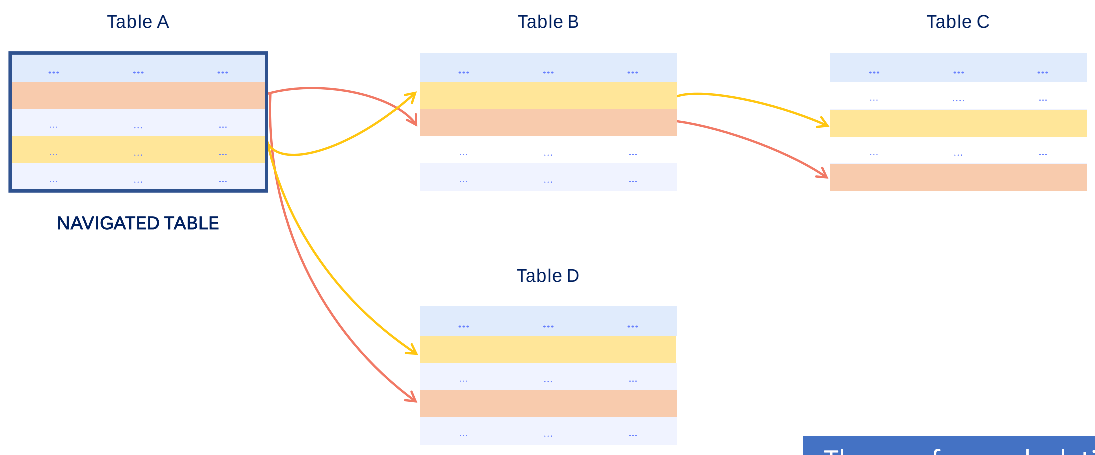
Aggregate Formulas

Review of aggregate formulas



-They perform calculations and searches over many records, in any table of the model and its extended table.

First, let's review some concepts.

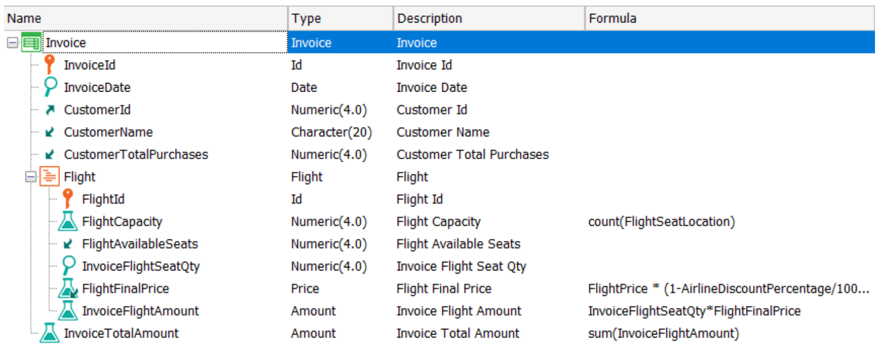
Aggregate formulas allow you to perform calculations or searches involving many records in a table and related values from the extended table of the table run through.

GeneXus

Review of aggregate formulas

Aggregate formulas:

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



Name	Type	Description	Formula
Invoice	Invoice	Invoice	
InvoiceId	Id	Invoice Id	
InvoiceDate	Date	Invoice Date	
CustomerId	Numeric(4.0)	Customer Id	
CustomerName	Character(20)	Customer Name	
CustomerTotalPurchases	Numeric(4.0)	Customer Total Purchases	
Flight	Flight	Flight	
FlightId	Id	Flight Id	
FlightCapacity	Numeric(4.0)	Flight Capacity	count(FlightSeatLocation)
FlightAvailableSeats	Numeric(4.0)	Flight Available Seats	
InvoiceFlightSeatQty	Numeric(4.0)	Invoice Flight Seat Qty	
FlightFinalPrice	Price	Flight Final Price	FlightPrice * (1-AirlineDiscountPercentage/100...
InvoiceFlightAmount	Amount	Invoice Flight Amount	InvoiceFlightSeatQty*FlightFinalPrice
InvoiceTotalAmount	Amount	Invoice Total Amount	sum(InvoiceFlightAmount)

Syntax:

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

optional optional optional optional

Aggregate formulas don't need a context, but if there is one you can filter the result.

These formulas allow us to count (with Count), sum (with Sum) or average (with Average) several records, perform searches such as finding the maximum (with Max) or minimum (with Min) value of an attribute in a set of records, or given many records in a table, find a value of an attribute (using Find) whose record is the first one found that meets certain conditions.

In the syntax, the aggregate expression will be the one searched for, maximized, minimized, summed or averaged, among the records that meet the aggregation condition. It can contain attributes (including formula attributes), constants and variables (user-created variables are only allowed in inline formulas). Only for the Count case, it is not an expression but an attribute. For Sum and Average, the result of the aggregation expression must be a numerical value.

The aggregation condition is the condition that records must verify to be considered in the aggregation. It can contain attributes, constants and variables (user variables only in inline formulas). It is optional and may not be included.

The default value will be returned when no record is found on which to perform the aggregation; it may be because none meets the aggregation condition, because it doesn't meet the implicit aggregation condition, or because the table is empty. It is a constant and is also optional.

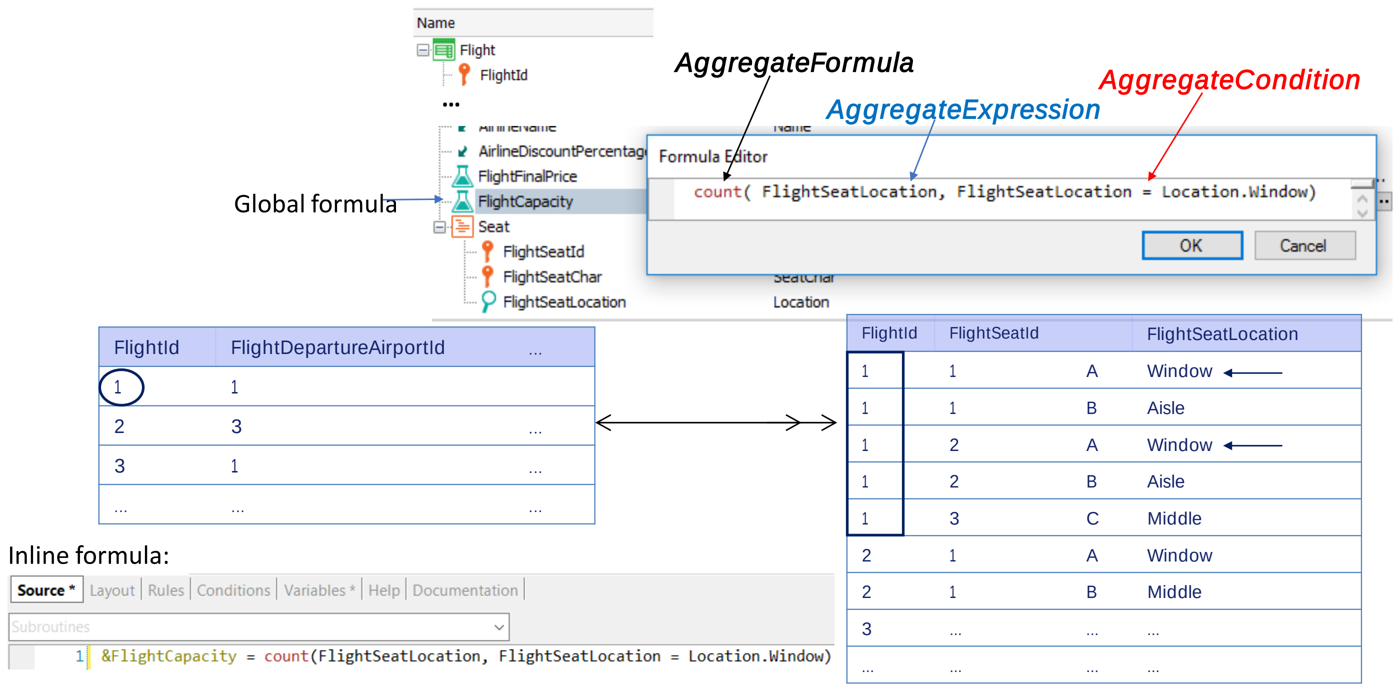
Review of aggregate formulas (continued)

The returned value is the attribute whose value is returned by the formula when it finds records that meet the aggregation condition. While the returned value may be a value (as in the case where the default value is returned), it is usually an attribute. Its inclusion is optional and only some formulas carry this fourth parameter.

The triggering condition is the one that determines that the formula is calculated with the aggregation expression that precedes it.
It is optional and the only attributes allowed are those belonging to the associated table and its extension. Triggering conditions can only be used in global aggregate formulas because in inline formulas they are not part of the formula definition, and their triggering is conditioned in the code by means of conditional clauses (e.g. if, else, do case, etc.).

While a horizontal formula needs a context to be evaluated, an aggregate formula does not necessarily need one. However, if a context table exists, not all the records that the formula explicitly states will be considered, but only those that also match the implicit conditions arising from that context.

Example: Count



```
count( FlightSeatLocation, FlightSeatLocation = Location.Window)
```

FlightId	FlightDepartureAirportId	...
1	1	
2	3	...
3	1	...
...	...	...

FlightId	FlightSeatId		FlightSeatLocation
1	1	A	Window
1	1	B	Aisle
1	2	A	Window
1	2	B	Aisle
1	3	C	Middle
2	1	A	Window
2	1	B	Middle
3	...	...	...
...	...	...	...

```
&FlightCapacity = count(FlightSeatLocation, FlightSeatLocation = Location.Window)
```

For example, the FlightCapacity attribute of the Flight transaction is a count formula that runs through the FLIGHTSEAT table and counts the number of seats on the flight.

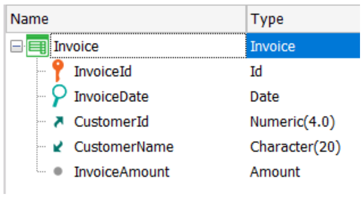
Since the table associated with the attribute is the Flight table, which has a 1 to N relationship with the FLIGHTSEAT table navigated by the Count, only the related records will be counted, i.e. the seats corresponding to the flight instantiated in Flight. If there were no relationship, all records in the FLIGHTSEAT table would be counted.

In addition, as we indicate that only seats that are windows will be counted, only those that meet the filter condition will be counted from the set of related records.

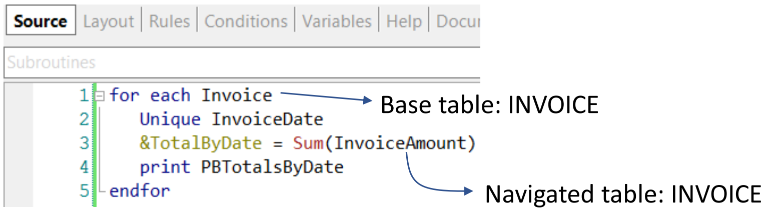
Since it is a Count formula, the aggregate expression is only an attribute that determines the table where the records will be counted – in this case, FlightSeatLocation – the aggregate condition is the filter that the seats must meet (that they are window). Also, since it is not a Max or Min formula, it has no default value, no return value and there is no trigger condition defined.

Here we see the same definition as an inline formula in the source of a procedure. In this case where the formula is “loose” there is no implicit condition: the window seats of all flights will be counted, not of a particular flight, as was the case with the global formula" or as would be the case if placed inside this For Each.

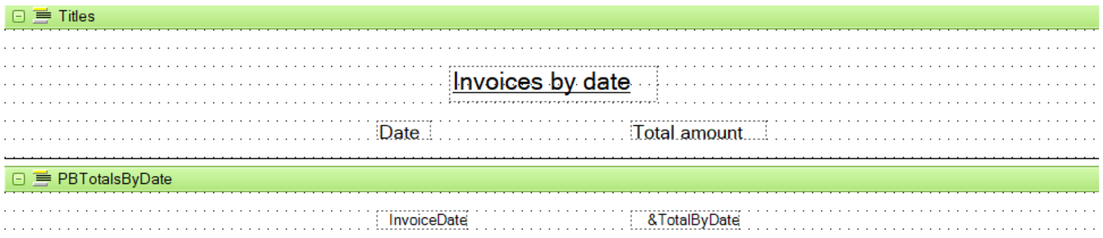
Example: associated table = navigated table



Name	Type
Invoice	Invoice
InvoiceId	Id
InvoiceDate	Date
CustomerId	Numeric(4.0)
CustomerName	Character(20)
InvoiceAmount	Amount



```
for each Invoice
   Unique InvoiceDate
   &TotalByDate = Sum(InvoiceAmount)
   print PBTotalsByDate
endfor
```



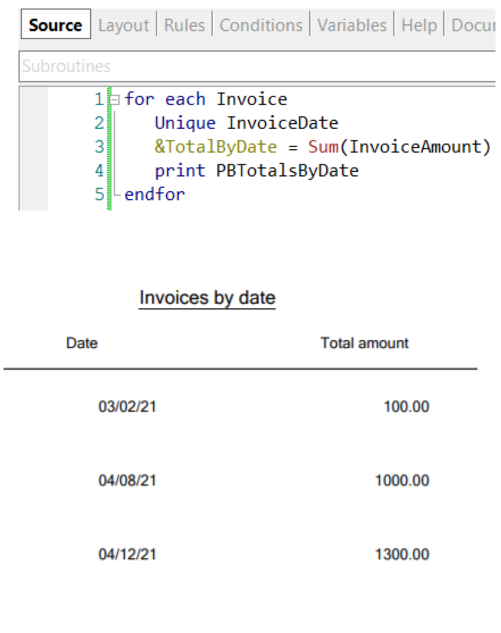
Let's analyze a particular case which is when the formula is in a certain context and the table navigated by the formula matches that context table.

In this example we want to print, for each invoice date, the sum total of all invoices with that invoice date.

The table navigated by the Sum formula is Invoice, and it matches the base table of the For Each, which is also Invoice.

In this case, due to the Unique clause by InvoiceDate, GeneXus will group the information by invoice date, both in the For Each and in the Sum formula. That is, the Sum formula will have an implicit equality condition by the InvoiceDate attribute, which will be considered as given.

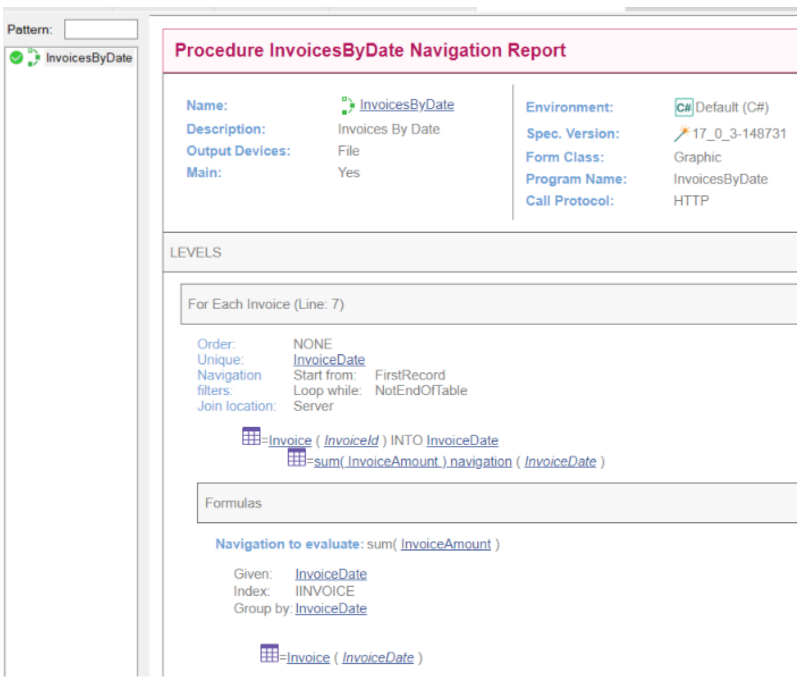
It is as if it were a control break, breaking by InvoiceDate. Let's see the navigation list.



```
for each Invoice
   Unique InvoiceDate
   &TotalByDate = Sum(InvoiceAmount)
   print PBTotalsByDate
endfor
```

Invoices by date

Date	Total amount
03/02/21	100.00
04/08/21	1000.00
04/12/21	1300.00



Procedure InvoicesByDate Navigation Report

Name:	InvoicesByDate	Environment:	Default (C#)
Description:	Invoices By Date	Spec. Version:	17_0_3-148731
Output Devices:	File	Form Class:	Graphic
Main:	Yes	Program Name:	InvoicesByDate
		Call Protocol:	HTTP

LEVELS

For Each Invoice (Line: 7)

```
Order:          NONE
Unique:         InvoiceDate
Navigation      Start from:   FirstRecord
filters:        Loop while:   NotEndOfTable
Join location:  Server

   =Invoice ( InvoiceId ) INTO InvoiceDate
      =sum( InvoiceAmount ) navigation ( InvoiceDate )
```

Formulas

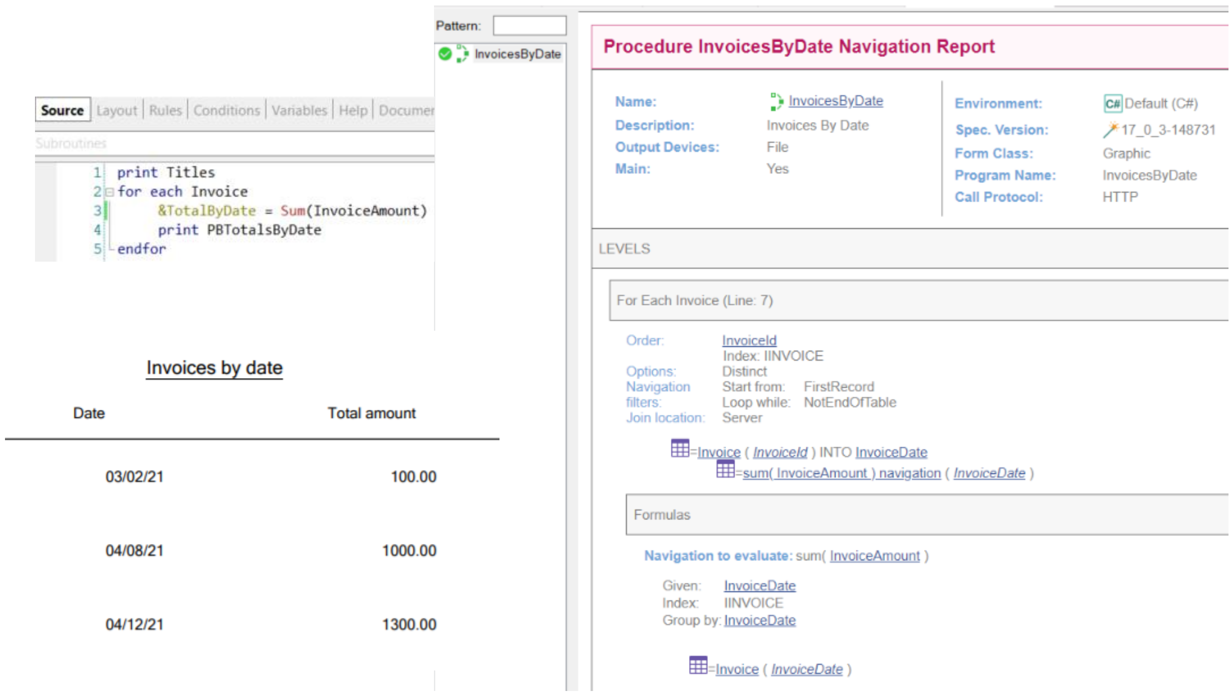
```
Navigation to evaluate: sum( InvoiceAmount )

   Given:    InvoiceDate
   Index:    IINVOICE
   Group by: InvoiceDate

      =Invoice ( InvoiceDate )
```

We check that the base table of the For Each is Invoice and that the Unique is by InvoiceDate.

Below we see the formula that is also grouping by InvoiceDate (note the Group by) and also InvoiceDate is Given. Therefore, it will add only the records of that date, just as we wanted.



```
print Titles
for each Invoice
    &TotalByDate = Sum(InvoiceAmount)
    print PBTotalsByDate
endfor
```

Invoices by date

Date	Total amount
03/02/21	100.00
04/08/21	1000.00
04/12/21	1300.00

Procedure InvoicesByDate Navigation Report

Name:	InvoicesByDate	Environment:	Default (C#)
Description:	Invoices By Date	Spec. Version:	17_0_3-148731
Output Devices:	File	Form Class:	Graphic
Main:	Yes	Program Name:	InvoicesByDate
		Call Protocol:	HTTP

LEVELS

For Each Invoice (Line: 7)

```
Order:            InvoiceId
                  Index: IINVOICE
Options:          Distinct
Navigation        Start from:    FirstRecord
filters:          Loop while:    NotEndOfTable
Join location:    Server

    =Invoice ( InvoiceId ) INTO InvoiceDate
        =sum( InvoiceAmount ) navigation ( InvoiceDate )
```

Formulas

```
Navigation to evaluate: sum( InvoiceAmount )

    Given:     InvoiceDate
    Index:     IINVOICE
    Group by:  InvoiceDate

      =Invoice ( InvoiceDate )
```

What would happen if we removed the Unique clause?
If we do not place the Unique, how does GeneXus know that we want to add only the invoice totals of the invoice date on which it is positioned? That is, if instead of writing the formula we decompose it in its calculation through a For Each...

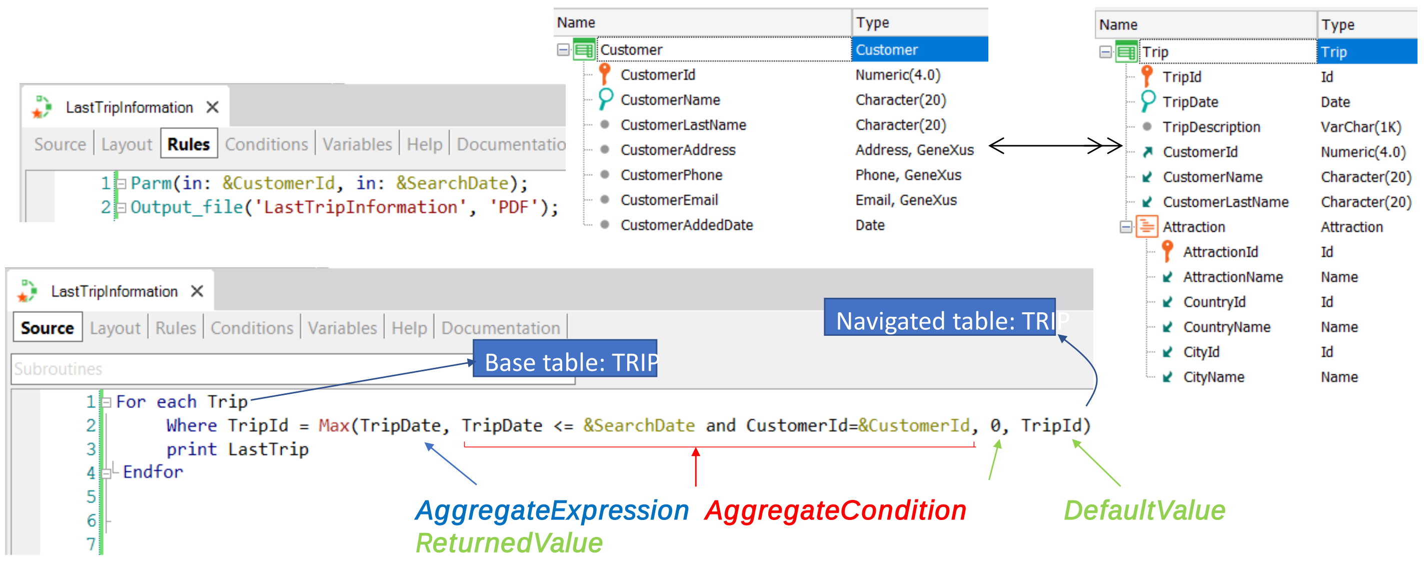
What will be printed? We do have a control break, but... we are not indicating the break criteria by InvoiceDate. We have not specified an order clause for the external For Each, so it will choose a primary key, and it will always stay in the internal For Each with only one record, that of InvoiceId.

However, let's look at the navigation list displayed by GeneXus.

Note that the Unique clause is no longer shown, but GeneXus automatically adds a DISTINCT clause, so the result remains the same. GeneXus had the intelligence to detect that we wanted to group by invoice date and reflected that in the code generation, making the list work the way we wanted.

Although GeneXus, depending on the case, is capable of automatically detecting what the developer wants to do, we should include the Unique clause, so that our intention is clear in the programming of the code.

Example 2: associated table = navigated table



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

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

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

Now let's see a similar case where a table is navigated and an aggregation is performed on the same table. In this case, the aggregation will be a max formula.

Let's suppose that given a customer who has many trips, you want to retrieve the data of a trip the customer made, which was immediately before a given date; i.e. the last trip before that date.

Suppose you want the information to be output in a list that prints the trip data. The procedure receives in a 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. Next, the data corresponding to that trip will be printed.

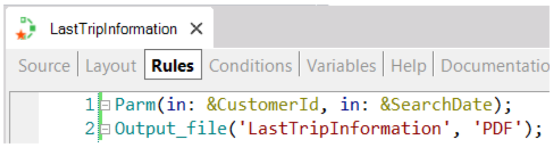
In the Max formula, the attribute to maximize is TripDate, since we want the trip that was immediately prior to a given date, so it will be the trip with the highest possible date, but lower than or equal to the search date.
The aggregation condition specifies that the trip must have a date lower than or equal to the given date and also belong to the customer requesting the information. The default value returned if no trip is found that meets these conditions is 0 and in case it is found it will be

the identifier of the trip found.

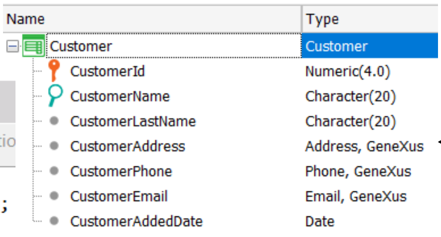
Now let's analyze what we have programmed. The For Each will iterate over the TRIP table and set that context for the Max formula.
The Max formula will also iterate on the TRIP table, but due to the context, an automatic filter will be set, because both tables are related. In this case, it is the same table, so the formula will be filtered by the TripId that is positioned in the For Each. Therefore, it will always return the TripId in which the For Each is located, and will be like not having written anything. This is not what we need.

In short, we need to be careful when an aggregate formula navigates the same table as that of its context because, most times, if we don't specify otherwise, it will apply a filter by primary key, which will cause that the navigated table will not be navigated at all and will always retrieve the same record in which it was positioned.

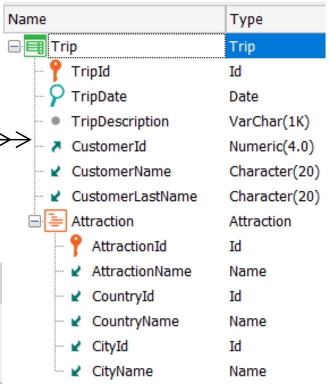
Example 2: associated table = navigated table



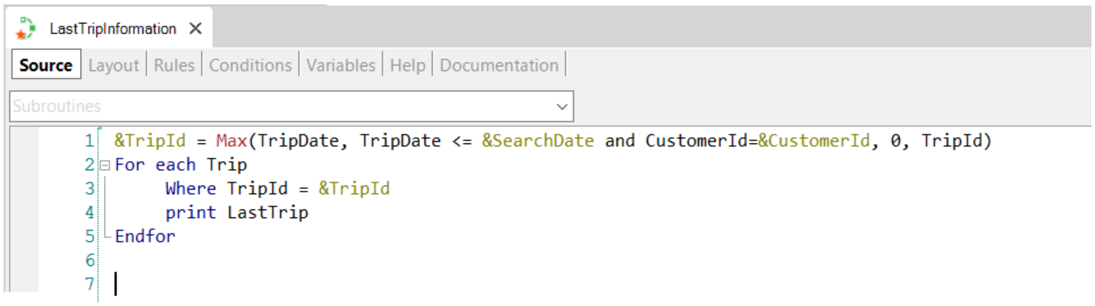
```
Parm(in: &CustomerId, in: &SearchDate);
Output_file('LastTripInformation', 'PDF');
```



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



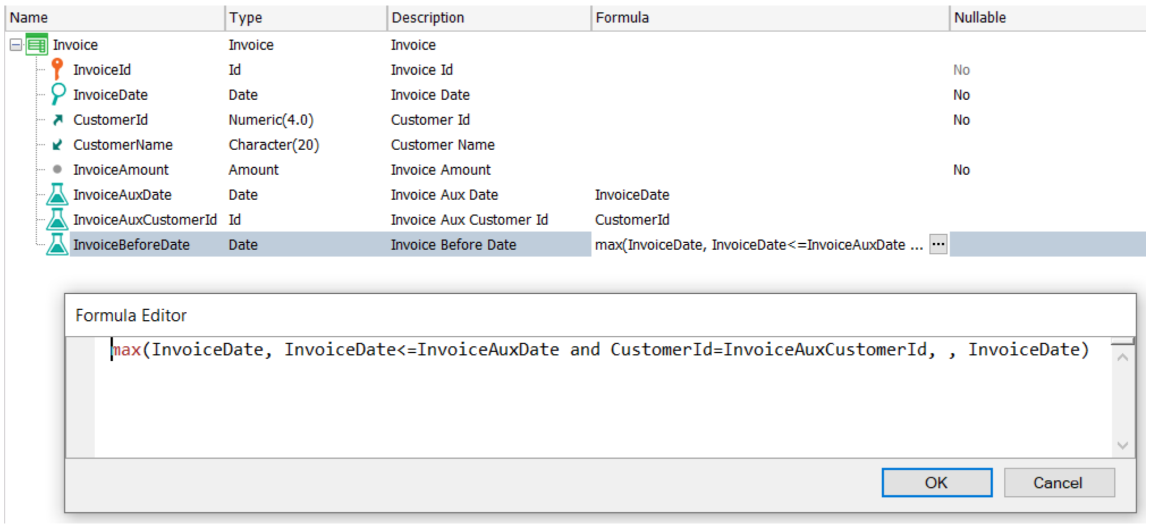
Name	Type
Trip	Trip
TripId	Id
TripDate	Date
TripDescription	VarChar(1K)
CustomerId	Numeric(4.0)
CustomerName	Character(20)
CustomerLastName	Character(20)
Attraction	Attraction
AttractionId	Id
AttractionName	Name
CountryId	Id
CountryName	Name
CityId	Id
CityName	Name



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

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

Example 3: associated table = navigated table



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

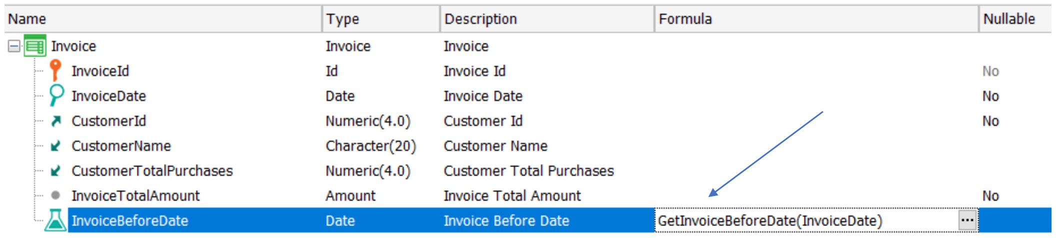
```
max(InvoiceDate, InvoiceDate<=InvoiceAuxDate and CustomerId=InvoiceAuxCustomerId, , InvoiceDate)
```

Now let's look at a similar case to the previous one in a global formula, in which 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, but defining an attribute as a global formula.

Note that again we are trying to define a formula that will navigate over the same table associated with the formula, so again the formula will be filtered by its context and will only navigate over a single record!

The difference between this case which is a global formula and the previous one which was an inline formula, is that in the previous one we could trigger the formula outside the context table to break the implicit filter, but in the case of a global formula this is not possible, so we can NEVER have a global aggregate formula that navigates the same table.

Example 3: associated table = navigated table



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	Numeric(4.0)	Customer Total Purchases		
InvoiceTotalAmount	Amount	Invoice Total Amount		No
InvoiceBeforeDate	Date	Invoice Before Date	GetInvoiceBeforeDate(InvoiceDate)	

Aggregate formulas don't need a context, but if there is one you can filter the result.

For this purpose, we will have to define it as a horizontal formula and perform the calculation within a procedure.

In summary, this confirms what we said before: that although aggregate formulas don't need a context to be evaluated (as horizontal formulas do), in the event that such a context exists, not all the records that the formula explicitly establishes will be considered, but only those that also match the implicit conditions that arise from that context.

And it is important to make sure that the context doesn't invalidate the objective we were looking for with the formula, as we saw in the last examples presented.

GeneXus™

training.genexus.com
wiki.genexus.com