

# **DISCUSSION OF QUESTIONS** for Instructor

Training Team - GeneXus

June 2023

## CONTENTS

CONTENTS .....	2
GENEXUS INSTRUCTOR EXAM PREPARATION.....	3
INTRODUCTION .....	3
Question 1 - DESIGN.....	3
Question 2 - DESIGN.....	7
Question 3 - DESIGN.....	9
Question 4 - DESIGN.....	11
Question 5 – DESIGN/NORMALIZATION .....	16
Question 6 – EXTENDED TABLE .....	18
Question 7 – HOW TO LEAVE A FOREIGN KEY WITHOUT A VALUE .....	19
Question 8 – USING INDEXES BY PRIMARY AND FOREIGN KEYS.....	20
Question 9 – DESIGN WITH SUBTYPE GROUPS .....	21
Question 10 – DESIGN WITH SUBTYPES (DOUBLE REFERENCE IN EXTENDED TABLE) .....	22
Question 11 – DESIGN WITH SUBTYPES – AVOID REFERENTIAL RELATIONSHIP .....	25
Question 12 – DESIGN- SPECIALIZATION .....	27
Question 13 – RULES AND EVENTS IN TRANSACTIONS .....	29
Question 14 – INLINE FORMULAS.....	32
Question 15 – UPDATE WITH FOR EACH COMMAND .....	33
Question 16 – EXAMPLES OF NESTED FOR EACH COMMANDS.....	37
Question 17 – NESTED GRIDS .....	44
Question 18 – NESTED GRIDS (CONTINUED) .....	48
Question 19 – UNIQUE OR CONTROL BREAK? .....	55

Question 20 – UNIQUE IN GRID (CONTINUED) .....	58
Question 21 – UNIQUE IN DATA PROVIDER AND DYNAMIC TRANSACTION (CONTINUED) .....	60
Question 22 – UNIQUE FOR AGGREGATION .....	66

## GENEXUS INSTRUCTOR EXAM PREPARATION

Candidates of the GeneXus Instructor exam have previously passed the GeneXus Senior Analyst exam, so they have already been technically evaluated.

## INTRODUCTION

Below are questions on some important topics, with answers that seek to expand their knowledge or their ability to integrate and articulate what they already know. In some cases, topics not addressed in the GeneXus course are explained. They won't be asked in the exam, but we understood that they helped to better understand GeneXus' logic and that's why they are explained here.

The ones you will find are just a few examples of some of the important topics (not all of them), so that you can have an idea of what kind of reasoning we will ask you in the exam.

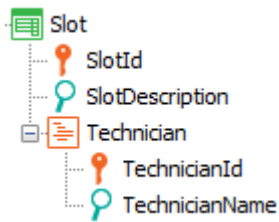
Note: Even though screen captures were made using GeneXus 16, they may vary in later versions.

## QUESTION 1 - DESIGN

A GeneXus application is used in a casino. It has transactions to register the slot machines as well as the technicians in charge of repairing them.

Knowing that a Slot machine can be repaired by several technicians (Technician), and that the same technician can repair several machines, select the right choice.

- a. A single transaction, Slot, with two levels:



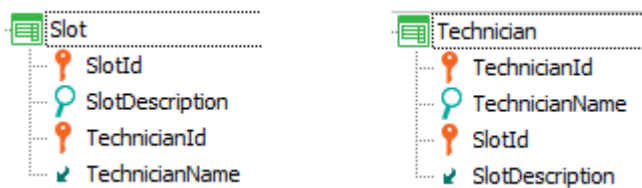
b. Two transactions: Slot and Technician:



c. Two transactions: Slot and Technician:



d. Two transactions: Slot and Technician:



e. None of the above.

## ANSWER

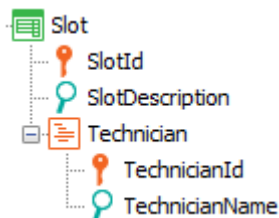
We were requested for an N to N relationship between the entities of reality Slot and Technician, and design **(b)** is the only one of those proposed that represents it.

Note that solutions (a) and (c) are equivalent and represent a different relationship from that requested between the entities. Which one? A 1 to N relationship, in which Technician wouldn't exist by itself, independently of the Slots; that is, Technician would be a weak entity, that to exist

depends on the existence of the Slot to which it is linked. This is evidenced by the fact that if we look at the table that is created to store its information, we see it has a primary key composed of SlotId and TechnicianId. So, how would one enter the system, for these alternatives, both (a) and (c), a technician without associating him in the same operation with a slot? Let's remember that it is impossible to leave a primary key without value (because in that case, how would we identify the table record?). SlotId is part of the primary key of the table that records technicians, so it would be impossible to enter a technician without entering a value for the SlotId attribute.

And if this happens, that is, if it is not possible to enter a technician without associating him/her with a slot, how would we make that technician be associated, in addition, with other slots?

For example, for solution (a):



Editing at runtime the information of Slot with Id 3 and description “Super Wizard” we enter in its grid the technician with Id 1, David Roberts, and confirm.

How do we associate with the same technician, David Roberts, another slot, for example 6, if the technician exists only as linked to slot 3?

One might be inclined to say that it is very easy: simply editing now the Slot 6 information, and adding to its grid a line with TechnicianId = 1 and TechnicianName = David Roberts.

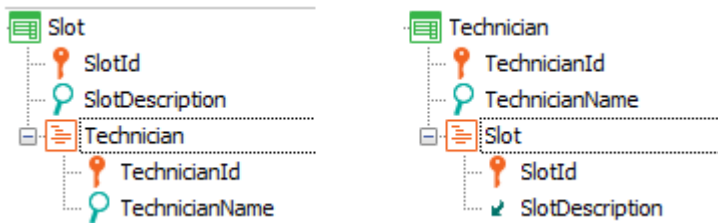
Note that there is no guarantee that there isn't another TechnicianId = 1 in that grid with another technician name, or even, without previously existing, that we can assign another name to the one we entered for that Id 1. Because TechnicianName is a physical attribute of that table, not an inferred one, as in the case of solution (b).

We could, however, use this design (which is 1 to N) ensuring "manually" that technician number 1 is not used other than for David Roberts. But what is demonstrated here is that we are not designing our reality correctly, because if we do it well, the system itself will not allow us to make mistakes, as is the case of the correct solution, (b), where the technician is identified only by his Id and then the Slots are assigned technicians (by their Id) who can repair the machine.

On the other hand, solution (d) cannot be correct, because both transactions determine the same physical table (the order in which the attributes appear does not alter the constitution of the table).

Here slots and technicians only exist if they are associated. Neither exists independently of the other.

What would happen if these other transactions had been created instead?

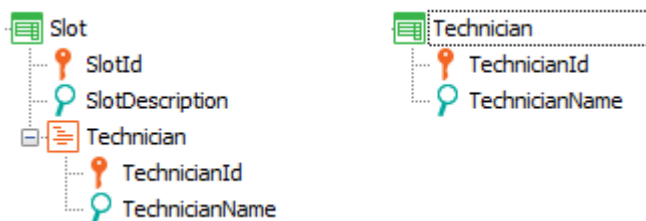


What tables will be created by GeneXus based on them? Although at first glance it may seem to be an incorrect solution, and in part it is not advisable, the tables to be created will be the correct ones, identical to that of solution (b). Note that the table corresponding to the second level of Slot will be exactly the same as that of the second level of Technician, since the primary key of both is the same, made up by both attributes, SlotId and TechnicianId. The problem of this design will only be operational: to be able to enter the technicians for a slot, first you will have to create those technicians using the Technician transaction, leaving the slots blank.

Of course, a solution such as:

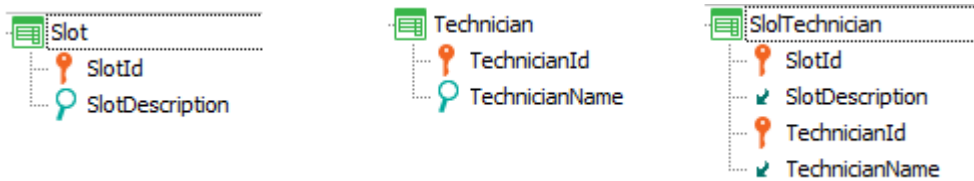


As far as the N to N representation is concerned it is completely equivalent to (b):



The only difference is given by the operation: What's more convenient? Enter all technicians without first associating them with the slots and then for each slot entered assign their technicians, or vice versa?

The tables generated are exactly the same in both solutions. Or you could even have this other solution, completely equivalent in terms of the N to N relationship:

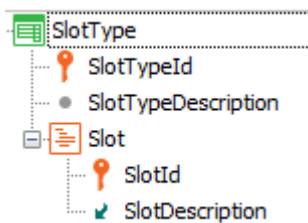


## QUESTION 2 - DESIGN

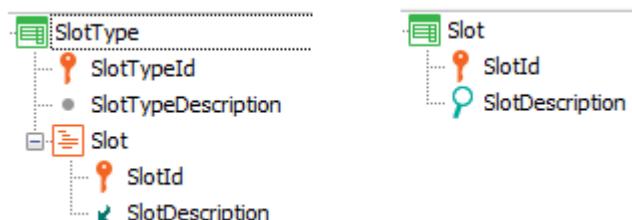
A GeneXus application is used in a casino. It has transactions to register the slot machines as well as the technicians in charge of repairing them.

Knowing that each machine (Slot) corresponds to a certain type (SlotType) and only one, and that there can be many slots of the same type, determine the correct option for the design of these transactions.

- a. A single transaction:



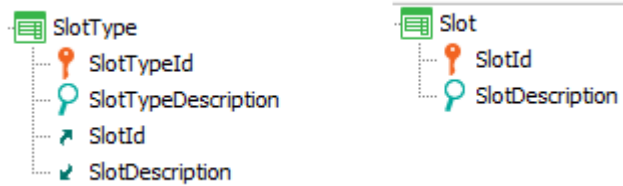
- b. Two transactions:



- c. Two transactions:



d. Two transactions:



e. None of the above.

## Answer

We are being asked to implement a strong 1-N relationship (this data is inferred from our knowledge of reality, as it has not been explained in the exercise instructions), so the correct answer is (c).

Here, in solution (c), each slot will have a unique “type” because in the Slot transaction the attribute SlotTypeId is on the same level as the identifier, SlotId. In this way, a given slot, identified by SlotId, will have a single description, SlotDescription, and will also have a single SlotTypeId (it would be different if this attribute also had the key symbol). On the other hand, nothing prevents another slot from having the same value for SlotTypeId (because the attribute is neither primary nor candidate key in this transaction). So, the requirement made in the instructions is met: a slot only has one type and several slots can have the same type.

Why is option (a) not correct, given that it also represents a 1-N relationship? Because in this representation the Slot entity would be weak, that is, it cannot identify itself only with its Id, but needs the type Id. We would be saying that slots only exist to the extent that there is a “slot type” on which they depend. It will not be possible to enter a Slot without having previously specified its SlotTypeId. In other words, under this representation, we would infer that the existence of the real world entity Slot, depends entirely on the “Slot type,” which contradicts the reality that we want to



model. In it, the “Slot Type” and “Slot” are related entities but they exist on their own. Each is identified regardless of the other. Also, if null values were admitted for SlotTypeId in Slot (in the correct solution, (c), “slots” could be entered without entering their “type.”

Option (b) clearly represents an N-N relationship (in this case for each “slot type,” many associated “slots” can be entered, and at the same time, each “slot,” represented by the Slot transaction, and identified with the attribute SlotId, can be repeated for many “slot types” (slot 1 can show it in its grid, as well as slot 2, etc.). This clearly does not correspond to the proposed reality.

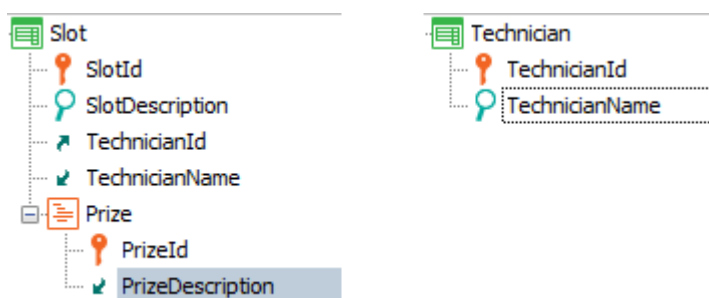
Option (d) represents a 1-N relationship but it is the opposite of what is being asked for. Note that here a “slot type” has a single “slot” associated with it and another “slot type” may have the same “slot” associated as the previous one. For this reason, here we are saying that each “slot” may have many “types,” and that a “type” only corresponds to one “slot.”

Summary of strong vs weak entity: an entity is strong, when it can be identified independently of the others, even though it is related to them. On the other hand, it will be weak when it does not exist independently of another entity. To exist, it depends on the existence of the other. The typical case is that of customer telephones. It doesn't make sense to have a phone number that is independent of the customer it belongs to.

### QUESTION 3 - DESIGN

A GeneXus application is used in a casino.

Given the following transactions, determine the relationship between the Slot and SlotPrize reality actors.



- 1 to 1 relationship (for each slot there is only one prize, and for each prize a slot that provides it).
- 1 to N relationship (for each slot there are several prizes, but where each prize corresponds only to that slot and not to another) where both entities are strong.

- c. 1 to N relationship (for each slot there are several prizes, but where each prize corresponds only to that slot and not to another) where the Slot entity is strong but SlotPrize is weak.
- d. N to N relationship (for each slot there are many prizes and each prize can be given for many slots).

---

#### Answer

The correct answer is (c). What does it mean to say that SlotPrize is a weak entity compared to Slot? That it won't exist as an independent entity. Its relationship with Slot is one of absolute dependence. A given prize exists only to the extent that the Slot to which it is associated is indicated. It's "such" award from "such" slot. You can never just say "it's such prize." It must always be indicated which slot it is. If, on the other hand, the relationship were 1-N strong, the prize would exist as an independent entity, identifiable by itself, without having to indicate the slot in order to know which prize we are talking about.

## QUESTION 4 - DESIGN

A GeneXus application is used in a casino. It has transactions to record clients as well as the VIP cards that are issued to them.

Knowing that each customer (Customer) can have a single VIP card (VIPCard) and that each VIP card can only belong to one customer, select the correct option for the design of these transactions.

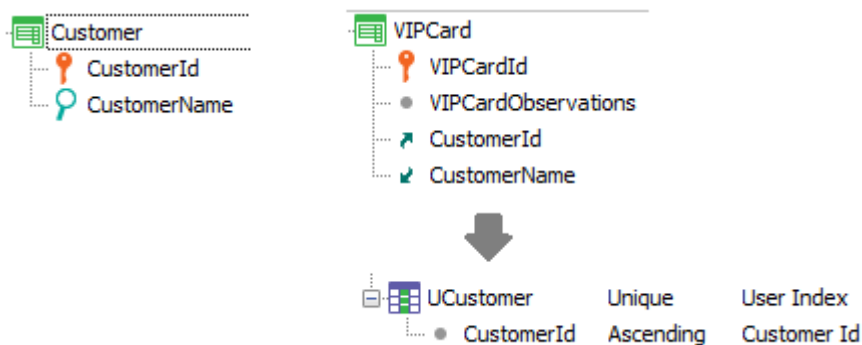
a. Two transactions:



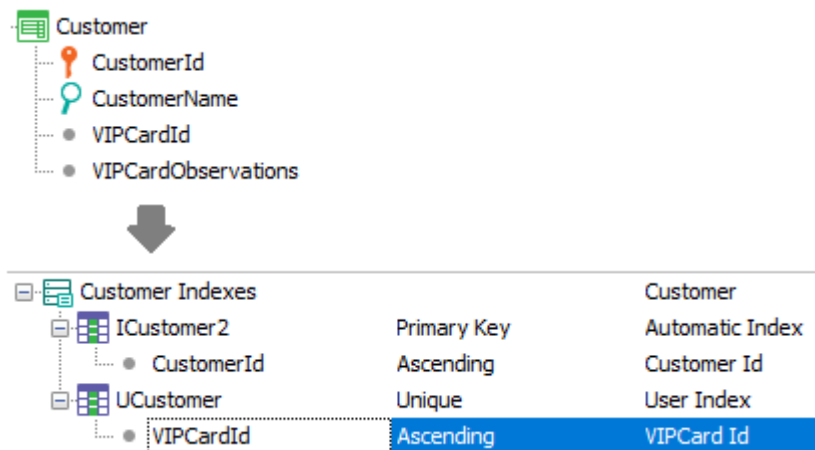
b. Two transactions:



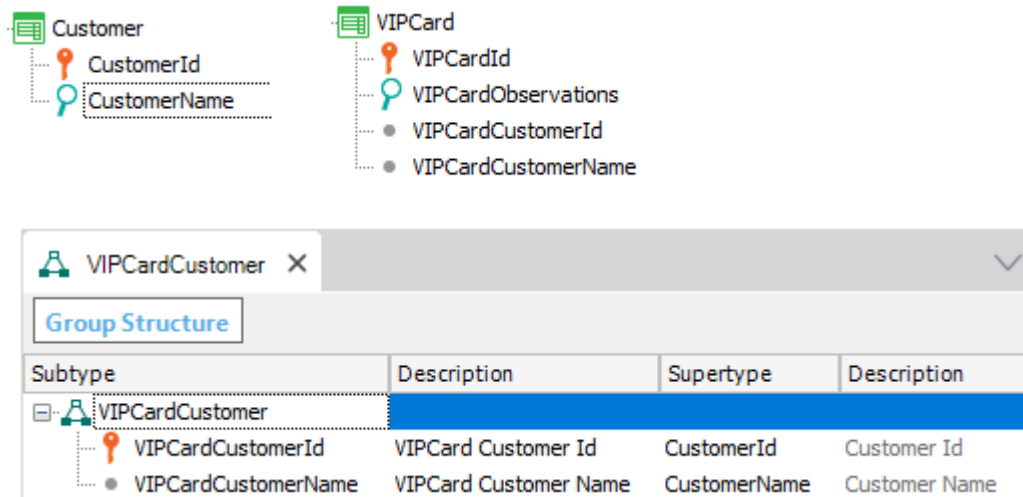
c. Two transactions and a unique index:



d. One transaction and one unique index:



e. Two transactions and a group of subtypes:



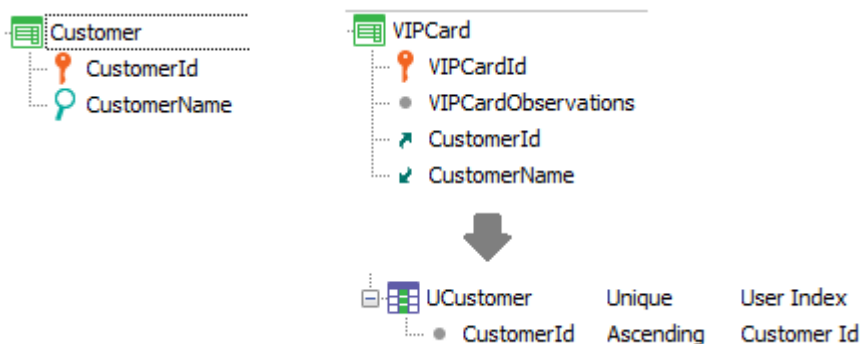
f. None of the above.

## Answer

We are asked for a 1 to 1 relationship between both entities. There is an implicit assumption that there are two separate entities, each with its own identifier. Each card will belong to only one customer, but it is strong enough to have an existence of its own. In other words, the system will work with the cards beyond the customer. There will be operations where the card must be entered and not the customer (for example, to pay). Therefore, the valid option will be (c) not (d).

Let's discuss the differences between both solutions and then we'll see why the others are far from right.

Let's look first at the right solution, (c):

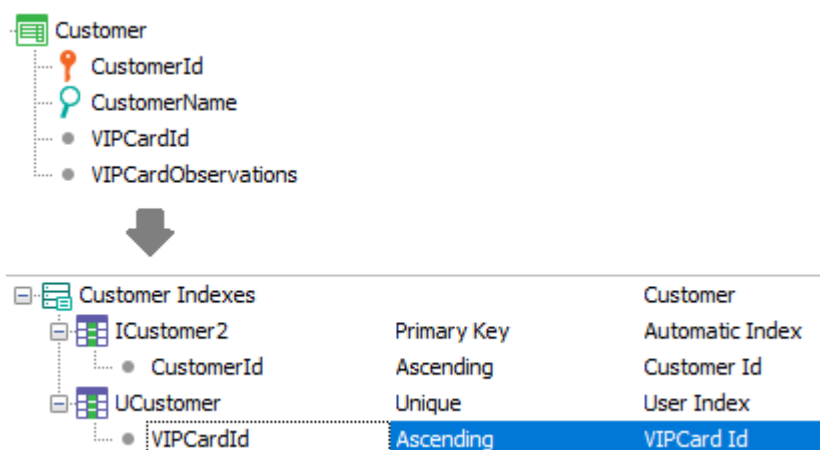


Here we have a customer identifier, **CustomerId**, and we also have a **VIPCardId** card identifier, which makes it possible for us to use them in any other transaction as foreign keys. For example, in an invoice we can place the attribute **VIPCardId**, and there the customer will be inferred, with no need to also specify it. As an observation, we cannot do the opposite, i.e. place the **CustomerId** attribute in the invoice structure and wait for the card to be inferred from there. We can look for it with a formula. Why is it so?

Because actually GeneXus is allowing us to model a 1 to N relationship that we make 1 to 1 through the definition of the unique index on what would be the foreign key. For this reason, at the **VIPCard** table level we define a Unique user index on the **CustomerId** attribute, which already had an automatic Foreign Key index defined.

In this way, each time a new card is entered into the system, if an attempt is made to assign an existing number for another card to the **CustomerId** attribute, the operation will fail because it will be found that the unique index value already exists and it will be informed.

In solution (d) we have defined a single transaction, with the customer ID as primary key, and with the card ID as candidate key. This also means that a card ID cannot be used for several customers (and, of course, that a customer has a single card ID), but we're not able to use the card in other entities. It can't be a foreign key anywhere. For example, we can't enter it in an invoice. We will necessarily have to enter the customer (and infer the card).



More specifically, in solution (c) there could eventually be customers without cards (what cannot happen is that a card doesn't have a specified customer). A good question is how to avoid this, that is, so that each client must have a card. This topic is linked to that of UTL. One possibility would be to write the following invocation rule to procedure in the Customer transaction:

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

Thus, once the customer has been added to the CUSTOMER table (and BEFORE COMMIT), the procedure CreateVIPCard is invoked, which will create the card for that customer. Assuming that VIPCardId is autonumbered, it could be programmed in this way:

Rules:

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

Source:

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

Note that even though the procedure will not commit (since BC operations do not make the procedure interpret that it will access the database and therefore does not place the implicit commit at the end of the source), given that it has been invoked from the Customer transaction before its Commit, when returning control to the transaction, it will commit and therefore the two records inserted in the database will be committed: the customer in the CUSTOMER table inserted by the transaction before calling the procedure and the card in the VIPCARD table inserted by the procedure itself when using the Insert method.

Remember that in web applications the UTL made up of records handled by one transaction iteration instance cannot be extended to also include records handled by another transaction. But it can happen between the transaction and batch objects such as the proc.

Let's briefly discuss why the other options in the question are incorrect.

Option (a) clearly represents a 1 to N relationship. It's not what has been requested.

Option (b) should be carefully considered:



At first glance, in the Customer transaction we say that a customer has only one VIPCard, and in the VIPCard transaction we say that a card has only one customer. So it would appear that the requirement is being met. However, transactions operate together and not in isolation. If we look at the group, we could say that VIPCardId would be a foreign key in the CUSTOMER table, and that symmetrically, CustomerId would be a foreign key in the VIPCARD table. And we could argue that the error of this design is that while it is true that a customer has only one card and that a card has only one customer, it is not true that customers and cards match in both tables.

That is, customer 1 can have VIPCardId 5, but when we go to VIPCardId 5, it turns out that this could have CustomerId 3, for example.

However, we did not realize that even these records cannot be entered due to referential integrity controls. How do I enter the client 1 with card 5 if I didn't enter it before? But to enter it, how do I go about it if the client doesn't exist yet? Well, it could be solved by allowing VIPCardId in CUSTOMER, as well as CustomerId in VIPCard to admit nulls. Thus, the first time I enter the customer without specifying the card, and then I enter the card, specifying the customer and return to the client in update mode to now specify the card.

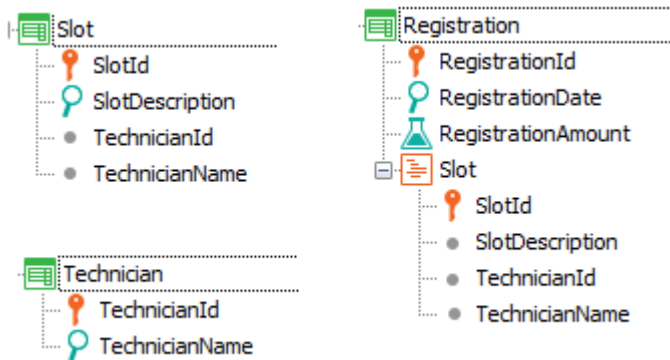
However, all of the above analysis started from a **false premise**. Are we sure that GeneXus will create two tables?

Remember that, first of all, GeneXus normalizes the tables to avoid inconsistencies. Note that by having cross-references we are saying on the one hand that CustomerId determines VIPCardId, and on the other hand that VIPCardId in turn determines CustomerId. So these two transactions will result in a single table that will have the composition of either of the two transactions (it may be the first or the last). And that, if we think about it, is completely logical. In this case, if the last one is used, the table will match the VIPCard structure, where it will also create a primary index by CustomerId (i.e. it will not allow entering the same customer for two different cards). It would become a case similar to that of solution (d), with everything we discussed there. Then this solution is closer to the required one, except for the fact that we need the customer and the card to exist separately, as entities.

The last solution, (e), is a variation of (a). It still represents a 1 to N relationship (where the definition of the subtype group seems completely unnecessary).

## QUESTION 5 – DESIGN/NORMALIZATION

An application is used for a casino. Given the following transaction design, determine the physical structure of the TABLES that will be designed and created by GeneXus.



a)

<b>TECHNICIAN</b> TechnicianId* TechnicianName	<b>SLOT</b> SlotId* SlotDescription TechnicianId	<b>REGISTRATION</b> RegistrationId* RegistrationDate	<b>REGISTRATIONSLOT</b> RegistrationId* SlotId* TechnicianId
------------------------------------------------------	-----------------------------------------------------------	------------------------------------------------------------	-----------------------------------------------------------------------

b)

<b>TECHNICIAN</b> TechnicianId* TechnicianName	<b>SLOT</b> SlotId* SlotDescription TechnicianId	<b>REGISTRATION</b> RegistrationId* RegistrationDate RegistrationAmount	<b>REGISTRATIONSLOT</b> RegistrationId* SlotId* TechnicianId
------------------------------------------------------	-----------------------------------------------------------	----------------------------------------------------------------------------------	-----------------------------------------------------------------------

c)

<b>TECHNICIAN</b> TechnicianId* TechnicianName	<b>SLOT</b> SlotId* SlotDescription TechnicianId	<b>REGISTRATION</b> RegistrationId* RegistrationDate	<b>REGISTRATIONSLOT</b> RegistrationId* SlotId*
------------------------------------------------------	-----------------------------------------------------------	------------------------------------------------------------	-------------------------------------------------------

d)

<b>TECHNICIAN</b> TechnicianId* TechnicianName	<b>SLOT</b> SlotId* SlotDescription TechnicianId	<b>REGISTRATION</b> RegistrationId* RegistrationDate RegistrationAmount	<b>REGISTRATIONSLOT</b> RegistrationId* SlotId*
------------------------------------------------------	-----------------------------------------------------------	----------------------------------------------------------------------------------	-------------------------------------------------------



---

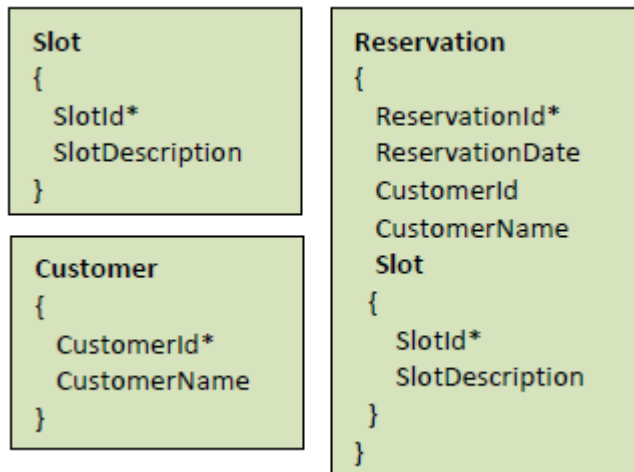
## Answer

The correct answer is (c).

- In the Slot transaction, the attribute TechnicianName is inferred from the foreign key TechnicianId.
- At the Slot level of the Registration transaction, the TechnicianId attribute is inferred through the foreign key SlotId (do not assume that because an attribute is a primary key in a table it automatically becomes a stored attribute and foreign key in any other table associated with a transaction in which it is present!). Knowing this, how would you make it possible for the second level of Registration to register a technician independent of the slot? You will have to use subtypes. Think about the solution and then see answer (d) to question 10.
- RegistrationAmount is a formula, so it will not be present in any table (unless explicitly defined as redundant).

## QUESTION 6 – EXTENDED TABLE

A GeneXus application is used in a casino. Having the following transactions, determine the extended table of the RESERVATIONSLOT table.



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

---

### Answer

The correct answer is (d).

The concept of extended tables is very important in GeneXus, because all its logic is based on abstracting the vision of the physical tables, and moving to a more conceptual vision (of univocally related information, which is scattered in different physical tables only for the purposes of normalization, but which nevertheless logically forms a unit). Hence, the For Each command allows you to work almost equally with all attributes of the extended table (and not only of the base table), as well as grids, Data Provider groups, transaction rules, etc.

Why is a database normalized? To avoid the duplication of data, which would bring the usual risk of having inconsistent data.

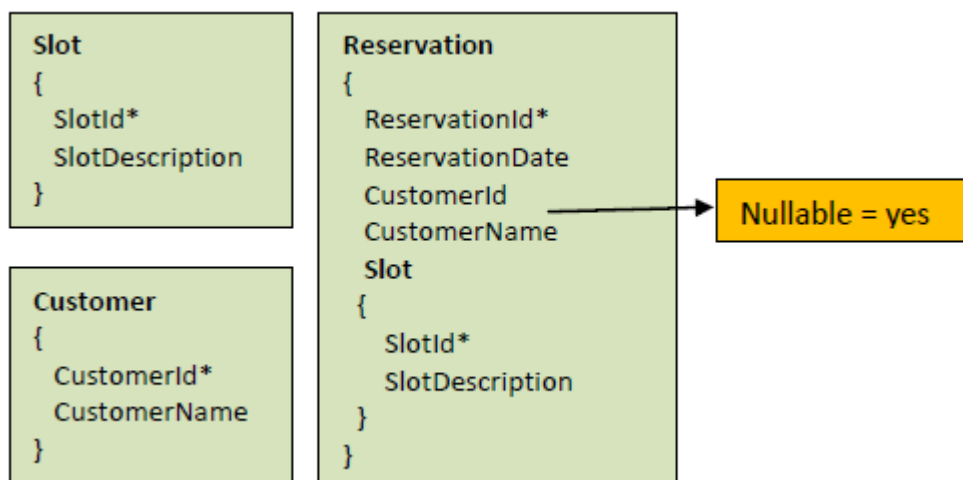
If we didn't need to avoid duplication, we would have a single table for reservation slots, with primary key {ReservationId, SlotId}. What attributes would that table have? It would have, in addition to the primary key, the description of the slot (SlotDescription), the reservation date (ReservationDate), the reservation customer (CustomerId) and his/her name (CustomerName). All

that information would be put together in a table, which we call “extended table.” Physically it does not exist, but it does exist as a conceptual tool.

#### QUESTION 7 – HOW TO LEAVE A FOREIGN KEY WITHOUT A VALUE

A GeneXus application is used in a Casino. It has a set of transactions to record the slots (Slot), and slot reservations (Reservation) by customers (Customer) as shown below.

Sometimes reservations are made without the need to specify the customer (CustomerId). From the proposed design, select the statement you consider correct:



- a. In spite of declaring that the attribute CustomerId (foreign key) in the RESERVATION table admits nulls (that is, it admits an unspecified value at the database level), GeneXus will always trigger the corresponding referential integrity checks against the CUSTOMER table, and will not admit entering an invalid value for that foreign key CustomerId.
- b. At the time of declaring that the CustomerId attribute (foreign key) in the RESERVATION table admits nulls (that is, it admits an unspecified value at the database level), then if a value is not entered in that foreign key, GeneXus does not trigger the referential integrity checks against the CUSTOMER table, but if a value is entered in that foreign key, GeneXus will trigger referential integrity checks against the table CUSTOMER.
- c. None of the above.

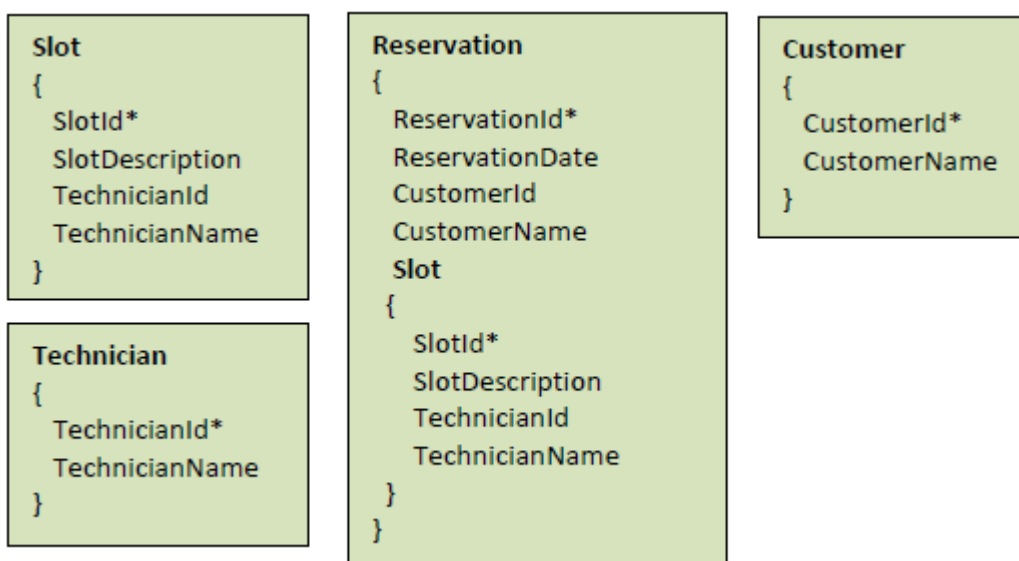
---

## Answer

The correct answer is (b).

### QUESTION 8 – USING INDEXES BY PRIMARY AND FOREIGN KEYS

Given the following transaction design, determine which index is used to efficiently validate, when attempting to eliminate a technician through the transaction, that there are no slots assigned to him.



- a. Index by TechnicianId in the table TECHNICIAN
- b. Index by TechnicianId in the table SLOT
- c. Index by SlotId in the table SLOT
- d. None of the above options are correct

---

## Answer

The correct answer is **(b)**. This question is not as important to the student as it is to the instructor (it is difficult to find in real exam questions). When trying to delete a record from the TECHNICIAN table through the transaction, as we know, logic is included in it to access all the tables that have TechnicianId (primary key) as foreign key, that is to say, all the subordinate tables that reference it. Each one of these tables must be accessed to check that there is no record that refers to the one to be deleted. In this example there is only one subordinate: the SLOT table. How to access this table to know if there is a record with the value for the foreign key attribute TechnicianId equal to that of the record you want to delete from TECHNICIAN? The indexes are the efficient mechanisms for

carrying out these searches, which is why GeneXus will use the index by foreign key defined in the SLOT table, on the foreign key attribute TechnicianId. In fact, it is to make more efficient referential integrity checks **when deleting the indexes by foreign key** automatically created by GeneXus in the database.

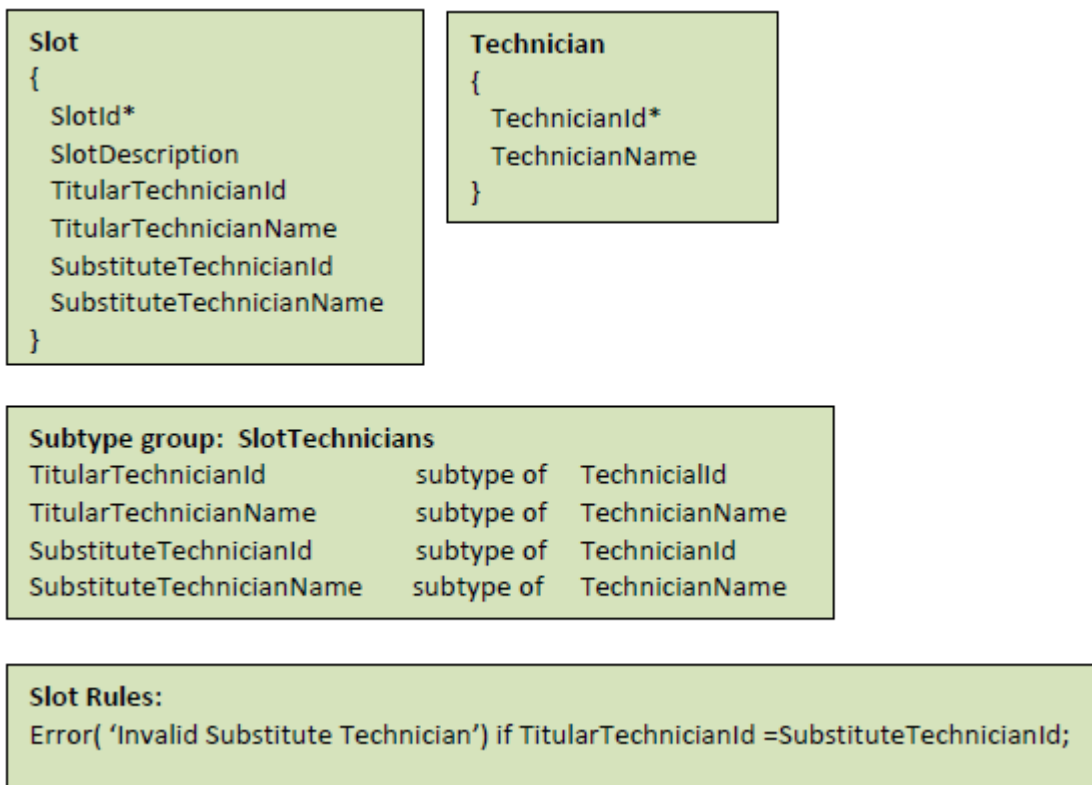
It is common to get confused and think that the correct answer would have been (a), but it is necessary to understand that the index by TechnicianId in the TECHNICIAN table accesses the TECHNICIAN table, where there is nothing to look for. The search must be carried out on SLOT. This is a complex issue, because in reality the strategy to access the tables used depends a little on the DBMS. But the conceptual logic of GeneXus would be correct.

## QUESTION 9 – DESIGN WITH SUBTYPE GROUPS

A GeneXus application is used in a casino. It has a set of transactions to record the slots (Slot) and the technicians (Technician) in charge of repairing them.

Each time a new slot is entered, a primary technician and a substitute technician must be associated with it. The system should check that it's not the same.

Determine whether it is true or false that the following alternative correctly meets the above requirement.



---

## Answer

False.

If subtypes are defined for the primary technician and for the substitute technician, they must be defined in two different groups to indicate the set of information that is handled jointly. In the solution presented, a single group of subtypes was defined, where both the subtypes corresponding to the primary technician and the substitute technician were entered, mixed. This is clearly incorrect. Why?

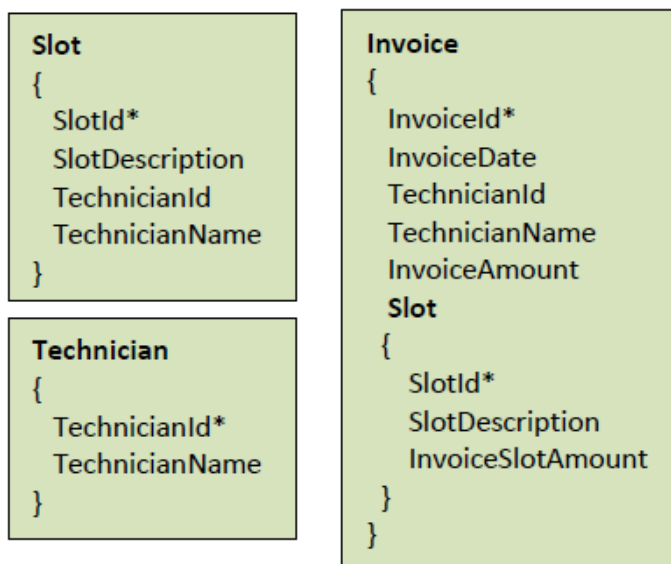
### QUESTION 10 – DESIGN WITH SUBTYPES (DOUBLE REFERENCE IN EXTENDED TABLE)

A GeneXus application is used in a casino. It has a set of transactions to record the slots (Slot) and the technicians in charge of repairs (Technician).

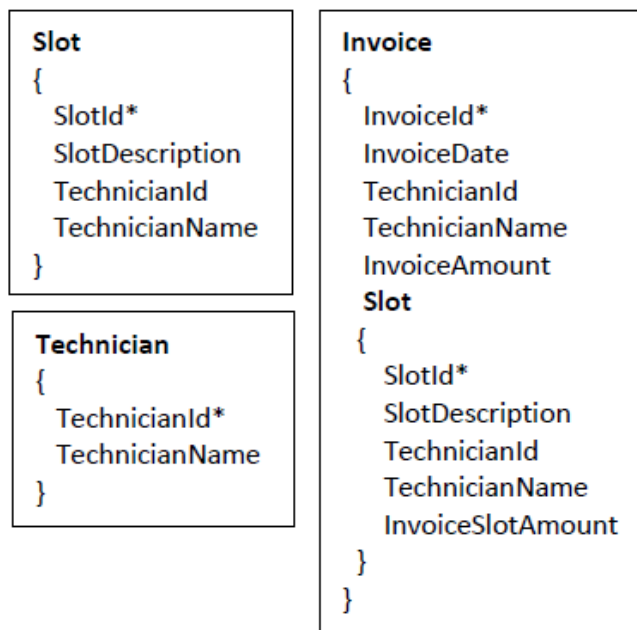
A slot can be repaired by a single technician, and each technician is assigned several slots to repair if necessary. Thus, when invoicing the services of a technician, it must be verified that the detailed slots are actually assigned to the technician of the invoice.

Determine which of the following options implements this requirement.

a.

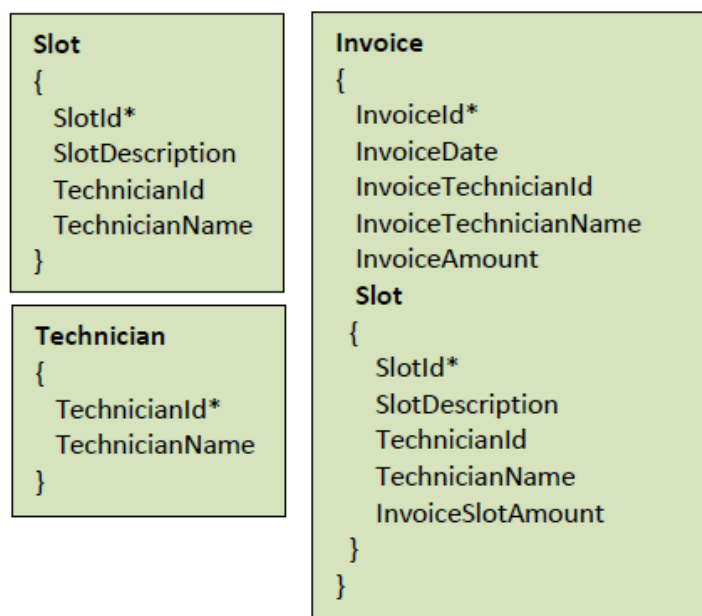


b.



**Invoice Rules:**  
Error( 'Invalid Slot') if TechnicianId <> TechnicianId;

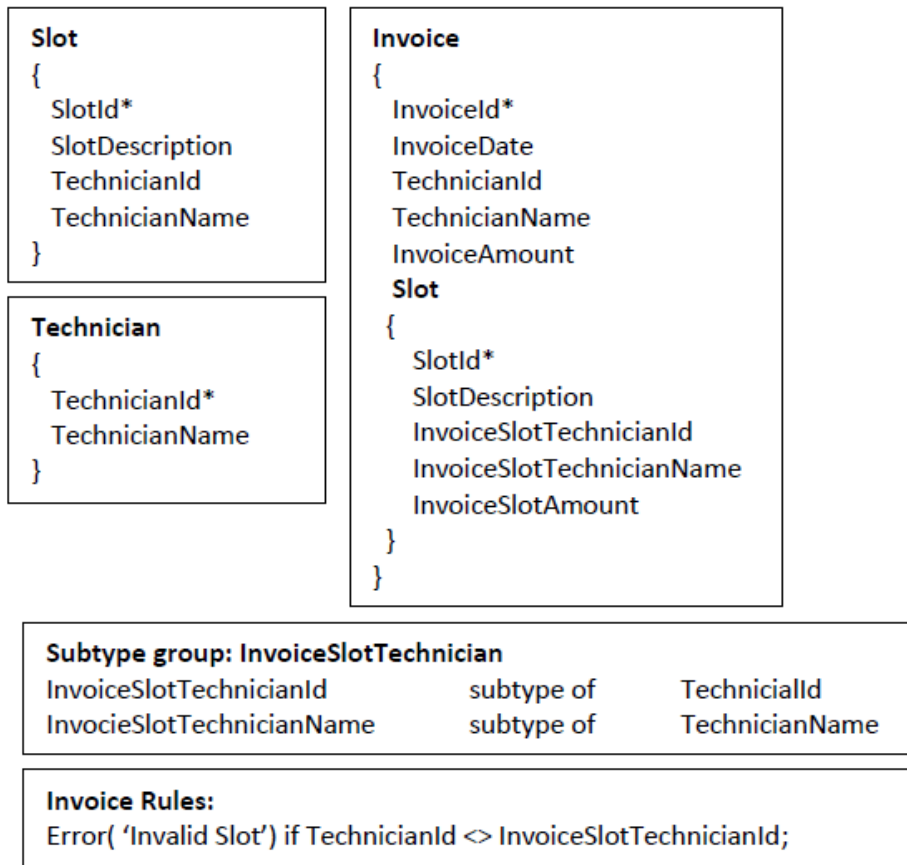
C.



Subtype group: InvoiceTechnician		
InvoiceTechnicianId	subtype of	TechnicianId
InvocieTechnicianName	subtype of	TechnicianName

**Invoice Rules:**  
Error( 'Invalid Slot') if TechnicianId <> InvoiceTechnicianId;

d.



## Answer

While there are other ways to implement it, the only option presented that does so is (c). Note that in the case of (d), as the subtype InvoiceSlotTechnicianId is not included in a group with a SlotId subtype, will be an independent technician from the one inferred from the Slot.

Although solution (c) is correct, in general it is not the one we usually advise. Why?

The need to change the name of the technician doesn't arise in the Invoice header because there is no ambiguity there. This ambiguity is not found in Slot, either.

This ambiguity appears on the second level of Invoice. In that level we have two technicians: the one of the invoice and the one that comes inferred of the slot.

So, the solution we recommend is to solve it here. But to solve it here we have to change the name of the TechnicianId and also the name of SlotId, and group all that. In this way, we are changing the name to both the primary key and the foreign key.

## Subtype group – InvoiceSlot

InvoiceSlotId	subtype of	SlotId
InvoiceSlotDescription	subtype of	SlotDescription



InvoiceSlotTechnicianId	subtype of	TechnicianId
InvoiceSlotTechnicianName	subtype of	TechnicianName

Where the Invoice transaction will be as follows:

```

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

```

The other transactions will remain unchanged.

With this solution, InvoiceSlotTechnicianId will be inferred through InvoiceSlotId.

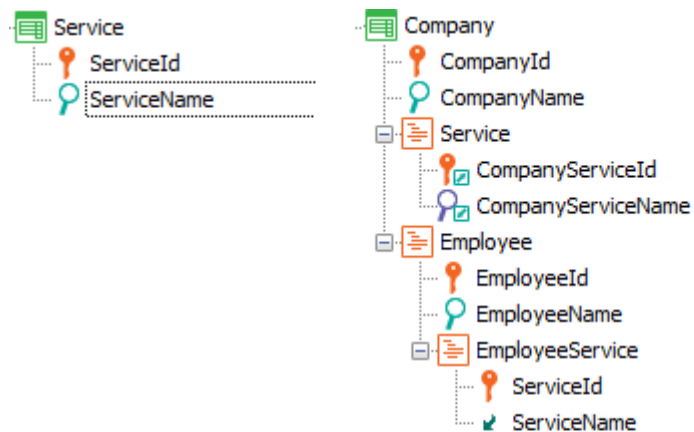
## QUESTION 11 – DESIGN WITH SUBTYPES – AVOID REFERENTIAL RELATIONSHIP

Transactions need to be modeled for a reality in which there are companies and services they can purchase (such as an emergency medical service, for example). At the same time, companies have employees who may also have contracted services that do not necessarily match those of the company for which they work. The services purchased by employees should be recorded so that, if, for example, many employees have purchased emergency medical service X, an agreement with that service can be sought to obtain a discount.

Employees can only work in one company, but they are not to be represented as a strong entity, but as dependent on the company.

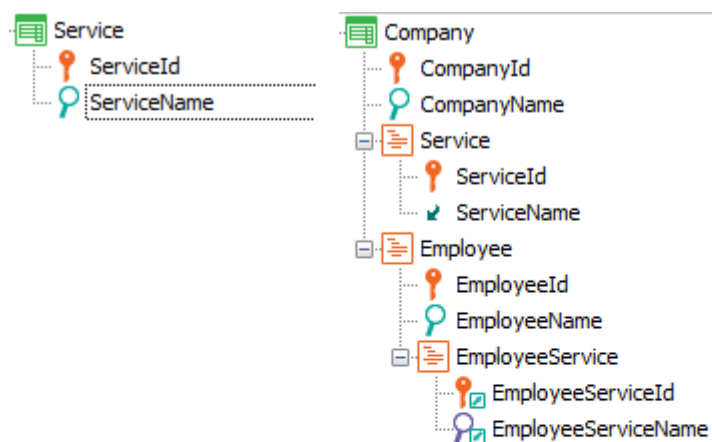
Explain which of the two solutions is the right one and why the other is not:

- a. Two transactions and the following group of subtypes:



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

b. Two transactions and a group of subtypes:



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

Answer

The correct answer is (a), not (b).

If we look closely, we have two parallel levels: Service and Employee. This means that everything that is inferred from any of them will correspond to the same company. However, we do not want the employee's service to exist as a company service, since in our reality the employee may have contracted services different from those of the company he or she works for. In other words: we don't want it to be checked when the user enters in the employee's services grid that the entered service exists as a record in the table corresponding to Company.Service.

Clearly, we need to define a group of subtypes because in the same transaction GeneXus will not allow us to repeat the same attribute name.

The question that arises is: does it make any difference whether you define it on one level or the other? The answer is No.

We could define two groups of subtypes and the problem is solved. But it is not good practice to define more subtypes than strictly necessary, because it is never exactly the same to have the subtype as to have the supertype, as this example will make clear.

Therefore, in order to solve the problem, we need only one group. Why is the correct answer (a) and not (b)?

The reason is that, if GeneXus would allow us to repeat the same attribute name, you would clearly find that in the table associated with the level Company.Employee.EmployeeService, with primary key {CompanyId, EmployeeId, ServiceId} the attributes {CompanyId, ServiceId} would form a foreign key to the table corresponding to the level Company.Service (because its primary key would be {CompanyId, ServiceId}).

But if we change the name (with a subtype) of ServiceId in the table in which this attribute is part of a foreign key, this does not delete for GeneXus its referential function.

On the other hand, if the attribute to which we change the name (using a subtype) is the one that plays the primary key role, then in the table in which the supertype attribute appears, it does not establish the referential relation.

This isn't explained in the GeneXus course, so you didn't have to know.

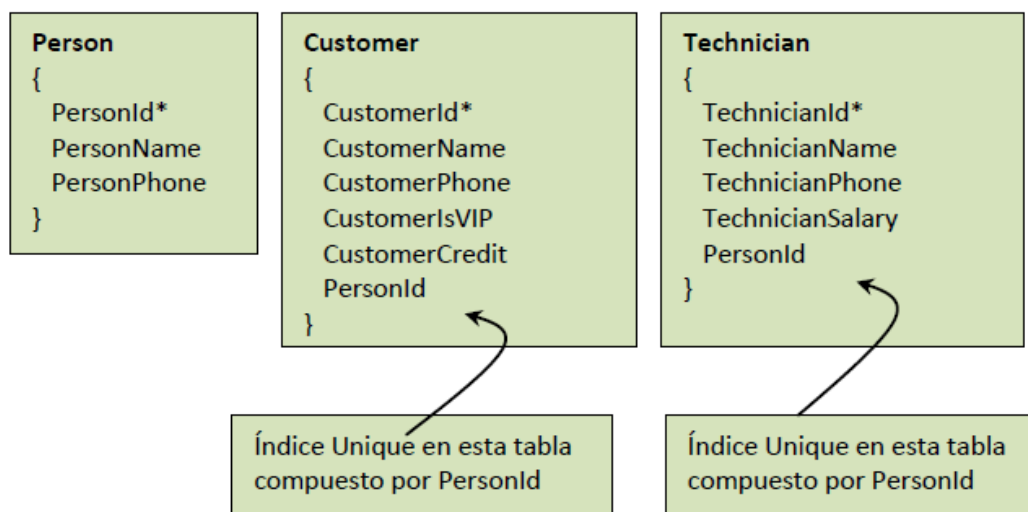
We set the example to help you reason about more complex cases that, however, are common in real life apps.

## QUESTION 12 – DESIGN- SPECIALIZATION

A GeneXus application is used in a casino.

The information that needs to be recorded includes the technicians who repair the slots, and the casino customers. As both technicians and customers are people, for whom a common set of information (name and telephone) is recorded, you need to record the general information only once, and then only record particular details (for example, if the person is a customer, then you need to record if he is a VIP customer and the credit provided by the casino, and if he is a technician, his salary).

Determine whether it is true or false that the following solution adequately resolves this specialization case in GeneXus.



## Answer

False.

Although defining the Unique index on PersonId in the CUSTOMER table establishes a 1-1 relationship with PERSON, we are not really representing that it is the same entity that had to be separated in two tables so that the specialized one only shows the attributes that are not common to the technicians. On the contrary, what is being said is that they are two different entities, with a 1 to 1 relationship, as could be that of a client with his VIP card (see question 4).

Here we are not saying that the existing 1 to 1 relationship between Person and Customer is a special one, where the customer “is” a person. To do this, note that the person will have an attribute to store the name (PersonName) and that the client who has that person associated may have another name (CustomerName).

The same analysis can be done between Person and Technician.

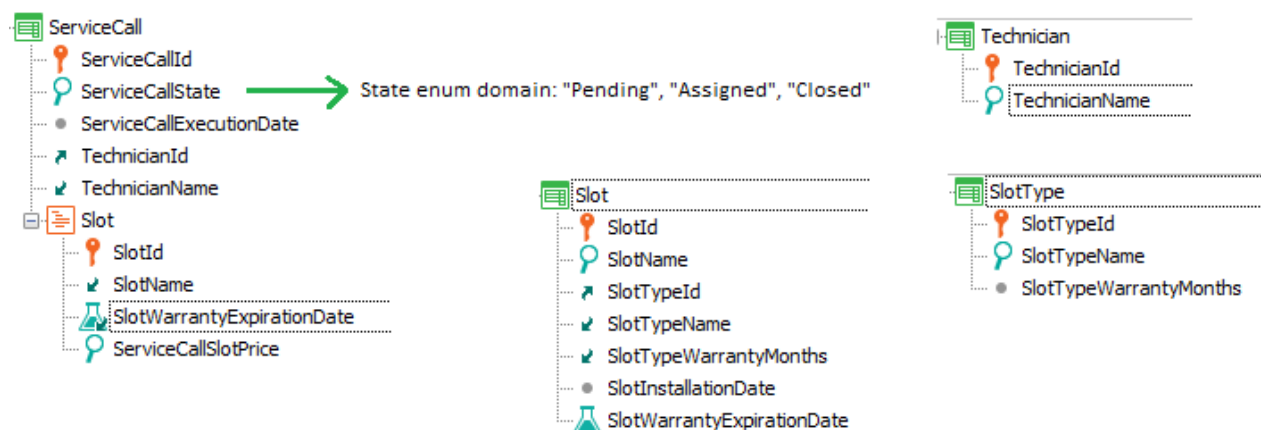
The solution to this specialization problem is given by subtype groups. How would it be? Why does defining CustomerId as a subtype of PersonId solves the problem for Customer? Think that setting it

as a subtype makes it a foreign key. In addition, setting it as the identifier of the Customer transaction turns it into a primary key. Therefore, CustomerId ends up being a primary key that is also a foreign key. So, it finally establishes a 1-1 relationship, through a key, and this is what causes the “is” relationship. Similar analysis for Technician.

### QUESTION 13 – RULES AND EVENTS IN TRANSACTIONS

A system is being developed for a company that supplies, installs and repairs Slots for casinos, bars, etc. The system must register the customers (casinos, bars, etc.), the slots installed for each customer with the date of installation and the end of the warranty, and the customers' calls for repairs of installed slots. Calls are created with status “pending.” When a technician is assigned to them, they become “assigned” and when the technician submits the service ticket corresponding to the call, with the date of completion and prices to be charged to the client for the repaired slots that were not under warranty, they become “closed.”

So, to close a call you must change its status, enter the date of completion and enter the prices of repairs.



A call cannot be allowed to be closed if the prices of all the repaired slots that were not under warranty have not been entered. To do so, the developer types the following in the rules section of the ServiceCall transaction:

```
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();
```

Here the rationale is as follows:

If the call status was “Assigned” and you want to update it, it is because you want to set it as closed. If the closing date is entered, then the status is changed to “Closed” knowing that anyway, if later on, at line level or immediately after, an error rule occurs (before the commit), the changes will be undone and both the status and the date will remain as they were before.

Analyze this solution and discuss other variations.

---

## Answer

Stating these rules will not cause the desired effect.

When working on a two-level transaction in Update mode, the only rules at the lines level to be triggered will be for those lines on which the user has performed some action. If the user does not edit a particular line at all, so that line clearly breaks a condition specified in an error rule, the rule will not be triggered.

Therefore, if you access the transaction in Update mode, enter a date and confirm, and there are slots in the lines that are not under warranty, the error rule will never be triggered and saving will be allowed.

Therefore, you have to check, after working with the lines (that is to say, after all the lines that the user has worked with have been effectively modified in the physical table), that none of the lines that are not under warranty have been left without a price. One possibility would be to invoke a procedure that runs through the records corresponding to the lines and counts how many are missing the price, and if one or more are found, then the rule of error is triggered. Would it be like this?

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

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

Although the error rule depends on the &Missing variable which is loaded by the procedure, since it is not conditioned with the same trigger event as the proc invocation, this rule will be evaluated when opening the transaction and not at the desired moment, which is **after** the proc trigger.

And if you had written:

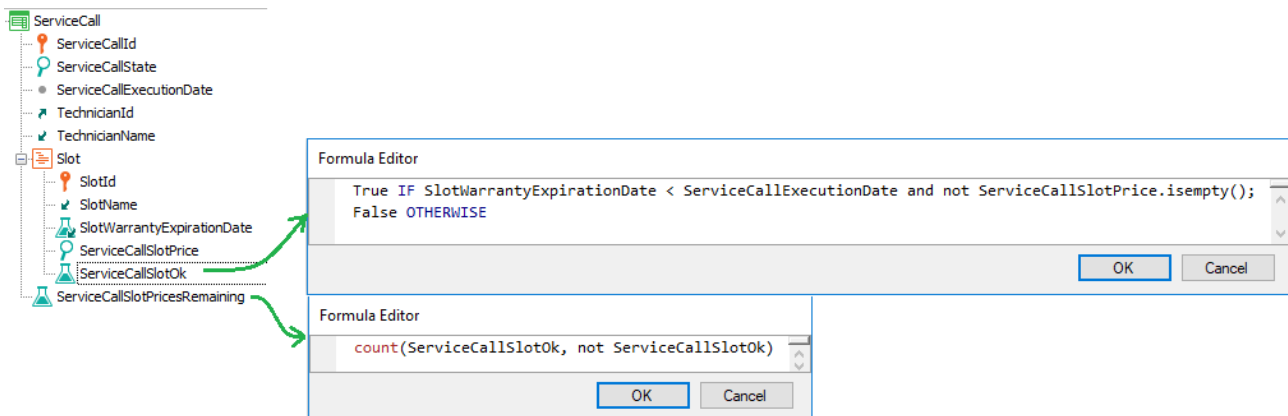
```
&Missing = MissingPrices( ServiceCallId ) if Update;  
  
Error( 'Service call could not be closed') if Update and &Missing <> 0 on AfterLevel  
Level ServiceCallSlotPrice;
```

Here you will have the problem that you will be triggering the procedure as soon as the transaction realizes that it is in Update mode, which is as soon as the identifier field is left. We have not given the user time to do anything, not even to enter the date of completion of the call. Clearly you have to trigger the proc after all the lines that the user wanted to update have been updated.

Therefore, you should have written:

```
&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;
```

Of course, you could have avoided using a procedure, defining the two formulas you indicated:



And in the rules:

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

And couldn't you also take advantage of changing the status to Closed later, when you are sure that everything will be done correctly?

```
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;
```

The last possible time to assign a value to a header attribute is immediately before the header is recorded, and this is well before on AfterLevel level 2.

Actually, the last possible time is on BeforeUpdate.

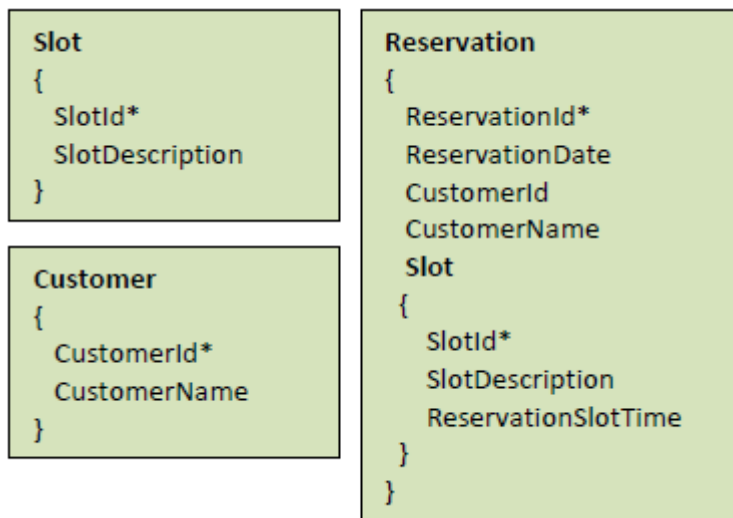
That is to say, you could write:

```
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;
```

## QUESTION 14 – INLINE FORMULAS

A GeneXus application is used in a casino. It has transactions to record the slots (Slot), and slot reservations (Reservation) made by customers (Customer).



Given the following procedure source:

```

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

```



Determine if there is a source table (at the time the formula is triggered), and what it is. Also, determine the table to be navigated by the formula to get your result.

- a. Source table: RESERVATION – Navigated table: RESERVATIONSLLOT
- b. Source table: RESERVATION – Navigated table: RESERVATION
- c. Source table: CUSTOMER – Navigated table: RESERVATION
- d. Source table: RESERVATIONSLLOT – Navigated table: RESERVATIONSLLOT

---

### Answer

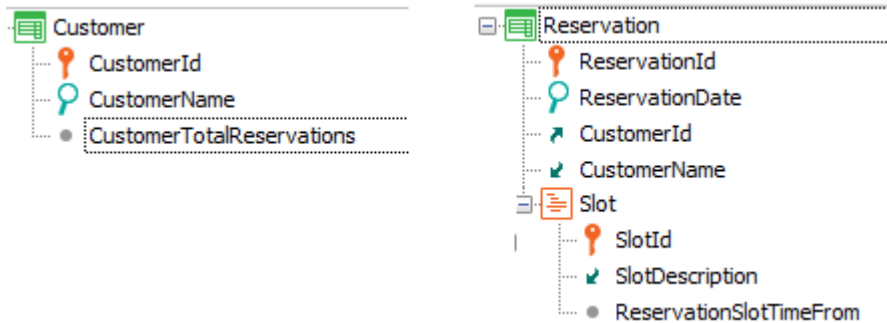
The correct answer is (a).

Anything that is found within a For Each command, by simply being there, refers to an iteration of this command, so its context is the record of the iteration in which the execution is working at a given moment. The base table of the For Each command, when the base transaction is specified, we know it is RESERVATION. If no base transaction was specified, the base table of the For Each command would be calculated with the attributes found “unattached” inside the For Each command (not those within an Aggregate formula, such as Count, or those within a For Each command nested in it, or another command that has its own resolution, such as a New, for example). In this case they will be those of the print block, which is why the table base will be RESERVATION. Therefore, the For Each command will iterate over this table, and for each reservation, the Count formula will be triggered. So, there will be a source table of the formula, which is the context it has when triggered (you are positioned on a reservation). What is counted? The records where ReservationSlotTime is located. What table is that attribute in? In RESERVATIONSLLOT. But when the formula is triggered, in an iteration of the For Each command, you have a ReservationId instantiated (the one from the record you’re working on). So it will count the related records, those corresponding to this reservation. As long as the formula doesn’t have a context (for example, if it is “unattached” inside the Source) it will not have a source table. For this reason, in this case, it would count all the records of the navigated table (RESERVATIONSLLOT), precisely because there is no context in which to specialize the calculation. Context is essential in GeneXus and it’s what makes it intelligent. It can infer things based on that context, without having to make it explicit, saving us programming work.

### QUESTION 15 – UPDATE WITH FOR EACH COMMAND

In the application for a casino that we are developing, we had defined the transactions Customer, Slot, and Reservation (the latter to allow customers to make reservations for slots to be used on a given day). Suppose the app was already in production, so the tables are already loaded with data,

when it is necessary to add an attribute, CustomerTotalReservations, to the Customer transaction, whose value will be the historical total of days in which the client has made reservations. In addition to adding the attribute to the Customer transaction, what else should you do?



Suppose a student develops the following solution:

A procedure is created with the following Source and it is executed after the reorganization:

```

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

Make a complete analysis of the problems of this solution.

## Answer

Clearly, the attribute CustomerTotalReservations corresponds to a known calculation: the sum of all the records in the Reservation table corresponding to the client CustomerId. Therefore, the first question that arises is why not define it as an attribute formula in the Customer transaction itself. Of course, it would be a global formula.

Name	Type	Description	Formula
Customer	Customer	Customer	
CustomerId	Id	Customer Id	
CustomerName	Name	Customer Name	
CustomerTotalReservations	Numeric(4,0)	Customer Total Reservations	count(ReservationDate)





In this way, you wouldn't have to do anything else, since each time the value is needed, the formula will be triggered and the calculation will be made. Note that you don't need to filter the Reservation records corresponding to the customer, since that is an implicit condition due to the context (when the CustomerTotalReservations formula is triggered, it will be because you're working with a certain CustomerId).

But what if the computer system has been in operation for a long time and there is a large number of Reservations for each customer?

Suppose that it is necessary to make frequent reports of the customers ordered according to the number of reservations made. The calculation will have to be triggered every time, for each customer, and this can imply a very high performance cost.

So, there may be a need for this attribute to be physically stored, and simply updated when each reservation is added or deleted. It is enough to define it as a redundant attribute (this is done by setting the Redundant column as visible in the transaction structure and selecting the check box for this attribute). Thus, GeneXus will do for us whatever is necessary. For the developer, it is as if the formula is still virtual, only it is not, so the performance will improve noticeably when making the report frequently requested (because the stored value will be taken directly and the formula will not be triggered again each time).


In fact, when you add a reservation for the customer, GeneXus will have generated code that will be executed updating the formula attribute without us having to worry about it (it will not trigger the formula again, but simply add or subtract 1 according to whether you are inserting or deleting a reservation).

Name	Type	Description	Formula	Redundant
 Customer	Customer	Customer		
 CustomerId	Id	Customer Id		
 CustomerName	Name	Customer Name		
 CustomerTotalReservations	Numeric(4,0)	Customer Total Reser...	count(ReservationDate)	<input checked="" type="checkbox"/>

The student might want to do the same thing, but manually with the procedure, which we already know will do what is expected, although its programming is defective, as we will see and this must be strongly indicated to the student.

Anyway, with your solution all you'll get is to initialize the attribute CustomerTotalReservations, which is not a formula, with each customer's total reservations at the time the proc is executed, but it is not solving what happens each time a new reservations is entered or deleted.

Therefore, your solution would still need to add in the Reservation transaction the CustomerTotalReservations attribute in the structure - it doesn't matter it is inferred: as you're going to use it in the rules, you need to declare it there; otherwise it will not let you save, warning you:

 error src0236: 'CustomerTotalReservations' must be referenced in the transaction's structure. (Transaction 'Reservation' Rules, Line: 1)

And the rule:

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

This is a good time to assess whether you understand what the Add rule does in all modes (Insert, Update, Delete).

In summary: the CustomerTotalAmount attribute is added to the Customer structure; a reorganization is made so that the attribute is added to the table; the procedure to initialize its value is executed only once, and the attribute CustomerTotalAmount is added to the Reservation transaction structure; also, the Add rule is added.

All this is what GeneXus does automatically by defining the formula as redundant.

Question: instead of the procedure, could we have given the Initial Value property of the CustomerTotalAmount attribute in the Customer transaction the value: count(ReservationDate), so that when you reorganize and add the attribute it is loaded with the initial value resulting from calculating that formula for each CustomerId?

Now let's analyze the student's procedure, abstracting the rest of the solution:

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

This procedure attempts to assign the value to the CustomerTotalAmount attribute which has been added to the table and is empty for each of the customers.

That's why the student decided to write a For Each command that will run through the CUSTOMER table, and for each record in the table, he wants to assign the value to CustomerTotalReservation, and for that what it does is code another For Each command that runs through the RESERVATION table, but only for those records corresponding to that CustomerId. And what it does is add one to a variable previously initialized at zero, so that when it finishes iterating through all of that customer's reservations, it will have its number.

A slightly better solution would have been:

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

Another possibility could have been:

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

It's interesting to discuss this last one. Why didn't the student think of that? For each reservation, the extended table is accessed; in this case, you can access the CustomerTotalReservation attribute and update it. The disadvantage could be that you're updating the attribute for each reservation the client has, instead of counting all the reservations and then updating the attribute. On the other hand, with this solution, if many customers have not made any reservations, they will not be run through unnecessarily. But we might as well have done the same thing in the following way:

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

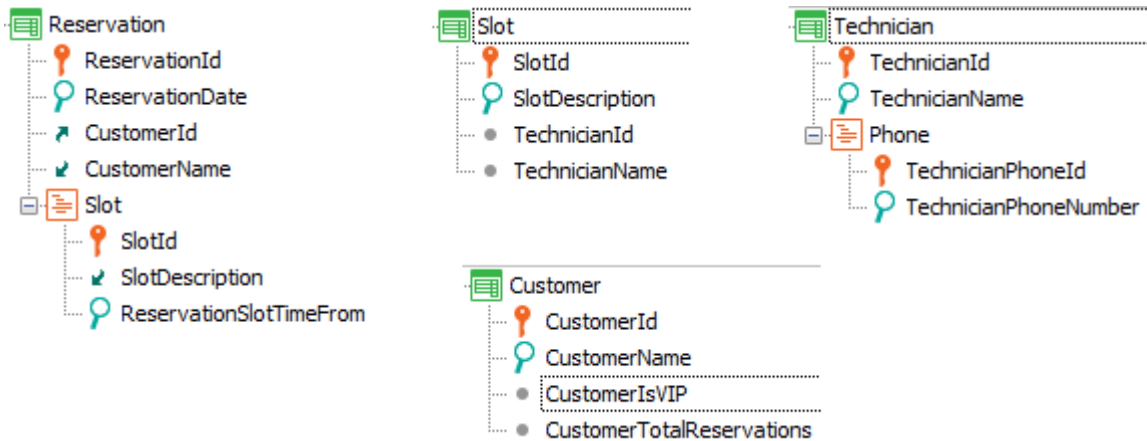
This is the same as implementing a control break per customer. That is to say, the reservations table is run through, grouping by CustomerId (it's what "unique CustomerId" means; that is, we are not keeping the repeated values of CustomerId), and for each unique CustomerId found, the records that the ReservationDate attribute has for that customer are counted (implicit condition). And that calculation is assigned to the CustomerTotalReservation attribute, which belongs to the extended table of Reservation.

It is important to understand the differences between the solutions in order to choose the most suitable one. What is the solution with the best performance?

## QUESTION 16 – EXAMPLES OF NESTED FOR EACH COMMANDS

In the application for a casino that we are developing, we had defined the transactions Customer, Slot, and Reservation (the latter to allow customers to make reservations for slots to be used on a given day). In addition, each slot has a technician assigned to carry out the repairs required:

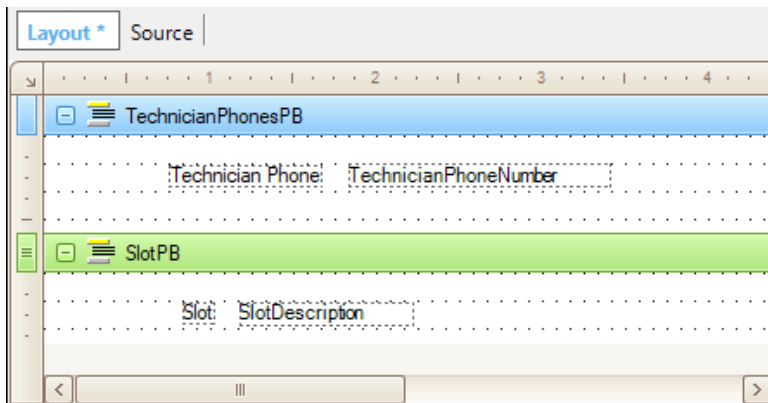
A PDF list needs to be created to show, for a given VIP client, from a given date, the slots he has reserved and for each one, the phones numbers of the slot technician, in case it is necessary to call him urgently to repair the slot.



To resolve the requirement, a procedure such as the one shown below has been implemented. Is it correctly programmed? Analyze it.

```
SlotsAndTechnicianPhones * X
Source * | Layout | Rules * | Conditions | Variables | Help | Documentation |
1 | parm( in: &ReservationDate, in: CustomerId );
2
3 | Output_file("Slots_TechnicianPhones", 'pdf');
4
```

```
SlotsAndTechnicianPhones * X
Source * | Layout | Rules * | Conditions | Variables | Help | Documentation |
Subroutines
1 | for each Reservation.Slot
2 |     where ReservationDate >= &ReservationDate
3 |     print SlotPB
4 |     for each Technician.Phone
5 |         print TechnicianPhonesPB
6 |     endfor
7 | endfor
8
```



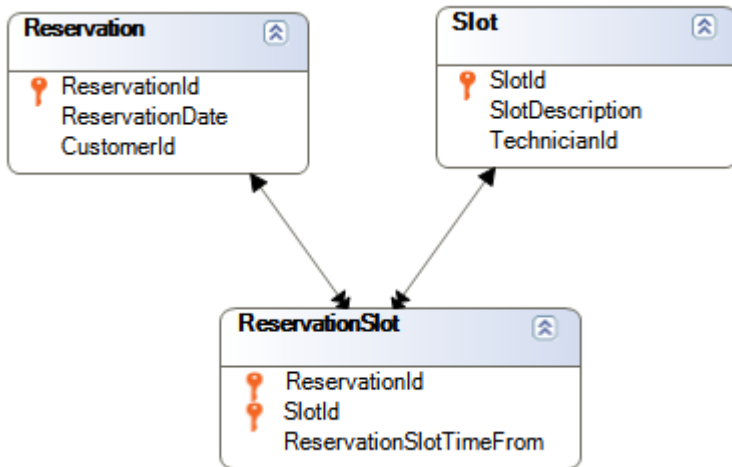
And in the properties, Call Protocol = HTTP

In the next question you should think about how to implement this as a web screen and not as a PDF.

### Answer

It is correctly programmed. At first glance one might think that the Layout is inverted. But remember that the order of the printblocks has nothing to do with how they will be listed on the output. Here they are only declared and designed. It is in the Source where they are sent to the output with the print command.

Let's analyze if the information listed is the one we are asked for. There is a couple of nested For Each commands. In the first one we explicitly say that the base table will be the one corresponding to the Slot level of the Reservation transaction; that is to say, the one called ReservationSlot. We should confirm that within that For Each command no attribute is being used that doesn't belong to the extended table of RESERVATIONSLOT. If so, the navigation list will show a warning that this attribute is not accessible. The attributes to be checked are those that, within the For Each command, are not within the second For Each command. In this case, there are only two attributes in this situation: that of the Where, and that of the printblock called SlotPB. These attributes are SlotDescription, included in the SLOT table, and ReservationDate, included in the RESERVATION table. In the table diagram:



We can clearly see that from RESERVATIONSLLOT we access a single record of the SLOT table and a single record of the RESERVATION table. So, GeneXus must access those two tables every time it iterates in the For Each command.

Which records of the base table will the For Each command work with? With those that comply with the following: when going to the RESERVATION table to evaluate the ReservationDate value, it is greater than or equal to the value of the &ReservationDate variable received in a parameter. Just that? No, the CustomerId value must also match the value directly received through a parameter in that attribute. Remember that “to receive in an attribute” is to establish that this attribute will be instantiated; that is to say, it will be part of the object context. In other words, whenever that attribute participates somewhere, it will do so with the value received when the object was invoked.

Since the For Each command being analyzed already accesses the RESERVATION table containing it, an automatic filter will be applied for that value of CustomerId.

An important observation: that the attribute received in the Parm rule belongs to the extended table of the For Each command DOES NOT make it automatically apply it as a filter. For this to happen, **the For Each command must access that particular table of the extended table** to do something.

To better understand it, let's think what would have happened if we didn't want to list the slots of a given customer *from* a given date, but *on* a given date. As in this case the two filters will be equality filters, we might want to receive both parameters directly in the corresponding attributes. That is to say, we would have specified the following:

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

And in the Source:

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



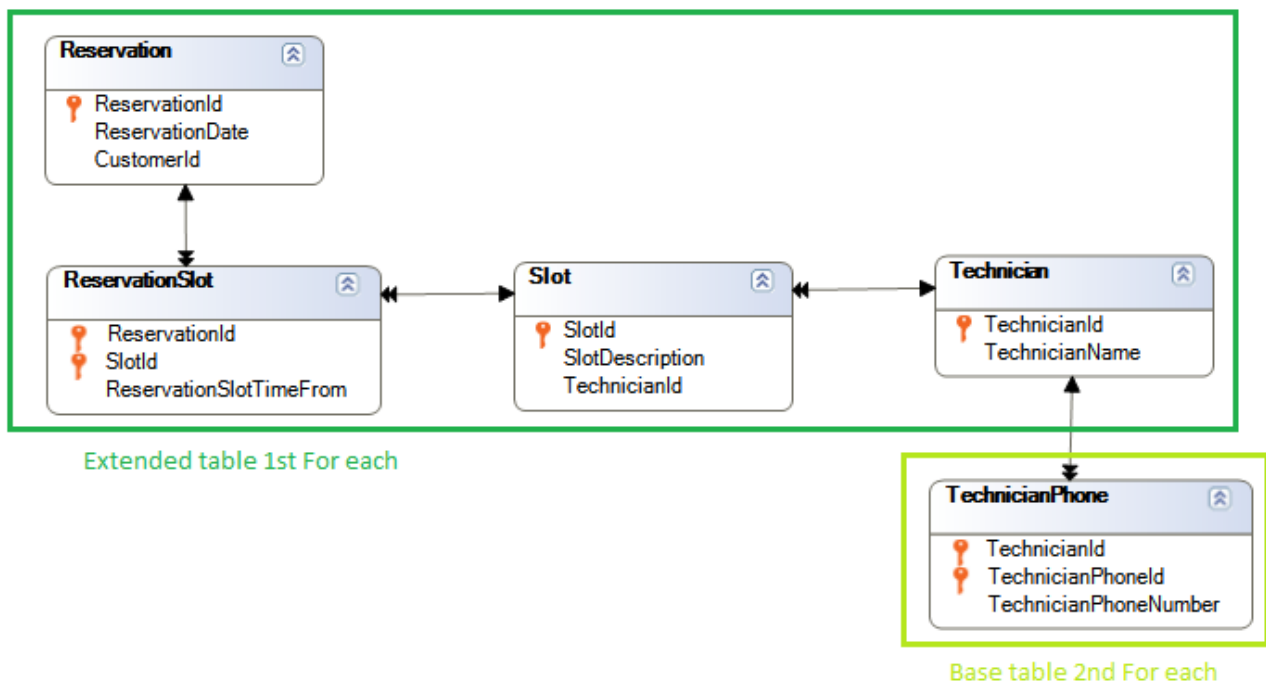
```
...  
endfor
```

But when doing this, the first For Each command will no longer have to access the Reservation table, but only Slot (to look for the value of SlotName). And in this case, then, the equality filters will not be applied, since they are on the Reservation table. You can confirm this by looking at the navigation list. The entire ReservationSlot table will be run through with no filters.

Therefore, it must at least declare one of the parameters as a variable, and use it in a Where clause filtering by equality. For example, “where ReservationDate = &ReservationDate.” Since CustomerId is in that same table, now it will apply this filter as well.

Think about what happens with the nested For Each command. Its base table will clearly be the one associated with the Phone level of the Technician transaction (that is to say, TechnicianPhone). The following question is: Does it establish implicit filters for the information it will use? Yes, it will show the phones of each slot’s technician. Why?

GeneXus looks for a relationship between the extended table of the external For Each command and the base table of the nested For Each command:



It's another way of looking for a 1 to N relationship, although in this case it is an indirect one. If each slot reservation slot has a TechnicianId, and in the table to be navigated there is also a TechnicianId, GeneXus understands that by the relationship between the information, it would be the same. That's why the join is made. It can clearly be seen in the navigation list:

<b>For Each ReservationSlot (Line: 1)</b>							
Order:	<u>ReservationId</u> , <u>SlotId</u> Index: IRESERVATIONSLOT						
Navigation filters:	Start from: FirstRecord Loop while: NotEndOfTable						
Constraints:	<u>CustomerId</u> = @CustomerId <u>ReservationDate</u> >= &ReservationDate						
Join location:	Server						
<table> <tr> <td></td> <td>=ReservationSlot ( <u>ReservationId</u> , <u>SlotId</u> )</td> </tr> <tr> <td></td> <td>=Reservation ( <u>ReservationId</u> )</td> </tr> <tr> <td></td> <td>=Slot ( <u>SlotId</u> )</td> </tr> </table>			=ReservationSlot ( <u>ReservationId</u> , <u>SlotId</u> )		=Reservation ( <u>ReservationId</u> )		=Slot ( <u>SlotId</u> )
	=ReservationSlot ( <u>ReservationId</u> , <u>SlotId</u> )						
	=Reservation ( <u>ReservationId</u> )						
	=Slot ( <u>SlotId</u> )						
<b>For Each TechnicianPhone (Line: 6)</b>							
Order:	<u>TechnicianId</u> Index: ITECHNICIANPHONE						
Navigation filters:	Start from: <u>TechnicianId</u> = @TechnicianId Loop while: <u>TechnicianId</u> = @TechnicianId						
<table> <tr> <td></td> <td>=TechnicianPhone ( <u>TechnicianId</u> , <u>TechnicianPhoneId</u> )</td> </tr> </table>			=TechnicianPhone ( <u>TechnicianId</u> , <u>TechnicianPhoneId</u> )				
	=TechnicianPhone ( <u>TechnicianId</u> , <u>TechnicianPhoneId</u> )						

The @ symbol always indicates context data (note the @CustomerId, which refers to the value received in the CustomerId attribute of the Parm rule). This @TechnicianId refers to the value of the attribute with that name in the Slot table accessed by the first For Each command.

If this is not what the programmer wants, is there a way to avoid these automatic filters? Yes, using a **subroutine** to encapsulate the code of this second For Each command (don't worry, you didn't have to know this because it isn't taught in the GeneXus course):

```

1 for each Reservation.Slot
2     where ReservationDate >= &ReservationDate
3
4     print SlotPB
5     Do 'ListTechnicianPhones'
6 endfor
7
8 Sub 'ListTechnicianPhones'
9     for each Technician.Phone
10        print TechnicianPhonesPB
11    endfor
12 endsub
13

```

If you do this, in the navigation list you will now see:

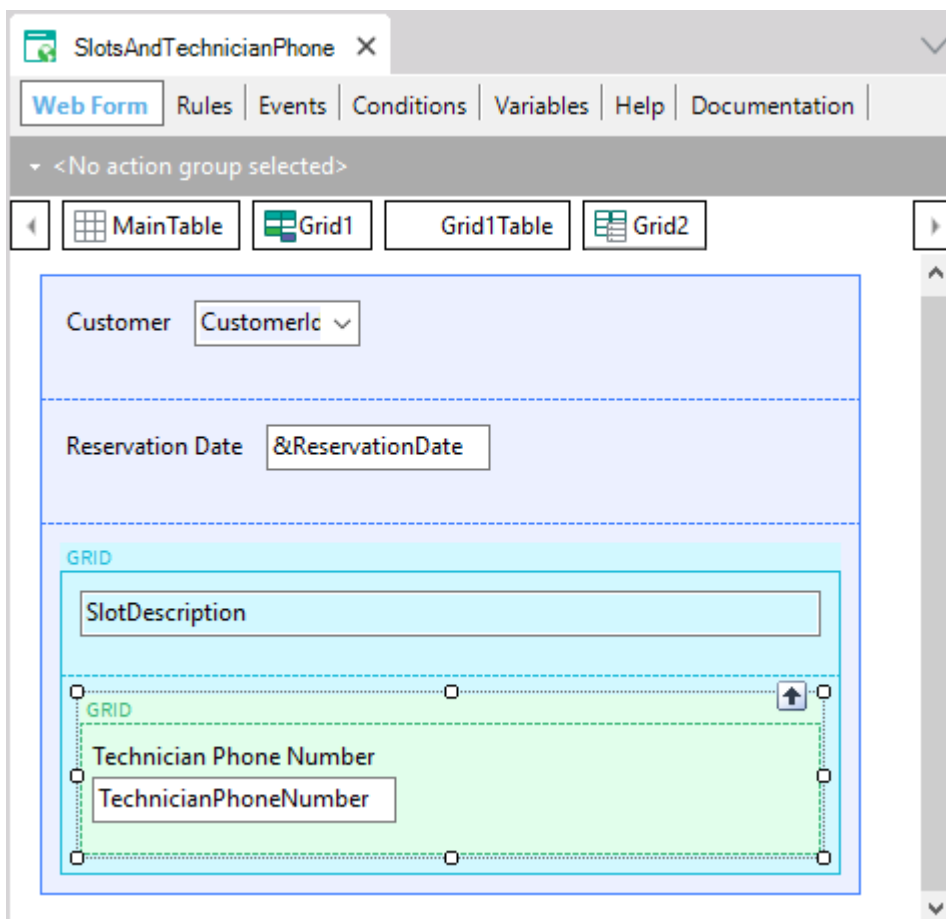
LEVELS	
For Each ReservationSlot (Line: 1)	
Order:	<a href="#">ReservationId</a> , <a href="#">SlotId</a> Index: IRESERVATIONSLOT
Navigation	Start from: FirstRecord
filters:	Loop while: NotEndOfTable
Constraints:	<a href="#">CustomerId</a> = @CustomerId <a href="#">ReservationDate</a> >= &ReservationDate
Join location:	Server
<div> <div></div> <div>=ReservationSlot ( <a href="#">ReservationId</a>, <a href="#">SlotId</a> )</div> </div> <div> <div></div> <div>=Reservation ( <a href="#">ReservationId</a> )</div> </div> <div> <div></div> <div>=Slot ( <a href="#">SlotId</a> )</div> </div>	
For Each TechnicianPhone (Line: 10)	
Order:	<a href="#">TechnicianId</a> , <a href="#">TechnicianPhoneId</a> Index: ITECHNICIANPHONE
Navigation	Start from: FirstRecord
filters:	Loop while: NotEndOfTable
<div> <div></div> <div>=TechnicianPhone ( <a href="#">TechnicianId</a>, <a href="#">TechnicianPhoneId</a> )</div> </div>	

## QUESTION 17 – NESTED GRIDS

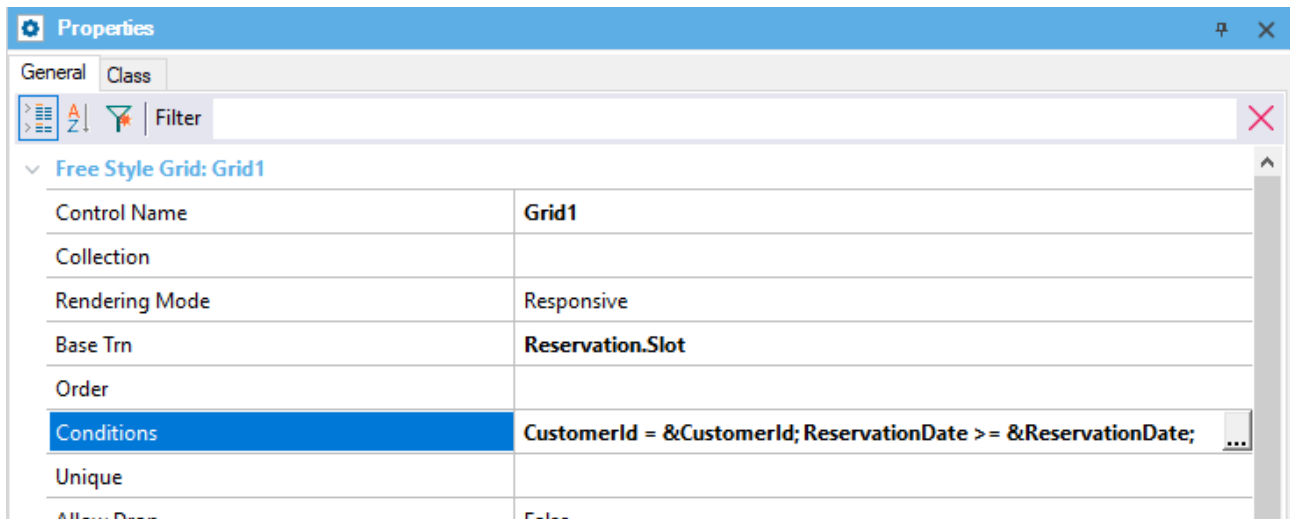
Now we want to implement the same requirement as in the previous question, but instead of doing it in a PDF list, we want to do it on a web screen.

### ANSWER

To implement this as a web screen and not a PDF, we could place two variables for the user to enter the desired customer and date values each time, and then have the information shown with a couple of nested grids, where the first one must **NECESSARILY** be a freestyle grid. It **CANNOT** be a standard grid. Why? Because a standard grid only admits fixed columns, of attribute/variable type. And here what we want to place inside the grid is, in addition to the SlotDescription attribute, **ANOTHER GRID**!

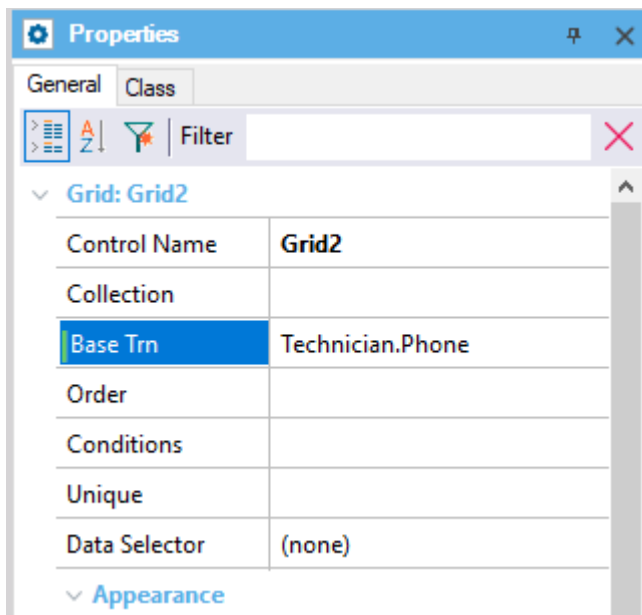


To the Freestyle grid we also indicate its base transaction, just as we did for the external For Each command in the procedure, and write the filters through the Conditions property of the grid (the equivalent to the Where clauses of the For Each command):










If we hadn't written the condition by CustomerId using a variable in the form itself, but we had received the CustomerId in a parameter, it would be worth exactly the same as we saw in the procedure case.

What happens with the information in the nested grid? Here we have placed a standard grid, but it could also be a freestyle grid. Note that all we have done is place in it the attribute TechnicianPhoneNumber. To ensure the navigation we want, we could also specify its base transaction, which we always suggest as good practice:



We can ask ourselves if in the case of nested grids the same will be valid as for the case that we already studied of nested For Each commands. The answer is yes.

That is, it will apply the filter by TechnicianId of the slot in question, exactly the same as for the case of nested For Each commands of the procedure.

FILL <u>CustomerId</u> WITH <u>CustomerId</u> , <u>CustomerName</u> IN	
 =Customer ( <u>CustomerId</u> ) INTO <u>CustomerId</u> <u>CustomerName</u> ORDER <u>CustomerName</u>	
Event Grid1.Load 	
Order:	<u>ReservationId</u> , <u>SlotId</u> Index: IRESERVATIONSLOT
Navigation	Start from: FirstRecord
filters:	Loop while: NotEndOfTable
Constraints:	<u>ReservationDate</u> >= &ReservationDate <u>CustomerId</u> = &CustomerId
Join location:	Server
 =ReservationSlot ( <u>ReservationId</u> , <u>SlotId</u> )  =Reservation ( <u>ReservationId</u> )  =Slot ( <u>SlotId</u> )	
Event Grid2.Load 	
Grid1	
Order:	<u>TechnicianId</u> Index: ITECHNICIANPHONE
Navigation	Start from: <u>TechnicianId</u> = @TechnicianId
filters:	Loop while: <u>TechnicianId</u> = @TechnicianId
 =TechnicianPhone ( <u>TechnicianId</u> , <u>TechnicianPhoneId</u> )	

Note, too, that we have modified the Control Type of the &CustomerId variable, to show at runtime a combo box with the CustomerName values of the Customer table, so that when the user selects the desired customer name, the variable is internally assigned its CustomerId. This is used by the property Control Type = “Dynamic Combo Box” of the control &CustomerId. The navigation list is indicating at the beginning the navigation of the Customer table to solve this.

If instead of having specified the grids as nested grids we had presented them as parallel grids, we would have had to explicitly program for the technicians’ phones grid the filter by TechnicianId in its conditions. The reason is that for parallel grids, just as it happens for parallel For Each commands, navigations are not related. And it is understandable; why should they be related automatically, if there is nothing to suggest that they should be related? In the case of nested grids, one would clearly think that if the information is linked in reality, it most likely should also to be presented as linked.

In the PDF list that we saw, when we didn’t want GeneXus to apply the implicit filters to the nested For Each command, we created a subroutine whose effect was to disregard the context. What

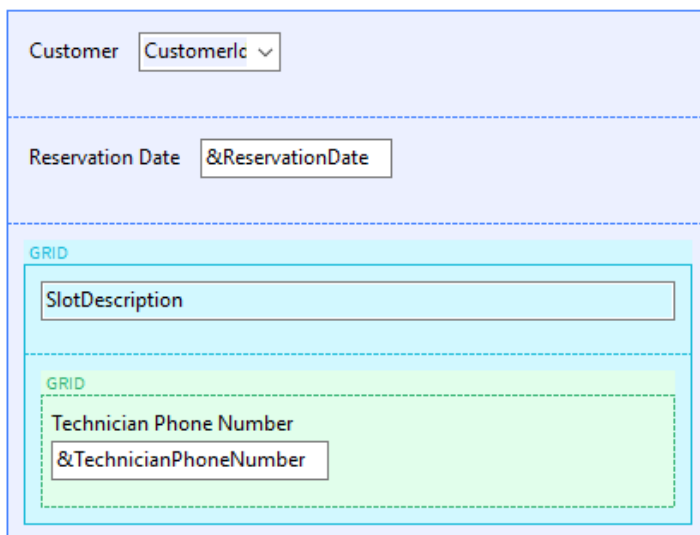
would we do here if for every SlotDescription of the reservation we wanted to see all the technicians' phones, not only those of the slot? In other words, how do we ensure that it doesn't apply the implicit filter, which is the one made by the join?

The easiest way is for the nested grid not to have a base table. When the grid doesn't have a base table, its loading is left to the developer, through the grid's Load event.

In the previous navigation list (that of grids with base tables), the Load event is indicated for each grid, and below it a table navigation is indicated. The student could then think that the Load event is triggered only once, where this sort of implicit For Each command is executed. Not to be confused: we know that when grids have a base table, a Load event is triggered for each record accessed in the navigation, immediately before loading its information into the grid.

The Load event of a grid is triggered only once when it **doesn't** have a base table.

Then what we would do is change in the nested grid the TechnicianPhoneNumber attribute for a variable:



But we must also be careful to remove any reference to a table or attribute anywhere else in relation to the grid. Of course, we need to empty the Base Trn property.

And in the Load event of the grid:

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

If you don't place the Load command inside the event, what will happen? No lines will be added to the grid. In other words, no phone will be loaded, even if its table is navigated.

In the navigation list, we now clearly see the difference between the grid with a base table and the grid without a base table:

```
FILL CustomerId WITH CustomerId, CustomerName IN
  [Table Icon]=Customer ( CustomerId ) INTO CustomerId CustomerName
  ORDER CustomerName
```

---

```
Event Grid2.Load
```

---

```
For Each TechnicianPhone (Line: 2)
```

---

```
Order:      TechnicianId , TechnicianPhoneId
Index: ITECHNICIANPHONE
Navigation  Start from:  FirstRecord
filters:    Loop while:  NotEndOfTable

[Table Icon]=TechnicianPhone ( TechnicianId, TechnicianPhoneId )
```

---

```
Event Grid1.Load
```

---

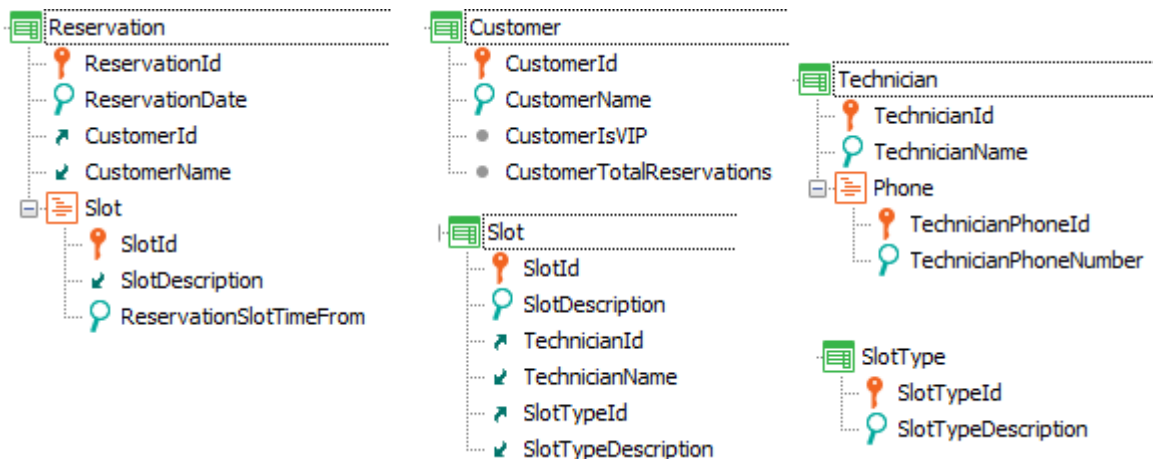
```
Order:      ReservationId , SlotId
Index: IRESERVATIONSLOT
Navigation  Start from:  FirstRecord
filters:    Loop while:  NotEndOfTable
Constraints: ReservationDate >= &ReservationDate
             CustomerId = &CustomerId
Join location: Server

[Table Icon]=ReservationSlot ( ReservationId, SlotId )
[Table Icon]=Reservation ( ReservationId )
[Table Icon]=Slot ( SlotId )
```

## QUESTION 18 – NESTED GRIDS (CONTINUED)

Extending the previous question, we have now added a type for each Slot, which is recorded in a separate transaction:





Now, the web panel that shows the slots reserved for a customer starting from a given date should also show:

1. For each listed slot the number of slots of the same type existing in the Casino; and
2. For each phone number of the associated technician, the corresponding telecommunications company (which we assume is not registered in the database, but is found with a proc from the analysis of the first digits of the phone number –eventually, it could be an external service).

Analyze why the solution below doesn't work (assuming that the CellPhoneCompany procedure is correctly programmed) and introduce a complete, well-developed solution. Try discussing other possibilities.

Web Form:

The web form is titled "Customer" and contains a dropdown menu for "CustomerId" with the value "Customer1c" selected. Below this is a section for "Reservation Date" with a text input field containing "&ReservationDate". The main content area is divided into two sections, each labeled "GRID". The top section, "Slots of this type", contains a table with two columns: "SlotDescription" and "Slots of this type". The bottom section, "Cell Phone Company", contains a table with two columns: "Technician Phone Number" and "Cell Phone Company".

Slots of this type	
SlotDescription	&SlotsQty

Cell Phone Company	
Technician Phone Number	&Company

Events:

```

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

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

```

Tip: One of the problems found by the student is that value 0 is always displayed for the Slots of this type, when actually there is always at least one slot of that type: the one that is being loaded in the grid.

Then, he tries the next modification, and it still shows 0:

```

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

```

---

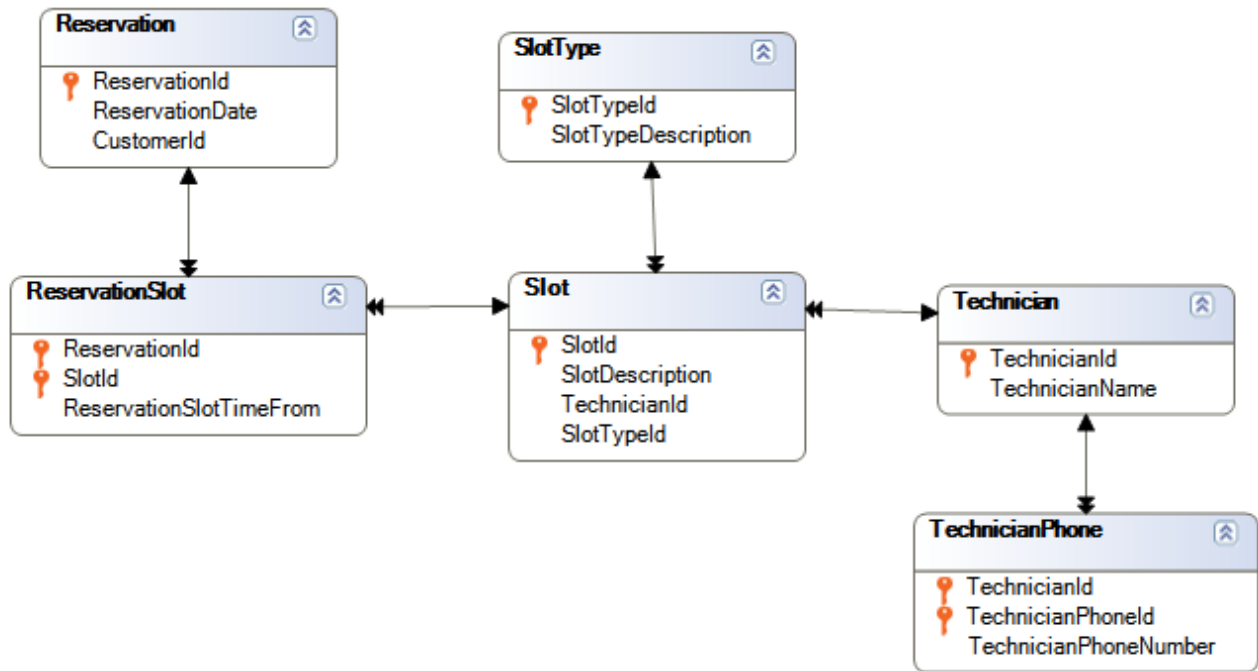
### Answer

Two variables have been added, &SlotsQty for the external grid (called Grid1) and &Company for the internal grid, &Company.

Both grids have a base table, so the attributes will be automatically loaded in both grids, according to their navigations.

But to load the variables, how do we do it? If a grid has a base table, we know that the grid's Load event will be triggered n times: once for each line to be loaded, immediately before loading it, and the developer has no need to type the Load **command**. Therefore, it is in the Load event where the variables that have been included in the grid have to be assigned a value.

Let's start by the outermost grid. In the Load event we have to calculate the existing number of slots of the same type as the one that is about to be loaded on that line triggered by the Load event. Therefore, there is a context for everything that is written inside the Load event: we are positioned on a record of the base table, ReservationSlot, and on each one of the records associated by extended table that we need. In other words, the attributes of the ReservationSlot table will be instantiated for that SlotId; of the Slot table for that slot if needed; of the SlotType table for that slot if needed (as well as of Technician for that slot, and of Reservation).



To calculate the number of slots of the Slot table that have the same SlotTypeId as that of the slot already instantiated within the Load event, how do we go about it? The student initially typed:

```

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

What did he tried to do? Why did he condition the records to be counted with the Count formula in this way?

Most likely, he has tried using the unconditional formula before adding that condition:

```

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

Perhaps he thought that only the slots of that type would be counted according to the context. Let's analyze why it isn't so. We know that inline formulas determine the table to be navigated by the attributes specified in it, regardless of the context. Then, they take into account the context to add the corresponding filters. In this case, the table to be navigated by the Count formula will be Slot. But since the formula is within a context in which we are positioned on a record of ReservationSlot that has a SlotId, this equality filter will be applied. Then there will always be a single record, the one corresponding to that SlotId.

Obviously we cannot use the formula within the context of the Load event, because the filter by SlotId must not be made. All the solutions sought by the student are incorrect because he can't disregard that context and, therefore, not have the filter added automatically by SlotId.

We will have no alternative but to do the calculation in a procedure, to do without that context.

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

We would program the SlotsQtyOfAType procedure:

Rules:

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

Source:

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

Or use a subroutine, so as not to leave the code itself, especially if the proc is not going to be used by any other object:

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

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

Parameters cannot be passed in subroutines, and since their code is inside the same program that invokes them, they see the status of the variables as they are when the subroutine was invoked. For this reason, we load the value of the &SlotTypeId variable, so the subroutine can use it.

What if the student had thought of defining in the SlotType transaction a formula attribute as shown below?

Name	Type	Description	Formula	Nullable
SlotType	SlotType	Slot Type		
SlotTypeId	Id	Slot Type Id		No
SlotTypeDescription	Description	Slot Type Description		No
SlotTypeQuantity	Numeric(4,0)	Slot Type Quantity	count( SlotDescription)	...

And in the web form placed that attribute instead of the &SlotsQty variable; or, if he didn't want to modify the web form, wrote in the Load event:

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

It will work perfectly! Note that a formula attribute (global) is not the same as an inline formula (local). This example clarifies the effect of a formula **ATTRIBUTE**.

Of course, defining too many formula attributes in a transaction just because it is useful to implement a certain panel will not be good practice. You have to find the cost/benefit balance. If this calculation is going to be required elsewhere, then it makes sense to define the formula at the level of an attribute in the transaction.

Another option would have been to implement the grid without a base table. That is to say, without a base transaction and without “unattached” attributes in the related events. And, of course, with variables in the grid (instead of the SlotDescription attribute we’ll insert a &SlotDescription variable). But we’ll have the same problem with nesting as before. The Load event will be as follows:

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

Don’t forget to type the Load command to effectively load the line into the grid.

---

#### LOAD COMMAND IN A LOAD EVENT OF A GRID WITH A BASE TABLE

Now we have to analyze the loading of the nested grid. Clearly, the Load command is not necessary.

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

But, what is the effect? In this case it has no effect. What we'll see at runtime is exactly the same. However, explicitly typing the Load command makes GeneXus not implicitly place this same command at the end of its code. This is useful if I want, as a developer, that only some records that "pass" through the Load event are effectively loaded into the grid, and not all of them.

For example, if the company is "X" I don't want to have the phone loaded in the grid:

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

Even if I want to load more than one grid line in the same Load event for a record, I will have to type the Load command into the event twice.

With these examples we show that it may be necessary in some cases to explicitly write the Load command in a Load event of a grid WITH base table.

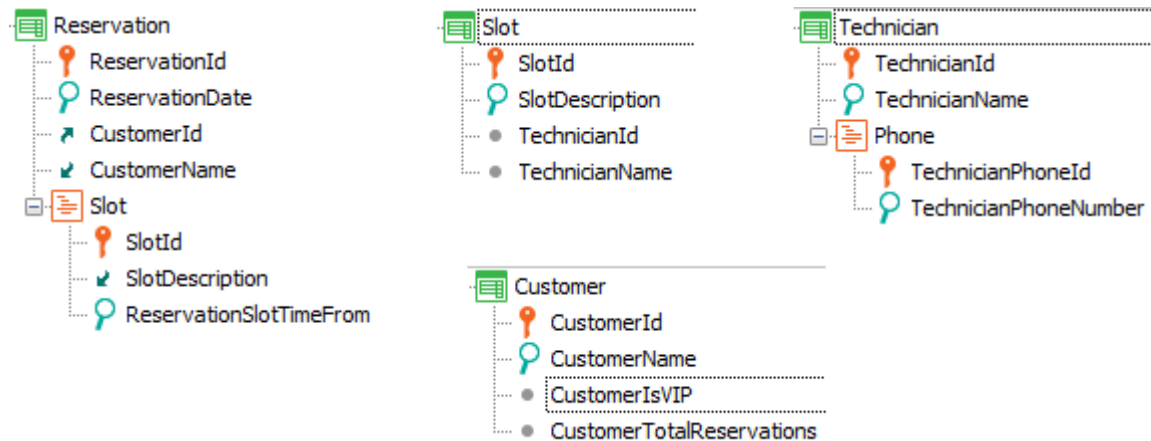
---

#### WHEN TO PLACE HIDDEN ATTRIBUTES IN A GRID

A common error that you will find in student implementations: place the TechnicianId attribute in the nested grid, but hidden. Why is it not necessary? Nothing you need in the Load event requires the attribute to be in the grid, because the time to trigger the Load event is when you are in the database, with access to the extended table. TechnicianId would have to be loaded in the grid only if it is needed later, in a user event, once the data was already loaded in the grid, and we are no longer connected to the database, but working exclusively with on-screen data.

## QUESTION 19 – UNIQUE OR CONTROL BREAK?

Moving on with the app:



Suppose a PDF list has to be made to show for each slot that has ever been reserved, the number of times it has been reserved.

Student A comes up with the following solution:

```

For each Slot
    &slotsInReservations = count( ReservationSlotTimeFrom)
    print SlotPB //with attribute SlotDescription and variable &slotsInReservations
endfor
  
```

On the other hand, student B creates this other solution:

```

For each Slot
    &slotsInReservations = 0
    For each Reservation.Slot
        &slotsInReservations += 1
    endfor
    print SlotPB //with attribute SlotDescription and variable &slotsInReservations
endfor
  
```

You tell them that both solutions are equivalent, even though student B is using the internal For Each command to do what the Count formula of student A does better, since it's meant to be optimized. In general –though not always– it is better to use a formula than to manually do what the formula already does.

But beyond that, you tell them that both solutions are wrong on one thing: they are not being asked to list **all** slots, including those that were never reserved.

Then student A proposes the following solution:

```

for each Reservation.Slot
    unique SlotId
    &slotsInReservations = count( ReservationSlotTimeFrom )
    print SlotPB //with attribute SlotDescription and variable &slotsInReservations
endfor

```

Meanwhile, student B creates this other solution:

```

for each Reservation.Slot
    order SlotId
    &slotsInReservations = 0
    for each Reservation.Slot
        &slotsInReservations += 1
    endfor
    print SlotPB //with attribute SlotDescription and variable &slotsInReservations
endfor

```

Will you tell them that both solutions are equivalent?

Answer

Yes, they are both equivalent. In fact, if you look at the navigation lists...

This is A's:

LEVELS

For Each ReservationSlot (Line: 1)

Order: SlotId

Index: IRESERVATIONSLOT2

Unique: SlotId

Navigation Start from: FirstRecord

filters: Loop while: NotEndOfTable

Join location: Server

ReservationSlot ( ReservationId, SlotId ) INTO SlotId

Slot ( SlotId ) INTO SlotDescription

count( ReservationSlotTimeFrom ) navigation ( SlotId )

Formulas

Navigation to evaluate: count( ReservationSlotTimeFrom )

Given: SlotId

Index: IRESERVATIONSLOT

Group by: SlotId

ReservationSlot ( SlotId )

You can see that the formula applies the filter by SlotId (Note “Given: SlotId”).

This one belongs to B:



LEVELS	
For Each ReservationSlot (Line: 1)	
Order:	SlotId
	Index: IRESERVATIONSLOT2
Navigation	Start from: FirstRecord
filters:	Loop while: NotEndOfTable
Join location:	Server
<div> <div>ReservationSlot ( ReservationId, SlotId ) INTO SlotId</div> <div>Slot ( SlotId ) INTO SlotDescription</div> </div>	
Break ReservationSlot (Line: 4)	
Order:	SlotId
	Index: IRESERVATIONSLOT2
Navigation	Loop while: SlotId = @SlotId
filters:	
<div> <div>ReservationSlot ( ReservationId, SlotId )</div> </div>	

B implemented a control break, while A used the Unique clause that was designed for this type of scenarios, where you want to navigate a table and make an aggregation on that same table, according to some attribute or attributes that you want to use only once in the output.

It is more likely that a solution that has been implemented to resolve certain scenarios and that contains a formula will have a better chance of being resolved efficiently than a solution in which we must implement what the formula already has implemented. But this isn't always the case.

Students could then develop the idea that whenever they need to implement a control break they could avoid it by using the Unique clause, but this is not true.

For example, if the request didn't involve an aggregation, but showing the reservation dates for each slot that has ever been reserved, we would have no alternative but to implement a control break:

```
for each Reservation.Slot
  order SlotId
  print SlotPB //with attribute SlotDescription
  for each Reservation.Slot
    print ReservationPB //with attribute ReservationDate
  endfor
endfor
```

There is no way to do this with a Unique clause.

Ask the student if there is any difference between placing in the Order clause the SlotId attribute or the SlotDescription attribute.

But, on the other hand, for the original problem, isn't the solution implemented by student A (join) better but simply by adding If?

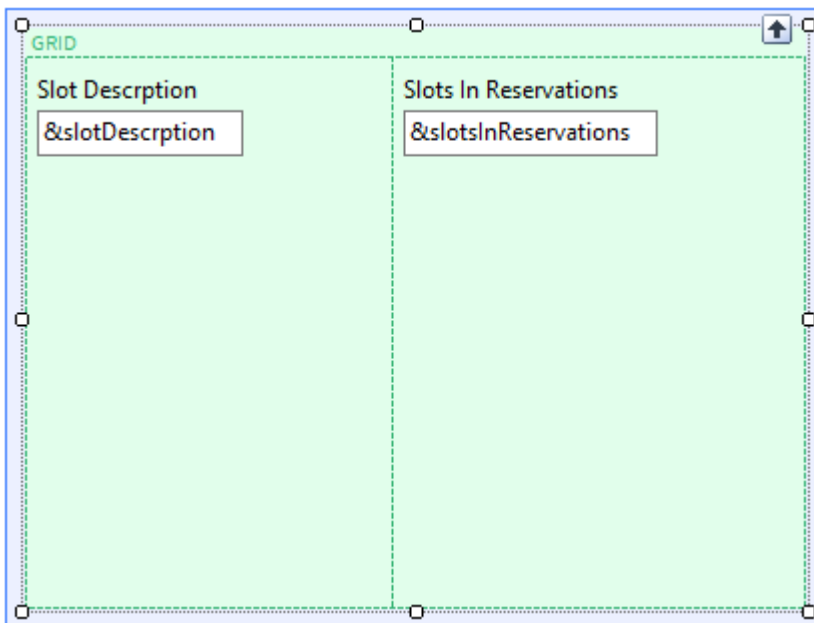
```
For each Slot
    &slotsInReservations = count(ReservationSlotTimeFrom)
    If &slotsInReservations > 0
        print SlotPB //with attribute SlotDescription and variable
&slotsInReservations
    endif
endfor
```

It is to be expected that there will be significantly fewer records in the Slot table than in the ReservationSlot. Will there be a difference between this solution and that of the same student but navigating only ReservationSlot with Unique?

## QUESTION 20 – UNIQUE IN GRID (CONTINUED)

Now it is requested that instead of solving the above as a PDF you solve it in a web panel.

The student then decides that he will have to insert a grid without a base table:



To be able to program his solution with the Unique clause in the Load event:

```
Event Grid1.Load()
    for each Reservation.Slot
```

```

        unique SlotId
        &slotDescription = SlotDescription
        &slotsInReservations = count( ReservationSlotTimeFrom )
        Load
    endfor

```

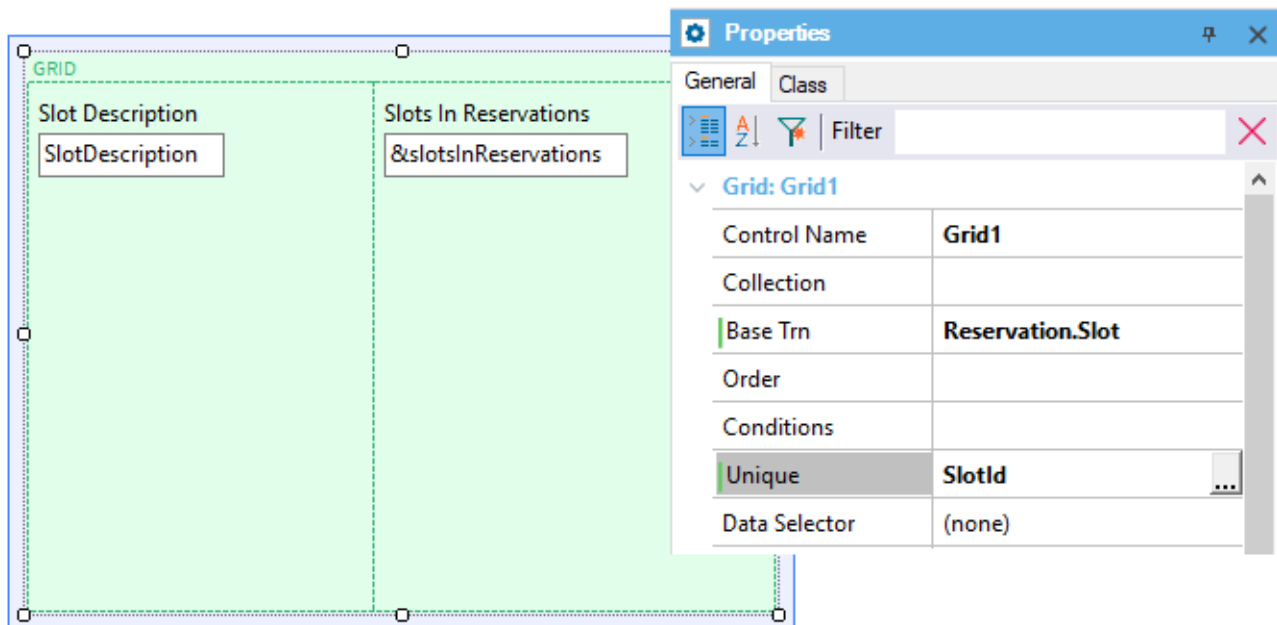
endevent

There is no way of doing the same using a grid with a base table. Explain how it would be done.

### Answer

It's odd that what can be done with a For Each command cannot be done with a grid with a base table, given that in this case there is some sort of underlying For Each command. The same can be said for a Data Provider group. They are equivalent elements in its logic.

Therefore, it will be enough to ask the student if there isn't a Unique property at the grid level.



```

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

```

## QUESTION 21 – UNIQUE IN DATA PROVIDER AND DYNAMIC TRANSACTION (CONTINUED)

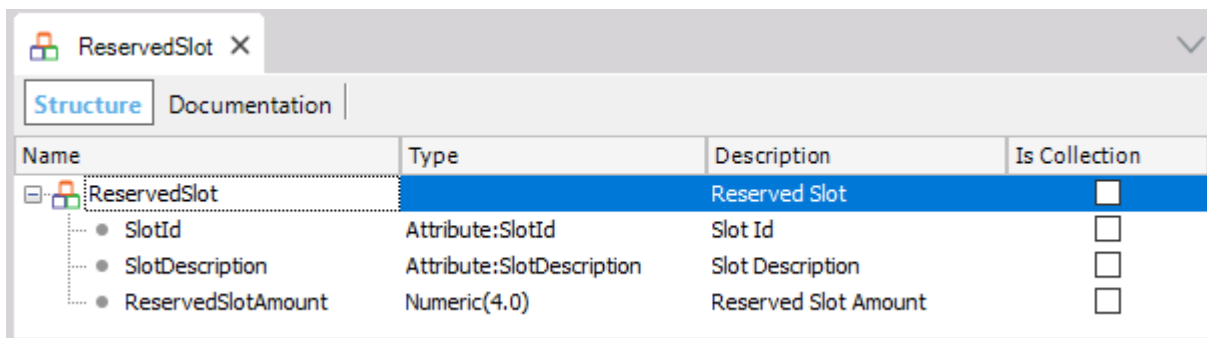
You want the students to be able to explore the most interesting solutions to solve the requirement, integrating the concepts offered in the course. So now you make them the following question: “If now we want to show in the grid (or in the PDF) the slots sorted by number of reservations, from highest to lowest, how do you implement it?”

Think of the possible answers.

### Answer

Some students may think of loading in a collection SDT the values of SlotDescription and &slotsInReservations and then in the web panel or procedure run through the ordered collection.

For example, define the SDT:



Name	Type	Description	Is Collection
ReservedSlot		Reserved Slot	<input type="checkbox"/>
• SlotId	Attribute:SlotId	Slot Id	<input type="checkbox"/>
• SlotDescription	Attribute:SlotDescription	Slot Description	<input type="checkbox"/>
• ReservedSlotAmount	Numeric(4,0)	Reserved Slot Amount	<input type="checkbox"/>

And load in a Data Provider:



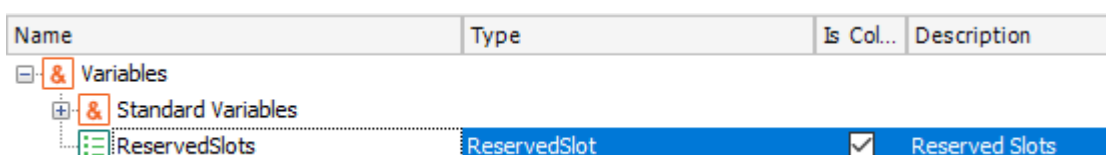
RetrieveReservedSlot	
Qualified Name	RetrieveReservedSlot
Object Visibility	Public
Output	
Infer Structure	No
Output	ReservedSlot
Collection	True
Collection Name	RetrieveReservedSlot
Network	

```

1 ReservedSlot from Reservation.Slot
2 unique SlotId
3 {
4   SlotId
5   SlotDescription
6   ReservedSlotAmount = count(ReservationSlotTimeFrom)
7 }
  
```

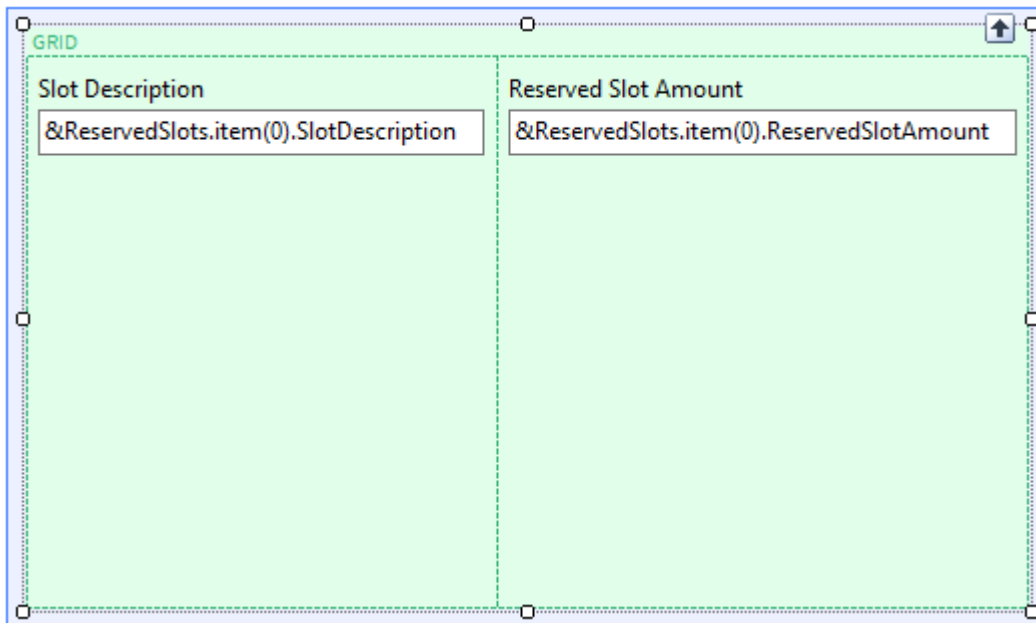
Note that the way of declaring the Data Provider group is similar to the way of coding the For Each command with the Unique clause in the list or in the web panel without a base table.

Then, in the web panel you could define an SDT collection variable:



Name	Type	Is Col...	Description
Variables			
• Standard Variables			
• ReservedSlots	ReservedSlot	<input checked="" type="checkbox"/>	Reserved Slots

Insert it in the web form, and GeneXus will automatically display it in a grid:



It will also automatically display with the content that the collection variable has before the Load event is executed.

Therefore, a good time to load it will be the Refresh event (if we do it at the Start event, it will run only once. On the other hand, the Refresh event will be executed every time it is refreshed, and since this data could vary anytime (it will be enough for new reservations to be created), perhaps the Refresh event is preferable).

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

Note that here the Data Provider is invoked so that it returns the loaded collection, and then the collection is sorted in descending order by ReservedSlotAmount.

Since after the Refresh event the grid load will be executed, this will resolve the requirement.

## SOLUTION WITH DYNAMIC TRANSACTION

Other students, however, will think of an equivalent solution, which is very similar but may be better: Instead of defining an SDT collection and a Data Provider to load it, why not define a dynamic transaction?

ReservedSlot * X				
Structure Web Form Win Form Rules Events Variables Help Documentation Patterns				
Name	Type	Description	Formula	Nullable
ReservedSlot	ReservedSlot	Reserved Slot		
ReservedSlotId	Id	Reserved Slot Id		No
ReservedSlotDescription	Description	Reserved Slot Description		No
ReservedSlotAmount	Numeric(4.0)	Reserved Slot Amount		No

#### ▼ Data

Data Provider	True
Used to	Retrieve data
Update Policy	Read Only

He will have to define the Data Provider in the same way as the other students did with the SDT collection:

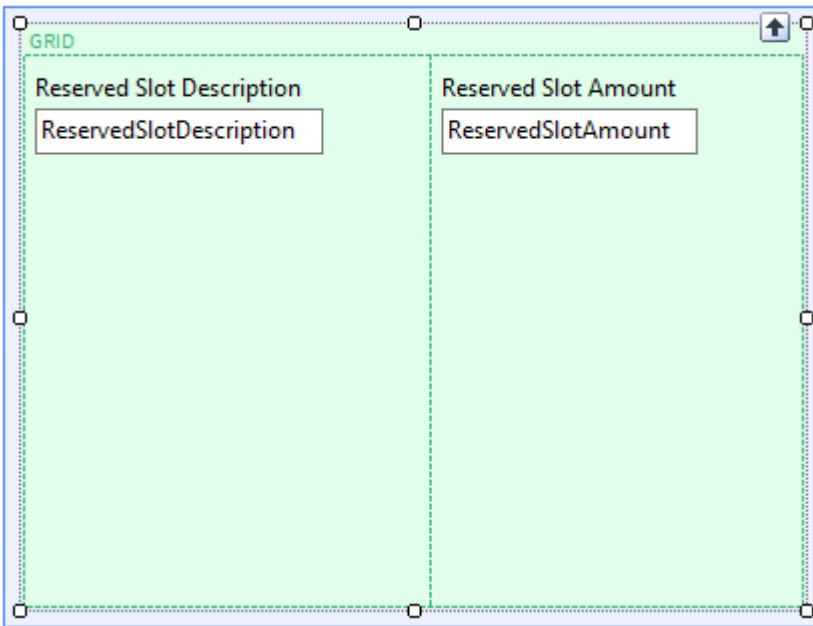
```

ReservedSlot_DataProvider * X
Source * Rules Variables Help Documentation

1 ReservedSlotCollection{
2     ReservedSlot from Reservation.Slot
3     unique SlotId
4     {
5         ReservedSlotId = SlotId
6         ReservedSlotDescription = SlotDescription
7         ReservedSlotAmount = count(ReservationSlotTimeFrom)
8     }
9 }
10

```

And in the web panel load the grid as if it were an ordinary transaction and not a dynamic one:



With the order defined in the grid properties:

Properties	
General Class	
Grid: Grid1	
Control Name	Grid1
Collection	
Base Trn	
Order	(ReservedSlotAmount)
Conditions	
Unique	
Data Selector	(none)

Note that nothing else needs to be programmed here.

The student came up with this solution. You congratulate him, and ask the following question: In the dynamic transaction, instead of using the already existing attributes SlotId and SlotDescription, why did you define two new attributes? I mean, why not define the transaction this way?

ReservedSlot * X				
Structure * Web Form Win Form Rules Events Variables Help Documentation Patterns				
Name	Type	Description	Formula	Nullable
ReservedSlot	ReservedSlot2	Reserved Slot2		
SlotId	Id	Slot Id		No
SlotDescription	Description	Slot Description		No
ReservedSlotAmount	Numeric(4,0)	Reserved Slot Amount		No

```

ReservedSlot_DataProvider * X
Source Rules Variables Help Documentation






1 ReservedSlotCollection{
2     ReservedSlot from Reservation.Slot
3     unique SlotId
4     {
5         SlotId
6         SlotDescription
7         ReservedSlotAmount = count(ReservationSlotTimeFrom)
8     }
9 }

```

So that in this way the slot of the dynamic transaction is effectively understood as a slot, in any other context.

If you make the student look at the navigation list, he'll understand why it's not possible:



Transaction ReservedSlot Navigation Report			
<b>Name:</b>  <a href="#">ReservedSlot</a> <b>Description:</b> Reserved Slot2		<b>Environment:</b>  Default (C#) <b>Spec. Version:</b>  16_0_4-134138 <b>Form Class:</b> HTML <b>Program Name:</b> ReservedSlot	
LEVELS			
Level Slot			
READ <a href="#">Slot</a> WHERE <a href="#">Slot</a> . <a href="#">SlotId</a> = <a href="#">SlotId</a> INTO <a href="#">SlotDescription</a> <a href="#">ReservedSlotAmount</a>  Referential integrity controls on delete: <ul style="list-style-type: none"> <li>• <a href="#">ReservationSlot</a> ( <a href="#">SlotId</a> )</li> </ul>			
Prompts			
<b>Table</b>  <a href="#">Slot</a>	<b>Program</b>  <a href="#">Gx0040</a>	<b>In Parameters</b>	<b>Out Parameters</b> <a href="#">SlotId</a>

It would be a parallel transaction to the Slot transaction, as they would share the same primary key. This is not possible. GeneXus, when trying to specify will show you the following error:

Table Slot specification			⌵
Table name: <a href="#">Slot</a>			
Problems found on table <a href="#">Slot</a> structure. It will not be created/reorganized. Slot needs conversion			
Errors			⌵
✖ <a href="#">rgz0043</a> Table ' <a href="#">Slot</a> ' associated to dynamic transaction cannot have parallel transactions ().			
Warnings			⌵
⚠ <a href="#">rgz0007</a> Attribute <a href="#">ReservedSlotAmount</a> does not allow nulls and does not have an Initial Value. An empty default value will be used.			
Table Structure			⌵
Attribute	Definition	Previous values Takes value from	
🔑 <a href="#">SlotId</a>	Numeric (4), Not null, Autonumber	<a href="#">Slot</a> <a href="#">SlotId</a>	
<a href="#">SlotDescription</a>	Character (20), Not null, NLS	<a href="#">Slot</a>	

What would happen if SlotId wasn't the primary key to the dynamic transaction, but only a foreign key? Is it possible to use the attribute in this case? Think about it and try to confirm your hypothesis.

The answer is yes, it is possible. Think of all the edges of the problem. For example, what happens if you want to remove a slot from the Slot table that has one related through this dynamic transaction? Is this referential integrity check carried out to avoid the deletion?

## QUESTION 22 – UNIQUE FOR AGGREGATION

Suppose you now need to list for each customer with slot reservations after a given date, their name and total number of slots booked from that date.

A student comes up with the following solution:

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

You mention that something is incorrectly programmed. Then, another student says the solution is not correct because the table navigated by the Count formula is not Reservation.

What will you answer?

### Answer

It is good practice to recommend students to analyze the navigation list, which already gives us clues to solve the problems.

The student who answered that the problem was that the Count formula did not navigate Reservation clearly thought that only in that scenario can a For Each command with Unique be used to make aggregations. This example shows that it isn't so.

In this case, the Count formula will navigate ReservationSlot, but filtering by the CustomerId (unique) of the For Each command.

**LEVELS**

**For Each Reservation (Line: 7)**

**Order:** CustomerId  
Index: IRESERVATION1

**Unique:** CustomerId

**Navigation filters:** Start from: FirstRecord  
Loop while: NotEndOfTable

**Constraints:** ReservationDate >= &fromDate

**Join location:** Server

=Reservation ( ReservationId ) INTO ReservationDate CustomerId

=Customer ( CustomerId ) INTO CustomerName

=count( ReservationSlotTimeFrom ) navigation ( CustomerId )

**Formulas**

**Navigation to evaluate:** count( ReservationSlotTimeFrom )

**Given:** CustomerId  
**Index:** IRESERVATION  
**Group by:** CustomerId

=ReservationSlot

=Reservation ( ReservationId )

That is, you can have the same CustomerId in many reservations (after the date &fromDate), but only one of these reservations will be used in the For Each command (through Unique) and then, within its body, the formula will be executed filtering through that CustomerId. All this is correct.

The error is quite simple: the records being counted are those of the ReservationSlot table corresponding to a customer reservation, regardless if they were made before &fromDate.

The right thing would be then:

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

Note that the Where clause filters the records used in the body of the For Each command, while the formula condition filters the records that will be counted. They are both necessary.



[www.genexus.com](http://www.genexus.com)

Copyright GeneXus S.A. 1988-2024.

All rights reserved. This document may not be reproduced by any means without the express permission of GeneXus S.A. The information contained herein is intended for personal use only.

Registered Trademarks:

GeneXus is trademark or registered trademark of GeneXus S.A. All other trademarks mentioned herein are the property of their respective owners.