Formulas

An encompassing look

In other videos, we have explained what global formulas and inline formulas are by analyzing the different types according to their navigation, including horizontal formulas, aggregation formulas, compound formulas and their corresponding use cases.

In this video we will try to provide a broader look, not a detailed one, to discover when it is convenient to use formulas according to their type and when we can use an alternative solution; also, in which cases we have restrictions and how we can lift them, what is the cost of adding redundancy and other observations that will help us integrate the concepts on this topic.

If you have any doubts about the concepts of the formulas on this video, or you want to review them before watching this synthesis, we suggest you watch the videos on formulas of the Advanced GeneXus Course.

Global formulas vs. Inline formulas (local formulas)

A **global formula**, also known as a “formula attribute,” is a calculation assigned to an attribute in a transaction structure.
They are called “global” because when defining the calculation in an attribute, its definition remains at the knowledge base level, and therefore all objects will have access to the attribute with that calculation.

Once we associate a calculation with an attribute, it will not be stored as a field of a table in the database, and for this reason they are also called “virtual attributes.” However, the global formula attribute is associated with the table to which it would have belonged if it had not been defined as a formula. This associated table represents the context of the formula; that is, at the moment the formula is triggered, it is positioned in a certain record of that table.

Only attributes can be defined as global formulas and not variables, since attributes have global scope in the knowledge base, while the scope of variables is limited to the object where they were defined and cannot be accessed from another object.

If we use a formula in a calculation that we implement when we write code in an object, such as in the source of a procedure or events of a panel or web panel, or in a data provider, or in the Conditions tab of an object, we are defining an **inline formula**.

Since we can write inline formulas in any object where we can write code, most of the time we assign this calculation to a variable, as we could only assign the value returned by a formula to an attribute if we are updating the table through a For Each in a procedure object.

Since variables have local scope to the object, inline formulas are also known as local formulas.

In the case of data providers, the elements on the left side of the assignments are part of a hierarchical structure that only has value within the data provider, so their scope is also local. The same applies if inline formulas are used in the Conditions tab of an object.

When to use horizontal formulas and when to use aggregation formulas

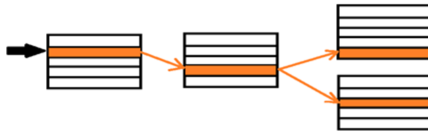
Name	Type	Description	Formula
Flight	Flight	Flight	
FlightId	Id	Flight Id	

Formula Editor

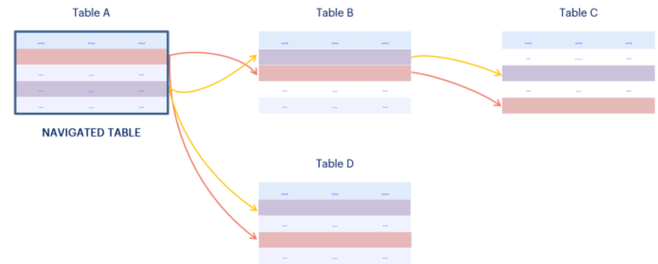
```
Occupancy.Low IF FlightCapacity < 5;
Occupancy.Medium IF FlightCapacity > 5 and FlightCapacity < 8;
Occupancy.High OTHERWISE
```

OK Cancel

Name	Type	Description	Formula
FlightCapacity	Numeric(4.0)	Flight Capacity	count(FlightSeatLocation)
FlightOccupancy	Occupancy	Flight Occupancy	Occupancy.Low IF FlightCapacity < 5; Occup...
Seat	Seat	Seat	
FlightSeatId	Id	Flight Seat Id	
FlightSeatChar	SeatChar	Flight Seat Char	
FlightSeatLocation	Location	Flight Seat Location	



Attribute =
expression$_1$ **if condition$_1$**;
expression$_2$ **if condition$_2$**;
...
expression$_n$ **if condition$_n$**;
expression$_0$ **otherwise**;



Aggregate formulas:

- Count
- Sum
- Average
- Max
- Min
- Find

The navigation of a formula determines whether it is a horizontal or aggregation formula.

When we have a formula that navigates a single record of a table (possibly accessing attributes of its extended table), we say it is a **horizontal formula**.

If the formula accesses many records of a table, then it is an **aggregation formula**.

Therefore, if we want to retrieve a value from a calculation that can be obtained with data from a single record and the associated records of its extended table, then we write a horizontal formula. This type of formula allows us to assign different values depending on certain conditions, and to take a default value in case none of the established conditions are met.

Horizontal formulas are essentially global (defined in the structure of a transaction) since when we write a calculation involving a single record in the code of an object, this calculation is part of the procedural (imperative) implementation, possibly within some conditional or repetitive structure and does not differ from the rest of the implemented code, so it does not make sense to talk about horizontal inline formulas.

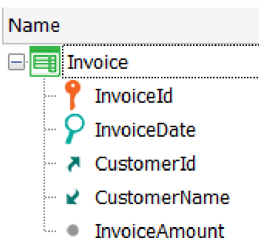
If, on the other hand, we want a value that depends on the search and/or processing of multiple records (and their associated records of the extended table), then we will use some of the aggregation formulas, such as count, sum, max, min, find, or average.

Aggregation formulas can be global (assigned to attributes at the transaction structure level, which indicates that those attributes always take the result of a calculation) or local (inline). They can be assigned to attributes or variables; to elements of a data provider; as part of filters (such as tab conditions or directly in For each commands, groups of Data Providers, etc.) that are evaluated at runtime.

Optional parameters in aggregation formulas

AggregateFormula(AggregateExpression, AggregateCondition, DefaultValue, ReturnedValue) if TriggeringCondition

optional — optional — optional — optional



Sum(InvoiceAmount) : Summarizes InvoiceAmount from all records of the INVOICE table.

Sum(InvoiceAmount. InvoiceDate=&today) : Summarizes the InvoiceAmount values from the invoices issued today.

Max(InvoiceAmount. InvoiceDate=&today, 0, InvoiceAmount) : Retrieves the maximum value of InvoiceAmount from the invoices issued today. If there are no invoices issued today, the returned value will be 0.

Max(InvoiceAmount. InvoiceDate=&today, 0, InvoiceAmount) if CustomerId = 4 : Only for issues of customer with Id=4, it retrieves the maximum value of InvoiceAmount from the invoices issued today. If there are no invoices issued today, the returned value will be 0.

Let's look at the parameters of an aggregation formula and their roles.

Aggregation formulas have a first mandatory parameter that is the aggregation expression, which establishes the table that will be run through by the formula.
It is recommended that this attribute not be a key, to avoid possible ambiguities when GeneXus determines the table to be navigated, especially if in the other parts of the formula there are no other attributes that allow determining the uniqueness of the table; for example, when the formula has no other parameters than the first one.
In particular, this attribute can be a formula attribute; it doesn't have to be a stored attribute, since every formula attribute has an associated table.

In addition to the first mandatory parameter, an aggregation formula can have other parameters, all of them optional.

The first one after the aggregation expression is the aggregation condition; that is, the condition that the records to be counted, summed, averaged, etc. must meet.

The next parameter is the default value returned by the formula if it doesn't find any record that meets the aggregation condition.

The last parameter will be the attribute containing the value to be returned in case records have been processed.

After the “if,” the triggering condition states what condition must be met for the formula to be executed. This condition is a logical expression as complex as necessary.

Restrictions on global aggregation formulas

Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
InvoiceAmount	Amount	Invoice Amount		No
InvoiceAuxDate	Date	Invoice Aux Date	InvoiceDate	
InvoiceAuxCustomerId	Id	Invoice Aux Customer Id	CustomerId	
InvoiceBeforeDate	Date	Invoice Before Date	max(InvoiceDate, InvoiceDate<InvoiceAuxDate ...	

Formula Editor

```
max(InvoiceDate, InvoiceDate<InvoiceAuxDate and CustomerId=InvoiceAuxCustomerId, , InvoiceDate)
```

OK Cancel

Since the attribute of the aggregation expression is arbitrarily chosen, the formula could navigate any table in the database. Therefore, we could make the table run through to be the same as the table associated with the formula attribute.

Let's analyze what would happen in this case.

Consider a case where given an invoice from a customer, we want to find the date of the previous invoice from the same customer. To solve this, we will use a Max formula similar to the one we defined before.
We realize that we must distinguish the attributes of the associated table record from the attributes corresponding to the records of the navigated table where the search will be made.
To do so, we add the following auxiliary attributes: InvoiceAuxCustomerId defined as a horizontal formula that takes the CustomerId value and the InvoiceAuxDate attribute that takes the InvoiceDate value.

Note that we are trying to define a formula that will navigate the same table associated with the formula. The formula is defined in the Invoice transaction, so its associated table is INVOICE and since the aggregation expression attribute is InvoiceDate, the table navigated is also INVOICE.

As we saw before, the associated table represents the context of the formula, so when the formula is triggered, it is positioned on a certain record of that table. Therefore, the formula will be filtered by the identifier of the record where it is positioned and will only navigate a single record!

That is to say, we will not be able to have a global aggregate formula that navigates the same table where it is defined, because it will not be able to run through it as we need in this case to find the maximum.

If we need this functionality, we could define the formula by invoking a procedure object that returns the calculation, as we will see next.

Global formulas that call procedures

Name	Type	Description	Formula	Nullable
Invoice	Invoice	Invoice		
InvoiceId	Id	Invoice Id		No
InvoiceDate	Date	Invoice Date		No
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	Character(20)	Customer Name		
CustomerTotalPurchases	Amount	Customer Total Purchases		
InvoiceAmount	Amount	Invoice Amount		No
InvoiceBeforeDate	Date	Invoice Before Date	GetInvoiceBeforeDate(InvoiceId)	

GetInvoiceBeforeDate* — Source * | Layout | Rules | Conditions | Variables | Help | Documentation

```
1 Parm(in:&InvoiceId, out:&InvoiceBeforeDate);
2
```

GetInvoiceBeforeDate — Source | Layout | Rules | Conditions | Variables | Help | Documentation

'GetPreviousInvoiceDate'

```
1  For each Invoice
2       Where InvoiceId=&InvoiceId
3       &InvoiceDate = InvoiceDate
4       &CustomerId = CustomerId
5       Do 'GetPreviousInvoiceDate'
6  Endfor
7
8  Sub 'GetPreviousInvoiceDate'
9       For each Invoice order (InvoiceDate)
10           Where CustomerId = &CustomerId
11           Where InvoiceDate < &InvoiceDate
12           &InvoiceBeforeDate = InvoiceDate
13           Exit
14      Endfor
15 EndSub
```

We will create a procedure that receives an invoice from a customer and returns the date of the previous invoice from the same customer.

So we define the InvoiceBeforeDate attribute as a formula and in the form editor we invoke the GetInvoiceBeforeDate procedure, passing as parameter the InvoiceId of the invoice of interest.

In the source we implement a For Each that runs through the INVOICE table filtering by the InvoiceId received by parameter and retrieve the date and customer of the invoice received. Next, we invoke a subroutine that runs through the invoice table again, sorted by invoice in descending order, and we get the invoice of the same customer whose date is the highest possible, but lower than the invoice received by parameter.

In this way, we are doing what we were trying to do with the max formula, which is to maximize the date of the invoice from the same customer but which is lower than the one of interest; in other words, the date of the previous invoice from the same customer.

The fact that we can define a formula that invokes a procedure to perform the necessary calculation opens many possibilities because this calculation can be as complex as we want, and then we can simply use the formula attribute from any object in our knowledge base.

How to define a formula attribute as redundant

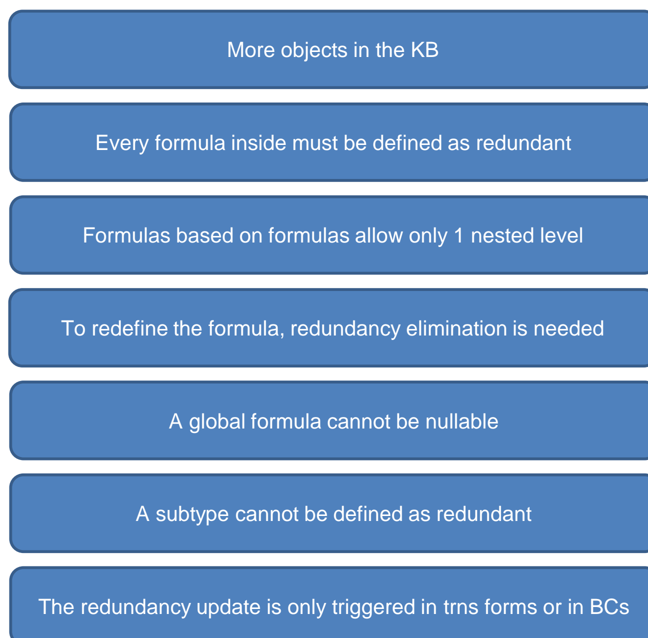
Name	Type	Description	Formula
Flight	Flight	Flight	
FlightId	Id	Flight Id	
FlightDepartureAirportId	Id	Flight Departure Airport Id	
FlightDepartureAirportName	Name	Flight Departure Airport Name	
FlightDepartureCountryId	Id	Flight Departure Country Id	
FlightDepartureCountryName	Name	Flight Departure Country Name	
FlightDepartureCityId	Id	Flight Departure City Id	
FlightDepartureCityName	Name	Flight Departure City Name	
FlightArrivalAirportId	Id	Flight Arrival Airport Id	
FlightArrivalAirportName	Name	Flight Arrival Airport Name	
FlightArrivalCountryId	Id	Flight Arrival Country Id	
FlightArrivalCountryName	Name	Flight Arrival Country Name	
FlightArrivalCityId	Id	Flight Arrival City Id	
FlightArrivalCityName	Name	Flight Arrival City Name	
FlightPrice	Price	Flight Price	
FlightDiscountPercentage	Percentage	Flight Discount Percentage	
AirlineId	Id	Airline Id	
AirlineName	Name	Airline Name	
AirlineDiscountPercentage	Percentage	Airline Discount Percentage	
FlightFinalPrice	Price	Flight Final Price	FlightPrice * (1-AirlineDiscountPercentage/100) ...
FlightCapacity	Numeric(4.0)	Flight Capacity	count(FlightSeatLocation)
Seat	Seat	Seat	
FlightSeatId	Id	Flight Seat Id	
FlightSeatChar	SeatChar	Flight Seat Char	
FlightSeatLocation	Location	Flight Seat Location	

Although redundant formulas are discussed extensively in another video, here we will review some concepts.

In some cases where the formula is triggered many times, for performance reasons it is convenient to define the global formula attribute as redundant. This will cause the attribute to be stored in the database; therefore, the time it will take to retrieve the value will be less than the time needed to perform the calculations.

To define an attribute as redundant, we do it from the transaction structure: right-click on the column bar, click on Column Chooser, and add the Redundant column by dragging it to the column bar.
Then in the Redundant column we select the checkbox to define the attribute as redundant. A “+” sign is added to the formula symbol to indicate that it is redundant.

Implications of defining a global formula as redundant



Defining an attribute as redundant has certain costs that must be considered.

First of all, for the database value to always be up to date, GeneXus will create procedure objects that will be triggered every time the data involved in the formula calculation changes. This implies that, for each formula attribute that we define as redundant, the number of objects in the knowledge base will increase.

In addition, if the formula attribute that we want to be redundant was defined based on other attributes that are also formulas, we must also define those other attributes as redundant.

Another limitation is that aggregation formulas that are made redundant, defined based on attributes that are also redundant formulas, allow only one nesting level.
If this limit is exceeded, its redundancies will not be correctly maintained.

To change the definition of a redundant formula, you must first remove the redundancy, make the change, and define the formula as redundant again.

Global formulas defined as redundant do not support null values, so we cannot set the Nullable property to Yes.

A subtype that is a formula cannot be defined as redundant.

Finally, the procedures in charge of updating the stored value will be fired only when the record is edited through the transaction form or through a business component of the transaction. This implies that if the data involved in the formula is being changed from a procedure object, the update procedures will not be triggered automatically and we will have to force this update explicitly, invoking a utility created by GeneXus for this purpose.

In short, we must think carefully when deciding to define a formula attribute as redundant, since these restrictions can make the implementation unfeasible.

Alternative solutions to a redundant formula

Vs.

```
1 Add(CustomerTripMiles, CustomerTotalMiles);
```

Customer* — Structure | Web Layout | Rules | Events | Variables | Help | Documentation | Patterns

Name	Type	Formula	Redundant	Nullable
Customer	Customer			
CustomerId	Numeric(4.0)			No
CustomerName	Character(20)			No
CustomerLastName	Character(20)			No
CustomerAddress	Address, GeneXus			No
CustomerPhone	Phone, GeneXus			No
CustomerEmail	Email, GeneXus			No
CustomerAddedDate	Date			No
CustomerTotalMiles	Numeric(4.0)	sum(CustomerTripMiles)	☐	
Trip	Trip			

Stored attribute

Customer — Structure | Web Form | Win Form | Rules | Events | Variables | Help | Documentation

Name	Type	Formula
Customer	Customer	
CustomerId	Id	
CustomerName	Name	
CustomerLastName	Name	
CustomerAddress	Address, GeneXus	
CustomerPhone	Phone, GeneXus	
CustomerEmail	Email, GeneXus	
CustomerAddedDate	Date	
CustomerTotalMiles	Numeric(4.0)	
CustomerIsVIP	Boolean	
Trip	Trip	

Stored attribute

When an attribute value can be obtained with a calculation, in general we define it as a formula in the transaction structure and this attribute is no longer stored in the database. If we need the attribute to always be stored, we can define it as redundant, so GeneXus automatically creates procedures in charge of updating its value and storing it in the database.

However, we could also assign the calculation to the attribute by means of a rule. The attribute doesn't disappear from the table just by assigning a value to it, so it doesn't become a virtual attribute, but continues to be a stored attribute.

Therefore, in terms of the update and the attribute being stored, updating the attribute by means of a rule and defining it as a redundant formula could be considered equivalent solutions.

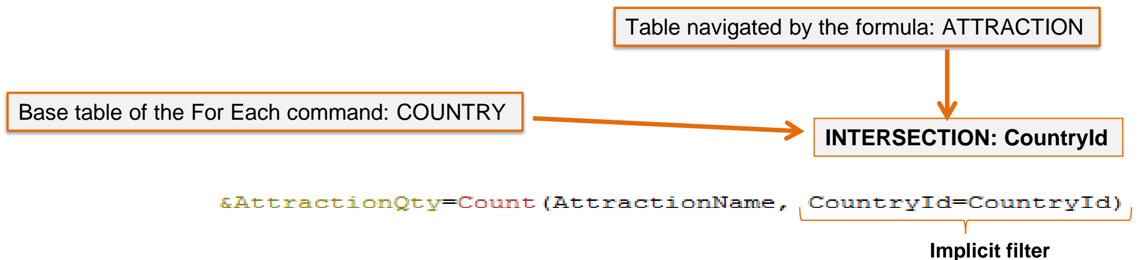
As an additional advantage, with the rule the attribute will still be editable in the transaction form, but with a drawback because the rule can't be triggered on demand. Therefore, it is not possible to force the attribute to be updated, which can be done with a redundant formula attribute.

The timing of when to use one solution or the other depends ultimately on how we will use the attribute in our application.

Inline formulas in a For Each: implicit filters and edge cases

```
Print header
For each Country
    Where Count(AttractionName) > 2
    &AttractionQty = Count(AttractionName)
    Print Country
Endfor
```

Name	Type	Is Collection	Description
Variables			
Standard Variables			
AttractionQty	Numeric(4.0)	☐	Attraction Qty



Let's look at the case of inline formulas defined in the body of a For each command.

As we have already seen, the formulas determine the table to navigate, by the attribute or attributes in brackets. In this case, we have included the AttractionName attribute, so the table to be navigated by the formula is ATTRACTION.

In the example we don't want to count all those in the ATTRACTION table, but those corresponding to the country where we are positioned in each iteration of the For each that will run through the COUNTRY table.

As the formula is defined within a For each command, it is in a context in which a table is being run through; in this case, the table of countries. Note that there is an attribute in common between both navigations—CountryId. That's why GeneXus determines an implicit filter by the attribute in common, so that for every country found by the For each, only the attractions from that country are counted.

Another requirement is that only countries that have more than two tourist attractions are listed.
To solve this, we can use a count formula as part of the filter expression in the Where clause of the For Each.

This count formula, defined as the previous one, will return the number of attractions of the country in which the For Each is positioned in each iteration (due to the implicit filter mentioned before); we use that to filter those countries for which the number of attractions is greater than two.

Inline formulas in a For Each: implicit filters and edge cases

Name	Type
Customer	Customer
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
CustomerAddress	Address, GeneXus
CustomerPhone	Phone, GeneXus
CustomerEmail	Email, GeneXus
CustomerAddedDate	Date

Name	Type
Trip	Trip
TripId	Id
TripDate	Date
TripDescription	VarChar(1K)
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
Attraction	Attraction
AttractionId	Id
AttractionName	Name
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name

LastTripInformation — Source | Layout | Rules | Conditions | Variables | Help | Documentation

```
1 Parm(in: &CustomerId, in: &SearchDate);
2 Output_file('LastTripInformation', 'PDF');
```

LastTripInformation — Source | Layout | Rules | Conditions | Variables | Help | Documentation

```
1 For each Trip
2     Where TripId = Max(TripDate, TripDate <= &SearchDate and CustomerId=&CustomerId, 0, TripId)
3     print LastTrip
4 Endfor
```

Base table: TRIP

Navigated table: TRIP

Now let's see a particular case, in which the table navigated by the formula matches the base table of the For Each.

Suppose that given a customer who has many trips, you want to retrieve the data of a trip made immediately prior to a given date; that is, the last trip before that date. The procedure receives by parameter the identifier of the customer who wants the information and the search date.

In the source of the procedure there is a For each that runs through the Trip table and the where will filter by the TripId returned by the Max formula. Then the data corresponding to that trip will be printed.

If we analyze what we have implemented, the For Each will iterate on the TRIP table and set that context for the Max formula.
The Max formula will also iterate on the TRIP table, but due to the context, an automatic filter will be set because both tables are related. In this case, it is the same table, so the formula will be filtered by the TripId that is positioned in the For Each.
Therefore, it will always return that TripId where the For Each is located, so the formula will never run through the TRIP table since it will only access one record.

Inline formulas in a For Each: implicit filters and edge cases

Name	Type
Customer	Customer
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
CustomerAddress	Address, GeneXus
CustomerPhone	Phone, GeneXus
CustomerEmail	Email, GeneXus
CustomerAddedDate	Date

Name	Type
Trip	Trip
TripId	Id
TripDate	Date
TripDescription	VarChar(1K)
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
Attraction	Attraction
AttractionId	Id
AttractionName	Name
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name

LastTripInformation — Source | Layout | Rules | Conditions | Variables | Help | Documentatio

```
1 Parm(in: &CustomerId, in: &SearchDate);
2 Output_file('LastTripInformation', 'PDF');
```

LastTripInformation — Source | Layout | Rules | Conditions | Variables | Help | Documentation

Subroutines

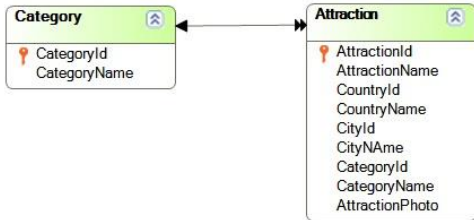
```
1 &TripId = Max(TripDate, TripDate <= &SearchDate and CustomerId=&CustomerId, 0, TripId)
2 For each Trip
3       Where TripId = &TripId
4       print LastTrip
5 Endfor
6
7
```

The solution in this case is to first run the formula to search for the trip identifier that meets the desired conditions, and then filter the For Each by that identifier value.

In this example, if we define an inline formula within a context (the base table of the For each), the same happens as we saw in the example of the global formula, whose associated table matched the navigated table, and the associated table provided the context.

Unlike horizontal formulas that need a context (since they can only use attributes of the associated table and its extended table), aggregation formulas do not need it, but if they are defined within a context, the context will act as a filter for the formula.

Inline formulas in a For Each: implicit filters and edge cases



Returns the list (without repeating elements) of all categories that have attractions, each with the corresponding number of registered attractions.

```
Print Title
For each Attraction
     Unique CategoryId
     &Quantity = Count(AttractionId)
     Print Categories
Endfor
```

Base table of the For each: ATTRACTION

Table navigated by the formula: ATTRACTION

Now let's see another case of an inline formula defined inside a For Each, where the table navigated by the formula matches the base table of the For Each. Note that, in both cases, the table is ATTRACTION.

However, in this example we have added a Unique clause by CategoryId and because of that, GeneXus will group the attractions by category, both in the For each and in the formula. Therefore, the Count formula will add from the context an implicit condition in its evaluation: it will count all the attractions for the attribute declared in the unique clause.

It’s as if it were a control break, breaking through CategoryId.

Thanks to this additional condition provided by the Unique clause, GeneXus can solve the problem that the table browsed by the formula matches the base table of the For each, since only the attractions of the category given in the Unique clause will be counted.

Compound formulas

```
Attribute  =   Max(...) if condition1;
               (2 * attrX) + 100  if condition2;
               Sum(attrY)  otherwise

Attribute = procedure(...) if condition1;
             Min(...)  if condition2;
             10   if condition3

Attribute  =  2 + Count(…) *   Sum(…) if condition;
              Attr1 + Attr2 * Attr3  otherwise


Attribute  =  Count(…) if condition1;
              Sum(…) if condition2;
              Find(…) if condition3;
```

In GeneXus, we can define complex formulas using compound formulas.

Compound formulas are those that integrate several conditional aggregate formulas, and may also contain horizontal expressions with triggering conditions.

Conditions are any valid logical expression, which can contain attributes belonging to the extended table of the table associated with the attribute being defined as a formula.

The first condition, when evaluated to True, will cause the result of the formula to be that of the expression to the left of that condition and the other expressions will not continue to be evaluated.

Example

Name	Type	Description	Formula	Nullable
Flight	Flight	Flight		
FlightId	Id	Flight Id		No
FlightDepartureAirportId	Id	Flight Departure Airport Id		No
FlightDepartureAirportName	Name	Flight Departure Airport Name		
FlightDepartureCountryId	Id	Flight Departure Country Id		
FlightDepartureCountryName	Name	Flight Departure Country Name		
FlightDepartureCityId	Id	Flight Departure City Id		
FlightDepartureCityName	Name	Flight Departure City Name		
FlightArrivalAirportId	Id	Flight Arrival Airport Id		No
FlightArrivalAirportName	Name	Flight Arrival Airport Name		
FlightArrivalCountryId	Id	Flight Arrival Country Id		
FlightFinalPrice	Price	Flight Final Price	FlightPrice * (1-AirlineDiscountPerce...	
FlightCapacity	Numeric(4.0)	Flight Capacity	count(FlightSeatLocation)	
FlightOccupancy	Character(1)	Flight Occupancy	Occupancy.Low IF count(FlightSe...	
Seat	Seat	Seat		
FlightSeatId	Id	Flight Seat Id		No
FlightSeatChar	SeatChar	Flight Seat Char		No
FlightSeatLocation	Location	Flight Seat Location		No

Formula Editor

```
Occupancy.Low IF count(FlightSeatLocation) < 5;
Occupancy.Medium IF count(FlightSeatLocation) >5 and count(FlightSeatLocation) < 8;
Occupancy.High OTHERWISE
```

OK Cancel

Let's see an example of this type of compound formulas in the travel agency reality.

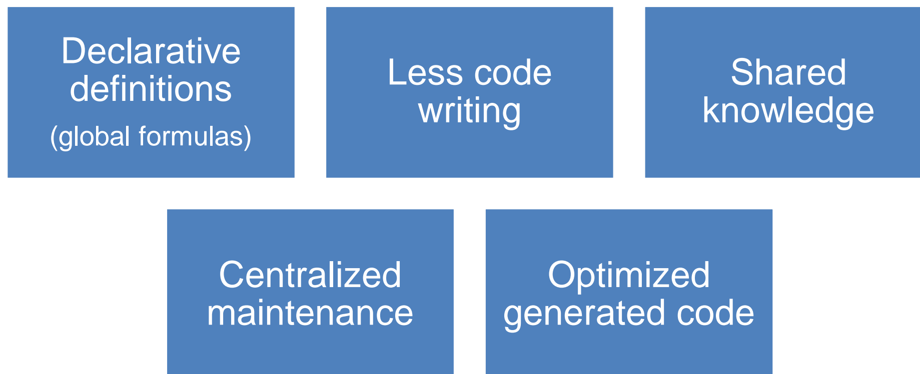
Here we see that the FlightOccupancy attribute was defined based on horizontal expressions that assign the corresponding value of the Occupancy domain (Low, Medium or High), depending on the number of seats on the flight, which are calculated with aggregate count formulas.

In particular, in this case, we could have replaced the aggregate formulas with the FlightCapacity attribute, but it is perfectly valid to leave it as it is defined.

In this implementation, the structure is that of a horizontal formula and the aggregate ones were included in the triggering conditions.

Compound formulas provide great flexibility to define calculations, and a large number of situations can be modeled.

Conclusions



After this conceptual review of the use of formulas in GeneXus, we can say they are very useful in many cases, and essentially provide the following advantages:

- Declarative definition instead of procedural code for global formulas.
- They allow us to save code, especially in the functionalities of aggregate formulas that process many records. In turn, this frees us from having to iterate on the records and implement the logic by code to count, add, maximize, etc.
- They are a way to share knowledge, for example, in the case of formula attributes that can be used in any object of the knowledge base.
- Maintenance is centralized, since in the case of a global formula, we change the definition in a single place—the transaction structure where the attribute was defined.
- Defining formulas is even better than writing procedures and invoking them. When a formula is defined, GeneXus is aware of its definition and is able to generate optimized statements by combining the formula query with the query in which the formula is present.

In summary, formulas make application development much easier and it is strongly recommended to learn how to use them.

To learn more about this topic, visit the wiki to continue reading.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications