz sentido ter um telefone independente do cliente ao qual pertence.

PERGUNTA 3 - DESENHO

Existe uma aplicação GeneXus para um cassino.

Dadas as seguintes transações, determine a relação entre os atores da realidade Slot e SlotPrize (prêmio do Slot).



a. Relacionamento 1 para 1 (para cada slot, existe apenas um prêmio e, para cada prêmio, um slot que o concede)
b. Relacionamento 1 para N (para cada slot, há vários prêmios, mas onde cada prêmio corresponde apenas a esse slot e não a outro), sendo ambas entidades fortes.
c. Relacionamento 1 para N (para cada slot, há vários prêmios, mas onde cada prêmio corresponde apenas a esse slot e não a outro), sendo a entidade Slot forte, mas SlotPrize fraca.

d. Relacionamento N para N (para cada slot, há muitos prêmios e cada prêmio pode ser concedido por muitos slots)

RESPOSTA

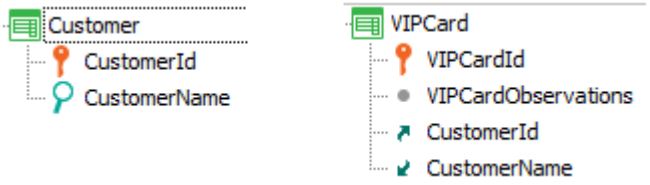
A correto é a c). O que significa dizer que SlotPrize é uma entidade fraca em relação a Slot? Que não existirá como entidade independente. Seu relacionamento com Slot é de absoluta dependência. Um determinado prêmio existe apenas na medida em se indica o Slot ao qual está associado. É o prêmio "tal" do slot "tal". Nunca poderá ser dito "é o prêmio tal" isoladamente. Sempre deverá ser indicado de qual slot se trata. Se, em vez disso, o relacionamento fosse 1-N forte, o prêmio existiria como entidade independente, identificável por si mesma, sem precisar indicar o slot para saber de que prêmio estamos falando.

PERGUNTA 4 - DESENHO

Existe uma aplicação GeneXus para um cassino. Possui transações para registrar os clientes, bem como os cartões VIP emitidos para eles.

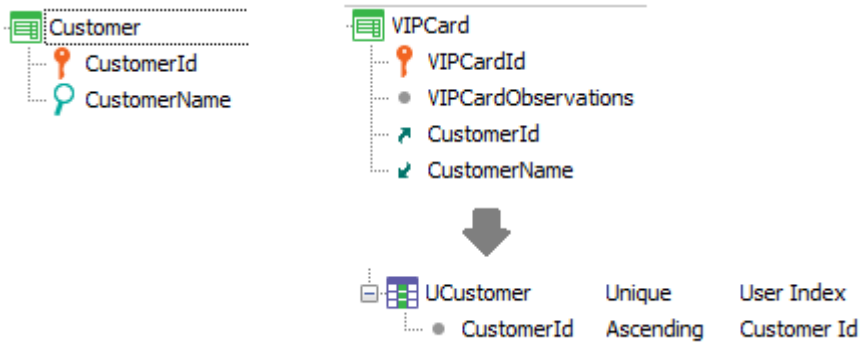
Sabendo que cada cliente (Customer) pode ter um único cartão VIP (VIPCard) e que cada cartão VIP pode pertencer a apenas a um cliente, determine a opção correta para o desenho destas transações.

a. Duas transações:

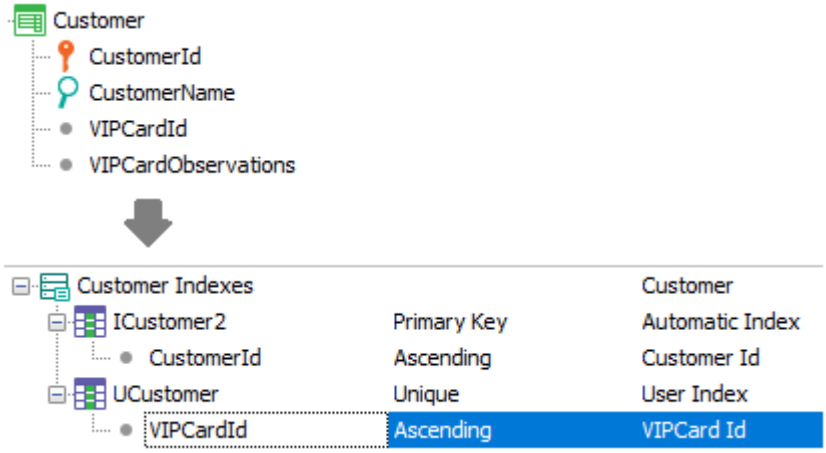


b. Duas transações:

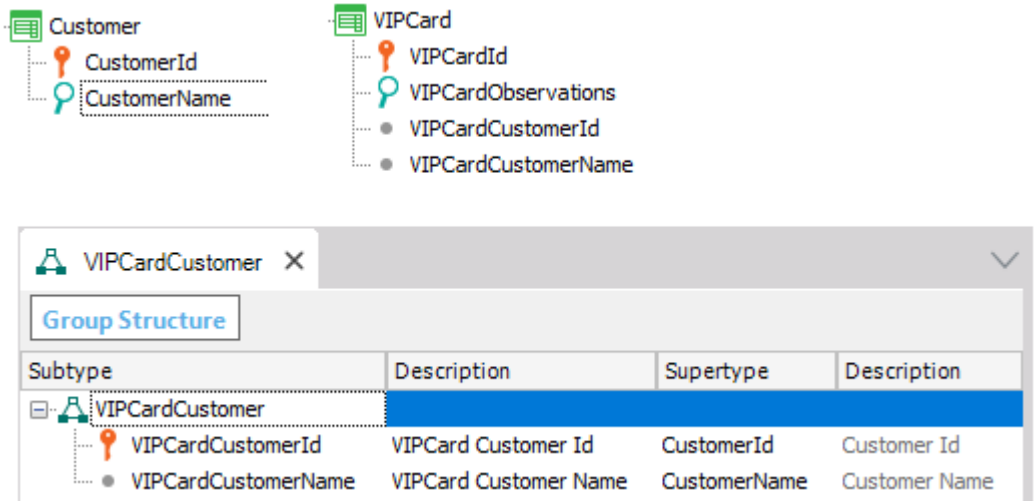
c. Duas transações e um índice unique:



d. Uma transação e um índice unique:



e. Duas transações e um grupo de subtipos:



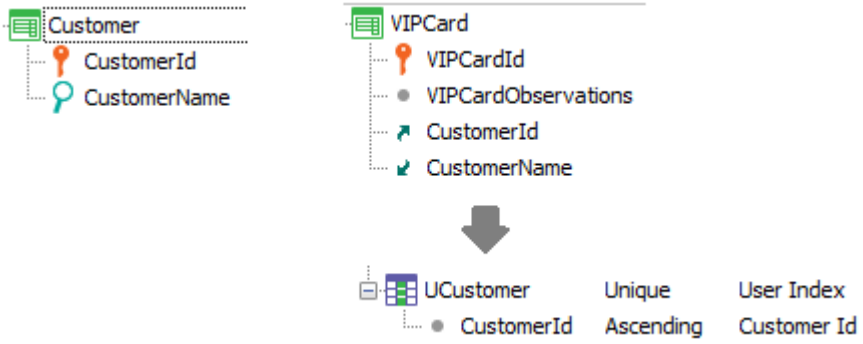
f. Nenhuma das anteriores

RESPOSTA

Nos é solicitado um relacionamento 1 para 1 entre as duas entidades. Há uma suposição implícita, é que se trata de duas entidades separadas, cada uma com seu identificador. Cada cartão corresponderá a um cliente e apenas a um, mas é suficientemente forte para ter existência própria. Ou seja, o sistema trabalhará com os cartões além do cliente. Haverá operações em que o cartão deva ser inserido e não o cliente (por exemplo, para pagar). Portanto, a opção válida será a **c**) e não a d).

Vamos discutir as diferenças entre as duas soluções e depois veremos por que as outras estão longe de estar corretas.

Observemos primeiro a solução correta, a c):



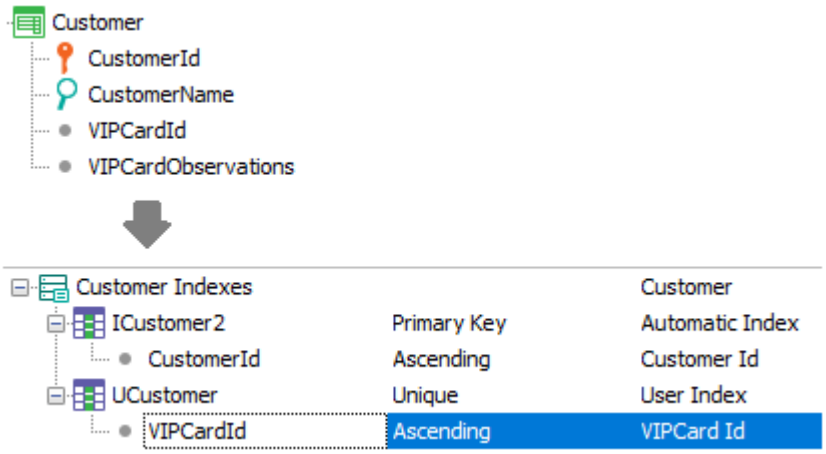
Aqui, temos um identificador para cliente, CustomerId, e também temos um identificador para cartão VIPCardId, o que faz com que possamos utilizá-los em qualquer outra transação como chaves estrangeiras. Por exemplo, em uma fatura, podemos colocar o atributo VIPCardId, e o cliente será inferido lá, sem precisar especificá-lo também. Como observação, não podemos fazer o inverso, ou seja, colocar na estrutura da fatura o atributo CustomerId e esperar que a partir daí o cartão seja inferido. Sim, poderemos procurá-lo com uma fórmula. Por que isto é assim?

Porque, na realidade, GeneXus está nos permitindo modelar um relacionamento 1 para N com o qual fazemos 1 para 1 através da definição do índice unique sobre o que seria a chave estrangeira. É por isso que, no nível da tabela VIPCard, definimos um índice de usuário Unique no atributo CustomerId, que já tinha definido um índice pela Foreign Key automático.

Dessa forma, toda vez que for inserido um novo cartão no sistema, se for feita uma tentativa de atribuir para o atributo CustomerId um número já existente para outro cartão, a operação falhará porque, utilizando esse índice unique, encontrará imediatamente que esse valor já existe e nos informará.

Na solução d), definimos uma única transação, com chave primária, o id do cliente e, com chave candidata, o id do cartão. Com isto, também obtemos que um id de cartão não pode ser repetido

entre clientes (e, é claro, que um cliente possua um único id de cartão), mas o que não podemos fazer é utilizar o cartão em outras entidades. Não poderá ser chave estrangeira em nenhum lugar. Por exemplo, não poderemos inseri-lo na fatura. Teremos necessariamente que colocar o cliente (e inferir o cartão).



Indo ao melhor, na solução c) poderia eventualmente haver clientes sem cartão (o que não poderá acontecer é que um cartão não tenha um cliente especificado). Uma boa pergunta a ser respondida é como se faria para evitar isso, ou seja, para que cada cliente tenha necessariamente um cartão. Este tema está vinculado ao da UTL. Uma possibilidade seria escrever a seguinte regra de chamada a procedimento na transação Customer:

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

Assim, depois que for inserido na tabela CUSTOMER o cliente (e ANTES DO COMMIT), será chamado o procedimento CreateVIPCard, o que criará o cartão para esse cliente. Supondo que o VIPCardId é autonumerado, poderia ser programado assim:

Rules:

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

Source:

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

Observe que, embora o procedimento não realizará commit (uma vez que as operações de BCs não fazem o procedimento interpretar que acessará a base de dados e, portanto, não coloca o commit implícito no final do fonte), dado que foi invocado a partir de transação Customer antes de seu

Commit, ao devolver o controle à transação, esta realizará o commit e, portanto, serão commitados os dois registros inseridos na base de dados: o cliente na tabela CUSTOMER inserido pela transação antes de chamar o procedimento e o cartão na tabela VIPCARD inserido pelo próprio procedimento ao utilizar o método Insert.

Lembre-se que em aplicações web, não pode ser estendida a UTL composta pelos registros manipulados por uma instância de iteração de transação para cobrir também os registros manipulados por outra transação. Mas sim entre transação e objetos batch como a proc.

Vamos discutir brevemente o motivo de estarem incorretas as outras opções da pergunta.

A opção a) representa claramente um relacionamento 1 para N. Não é o que nos pedem.

A opção b) deve ser estudada em detalhes:

À primeira vista, olhando para a transação Customer, estamos dizendo que um cliente possui um único VIPCard e olhando para a transação VIPCard, dizemos que um cartão tem um único cliente. Assim, pode parecer que o requisito está sendo atendido. No entanto, as transações operam em conjunto e não sozinhas. Se olharmos para o conjunto, poderíamos dizer que VIPCardId seria uma chave estrangeira na tabela CUSTOMER e, simetricamente, CustomerId seria uma chave estrangeira na tabela VIPCARD. E poderíamos arriscar que o erro desse desenho é que, embora cumpre-se que um cliente tenha apenas um cartão e que um cartão tenha apenas um cliente, não é verdade que correspondam clientes e cartão nas duas tabelas.

Ou seja, o cliente 1 pode ter o VIPCardId 5, mas quando vamos para o VIPCardId 5, verifica-se que ele poderia ter o CustomerId 3, por exemplo.

No entanto, não percebemos que esses registros nem sequer podem ser inseridos devido aos controles de integridade referencial. Como faço para inserir o cliente 1 com cartão 5 se não o inseri antes? Mas para inseri-lo, como faço se o cliente ainda não existe? Bem, isso poderia ser resolvido permitindo que VIPCardId em CUSTOMER, bem como CustomerId em VIPCard permitam nulos. Assim, a primeira vez insiro o cliente sem especificar um cartão, então insiro o cartão, especificando o cliente e retorno ao cliente em update para que agora possa especificar o cartão.

No entanto, toda a análise anterior foi baseada em uma premissa falsa. Temos certeza de que o GeneXus criará duas tabelas?

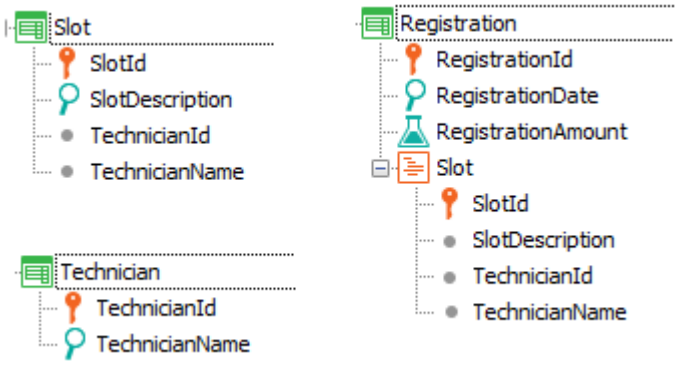
Lembremos que, em primeiro lugar, GeneXus normaliza as tabelas, para evitar inconsistências. Se observarmos, ao ter as referências cruzadas, estamos dizendo, por um lado, que CustomerId determina VIPCardId, e por outro lado, que VIPCardId determina CustomerId. Portanto, essas duas transações darão origem a uma única tabela que terá a composição de qualquer uma das duas transações (pode ser da primeira ou da última). E, se pensarmos sobre isso, é completamente lógico. Nesse caso, se escolher a última, a tabela coincidirá com a estrutura de VIPCard, mas onde também criará um índice primário também por CustomerId (ou seja, não permitirá inserir para dois cartões diferentes o mesmo cliente). Se transformaria em um caso análogo ao da solução d), com tudo o que discutimos lá. Então, esta solução está mais próxima da requerida, exceto pelo fato de que precisamos que existam o cliente e o cartão separadamente, como entidades.

A última solução, a e), é uma variação da a). Continua representando um relacionamento 1 para N (onde parece completamente desnecessária a definição do grupo de subtipos).

PROVA

PERGUNTA 5 – DESENHO/NORMALIZAÇÃO

Existe uma aplicação para um cassino. Dado o seguinte desenho de transações, determine a estrutura física das TABELAS que GeneXus desenhará e criará.



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	

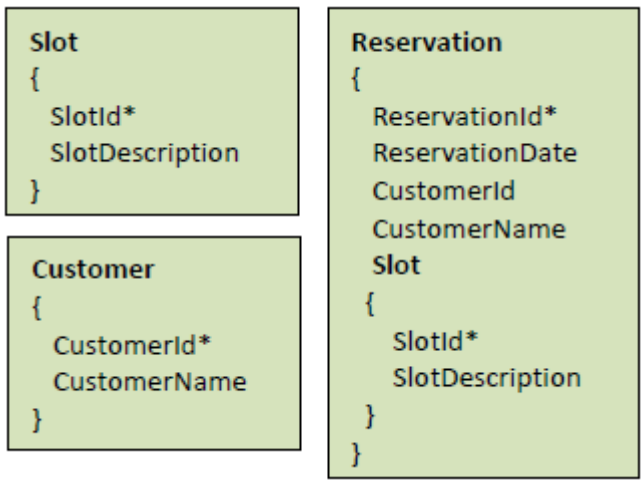
RESPOSTA

A correta é a c)

- Na transação Slot, o atributo TechnicianName é inferido a partir da chave estrangeira TechnicianId.
- No nível Slot da transação Registration, o atributo TechnicianId é inferido por meio da chave estrangeira SlotId (tenha o cuidado em acreditar que, porque um atributo é chave primária em uma tabela, que isso o converte automaticamente em um atributo armazenado e chave estrangeira em qualquer outra tabela associada a uma transação na qual apareça!) Sabendo disso, como faria para que o segundo nível de Registration permitisse registrar um técnico independente do slot? Terá que utilizar subtipos. Pense na solução e então veja a resposta d) à pergunta 10.
- RegistrationAmount é fórmula, portanto não estará presente em nenhuma tabela (a menos que seja explicitamente definida como redundante).

PERGUNTA 6 – TABELA ESTENDIDA

Existe uma aplicação GeneXus para um cassino. Tendo as seguintes transações, determine a tabela estendida da tabela RESERVATIONSLOT



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

RESPOSTA

A correta é a d).

O conceito de tabela estendida é muito importante em GeneXus, porque toda a sua lógica é baseada em abstrair a visão das tabelas físicas e em passar para uma visão mais conceitual (de informação univocamente relacionada, que está espalhada em diferentes tabelas físicas apenas para fins de normalização, mas que, no entanto, forma logicamente uma unidade). Portanto, o for each permite trabalhar quase igualmente com todos os atributos da tabela estendida (e não apenas da tabela base), bem como grids, grupos de Data Providers, regras de transações, etc.

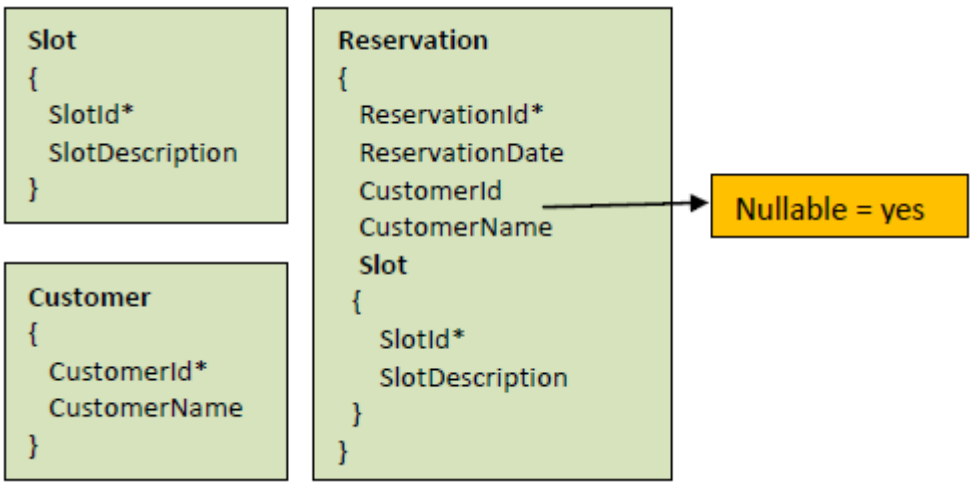
Para que uma base de dados é normalizada? Para evitar a duplicação de dados, que traria o conhecido risco de ter dados inconsistentes.

Se não precisarmos evitar a duplicação, teríamos uma única tabela para os slots das reservas, com chave primária {ReservationId, SlotId}. Quais atributos essa tabela teria? Além da chave primária, teria a descrição do slot (SlotDescription), a data da reserva (ReservationDate), o cliente da reserva (CustomerId) com seu nome (CustomerName). Toda essa informação estaria reunida em uma tabela, que chamamos de “tabela estendida”. Fisicamente não existe, mas sim como ferramenta conceitual.

PERGUNTA 7 – COMO DEIXAR SEM VALOR UMA CHAVE ESTRANGEIRA

Existe uma aplicação GeneXus para um Cassino, que possui um conjunto de transações para registrar os slots (Slot) e a reserva de slots (Reservation) por parte dos clientes (Customer), conforme mostrado.

Em algumas ocasiões, as reservas são feitas sem a necessidade de especificar o cliente (CustomerId). A partir do desenho proposto, indique a afirmação que considere correta:



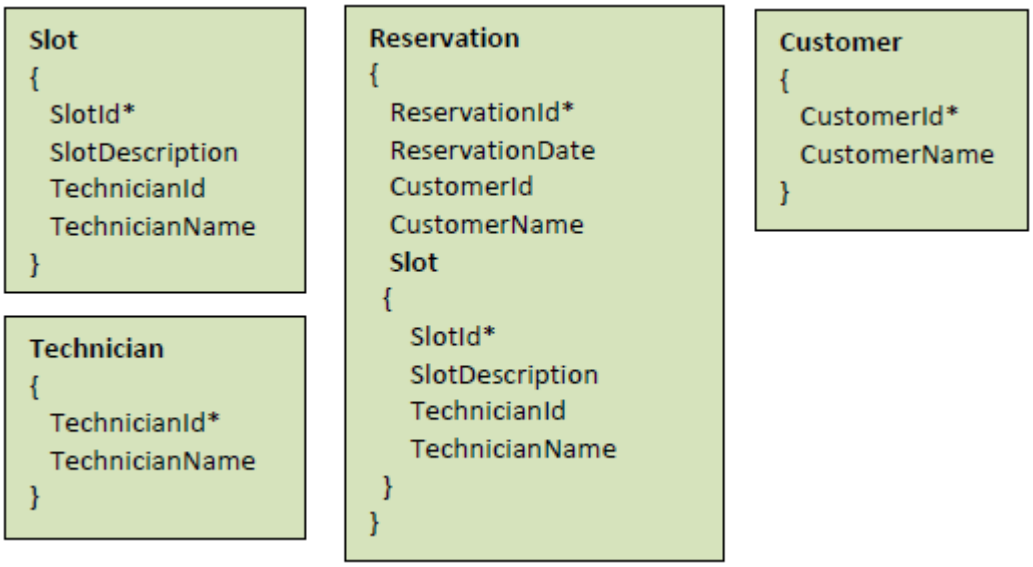
a. Apesar de declarar que o atributo CustomerId (chave estrangeira) na tabela RESERVATION permite nulos (ou seja, admite um valor não especificado no nível da base de dados), GeneXus sempre disparará os correspondentes controles de integridade referencial na tabela CUSTOMER, e não admitirá que não seja inserido um valor válido para essa chave estrangeira CustomerId.
b. No momento de declarar que o atributo CustomerId (chave estrangeira) na tabela RESERVATION admite nulos (ou seja, admite um valor não especificado no nível da base de dados), então, se não for inserido um valor nessa chave estrangeira, GeneXus não dispara os controles de integridade referencial na tabela CUSTOMER, mas se for inserido um valor nessa chave estrangeira, GeneXus disparará os controles de integridade referencial na tabela CUSTOMER.
c. Nenhuma das anteriores

RESPOSTA

A correta é a b)

PERGUNTA 8 – UTILIZAÇÃO DE ÍNDICES POR CHAVES PRIMÁRIA E ESTRANGEIRA

Dado o desenho de transações a seguir, determine qual índice é utilizado para validar com eficiência ao tentar eliminar um técnico (Technician) por meio da transação, que não existam slots que o tenham atribuído.



a. Índice por TechnicianId na tabela TECHNICIAN
b. Índice por TechnicianId na tabela SLOT
c. Índice por SlotId na tabela SLOT
d. Nenhuma das anteriores está correta

RESPOSTA

A correta é a b). Esta pergunta não é tão importante para o aluno quanto para o instrutor (dificilmente a encontre em perguntas reais de exame). Ao tentar excluir um registro da tabela TECHNICIAN por meio da transação, como sabemos, é incluída lógica nela para acessar todas as tabelas que tenham TechnicianId (chave primária) como chave estrangeira, ou seja, para todas as tabelas subordinadas, que a referenciam. Deve ser acessada cada uma dessas tabelas para confirmar que não exista nenhum registro que referencie àquele que deseja excluir. Neste exemplo, há apenas uma subordinada: a tabela SLOT. Como acessar esta tabela para saber se há um registro com o valor para o atributo chave estrangeira TechnicianId igual ao do registro que deseja excluir de TECHNICIAN? Os índices são os mecanismos eficientes para realizar essas pesquisas; portanto, GeneXus utilizará o índice por chave estrangeira definido na tabela SLOT, sobre o atributo chave estrangeira TechnicianId. De fato, é para tornar eficientes os controles de integridade referencial na eliminação que os índices por chave estrangeira são criados automaticamente pelo GeneXus na base de dados.

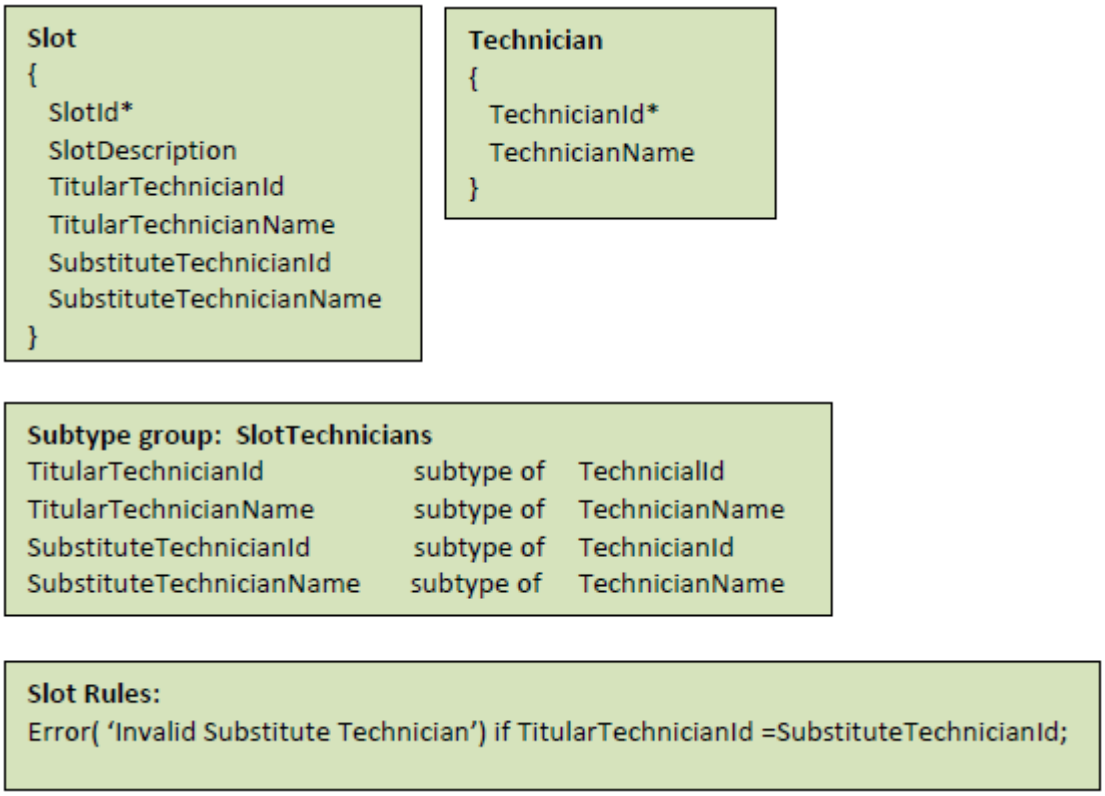
É comum confundir-se e pensar que a resposta correta seria a), mas, pelo raciocínio, entende-se que o índice por TechnicianId na tabela TECHNICIAN, o que faz é acessar a tabela TECHNICIAN, onde não há nada a procurar. A pesquisa deve ser realizada sobre SLOT. Este é um tema complexo, porque, na realidade, a estratégia de acesso às tabelas utilizadas depende um pouco do DBMS. Mas a lógica conceitual de GeneXus seria a indicada.

PERGUNTA 9 – DESENHO COM GRUPOS DE SUBTIPOS

Existe uma aplicação GeneXus para um cassino. Esta, possui um conjunto de transações para registrar os slots (Slot) e os técnicos responsáveis pelos reparos (Technician).

Cada vez que um novo slot é inserido, deve ser associado um técnico responsável e um substituto. O sistema deverá controlar que não seja o mesmo.

Determine se é verdadeiro ou falso que a alternativa a seguir resolve corretamente o requisito anterior.



RESPOSTA

Falso.

Para definir subtipos para o técnico titular e para o substituto, devem ser definidos em dois grupos diferentes, para indicar o conjunto de informações que são tratadas conjuntamente. Na solução apresentada, foi definido um único grupo de subtipos, onde foram colocados misturados tanto os subtipos correspondentes ao titular quanto ao substituto. Isto está claramente incorreto. Por quê?

PERGUNTA 10 –DESENHO COM SUBTIPOS (REFERÊNCIA DUPLA EM TABELA ESTENDIDA)

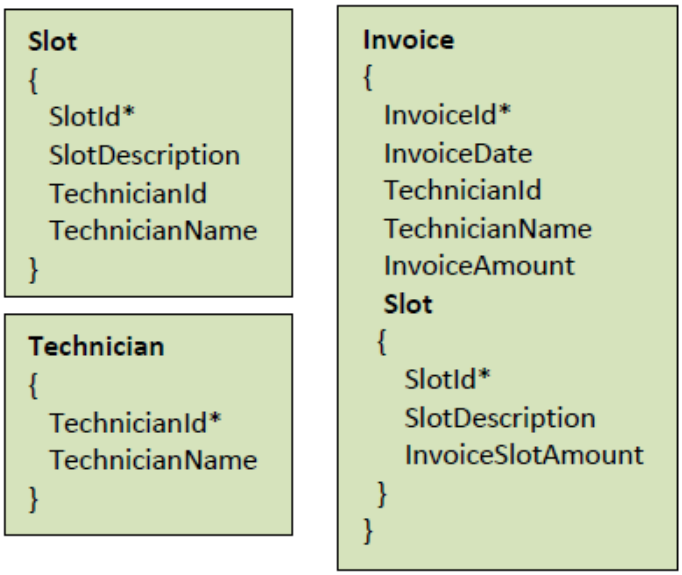
Existe uma aplicação GeneXus para um cassino, com um conjunto de transações para registrar os slots (Slot) e os técnicos encarregados dos reparos (Technician).

Um slot pode ser reparado por um único técnico e cada técnico tem atribuídos vários slots para reparar em caso de necessidade. Portanto, ao faturar os serviços de um técnico (Invoice), é

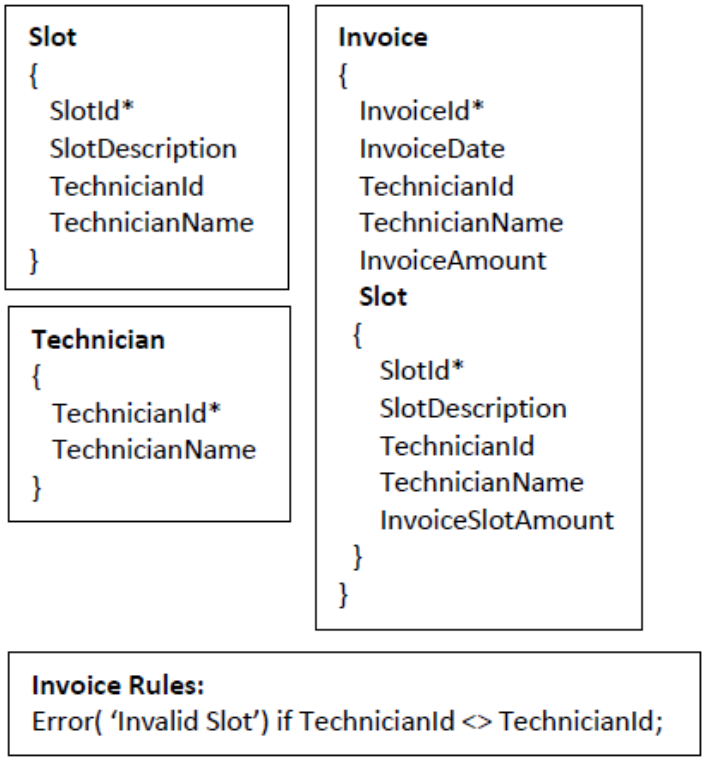
necessário verificar que os slots detalhados estão realmente sob responsabilidade do técnico da fatura.

Determine, das seguintes opções, a que implementa este requisito.

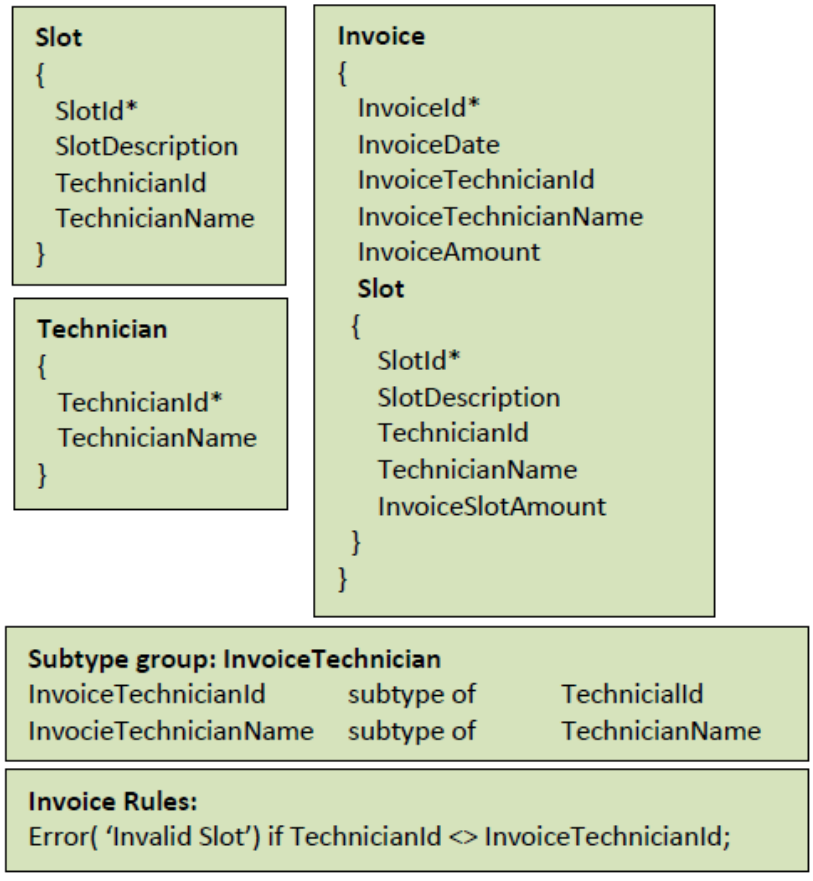
a.



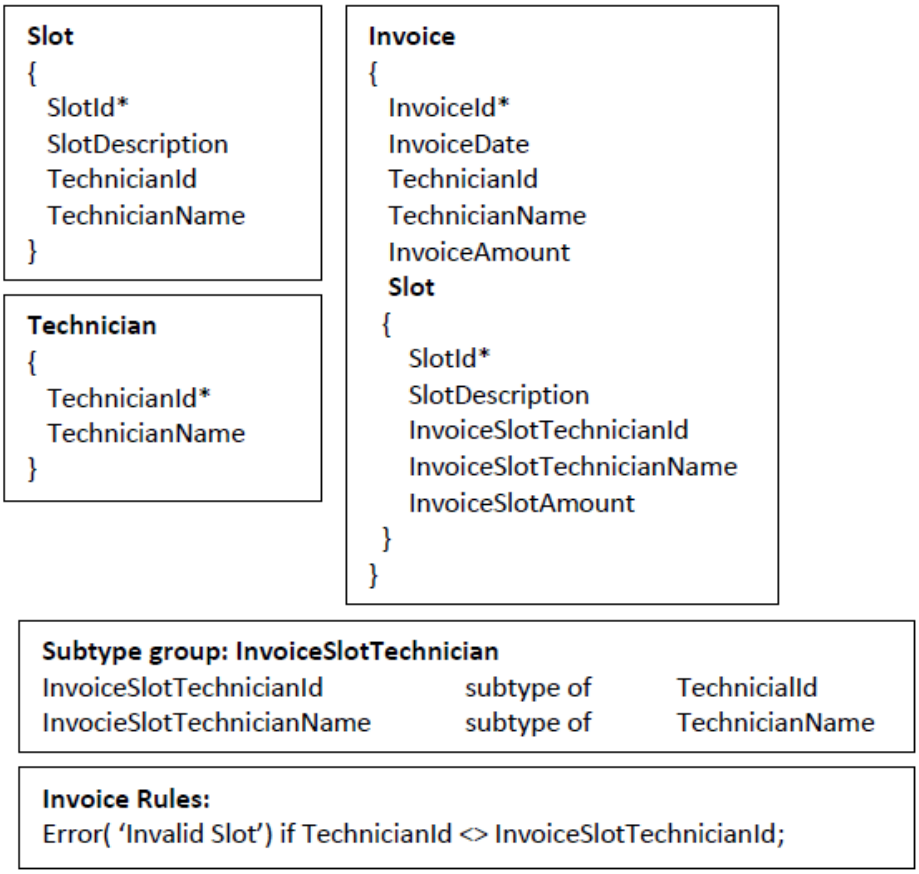
b.


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.



RESPOSTA

Embora existam outras maneiras de implementá-lo, a única das opções apresentadas que o atinge é a c). Observe que, no caso da d), como o subtipo InvoiceSlotTechnicianId não está em um grupo com um subtipo SlotId, será um técnico independente daquele que vem inferido do Slot.

Embora a solução c) esteja correta, em geral não é a que costumamos aconselhar. Por quê?

Não é no cabeçalho da Invoice que surge a necessidade de alterar o nome do técnico, pois não há nenhuma ambiguidade lá. Também não é no Slot onde a ambiguidade se encontra.

É no segundo nível da Invoice que aparece essa ambiguidade. É que nesse nível temos dois técnicos: o da fatura e o que vem inferido do slot.

Portanto, a solução que recomendamos é resolvê-la aqui. Mas para resolvê-la aqui, temos que mudar o nome do TechnicianId e também o nome do SlotId e agrupar tudo isso. Desta forma, estamos alterando o nome tanto para a chave primária como para a estrangeira.

Subtype group – InvoiceSlot

InvoiceSlotId	suptype of	SlotId
InvoiceSlotDescription	subtype of	SlotDescription

InvoiceSlotTechnicianId subtype of TechnicianId
InvoiceSlotTechnicianName subtype of TechnicianName

Onde a transação Invoice ficará:

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

E as outras transações não modificadas.

Com esta solução, InvoiceSlotTechnicianId será inferido através de InvoiceSlotId.

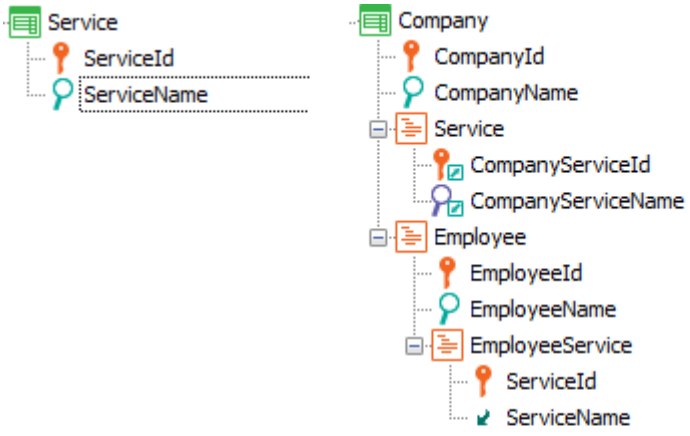
PERGUNTA 11 – DESENHO COM SUBTIPOS – EVITAR RELACIONAMENTO REFERENCIAL

É necessário modelar as transações para uma realidade na qual existem empresas e serviços que estas podem contratar (como, por exemplo, um serviço de emergência médica). Por sua vez, as empresas possuem funcionários que também podem ter contratados serviços que não precisam corresponder aos da empresa para a qual trabalham. Interessa registrar esses serviços dos funcionários porque, por exemplo, se muitos funcionários contrataram o serviço de emergência médica X, é possível tentar um acordo com esse serviço para obter algum desconto.

Os funcionários podem trabalhar apenas em uma empresa, mas não é requerido ser representados como uma entidade forte, mas dependente da empresa.

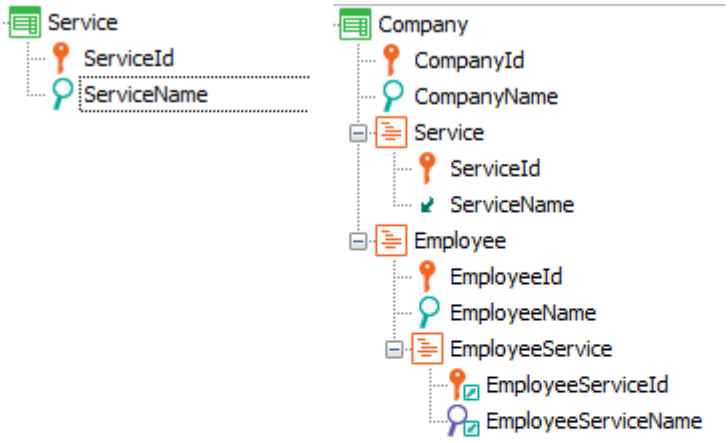
Explique qual das duas soluções é a correta e porque a outra não é:

a. Duas transações e o seguinte grupo de subtipos:



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

b. Duas transações e um grupo de subtipos:



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

RESPOSTA

A solução correta é a) e não a b).

Se olharmos atentamente, temos dois níveis paralelos: Service e Employee. Isso significa que tudo o que é inferido de qualquer um deles corresponderá à mesma empresa. No entanto, não queremos que o serviço do funcionário exista como serviço da empresa, pois, em nossa realidade, o funcionário poderia ter contratados serviços diferentes daqueles da empresa para a qual trabalha. Em outras palavras: não queremos que seja verificado quando o usuário insere no grid de serviços do empregado, que o serviço inserido exista como registro na tabela correspondente a Company.Service.

É claro que precisamos definir um grupo de subtipos, pois na mesma transação GeneXus não nos permitirá repetir o mesmo nome de atributo.

A questão que surge é: dá no mesmo defini-lo em um nível do que no outro? A resposta: não.

Poderíamos definir dois grupos de subtipos e o problema acabou. Mas não é uma boa prática definir mais subtipos do que aqueles estritamente necessários, pois nunca é exatamente o mesmo ter o subtipo que ter o supertipo, como ficará claro neste exemplo.

Portanto, para resolver o problema, temos apenas um grupo. Por que a solução correta é a) e não a b)?

É que, se GeneXus nos permitisse repetir o mesmo nome de atributo, claramente encontraria que na tabela associada ao nível Company.Employee.EmployeeService, de chave primária {CompanyId, EmployeeId, ServiceId}, os atributos {CompanyId, ServiceId} formariam uma chave estrangeira para a tabela correspondente ao nível Company.Service (já que sua chave primária seria {CompanyId, ServiceId}).

Mas se o que fazemos é alterar o nome (com um subtipo) de ServiceId na tabela em que este atributo faz parte de uma chave estrangeira, isto não elimina para GeneXus sua função referencial.

Por outro lado, se o atributo para o qual alteramos o nome (usando um subtipo) é aquele que cumpre a função de chave primária, então na tabela em que o atributo supertipo aparece, não estabelece o relacionamento referencial.

Isto não é explicado no curso GeneXus, então não teria como saber.

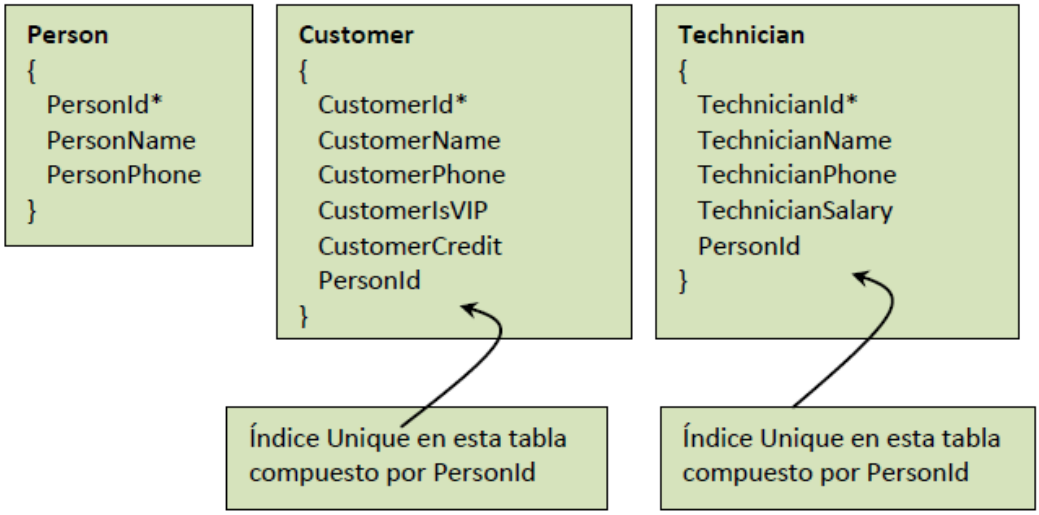
Damos o exemplo para ajudá-lo a raciocinar sobre casos um pouco mais complexos que, no entanto, são comuns em apps da vida real.

PERGUNTA 12 – DESENHO - ESPECIALIZAÇÃO

Existe uma aplicação GeneXus para um cassino.

Entre as informações que precisam ser registradas, está a dos técnicos que consertam os slots, bem como a dos clientes do cassino. Como tanto os técnicos como os clientes são pessoas, das quais se registra um conjunto de informações comuns (nome e telefone), deseja-se registrar as informações gerais apenas uma vez e depois registrar apenas as informações específicas (por exemplo, se a pessoa é um cliente, deve se registrar se é um cliente VIP e o crédito fornecido pelo cassino e se for um técnico, seu salário).

Determine se é verdadeiro ou falso que a seguinte solução resolve este caso de especialização adequadamente em GeneXus.



RESPOSTA

Falso.

Embora ao definir o índice Unique em PersonId na tabela CUSTOMER se estabelece um relacionamento 1-1 com PERSON, na verdade não estamos representando que se trata da mesma entidade que deveria se separar em duas tabelas para que na especializada só apareçam os atributos que não são comuns aos técnicos. Pelo contrário, o que está sendo dito é que são duas entidades diferentes, com um relacionamento 1 para 1, como poderia ser a de um cliente com seu cartão VIP (ver pergunta 4).

Aqui não estamos dizendo que o relacionamento 1 para 1 existente entre Person e Customer é uma especialização, onde o cliente "é" uma pessoa. Para fazer isso, observe que a pessoa terá um atributo

para armazenar o nome (PersonName) e que o cliente que tenha associada essa pessoa poderá ter outro nome (CustomerName).

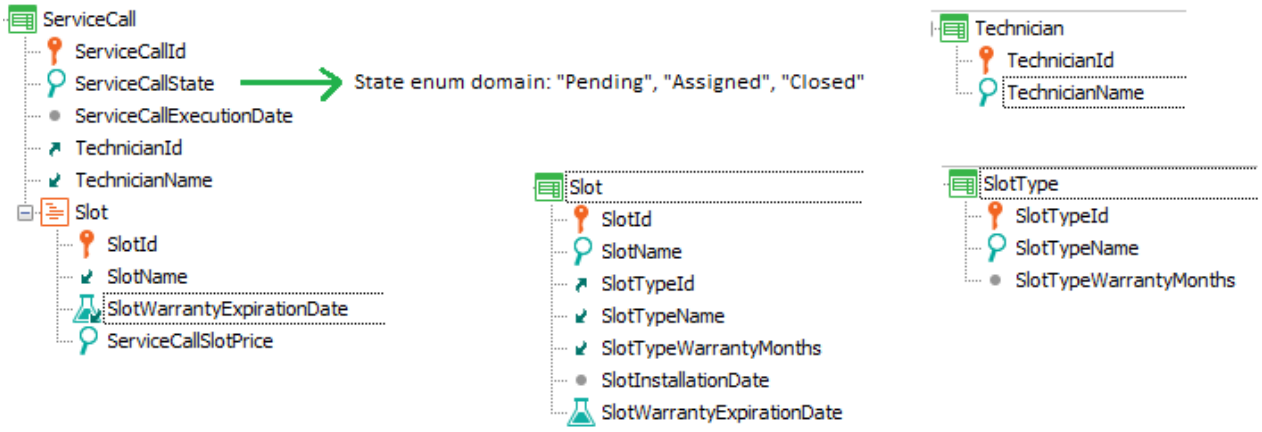
A mesma análise pode ser feita entre Person e Technician.

A solução para este problema de especialização é dada por grupos de subtipos. Como seria? Por que definir CustomerId como subtipo de PersonId resolve o problema para Customer? Pense que dizer que é um subtipo o converte em chave estrangeira. Mas, além disso, dizer que é identificador da transação Customer, o converte em chave primária. Portanto, CustomerId acaba sendo a chave primária que também é estrangeira. Então, finalmente, estabelece um relacionamento 1-1, por chave, e este último é o que provoca a relação "é". Análise análoga para Technician.

PERGUNTA 13 – REGRAS E EVENTOS EM TRANSAÇÕES

Está sendo desenvolvido um sistema para uma empresa que fornece, instala e repara Slots para cassinos, bares, etc. O sistema deve registrar os clientes (cassinos, bares, etc.), os Slots instalados em cada cliente com a data de instalação e de final da garantia, e os chamados dos clientes para que sejam realizados reparos de Slots instalados. Os chamados são criados com status: "pendente". Quando são atribuídos a um técnico, passam para "atribuído" e, quando o técnico apresenta o tíquete de serviço correspondente ao chamado, com a data de conclusão e os preços a serem cobrados do cliente pelos Slots reparados que não estavam em garantia, passam para "fechado".

Para fechar, então, um chamado, deve alterar o status, inserir a data de conclusão e inserir os preços dos reparos.



Não pode ser permitido fechar um chamado se não foram inseridos os preços de todos os slots reparados que não estavam em garantia. Para fazer isso, o desenvolvedor escreve na seção de regras da transação 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();
```

Raciocinando da seguinte forma:

Se o status do chamado era 'Assigned' e queremos atualizá-lo, é porque queremos fechá-lo. Se for inserida a data de fechamento, então o status é alterado para 'Closed', sabendo que, de qualquer maneira, se a seguir, no nível das linhas ou imediatamente depois, ocorre uma regra de erro (antes do commit), as alterações serão desfeitas e tanto o status como a data ficarão como estavam antes.

Analise esta solução e discuta outras variações.

RESPOSTA

Esta declaração de regras não produzirá o efeito desejado.

Ao trabalhar em uma transação de dois níveis no modo Update, as únicas regras no nível das linhas que são disparadas serão para aquelas linhas sobre as quais o usuário tenha realizado alguma ação. Se o usuário não editar em nada uma linha específica, para que essa linha viole claramente uma condição especificada em uma regra de erro, a regra não será disparada.

Portanto, se a transação for acessada em modo update, uma data é inserida e confirmada, tendo slots nas linhas que não estão em garantia, a regra de erro nunca será disparada e a gravação será permitida.

Portanto, temos que verificar, depois de trabalhar com as linhas (ou seja, depois de terem sido efetivamente modificadas na tabela física todas aquelas com as quais o usuário trabalhou), que não tenha ficado nenhuma das que não estão em garantia, sem preço. Uma possibilidade seria invocar um procedimento que percorra os registros correspondentes às linhas e conte para quantos está faltando o preço e, se para um ou mais estiver faltando, então seja disparada a regra de erro. Seria assim?

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

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

Embora a regra de erro dependa da variável &Missing que é carregada pelo procedimento, como não está condicionada com o mesmo evento de disparo que a invocação à proc, esta regra será avaliada ao abrir a transação e não no momento desejado, que é após o disparo da proc.

E se então tivéssemos escrito:

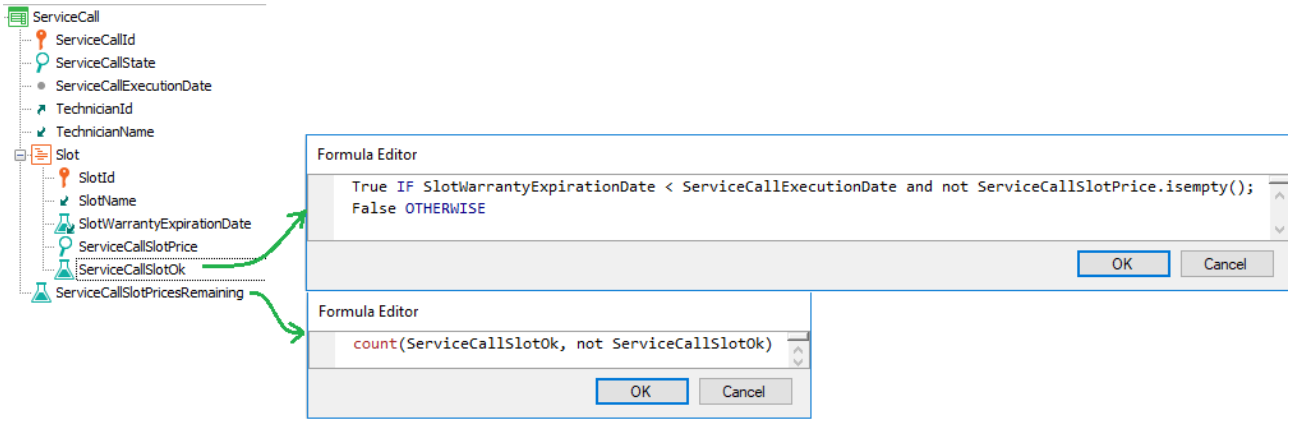
```
&Missing = MissingPrices( ServiceCallId ) if Update;

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

Aqui teremos o problema que estaremos disparando o procedimento assim que a transação reconhecer que está em modo Update, que é assim que o campo identificador for abandonado. Não demos ao usuário tempo para fazer nada, nem mesmo para inserir a data de atendimento do chamado. Claramente temos que disparar a proc depois que tenham sido atualizadas todas as linhas que o usuário quis atualizar.

Portanto, deveríamos ter 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;
```

Claro que poderíamos ter evitado utilizar um procedimento, definindo as duas fórmulas que indicamos:



E nas regras:

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

E não poderíamos aproveitar para também alterar o status para Closed depois, quando tivermos certeza de que tudo será feito corretamente?

```
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;
```

O último momento possível para atribuir um valor para um atributo do cabeçalho é imediatamente antes do cabeçalho ser gravado, e isso é muito antes de on AfterLevel level segundo nível.

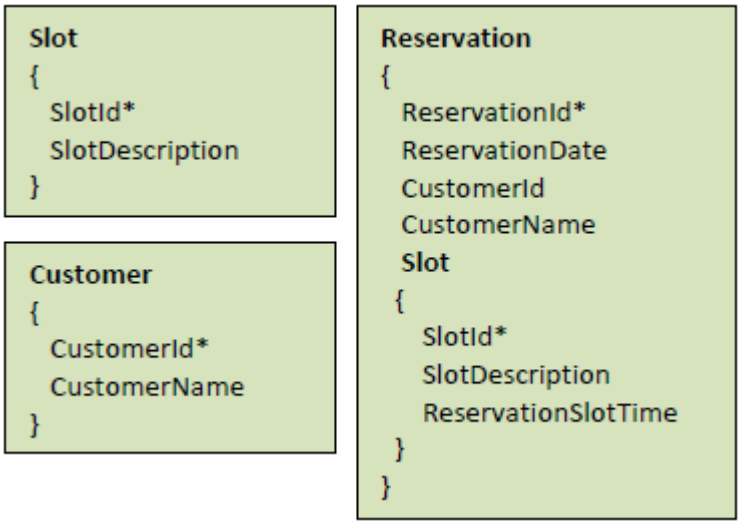
De fato, o último momento possível é on BeforeUpdate.

Quer dizer que no máximo nós poderíamos escrever:

```
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;
```

PERGUNTA 14 – FÓRMULAS INLINE

Existe uma aplicação GeneXus para um cassino. Esta possui transações para registrar os slots (Slot) e a reserva dos slots (Reservation) por parte dos clientes (Customer)



Dado o seguinte source de um procedimento:

For each Reservation
&Slots = count(ReservationSlotTime)

print Info //ReservationDate, CustomerName, &Slots
Endfor

Determine se existe uma tabela de partida (no momento de disparo da fórmula) qual é, e determine a tabela a ser navegada pela fórmula para obter seu resultado.

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

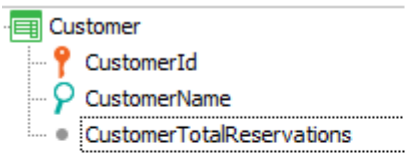
RESPOSTA

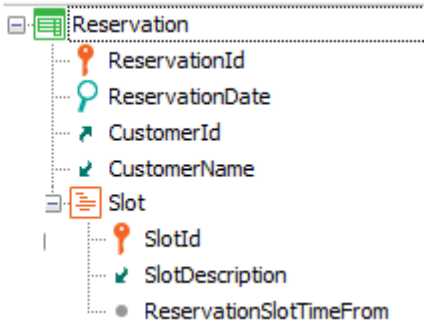
A resposta correta é a).

Qualquer coisa que esteja dentro de um for each, apenas por estar ali, refere-se a uma iteração dele, portanto, tem como contexto o registro da iteração em que se encontra trabalhando a execução em um determinado momento. A tabela base do for each, ao especificar a transação base, sabemos que é RESERVATION. Se não tivesse sido especificada transação base, a tabela base do for each é calculada com os atributos que estão dentro do for each "soltos" (não aqueles que estão dentro de uma fórmula aggregate, como a Count, ou aqueles que estejam dentro de um for each aninhado a este, ou outro comando que tenha sua própria resolução, como um new, por exemplo). Neste caso, serão os do print block, então a tabela base será RESERVATION. Portanto, o for each irá iterar sobre essa tabela e, para cada reserva, é disparada a fórmula count. Então, haverá tabela de partida da fórmula, que é o contexto que tem ao disparar (estamos posicionados em uma reserva). O que se conta? Os registros onde se encontra ReservationSlotTime. Em que tabela está esse atributo? Em RESERVATIONSLOT. Mas quando a fórmula é disparada, em uma iteração do for each, temos um ReservationId instanciado (o do registro no qual estamos trabalhando). Portanto, contará os registros relacionados, aqueles que correspondem a esta reserva. Sempre que a fórmula não tiver um contexto (por exemplo, se estiver "solta" dentro do Source), não terá tabela de partida. Por esse motivo, nesse caso, contaria todos os registros da tabela navegada (RESERVATIONSLOT), justamente porque não há um contexto que permita especializar o cálculo. O contexto é fundamental em GeneXus e o que o torna inteligente. Porque pode inferir coisas com base nesse contexto, sem precisar explicitá-lo, economizando trabalho de programação.

PERGUNTA 15 – ATUALIZAÇÃO COM FOR EACH

Na aplicação para um Cassino que estamos desenvolvendo, definimos as transações Customer, Slot e Reservation (esta última para permitir aos clientes realizar reservas de slots para serem utilizados um determinado dia). Suponhamos que a aplicação já estava em produção, portanto, as tabelas já estão carregadas com dados, quando surge a necessidade de adicionar um atributo, CustomerTotalReservations para a transação Customer, cujo valor será o total histórico de dias em que o cliente realizou reservas. Além de adicionar o atributo à transação Customer, o que mais deve fazer?





Suponha que um aluno elabora a seguinte solução:

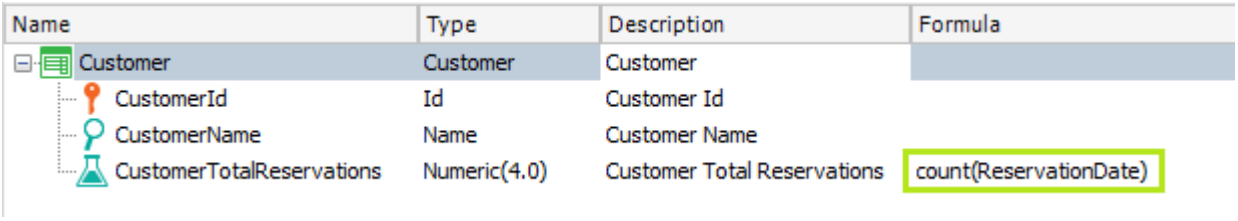
Cria um procedimento com o seguinte Source e o executa após a reorganização:

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

Realize uma análise completa dos problemas desta solução.

RESPOSTA

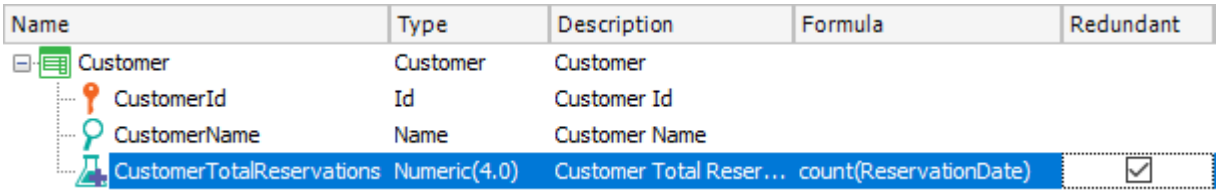
Claramente, o atributo CustomerTotalReservations responde a um cálculo conhecido: corresponde a somar todos os registros da tabela Reservation correspondentes ao cliente CustomerId. Portanto, a primeira pergunta que surge é por que não defini-lo como atributo fórmula na própria transação Customer. Seria uma fórmula global, claro.



Dessa forma, não precisaríamos fazer mais nada, pois toda vez que o valor seja necessário, a fórmula será disparada o cálculo será realizado. Observemos que não precisamos filtrar os registros de Reservation que correspondem ao cliente, pois essa é uma condição implícita devido ao contexto (quando a fórmula for disparada, CustomerTotalReservations, será porque estamos trabalhando com um CustomerId determinado).

Mas, o que acontece se o sistema estiver em operação há muito tempo e a quantidade de Reservas de cada cliente for enorme? Suponhamos que seja necessário fazer com certa frequência relatórios dos clientes classificados de acordo com a quantidade de reservas feitas. Cada vez terá que disparar o cálculo, para cada cliente, e isto pode implicar em um alto custo de desempenho.

Então, pode surgir a necessidade de que esse atributo seja armazenado fisicamente e simplesmente seja atualizado quando é adicionada ou excluída cada reserva. Para fazer isso, basta defini-lo como um atributo redundante (e isto é feito colocando visível a coluna Redundant na estrutura da transação e marcando o check box para este atributo). A partir desta alternativa GeneXus fará por nós todo o necessário. Para os propósitos do desenvolvedor, é como se a fórmula permanecesse virtual, só que não é, pelo que o desempenho melhorará significativamente ao fazer o relatório solicitado frequentemente (porque será obtido diretamente o valor armazenado e a fórmula não será disparada novamente toda vez). De fato, quando uma reserva é adicionada para o cliente, GeneXus haverá gerado código que será executado atualizando o atributo fórmula sem que tenhamos que nos preocupar (e não voltará a disparar a fórmula, mas simplesmente adicionando ou subtraindo 1 conforme esteja inserindo ou excluindo uma reserva).

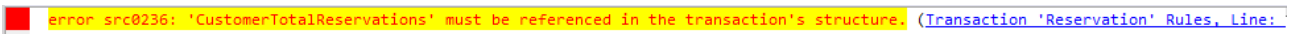


O aluno poderia querer fazer o mesmo, mas manualmente com o procedimento, que já sabemos que realizará o esperado, embora sua programação esteja com falha, como veremos e isso deve ser enfatizado para o aluno.

De qualquer forma, com sua solução, a única coisa que será conseguida será inicializar o atributo CustomerTotalReservations, que não é fórmula, com o total de reservas de cada cliente no momento

de executar a proc, mas não está resolvendo o que acontece cada vez que se insira ou elimine uma nova reserva.

Portanto, para sua solução estaria faltando adicionar na transação Reservation o atributo CustomerTotalReservations na estrutura –não importa que seja inferido: como vamos utilizá-lo nas regras, precisamos declará-lo ali; caso contrário, não nos permitirá gravar, avisando-nos:



e a regra:

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

Este é um bom momento para avaliar se entende-se o que realiza a regra Add em todos os modos (Insert, Update, Delete).

Resumindo: é adicionado o atributo CustomerTotalAmount à estrutura de Customer, é reorganizado para que o atributo seja adicionado à tabela, executa-se apenas uma vez o procedimento para inicializar seu valor, e se adiciona o atributo CustomerTotalAmount à estrutura da transação Reservation, e adiciona-se a regra Add.

Tudo isso é o que GeneXus faz automaticamente ao definir a fórmula como redundante.

Pergunta: em vez do procedimento, poderíamos ter dado à propriedade Initial Value do atributo CustomerTotalAmount na transação Customer o valor: count (ReservationDate), para que, ao reorganizar e adicionar o atributo, seja carregado com o valor inicial resultante do cálculo dessa fórmula para cada CustomerId?

Agora vamos analisar o procedimento do aluno, abstraindo o restante da solução:

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

Este procedimento tenta atribuir o valor ao atributo CustomerTotalAmount que foi adicionado à tabela e está vazio para cada um dos clientes.

Por isso, o aluno decidiu escrever um for each que percorrerá a tabela CUSTOMER e para cada registro da tabela, desejará atribuir o valor para CustomerTotalReservation, e para isso o que faz é

codificar outro for each que percorra a tabela RESERVATION, mas apenas para aqueles registros que correspondam a esse CustomerId. E o que faz é somar um a uma variável anteriormente inicializada como zero, de forma que, quando termina de iterar por todas as reservas desse cliente, terá sua quantidade.

Uma solução um pouco melhor teria sido.

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

E outra possibilidade teria sido:

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

É interessante discutir esta última. Por que isso não ocorreu ao aluno? Para cada reserva é acessada a tabela estendida, nesse caso, podemos acessar o atributo CustomerTotalReservation e atualizá-lo. A desvantagem poderia ser que estamos atualizando o atributo para cada reserva que o cliente possua, em vez de contar todas as reservas e apenas lá atualizar o atributo. Por outro lado, com esta solução, se muitos clientes não fizeram nenhuma reserva, não serão percorridos inutilmente. Mas isso também poderia ter sido alcançado da seguinte maneira:

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

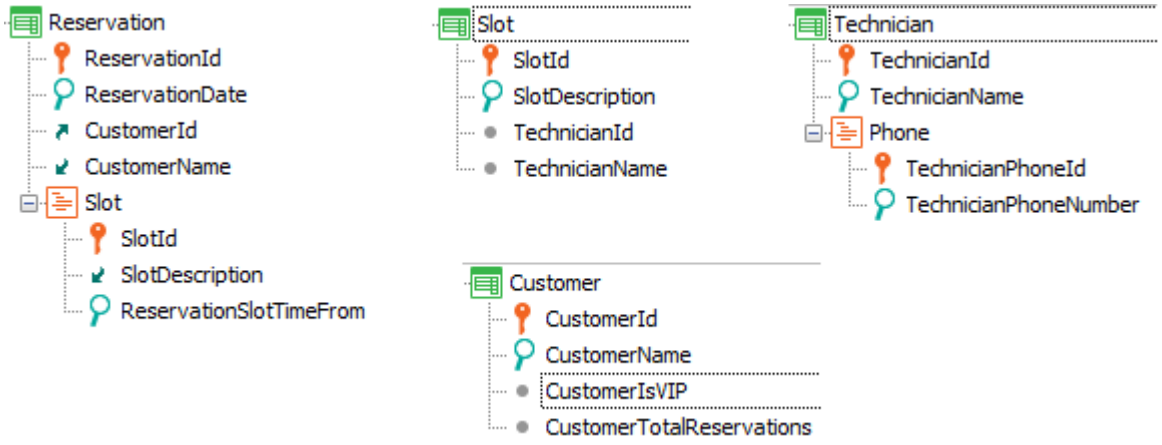
Que é como implementar um corte de controle por cliente. Ou seja, a tabela de reservas é percorrida, agrupando por CustomerId (é o que significa “unique CustomerId”, ou seja, que não manteremos os valores repetidos de CustomerId) e, para cada CustomerId único encontrado, são contados os registros que possuem o atributo ReservationDate, para esse cliente (condição implícita). E essa conta é atribuída ao atributo CustomerTotalReservation, que é da tabela estendida de Reservation.

É importante entender as diferenças entre as soluções, para escolher a que mais nos convém. Qual é a solução com melhor desempenho?

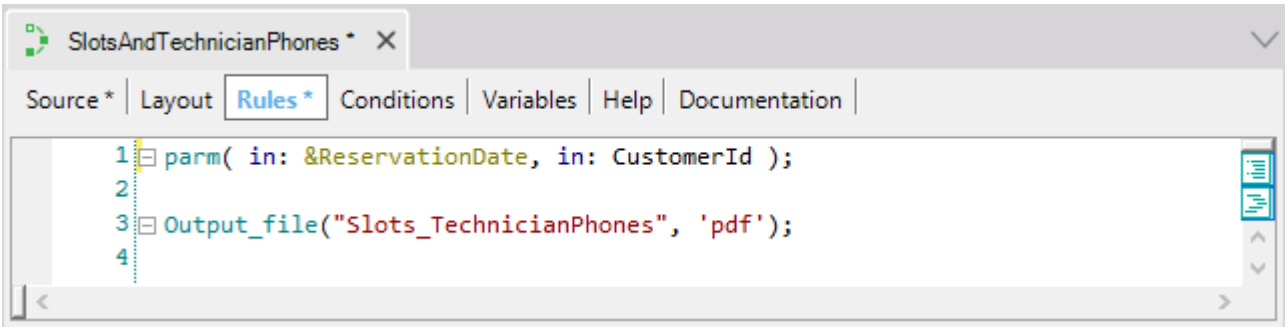
PERGUNTA 16 – CASOS DE FOR EACHS ANINHADOS

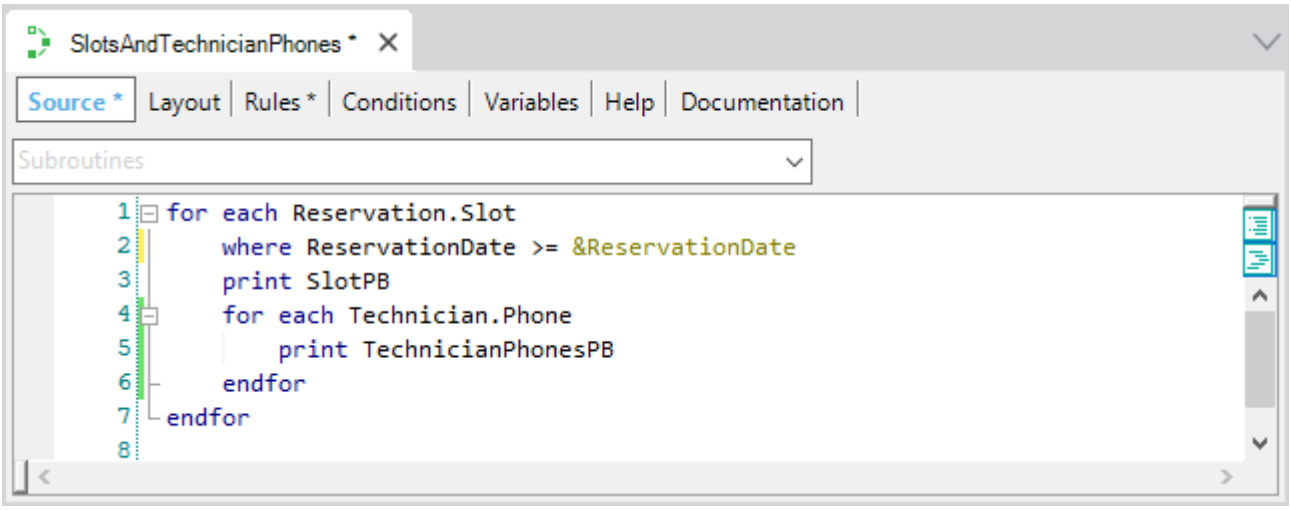
Na aplicação para um Cassino que estamos desenvolvendo, definimos as transações Customer, Slot e Reservation (esta última para permitir aos clientes realizar reservas de slots para serem usados em um determinado dia). Além disso, cada slot tem um técnico designado para executar os reparos que necessite:

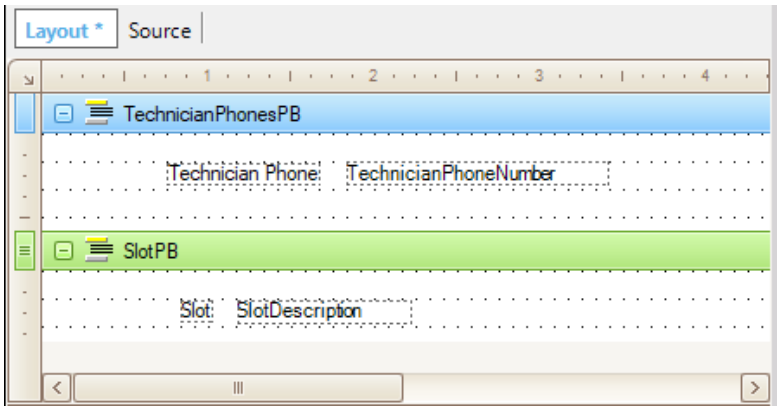
Deseja-se criar uma lista pdf que mostre para um cliente VIP determinado, a partir de uma determinada data, os slots que tem reservados e para cada um, os telefones do técnico do slot, caso seja necessário chamá-lo com urgência para reparar o slot.



Para resolver o requisito, é implementado um procedimento como o apresentado. Está bem programado? Analise-o.







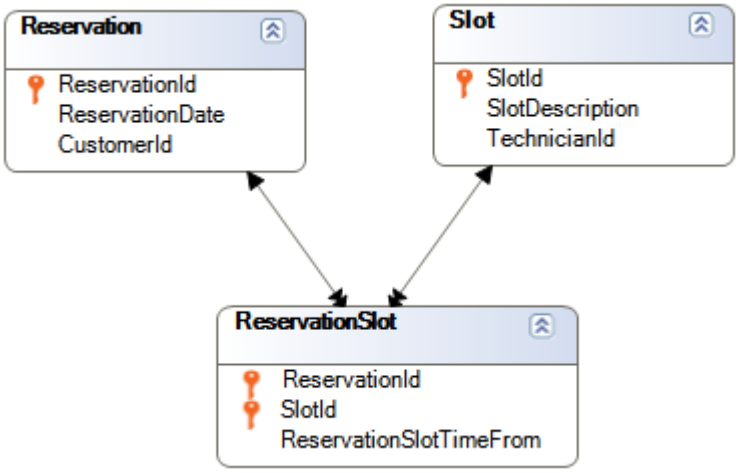
E nas propriedades, Call Protocol = HTTP

Na próxima pergunta, deverá pensar em como implementar isto, mas como uma tela web e não como um pdf.

RESPOSTA

Está bem programado. À primeira vista, poderia pensar que o Layout está invertido. Mas lembre-se de que a ordem dos printblocks não diz nada sobre como eles serão listados na saída. Aqui são apenas declarados e desenhados. É no Source onde são enviados para a saída com o comando print.

Vamos analisar se a informação listada é a solicitada. Temos um par de for each aninhados. No primeiro, dizemos explicitamente que a tabela base será a correspondente ao nível Slot da transação Reservation, ou seja, a de nome ReservationSlot. Devemos confirmar que dentro deste for each, não esteja sendo usado nenhum atributo que não pertença à tabela estendida de RESERVATIONSLOT, pois, nesse caso, a lista de navegação emitirá um aviso indicando que esse atributo não é acessível. Os atributos que devemos confirmar são aqueles que, dentro do for each, não estão dentro do segundo for each. No nosso caso, temos apenas dois atributos nessa situação: o do where e o do printblock chamado SlotPB. Estes atributos são SlotDescription, presente na tabela SLOT, e ReservationDate, presente na tabela RESERVATION. Se observarmos o diagrama de tabelas:



vemos claramente que a partir de RESERVATIONSLOT acessamos um único registro da tabela SLOT e um único registro da tabela RESERVATION. GeneXus deverá, então, acessar essas duas tabelas cada vez que itere no for each.

Com quais registros da tabela base trabalhará o for each? Claramente com aqueles que cumpram que, ao ir para a tabela RESERVATION para avaliar o valor de ReservationDate, este seja maior ou igual ao valor da variável &ReservationDate recebido por parâmetro. Só isso? Não, também deverá cumprir que o valor de CustomerId coincida com o que foi recebido por parâmetro diretamente nesse atributo. Lembremos que "receber em atributo" é estabelecer que esse atributo estará instanciado, ou seja, fará parte do contexto do objeto. Em outras palavras, sempre que esse atributo participar em algum lugar, o fará com o valor recebido quando o objeto foi chamado.

Como no for each que estamos analisando, já é acessada a tabela RESERVATION que o contém, então será aplicado um filtro automático por esse valor de CustomerId.

Uma observação importante: que o atributo recebido na regra parm pertença à tabela estendida do for each NÃO FAZ com que este o aplique automaticamente como filtro. Para que isso aconteça, o for each deve estar acessando particularmente a essa tabela da estendida para fazer algo.

Para entender melhor, vamos pensar no que teria acontecido se não quiséssemos listar os slots de um determinado cliente *a partir* de uma data dada, mas *em* uma determinada data. Como nesse caso, os dois filtros serão por igualdade, poderíamos querer receber os dois parâmetros diretamente nos atributos correspondentes. Ou seja, teríamos especificado:

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

E no Source:

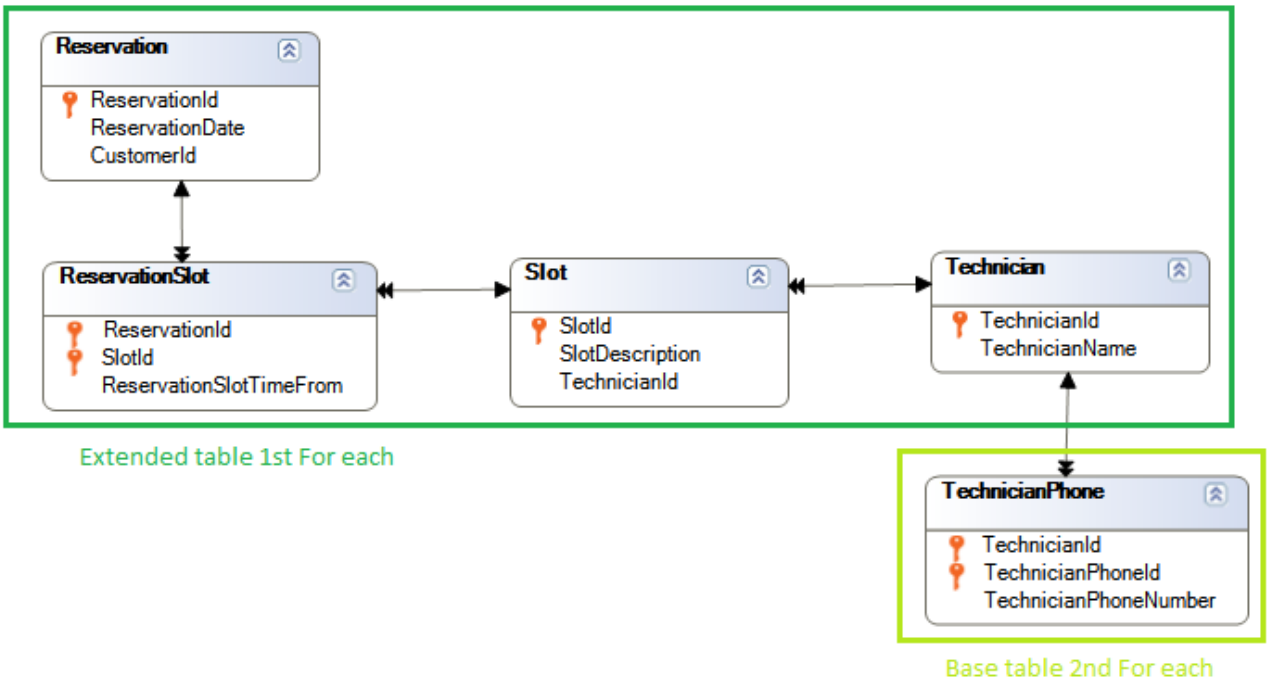
```
For each Reservation.Slot
print SlotPB
…
endfor
```

Mas, ao fazer isto, o primeiro for each já não precisará acessar a tabela Reservation, mas apenas Slot (para buscar o valor de SlotName). E, nesse caso então, os filtros por igualdade não serão aplicados, pois são sobre a tabela Reservation. Pode verificá-lo, observando a lista de navegação. Será percorrida toda a tabela ReservationSlot sem nenhum filtro.

Portanto, deverá, pelo menos, declarar um dos parâmetros como variável e usá-la em um where filtrando por igualdade. Por exemplo "where ReservationDate = &ReservationDate". Como CustomerId está nessa mesma tabela, agora sim aplicará esse filtro também.

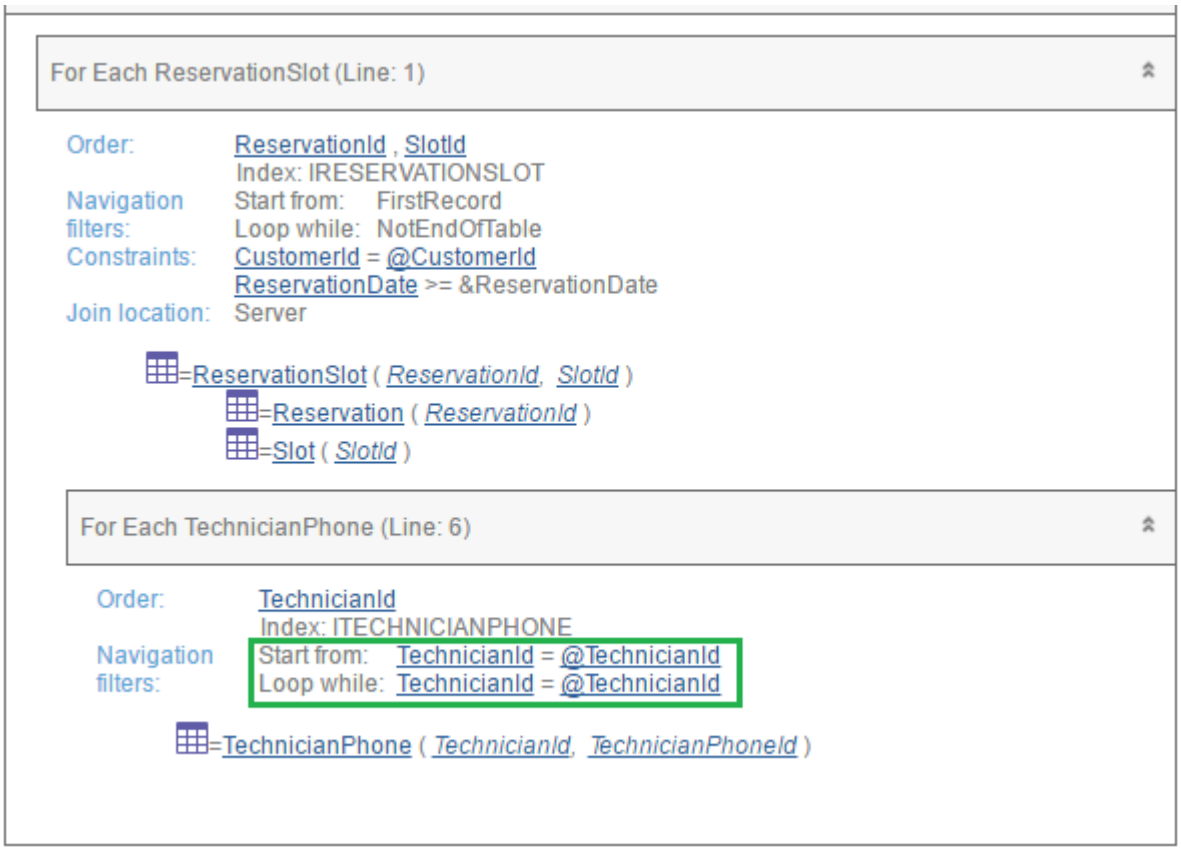
Agora vamos pensar no que acontece com o for each aninhado. Sua tabela base será claramente a associada ao nível Phone da transação Technician (ou seja, TechnicianPhone). A pergunta seguinte é: estabelece filtros implícitos para a informação que utilizará? Sabemos que sim, que mostrará os telefones do técnico de cada slot. Por quê?

GeneXus busca se existe relação entre a tabela estendida do for each externo e a tabela base do for each aninhado:



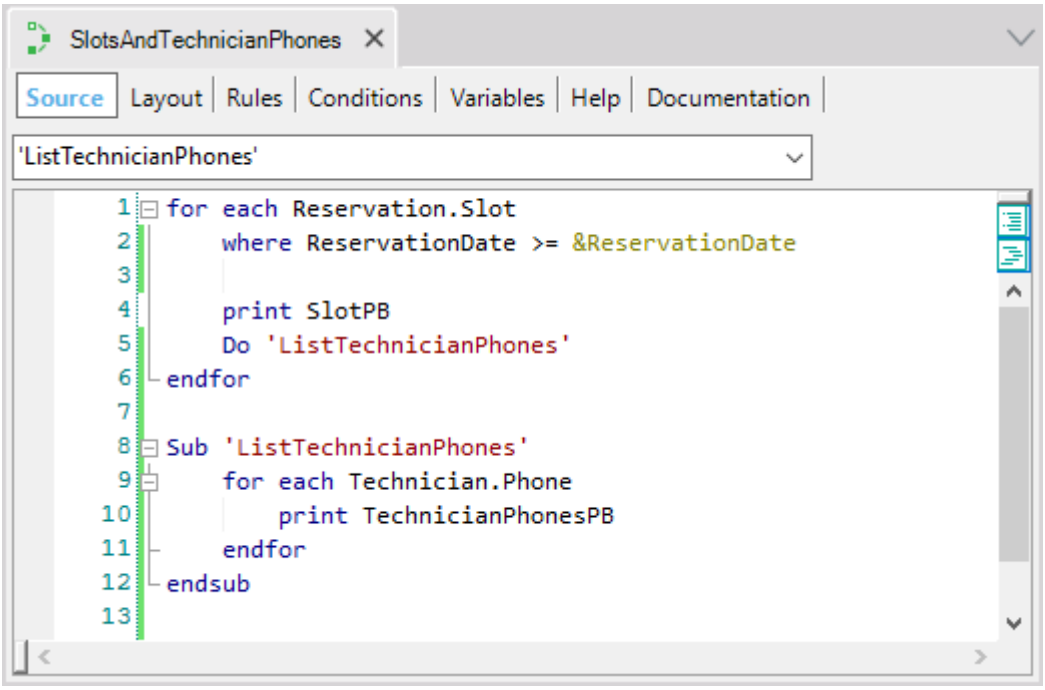
É outra maneira de procurar um relacionamento 1 para N, embora neste caso indireto. Se cada reservation slot possui um TechnicianId e na tabela a ser navegada também existe um TechnicianId,

então GeneXus entende que, pela relação entre a informação, se trataria do mesmo. Por isso realiza o join. Vemos isso claramente na lista de navegação:

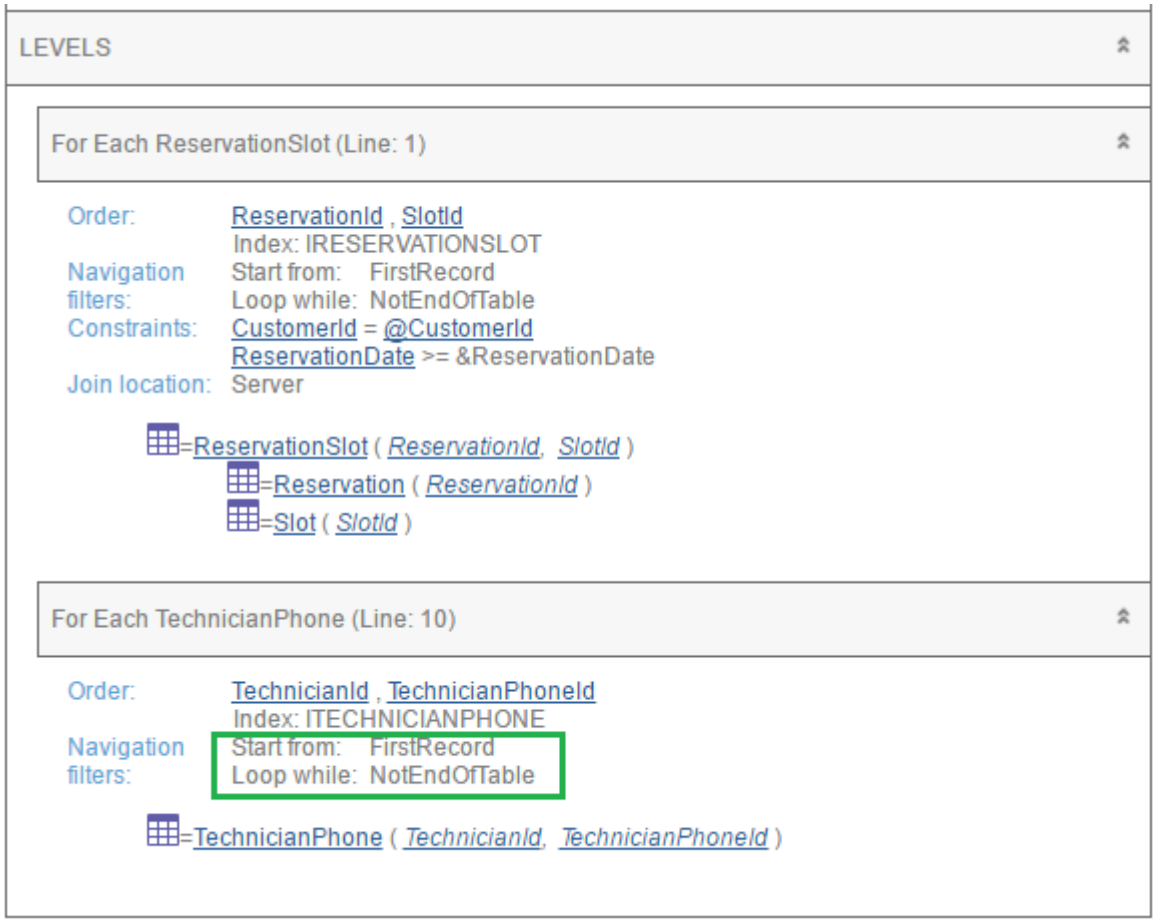


Onde o @ sempre está indicando informação contextual (observe o @CustomerId, que se refere ao valor recebido no atributo CustomerId na regra parm). Este @TecnhicianId refere-se ao valor do atributo desse nome da tabela Slot acessada pelo primeiro for each.

Se não fosse isso que o programador queria, existe uma maneira de evitar esses filtros automáticos? A resposta é sim, utilizando uma sub-rotina para encapsular o código deste segundo for each (relaxe, não tinha por que saber; isso não é dado no curso GeneXus):



Se fizer isto, na lista de navegação, verá agora:

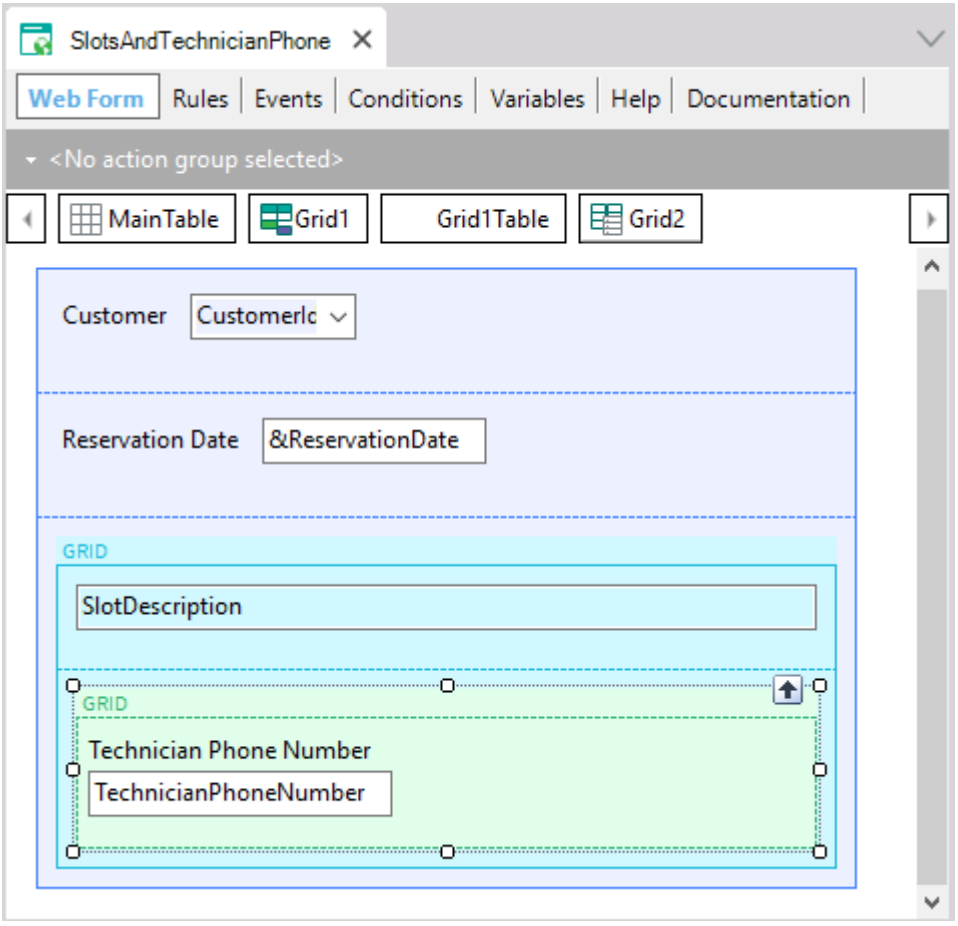


PERGUNTA 17 – GRIDS ANINHADOS

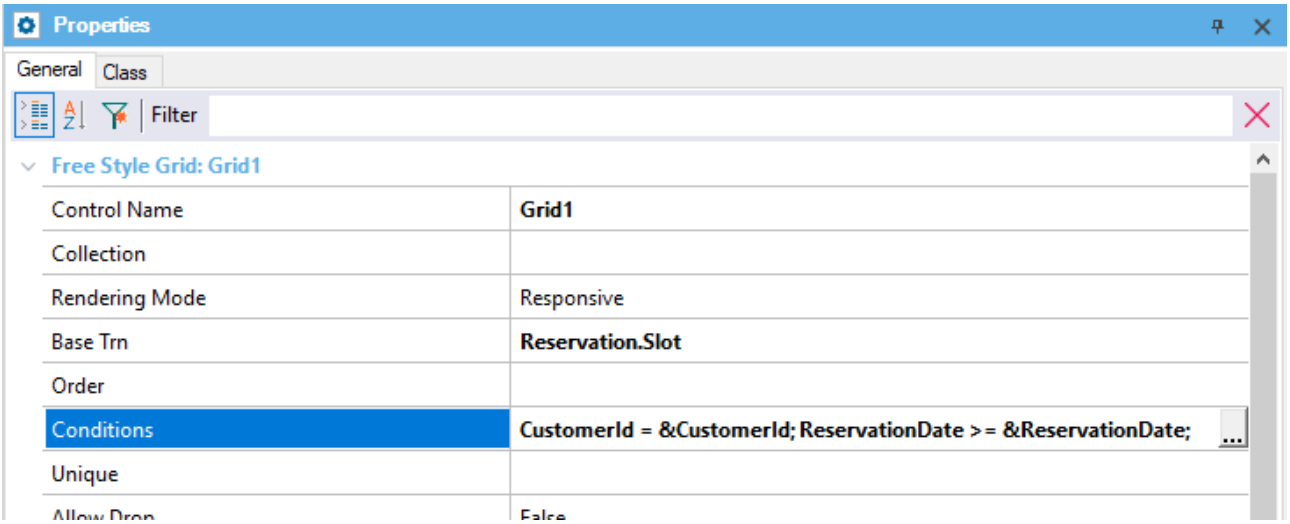
Agora, queremos implementar o mesmo requisito da pergunta anterior, mas, em vez de fazê-lo em uma lista pdf, queremos fazê-lo em uma tela web.

RESPOSTA

Para implementar isto mesmo como uma tela web e não como um pdf, poderíamos colocar duas variáveis para o usuário inserir os valores desejados de cliente e data em cada oportunidade e, em seguida, a informação seja exibida com um par de grids aninhados, onde o primeiro, NECESSARIAMENTE deve ser freestyle. NÃO pode ser um grid padrão. Por quê? Porque um grid padrão suporta apenas colunas fixas, do tipo atributo/variável. E aqui o que queremos colocar dentro do grid é, além do atributo SlotDescription, OUTRO GRID!

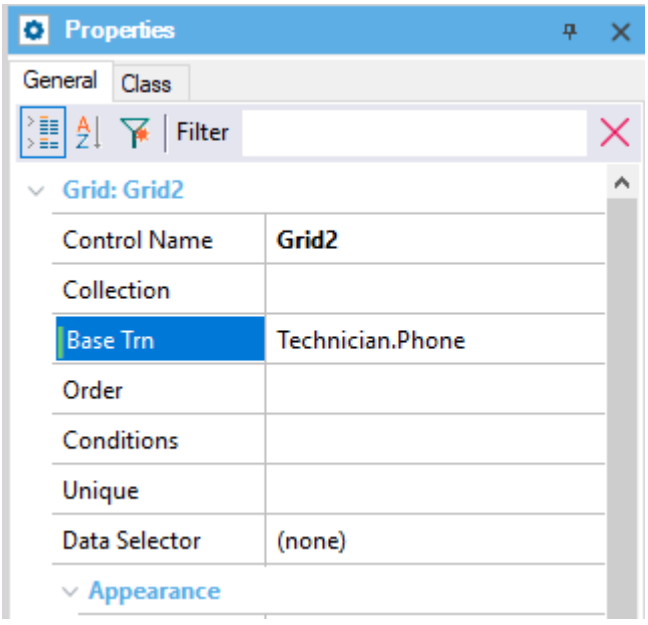


Para o grid Freestyle indicamos também sua transação base, igual ao que fizemos com o For each externo no procedimento, e os filtros escrevemos através da propriedade Conditions do grid (que é equivalente às cláusulas where do for each):



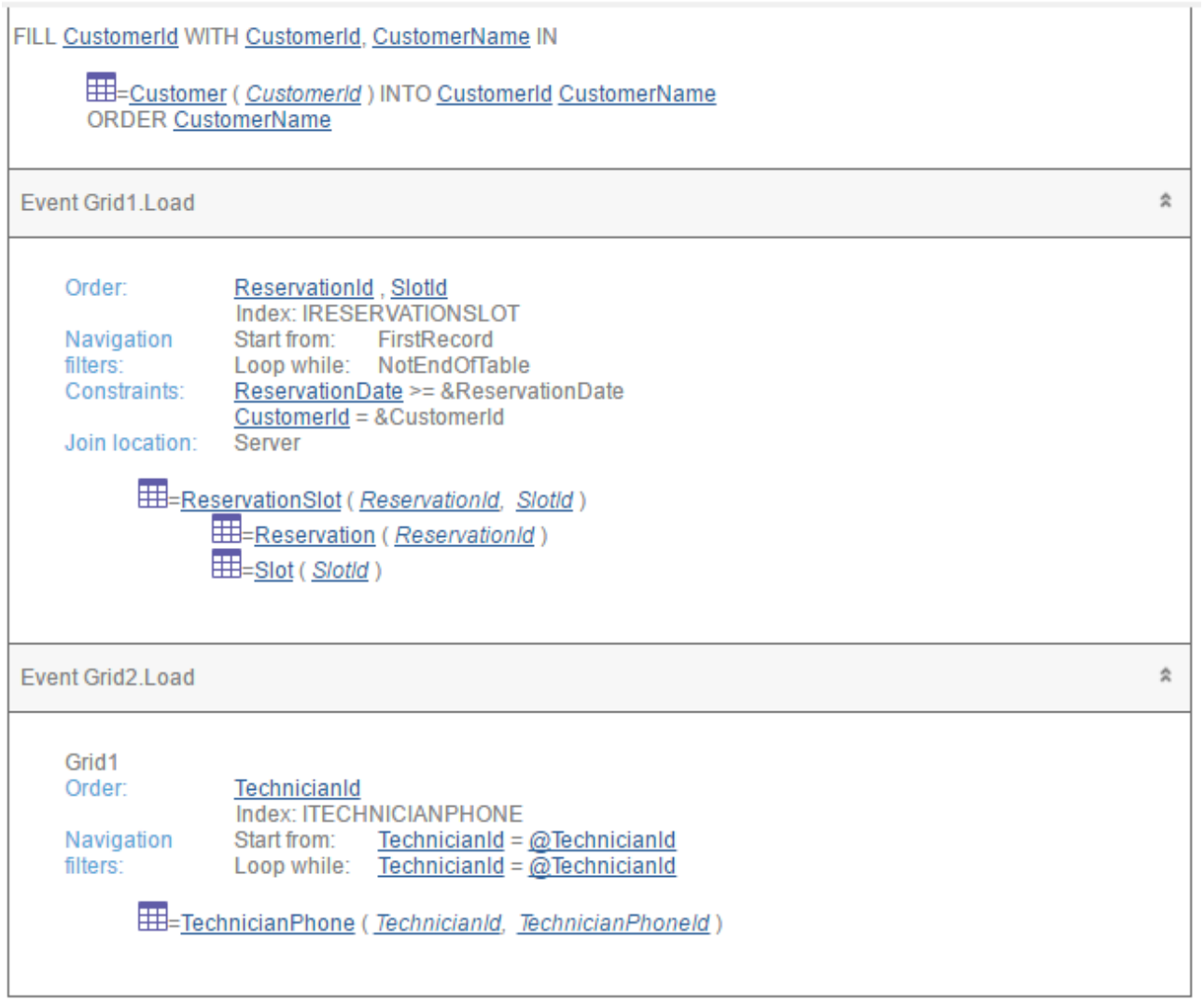
Se a condição por CustomerId não tivéssemos escrito utilizando uma variável no próprio form, mas tivéssemos recebido o CustomerId por parâmetro, valeria exatamente o mesmo que vimos no caso do procedimento.

O que acontece com a informação do grid aninhado? Aqui nós colocamos um grid padrão, mas poderia ser um freestyle também. Observemos que tudo o que fizemos foi colocar nele o atributo TechnicianPhoneNumber. Para garantir a navegação que queremos, também poderíamos especificar sua transação base, o que sempre sugerimos como boa prática:



Podemos nos perguntar se no caso de grids aninhados valerá o mesmo que no caso que já estudamos de for eachs aninhados. E a resposta é sim.

Ou seja, aplicará o filtro por TechnicianId do slot em questão, exatamente igual ao caso de for eachs aninhados do procedimento.



Observe, también, que modificamos o Control Type da variável &CustomerId, para exibir em execução um combo box com os valores de CustomerName da tabela Customer, de maneira que, quando o usuário selecione o nome do cliente desejado, internamente se atribui na variável seu CustomerId. Isto é utilizando a propriedade Control Type = “Dynamic Combo Box” do controle &CustomerId. A lista de navegação está nos indicando no início a navegação da tabela Customer para resolver isto.

Se, em vez de ter especificado os grid como aninhados, os tivéssemos apresentado como paralelos, lá deveríamos ter programado explicitamente para o grid de telefones dos técnicos o filtro por TechnicianId em suas condições. Porque para grids paralelos, assim como acontece para for eachs paralelos, as navegações não são relacionadas. E é lógico, por que se relacionariam automaticamente, se não há nada que faça entender que desejam vincular? No caso de grids aninhados, alguém claramente pensaria que o mais provável é que, se a informação está vinculada na realidade, também se queira apresentá-la vinculada.

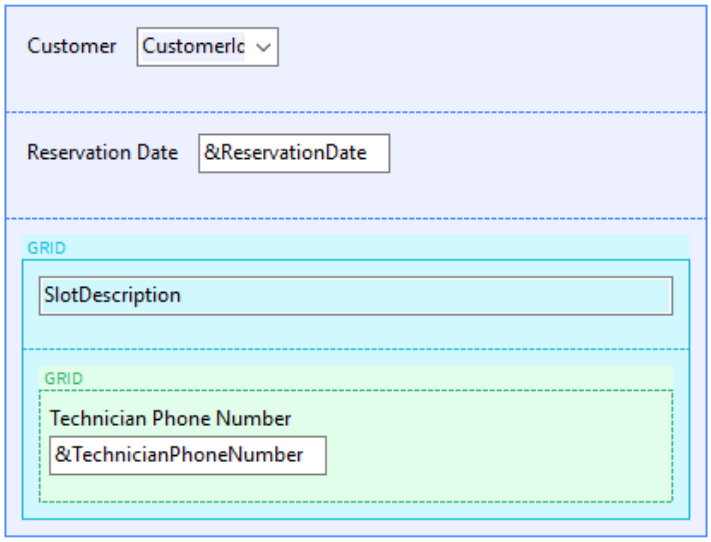
No caso da lista pdf que vimos, quando não queríamos que GeneXus aplicasse os filtros implícitos ao for each aninhado, criávamos uma sub-rotina, cujo efeito era desconsiderar o contexto. Como faríamos aqui se, para cada SlotDescription da reserva, quiséssemos ver no grid aninhado todos os telefones dos técnicos, e não os do slot? Ou seja, como fazemos para não aplicar o filtro implícito, que é o que realiza o join?

A maneira mais fácil será que o grid aninhado seja sem tabela base. Quando o grid está sem tabela base, toda sua carga fica nas mãos do desenvolvedor, por meio do evento Load do grid.

Vendo a lista de navegação anterior (a dos grids com tabelas base), se indica o evento Load para cada grid e, abaixo, se indica uma navegação de tabela. O aluno poderia pensar que o evento Load é disparado apenas uma vez, onde é executado esse tipo de for each implícito. Não deve ser confundido: sabemos que quando os grids têm tabela base, é disparado um evento Load para cada registro acessado na navegação, imediatamente antes de carregar suas informações no grid.

Apenas acontece que o evento Load de um grid é disparado uma única vez quando não possui tabela base.

Portanto, o que faríamos seria modificar no grid aninhado o atributo TechnicianPhoneNumber por uma variável:



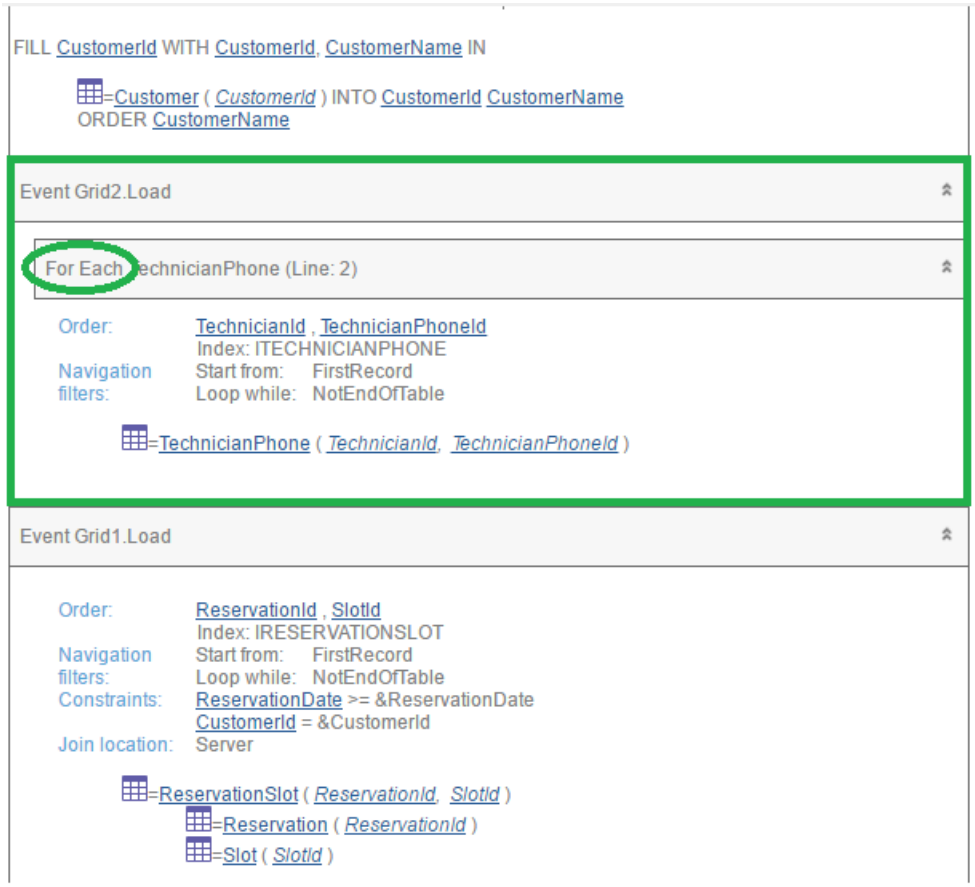
Mas também devemos ter cuidado de remover qualquer referência à tabela ou atributo em qualquer outro lugar relacionado com o grid. Obviamente, devemos esvaziar a propriedade Base Trn.

E no evento Load do grid:

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

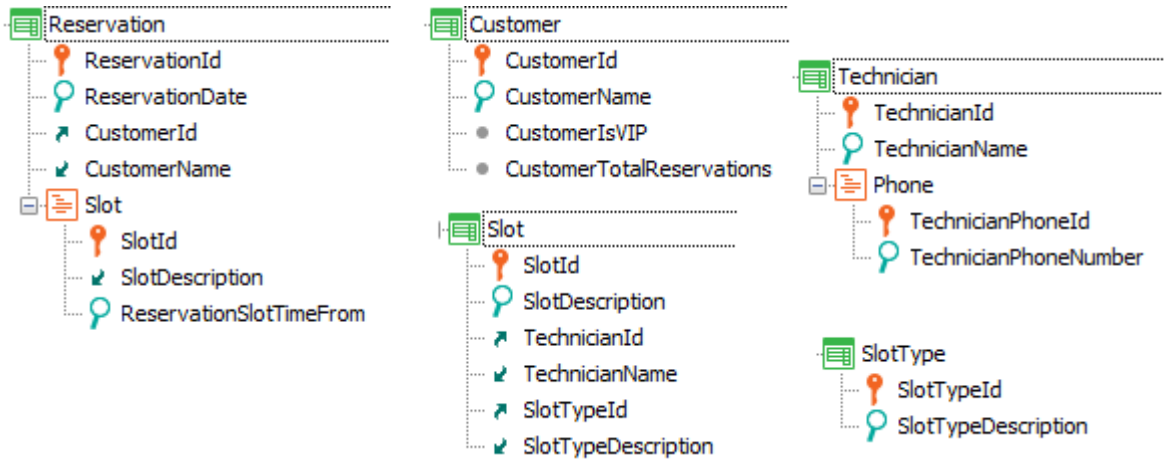
Se não colocarmos o comando Load dentro do evento, o que acontecerá? Nenhuma linha será adicionada ao grid. Ou seja, nenhum telefone será carregado, mesmo que sua tabela seja navegada.

Olhando para a lista de navegação, agora, vemos claramente a diferença entre o grid com tabela base e o grid sem tabela base:



PERGUNTA 18 – GRIDS ANINHADOS (CONTINUAÇÃO)

Estendendo a pergunta anterior, agora adicionamos para cada Slot um tipo, que é registrado em uma transação separada:

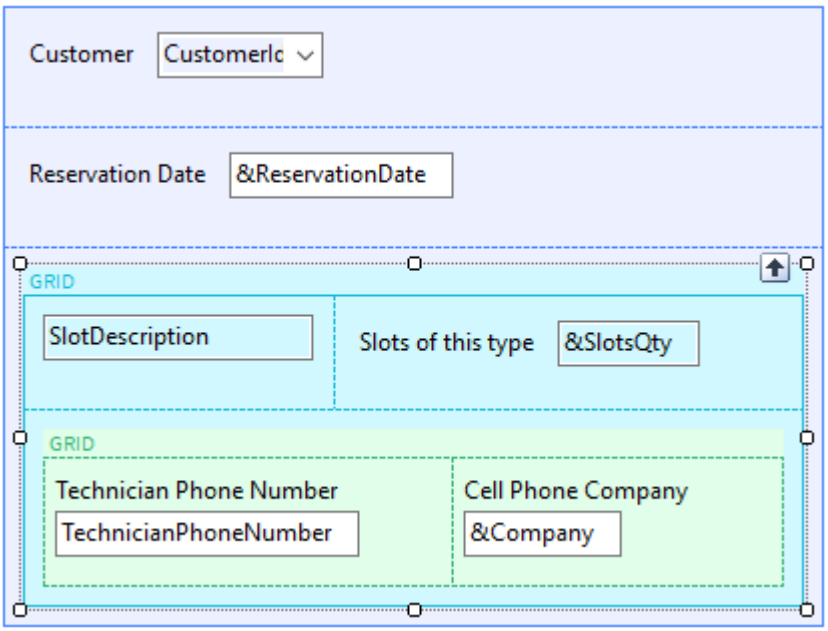


Deseja-se agora, que o web panel que mostra os slots reservados para um cliente a partir de uma determinada data, mostre também:

1. para cada slot listado, a quantidade de slots do mesmo tipo existente no Cassino; e
2. para cada telefone do técnico associado, a companhia telefônica correspondente (que suponhamos, não está registrada na base de dados, mas que se encontra com uma proc a partir da análise dos primeiros dígitos do número –eventualmente poderia ser um serviço externo-).

Analise por que a solução encontrada a seguir não funciona (supondo que o procedimento CellPhoneCompany esteja bem programado) e proponha uma solução completa e bem desenvolvida. Tente discutir outras possibilidades.

Web Form:



Events:

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

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

Como ajuda: um dos problemas detectados pelo aluno é que sempre é mostrado valor 0 para os Slots desse tipo, quando na verdade sempre há pelo menos um slot desse tipo: o que está sendo carregado no grid.

Então, tenta a seguinte modificação e ainda mostra 0:

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

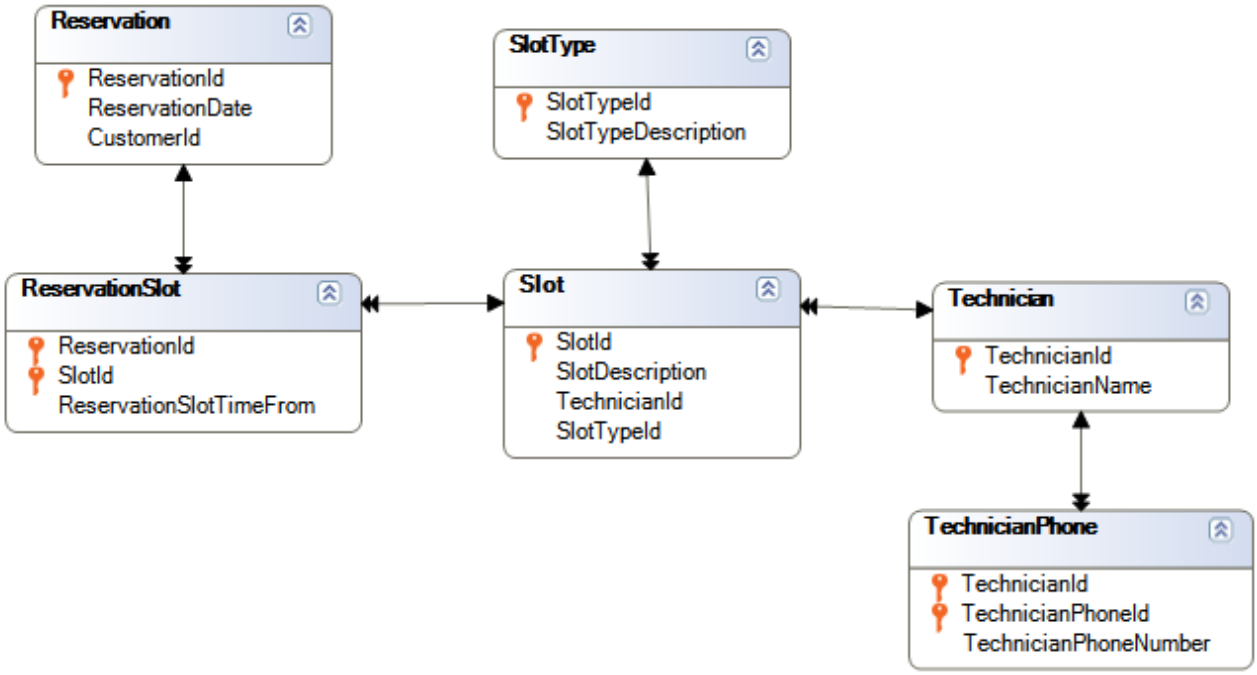
RESPOSTA

Foram adicionadas duas variáveis, &SlotsQty para o grid externo (o de nome Grid1) e &Company para o grid interno, &Company.

Ambos os grids estão com tabela base, portanto, os atributos serão carregados automaticamente nos dois grids, de acordo com suas navegações.

Mas, para carregar as variáveis, como fazemos? Se um grid tiver tabela base, sabemos que o evento Load do grid será disparado n vezes: uma vez para cada linha a ser carregada, imediatamente antes de carregá-la, e o desenvolvedor não precisará escrever o comando Load. Portanto, é no evento Load onde deve ser atribuído valor às variáveis que foram incluídas no grid.

Comecemos pelo grid mais externo. É no evento Load que temos que realizar o cálculo da quantidade existente de slots do mesmo tipo que o que se está por carregar nesta linha que disparou o Load. Portanto, existe um contexto para tudo o que se escreva dentro do Load: estamos posicionados em um registro da tabela base, ReservationSlot, e em cada um dos registros associados por tabela estendida que necessitemos. Em outras palavras, estarão instanciados os atributos da tabela ReservationSlot para esse SlotId, da tabela Slot para esse slot se precisarmos deles, da tabela SlotType para esse slot, se precisarmos (assim como de Technician para esse slot, e de Reservation).



Para calcular a quantidade de slots da tabela Slot, que possuem o mesmo SlotTypeId que o slot instanciado já dentro do Load, como fazemos? O aluno escreveu inicialmente:

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

Endevent
```

O que tentou fazer? Por que condicionou dessa forma os registros a serem contados com a fórmula count?

Certamente tenha tentado antes de adicionar essa condição com a fórmula sem condições:

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

Talvez pensando que, por contexto, já fosse contar apenas os slots desse tipo. Vamos analisar por que não é assim. Sabemos que as fórmulas inline determinam a tabela a ser navegada pelos atributos especificados nela, sem considerar o contexto. Em seguida, atendem ao contexto para adicionar os filtros correspondentes. No nosso caso, a tabela a ser navegada pelo count será claramente Slot. Mas como a fórmula está dentro de um contexto em que estamos posicionados em um registro de ReservationSlot que possui um SlotId, aplicará esse filtro por igualdade. Então, sempre contará um único registro, o que corresponde a esse SlotId.

Obviamente, não podemos usar a fórmula dentro do contexto do evento Load, porque precisamos que não execute esse filtro por SlotId. Todas as soluções procuradas pelo aluno estão incorretas, pois não pode romper com esse contexto e, portanto, que seja adicionado automaticamente o filtro por SlotId.

Não teremos alternativa senão fazer o cálculo em um procedimento, para romper com essa contextualização.

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

E programaríamos o procedimento SlotsQtyOfAType:

Rules:

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

Source:

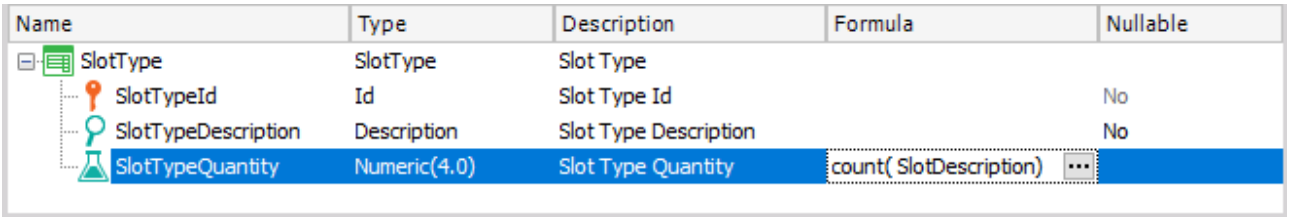
```
	&SlotsQty = count( SlotDescription, SlotTypeId = &SlotTypeId)
```

Ou utilizar uma sub-rotina, para não sairmos do próprio código, especialmente se a proc não vá ser utilizada por nenhum outro objeto:

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

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

Nas sub-rotinas não podem ser passados parâmetros e, como seu código se encontra no mesmo programa que as chama, veem o estado das variáveis como estão no momento em que a sub-rotina foi chamada. É por isso que carregamos o valor da variável &SlotTypeId, para que a sub-rotina possa usá-la.

E se o aluno tivesse pensado em definir na transação SlotType um atributo fórmula, como apresentamos a seguir?



E no web form, colocar esse atributo em vez da variável &SlotsQty, ou, se não quiser tocar no web form, escrever no Load:

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

Funcionará corretamente! Observe que não é o mesmo, portanto, um atributo fórmula (global) e uma fórmula inline (local). Este exemplo torna mais claro o efeito de um ATRIBUTO fórmula.

Obviamente, não será uma boa prática preencher uma transação com atributos fórmula apenas porque é útil para o caso específico da implementação de um painel. Deve ser encontrada o balanço custo/benefício. Se esse cálculo for necessário em outros lugares, então faz todo sentido a definição da fórmula no nível de um atributo na transação.

Outra opção teria sido implementar o grid sem tabela base. Ou seja, sem transação base e sem atributos "soltos" nos eventos relacionados. E, é claro, com variáveis no grid (então, em vez do atributo SlotDescription, inseriremos uma variável &SlotDescription). Mas teremos o mesmo problema de antes com o aninhamento. O evento Load será:

```
Event Grid1.Load()
	for each Reservation.Slot
		&SlotDescription = SlotDescription
		&SlotTypeId = SlotTypeId
	Do 'SlotsQuantity'
		Load
endfor
endevent

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

Endsub
```

Não podemos esquecer de escrever o comando Load para carregar efetivamente a linha no grid.

COMANDO LOAD EM UM EVENTO LOAD DE UM GRID COM TABELA BASE

Agora temos que analisar a carga do grid aninhado. Claramente, o comando Load é desnecessário.

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

Mas, qual é o efeito? Não tem efeito neste caso. O que veremos em execução é exatamente o mesmo. Entretanto, escrever explicitamente o comando Load faz com que GeneXus não coloque implicitamente no final de seu código esse mesmo comando. Isto é útil se eu quiser, como desenvolvedor, que apenas alguns registros que "passam" pelo evento Load sejam efetivamente carregados no grid, e não todos.

Por exemplo, se eu quiser que, se a empresa for "X", não seja carregado o telefone no grid:

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

endevent
```

Inclusive, se quiser carregar mais de uma linha do grid no mesmo evento Load para um registro, então terei que escrever duas vezes o comando Load dentro do evento.

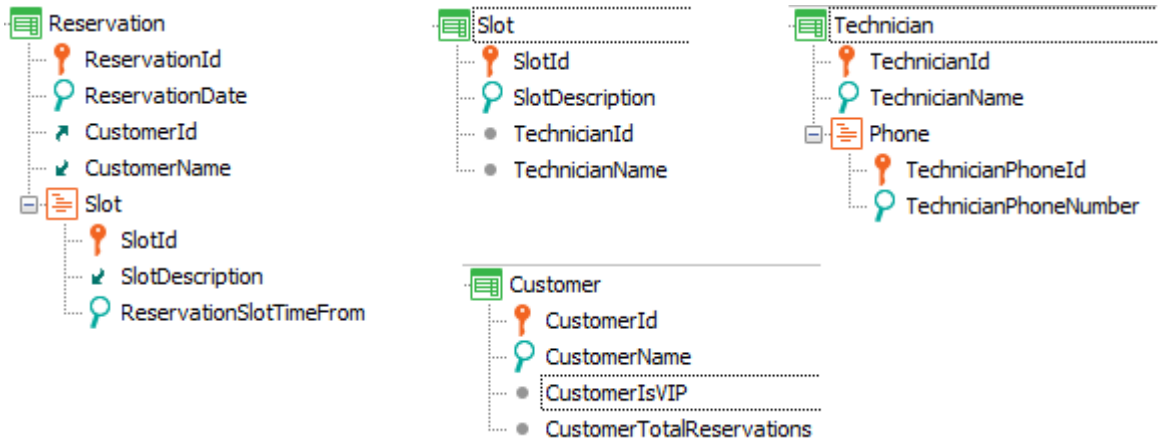
Com estes exemplos, mostramos que, em alguns casos, pode ser necessário escrever explicitamente o comando Load em um evento Load de um grid COM tabela base.

QUANDO COLOCAR ATRIBUTOS OCULTOS EM UM GRID

Um erro frequente que encontrará nas implementações de alunos: colocar o atributo TechnicianId no grid aninhado, mas oculto. Por que é desnecessário? Nada do que precise no evento Load exige que o atributo esteja no grid, porque no momento de disparar o evento Load é quando se está na base de dados, com acesso à tabela estendida. Seria necessário deixar o TechnicianId carregado no grid somente se ele for necessário posteriormente, em um evento do usuário, uma vez que a informação já foi carregada no grid e já não estamos mais conectados à base de dados, mas estamos trabalhando com a informação da tela, exclusivamente.

PERGUNTA 19 – UNIQUE OU CORTE DE CONTROLE?

Continuando com a aplicação:



Suponha que se queira criar uma lista PDF que mostre para cada slot que tenha sido reservado alguma vez, a quantidade de vezes que foi reservado.

Um aluno A apresenta a seguinte solução:

```
For each Slot
	&slotsInReservations = count( ReservationSlotTimeFrom)
	print SlotPB //com o atributo SlotDescription e a variável &slotsInReservations

endfor
```

Enquanto um aluno B apresenta esta outra:

```
For each Slot
	&slotsInReservations = 0
	For each Reservation.Slot
		&slotsInReservations += 1
	endfor
	print SlotPB // com o atributo SlotDescription e a variável &slotsInReservations
endfor
```

Você indica que ambas as soluções são equivalentes, embora o aluno B esteja implementando com o for each interno o que a fórmula count do aluno A já realiza e faz melhor, pois foi projetada para ser otimizada. Geralmente (mas nem sempre) é melhor usar uma fórmula do que fazer manualmente o que a fórmula já faz.

Mas, além disso, você diz a eles que ambas as soluções estão erradas em uma coisa: não está sendo solicitada uma lista de todos os slots, incluindo aqueles que nunca foram reservados.

Então o aluno A apresenta a seguinte solução:

```
for each Reservation.Slot
       unique SlotId
       &slotsInReservations = count( ReservationSlotTimeFrom )
       print SlotPB //com o atributo SlotDescription e a variável &slotsInReservations
endfor
```

Enquanto o B apresenta esta outra:

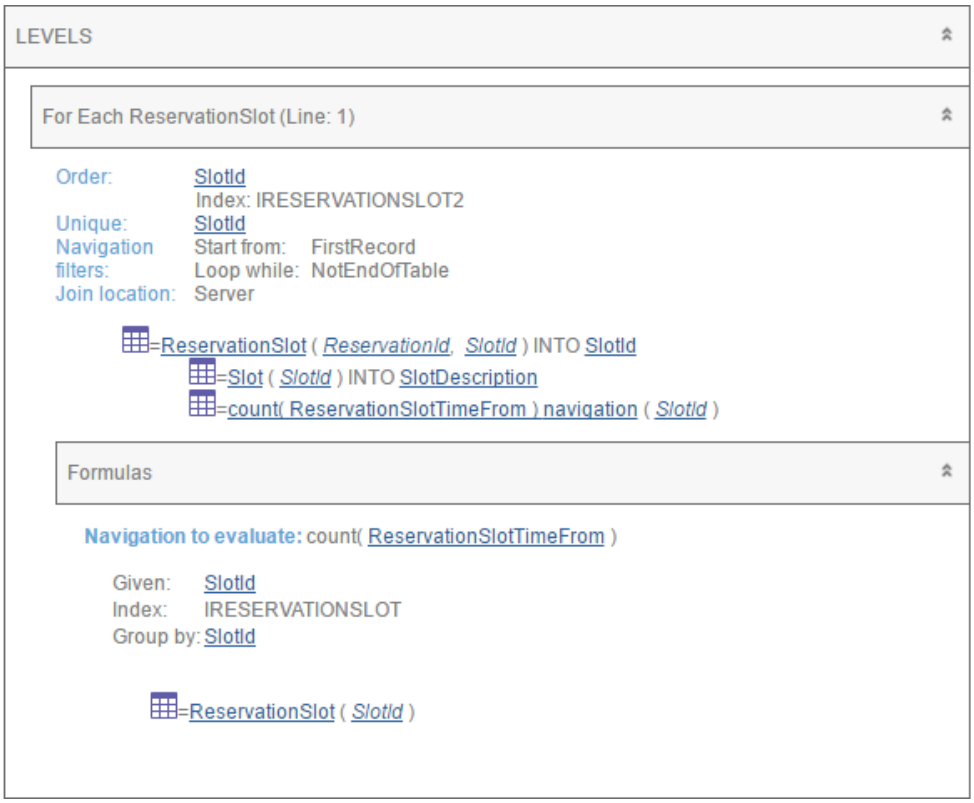
```
for each Reservation.Slot
       order SlotId
       &slotsInReservations = 0
       for each Reservation.Slot
              &slotsInReservations += 1
       endfor
       print SlotPB //com o atributo SlotDescription e a variável &slotsInReservations

endfor
```

Você dirá a eles que as duas soluções são equivalentes?

RESPOSTA

Sim, ambas serão equivalentes. De fato, se olhar as listagens de navegação.

Esta é a de A:



Onde pode ver que a fórmula aplica o filtro por SlotId (observe o “Given: SlotId”).

E esta de B:

B implementou um corte de controle, enquanto A usou a cláusula unique destinada a esse tipo de cenários, em que deseja navegar em uma tabela e realizar uma agregação nessa mesma tabela, de acordo com algum atributo ou atributos que queremos usar apenas uma vez na saída.

É mais provável que uma solução que tenha sido implementada para resolver certos cenários e que contenha uma fórmula tenha mais chances de ser resolvida com eficiência, em vez de uma solução na qual devemos implementar o que a fórmula já traz implementado. Mas isto nem sempre é assim.

Aos alunos poderia então ficar a ideia de que sempre que precisarem implementar um corte de controle, poderiam evitá-lo usando a cláusula unique, mas isto não é verdade.

Por exemplo, se o que é solicitado não foi uma agregação, mas que para cada slot reservado alguma vez, sejam mostradas as datas de reserva, não teremos outra alternativa a não ser implementar um corte de controle:

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

endfor
```

Não há maneira de fazer isso com uma cláusula unique.

Pergunte ao aluno se existe alguma diferença entre colocar na cláusula order o atributo SlotId e colocar o atributo SlotDescription.

Mas, por outro lado, para o problema original, não é uma solução melhor a que o aluno A implementou (join), mas simplesmente adicionando um if?:

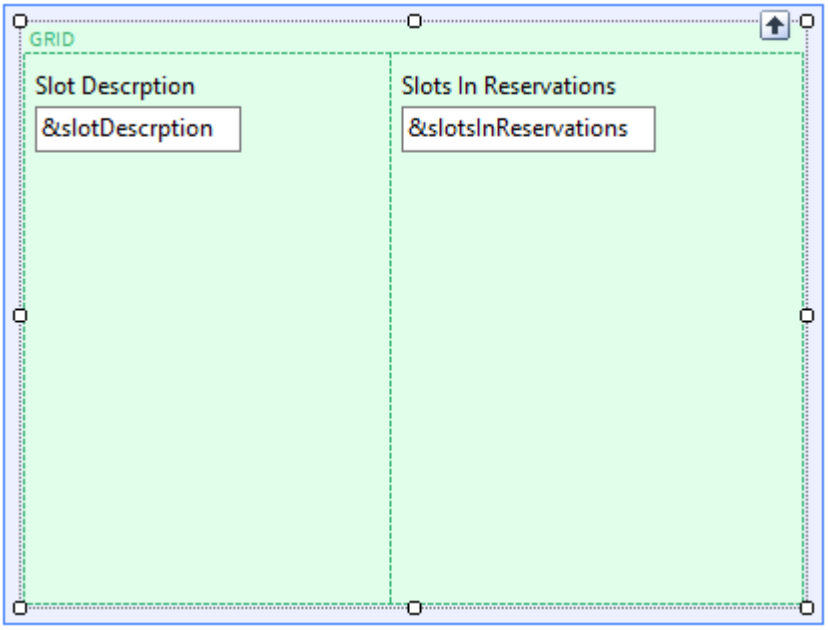
```
For each Slot
	&slotsInReservations = count(ReservationSlotTimeFrom)
	If &slotsInReservations > 0
		print SlotPB //com o att SlotDescription e a var &slotsInReservations
	endif

endfor
```

Espera-se que haja uma quantidade significativamente menor de registros na tabela Slot do que em ReservationSlot. Haverá diferença entre esta solução e a do mesmo aluno, mas navegando apenas ReservationSlot com unique?

PERGUNTA 20 – UNIQUE EM GRID (CONTINUAÇÃO)

Agora, solicita-se que, em vez de resolver o anterior, como pdf, resolva-o em um web panel.

O aluno então decide que terá que inserir um grid sem tabela base:



Para poder codificar no evento Load sua solução com a cláusula unique:

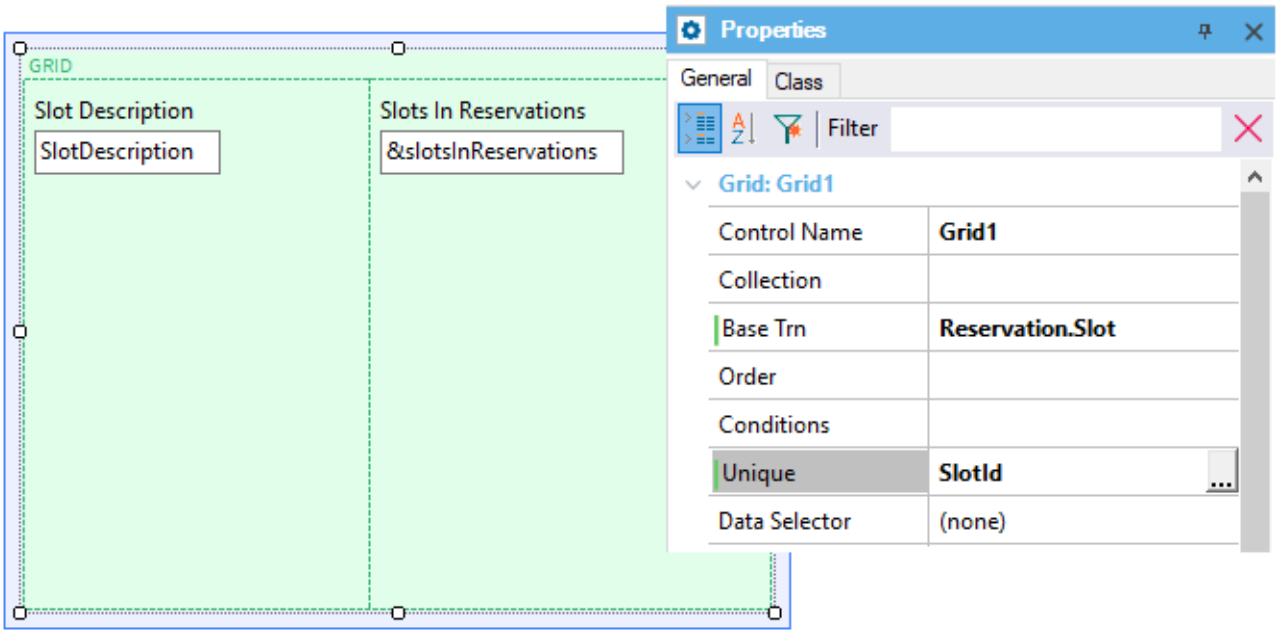
```
Event Grid1.Load()
	for each Reservation.Slot
		unique SlotId
		&slotDescrption = SlotDescription
		&slotsInReservations = count( ReservationSlotTimeFrom )
		Load
	endfor

endevent
```

Como não há maneira de fazer isso com um grid com tabela base. Explique como isso seria feito.

RESPOSTA

É raro que o que pode ser feito com for each não possa ser feito com um grid com tabela base, pois nesse caso por trás existe um tipo de for each. O mesmo pode ser dito sobre um grupo de data provider. São elementos equivalentes em sua lógica.

Portanto, será suficiente perguntar ao aluno se não haverá talvez uma propriedade Unique no nível do grid.




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

PERGUNTA 21 – UNIQUE EM DATA PROVIDER E TRANSAÇÃO DINÂMICA (CONTINUAÇÃO)

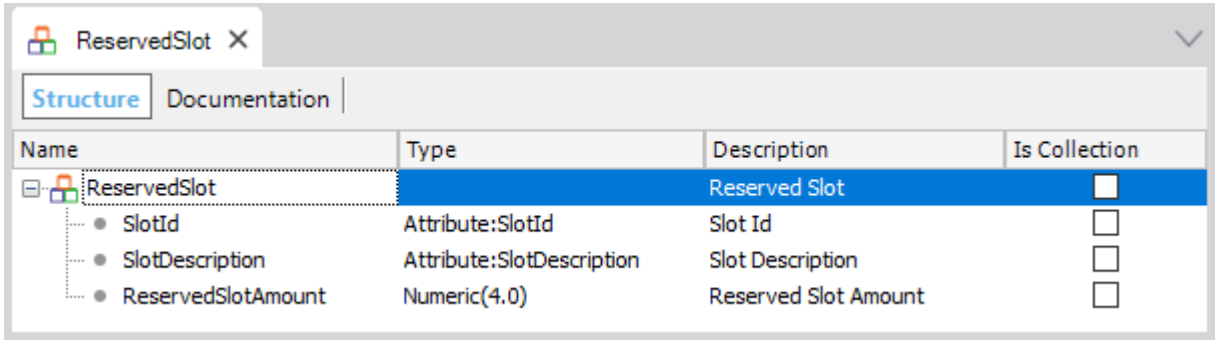
Você deseja que os alunos possam explorar as soluções mais interessantes para resolver o requisito, integrando os conceitos fornecidos no curso. Pelo que agora propõe: “Se agora queremos mostrar no grid (ou no pdf) os slots classificados por quantidade de reservas, da maior para a menor, como implementa? ”

Imagine as possíveis respostas.

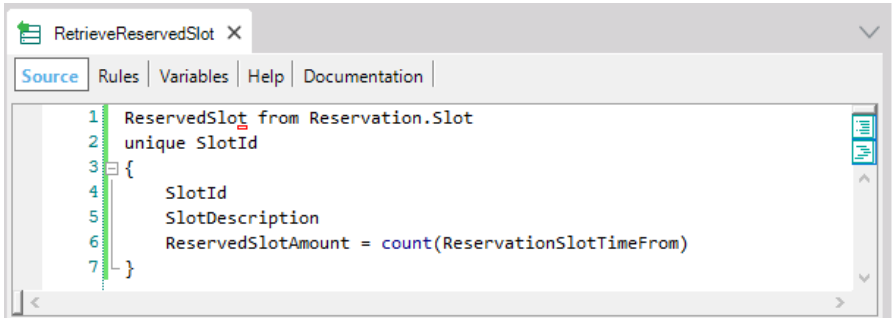
RESPOSTA

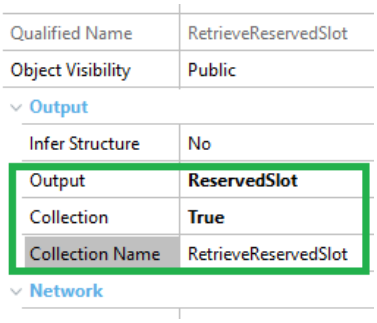
Alguns alunos poderão pensar em carregar em um SDT coleção os valores de SlotDescription e &slotsInReservations e, em seguida, no web panel ou procedimento, percorrer a coleção ordenada.

Assim, definir por exemplo o SDT:



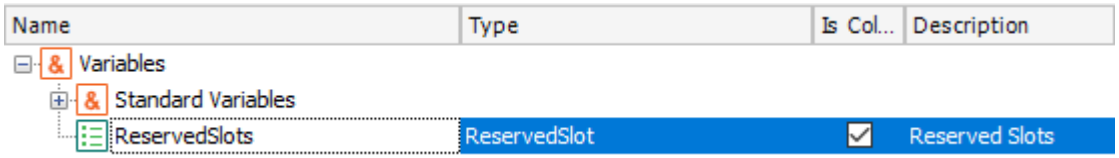
E carregá-lo com um Data Provider::



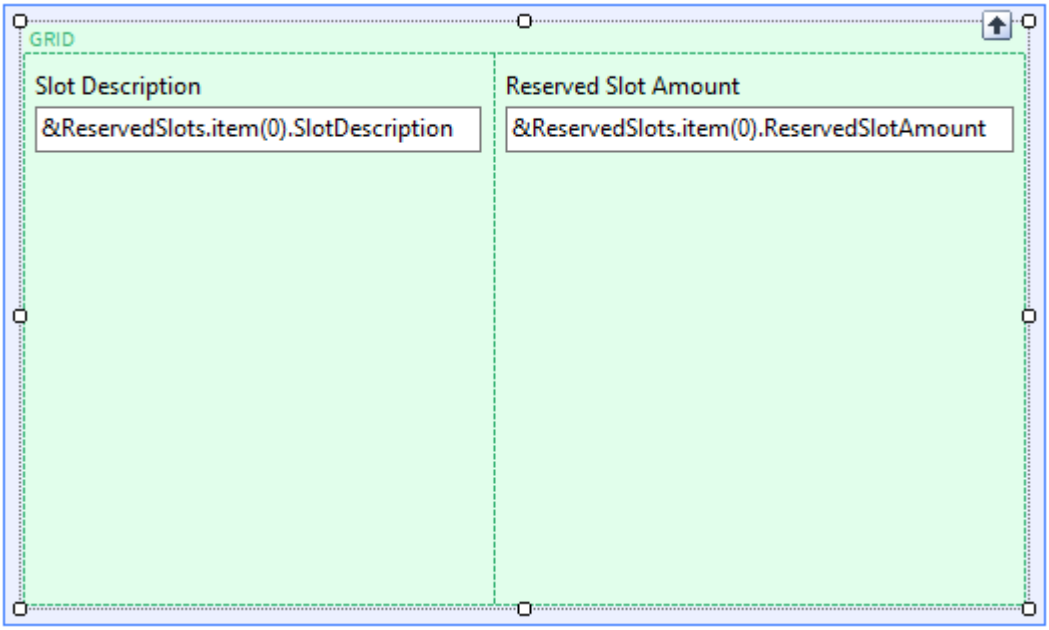


Observe que é análoga a forma de declarar o grupo do data provider e codificar o for each com a cláusula unique na lista ou no web panel sem tabela base.

Então, em seguida, no web panel poderia ser definida uma variável coleção do SDT:



Inseri-la no web form, após o que GeneXus a apresentará automaticamente em um grid:



Que também mostrará automaticamente com o conteúdo que tenha a variável coleção antes que o Load seja executado.

Portanto, um bom momento para carregá-la será o evento Refresh (se o fizermos no Start, será executado apenas uma vez. Em vez disso, o Refresh será executado toda vez que for atualizada e como esses dados poderiam variar de um momento para outro (obterá com que se criem novas reservas); talvez seja preferível o evento Refresh).

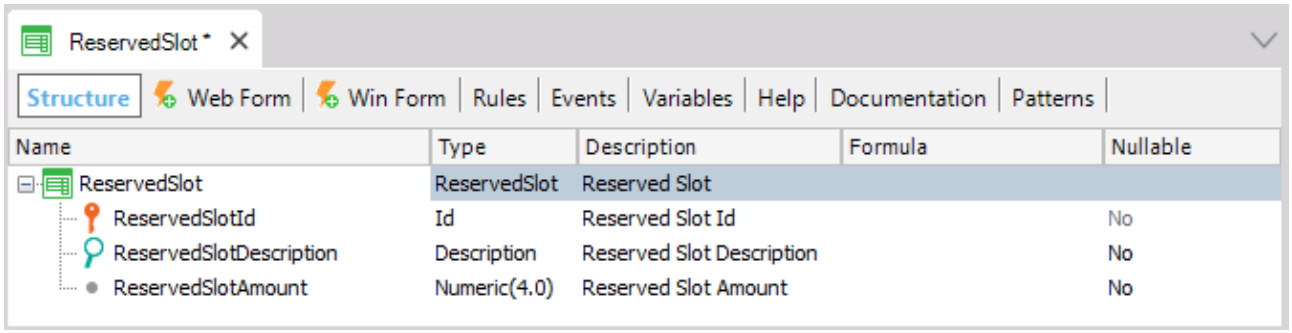
```
Event Grid1.Refresh()
	&ReservedSlots = RetrieveReservedSlot()
	&ReservedSlots.Sort("[ReservedSlotAmount]")
endevent
```

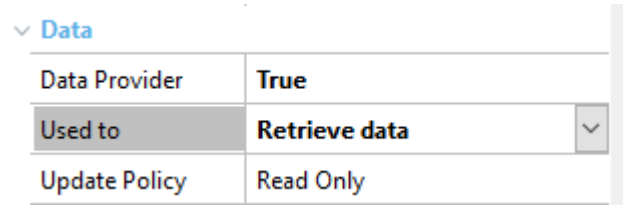
Observe que aqui invocamos o Data Provider para que nos retorne a coleção carregada e, em seguida, o que fazemos é classificá-la em ordem decrescente por ReservedSlotAmount.

Como após o evento refresh, será executada a carga do grid, com isto teremos o requisito resolvido.

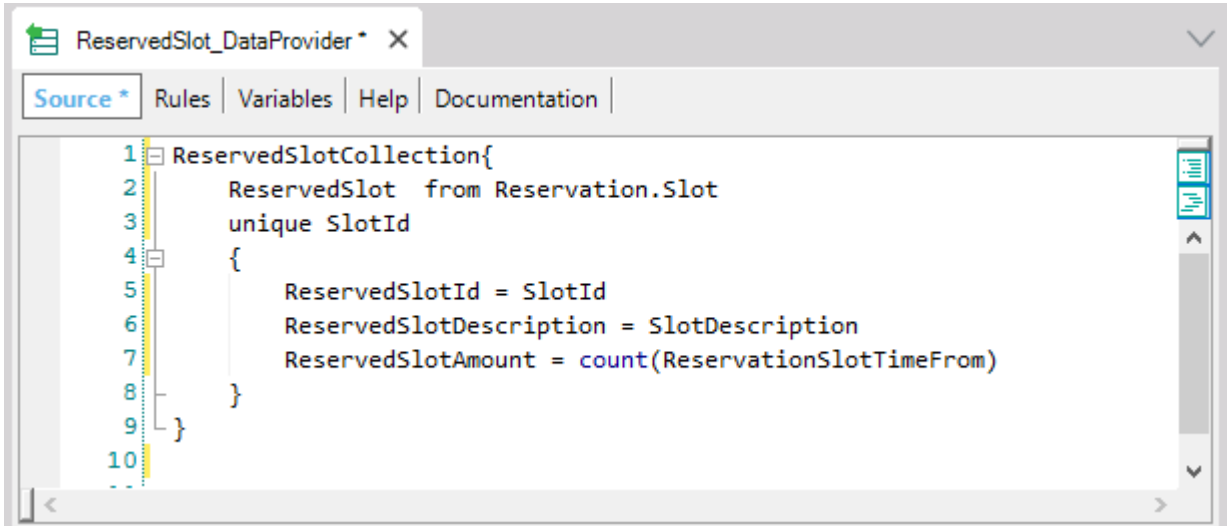
SOLUÇÃO COM TRANSAÇÃO DINÂMICA

Outros alunos, no entanto, pensarão em uma solução equivalente, muito parecida, mas que pode ser melhor: porque em vez de definir um SDT coleção e um Data Provider que o carregue, não definir uma transação dinâmica?

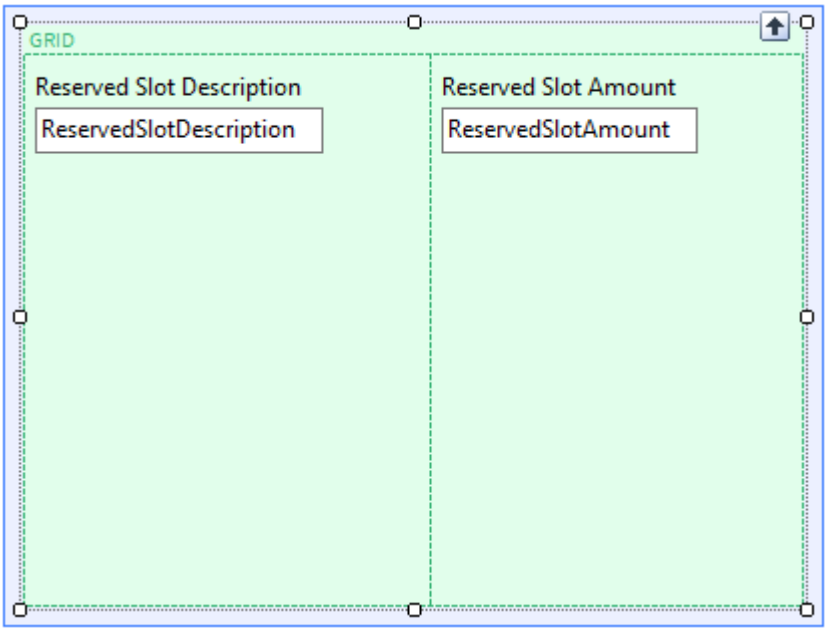




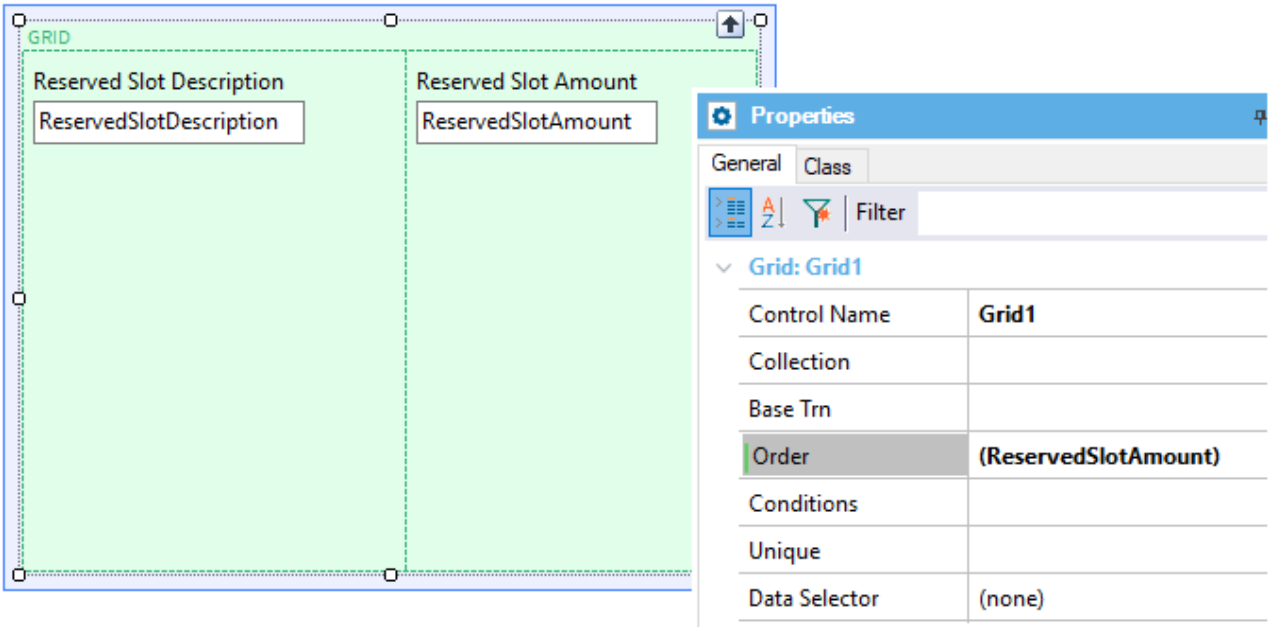
Deverá definir o Data Provider de maneira análoga à como fizeram os outros alunos com o SDT coleção:



E, no web panel carregar o grid como se fosse uma transação comum e atual e não uma dinâmica:

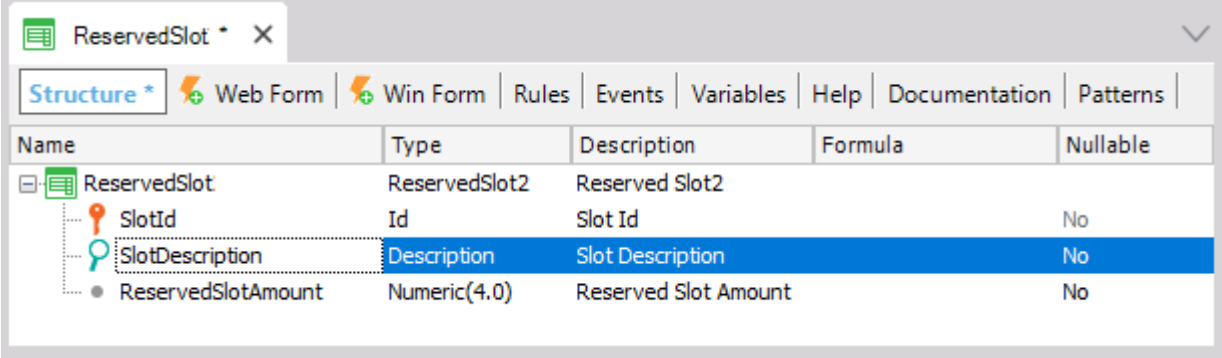


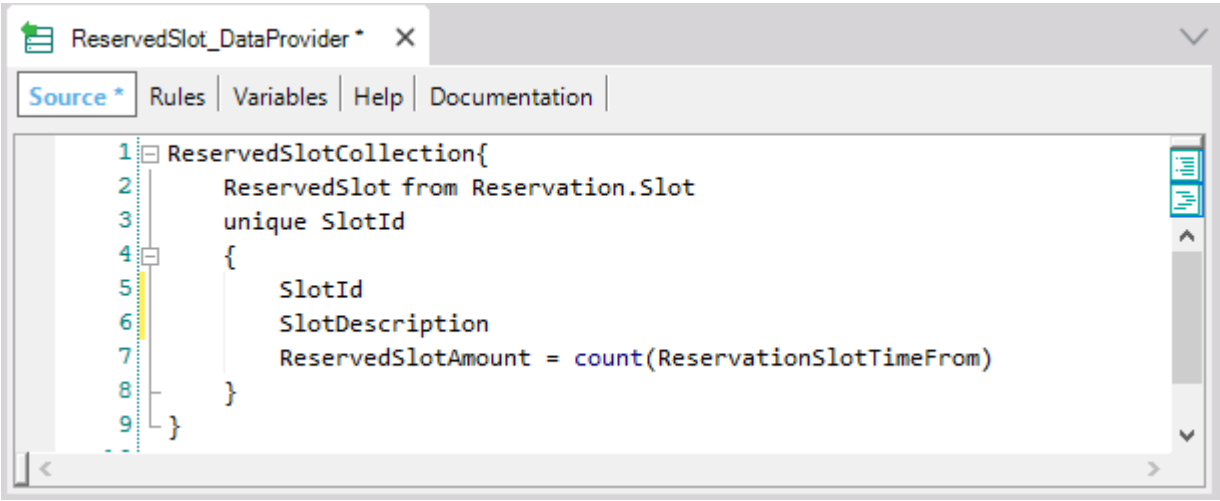
Com a ordem definida nas propriedades do grid:



Observemos que aqui não tem que programar absolutamente nada mais.

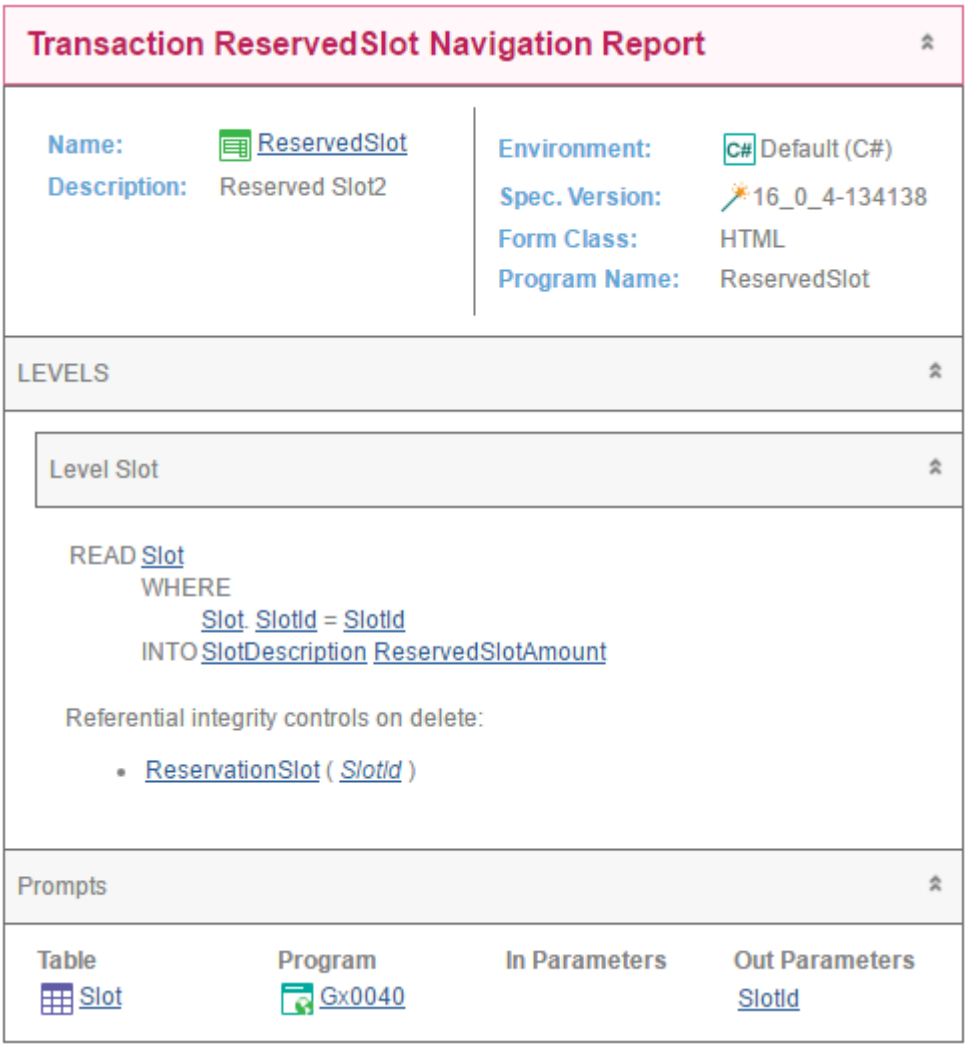
O aluno chegou a esta solução. Você o parabeniza. E faz a seguinte pergunta: por que na transação dinâmica, em vez de usar os atributos já existentes SlotId e SlotDescription, você definiu dois novos atributos? Ou seja, por que não definir a transação dessa outra maneira?:



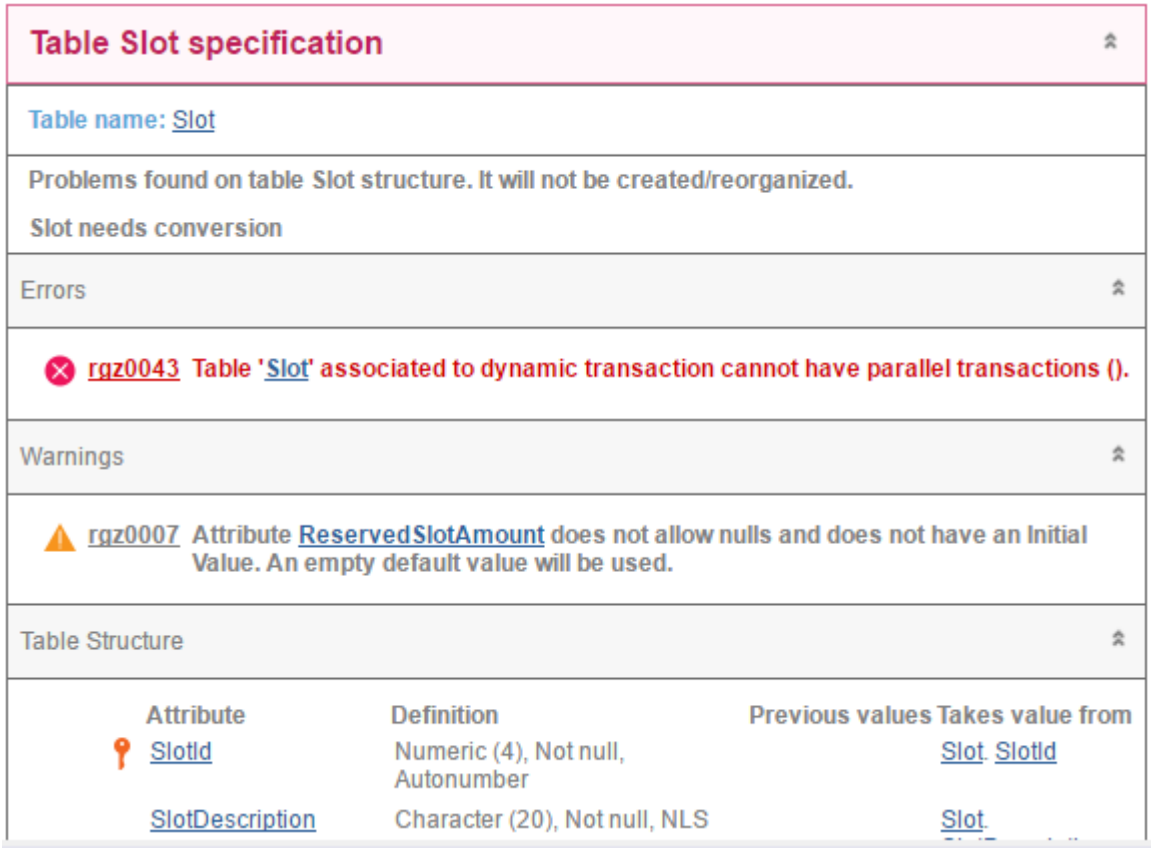


Para que, dessa maneira, o slot da transação dinâmica seja entendido como um slot efetivamente, em qualquer outro contexto.

Se faz o aluno olhar a lista de navegação, entenderá por que não é possível:



É que seria uma transação paralela à transação Slot, pois compartilhariam a mesma chave primária. Isso não é possível. GeneXus, ao tentar especificar, mostrará o seguinte erro:



O que aconteceria se SlotId não fosse a chave primária da transação dinâmica, mas apenas uma chave estrangeira? É possível usar o atributo nesse caso? Pense nisso e tente confirmar sua hipótese.

Mas a resposta é que sim, é possível. Pense em todas as arestas do problema. Por exemplo, o que acontece se você quer remover um slot da tabela Slot que tenha algum relacionado por essa transação dinâmica? É realizado esse controle de integridade referencial para evitar a eliminação?

PERGUNTA 22 – UNIQUE PARA AGREGAÇÃO

Suponha que agora precisa listar para cada cliente com reservas de slots posteriores a uma determinada data, seu nome e quantidade total de slots reservados a partir dessa data.

Um aluno propõe a seguinte solução:

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

Você menciona que há algo que está mal programado. Então outro aluno afirma que a solução não está correta porque a tabela navegada pela fórmula count não é Reservation.

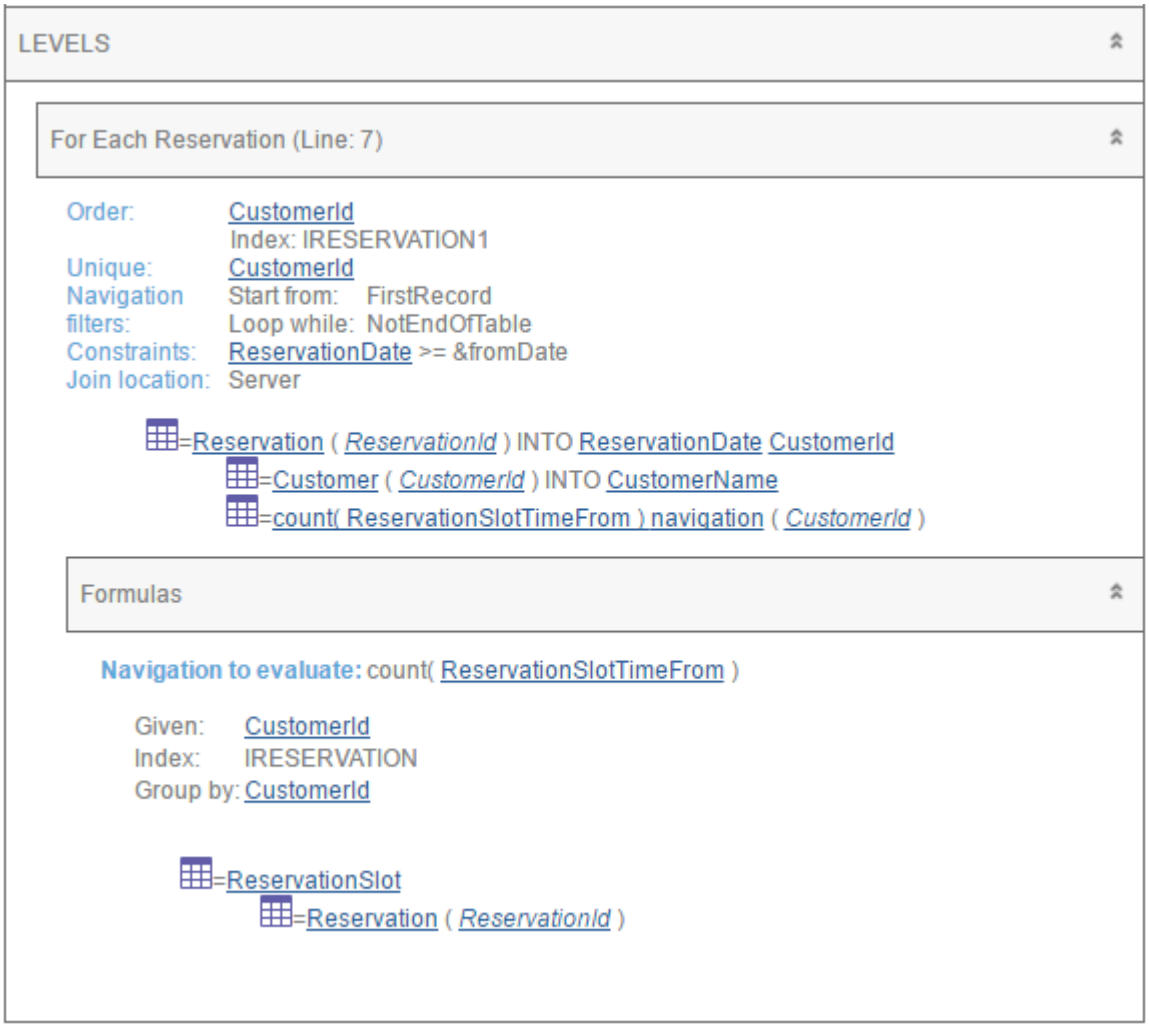
O que você vai responder?

RESPOSTA

É boa prática recomendar aos alunos que analisem a lista de navegação, que já nos fornece pistas para resolver os problemas.

O aluno que respondeu que o problema era que a fórmula count não navegava Reservation ficou claramente com a ideia de que somente nesse cenário é possível usar um for each com unique para fazer agregações. Este exemplo mostra que isso não assim.

Nesse caso, a fórmula count navegará ReservationSlot, mas filtrando pelo CustomerId (unique) do for each.



Ou seja, podemos ter o mesmo CustomerId em muitas reservas (posteriores à data &fromDate), mas será levada apenas uma dessas reservas no for each (pelo unique) e então, dentro de seu corpo, será executada a fórmula, que filtrará por esse CustomerId. Tudo isso está correto.

O erro é muito simples: estão sendo contados os registros da tabela ReservationSlot que correspondem a uma reserva do cliente, mesmo de datas anteriores a &fromDate.

O correto seria então:

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

endfor
```

Observe que a cláusula where filtra os registros sobre os quais será operado no corpo do for each, enquanto a condição da fórmula filtra os registros que serão contados. Ambas são necessárias.

www.genexus.com