# Redundant attributes and their maintenance

GeneXus

In this video, we will see how to define inferred attributes or formulas–which, by definition, are not stored–as redundant and make them attributes of a database table.

Name	Туре
🖃 🥅 Trip	Trip
- 📍 TripId	Id
- 🖓 TripDate	Date
<ul> <li>TripDescription</li> </ul>	VarChar(1K)
- 🔁 CustomerId	Numeric(4.0)
🛛 🖌 CustomerName	Character(20)
🛛 🖌 CustomerLastName	Character(20)

Name
Trip Structure
📍 📍 TripId
<ul> <li>TripDate</li> </ul>
<ul> <li>TripDescription</li> </ul>
<ul> <li>CustomerId</li> </ul>

Name		Туре
🖃 🥅 Ci	ustomer	Customer
- 9	CustomerId	Numeric(4.0)
- 9	CustomerName	Character(20)
	CustomerLastName	Character(20)
	CustomerFulName	Name
••••	CustomerAddress	Address, Gen
•	CustomerPhone	Phone, GeneX
•	CustomerEmail	Email, GeneXus
•	CustomerAddedDate	Date



As we know, GeneXus automatically normalizes the database in Third Normal Form, which implies that the only attributes that can be in more than one table are those attributes that are a primary key and perform foreign key functions.

The rest of the attributes, which are called secondary, are stored in a single table and determined by the primary key. If they are added to a different transaction, GeneXus will infer them, retrieving their value from the table where they are stored by means of the foreign key.

In addition, when we define an attribute as a formula in a transaction, it is no longer stored and becomes a virtual attribute.

However, for performance reasons, in some cases we want to allow an inferred attribute to be stored in the table associated with the transaction where it is inferred, or a formula attribute that must perform many calculations and is time consuming whenever its value is retrieved to be stored in its associated table in order to obtain the value more quickly.

GeneXus allows storing an attribute that by default is not stored in a table, defining it as redundant.

## **Referential redundancy**



When we make an inferred attribute redundant, it is called **referential redundancy**.

This approach to attribute redundancy was mainly motivated by the need to improve performance.

Let's suppose that the table of tourist attractions had millions of records, and we would like to retrieve those corresponding to countries whose name is alphabetically later than a given value, and whose city name is later than another.

For optimization purposes, we know that sorting by filter attributes is the most convenient option.

If we look at the navigation list, on one hand we see that the search is optimized in terms of navigation filters, but note that two joins will have to be made since CountryName and CityName are not in the Attraction table being run through. However, if they were in the table...

## **Referential redundancy**

Name	Redundant	_			
- Attraction		Country			
-      AttractionId     AttractionName     CountryId		CountryId	Warnings	\$	"
Countrywame	$\checkmark$	CityName	A <u>spc0038</u>	There is no index for order <u>CountryName, CityName;</u> poor performance may be noticed in group starting at line 1.	y
CityName	$\checkmark$				
AttractionPhoto			LEVELS	\$	*
					_
Source * Layout Rule	es Conditior	ns Variables Help Documentation	For Each Attra	ction (Line: 1)	~
Subroutines		~	Order:	CountryName , CityName	
1 ⇒ for each 2 ord 3 whe 4 whe 5 pri 6 endfor	Attractic er Country re Country re CityNam nt attract	<pre>con /Name, CityName /Name &gt;= &amp;CountryFilter ne &gt;= &amp;CityFilter tion_info</pre>	Navigation filters:	Start from: <u>CountryName</u> >= &CountryFilter <u>CityName</u> >= &CityFilter Loop while: NotEndOfTable Attraction ( <u>AttractionId</u> )	
0 - End tor	e =	attraction info			
		AttractionName			

...there would be no join to perform, so the performance would improve.

In both cases, we are informed that because there is no index for the order we defined, we could notice performance problems.

The intelligence and capabilities provided will depend on the DBMS being used. We know that DBMSs now are much more intelligent than in the past and have strategies to optimize searches.

However, in some cases to solve a query it will be necessary to create a temporary index that will be deleted after the query, and so on every time the query is executed.

If this were the case, and we needed, precisely to avoid this continuous creation of an index and its subsequent elimination, to create a user index for these attributes...

## Referential redundancy

Name	Redundant	Attraction *	×	$\sim$
Attraction		Structure Ind	exes *	
AttractionId		Attribute	Order	Description
AttractionName		🖃 🚍 Attraction In	Idexes	Attraction
CountryId		Attractio	on Primary Key	Automatic Index
Countrylu		<ul> <li>Attrac</li> </ul>	tionId Ascending	Attraction Id
Countryivame		IAttractio	on1 Foreign Key	Automatic Index
- 🔁 CityId	_	-  Count	tryId Ascending	Country Id
🖌 CityName		<ul> <li>CityId</li> </ul>	Ascending	City Id
AttractionPhoto		🗐 📑 UAttracti	on Duplicate	User Index
_		- • 😣 Co	untryName Ascending	
			🐼 Attribute Country	lame does not exist in table structure
Name	Redundant	Attraction *	x	~
	Redundanc	Structure Ind	exes *	
🛛 📍 AttractionId		Attribute	Order	Description
- 🖓 AttractionName		🖃 🚍 Attraction In	idexes	Attraction
CountryId		IAttractio	on Primary Key	Automatic Index
		<ul> <li>Attrac</li> </ul>	tionId Ascending	Attraction Id
Cibud	-	🗐 📑 IAttractio	on1 Foreign Key	Automatic Index
CityIu		-  Count	tryId Ascending	Country Id
CityName	$\checkmark$	CityId	Ascending	City Id
-AttractionPhoto		🖨 🎫 UAttracti	on Duplicate	User Index
		- • Count	ryName Ascending	Country Name
		- • CityNa	ame Ascending	City Name

...we will only be able to do it if both are in the Attraction table.

GXExpert3 - GeneXus 17								-	- 0	$\times$
File Edit View Layout Build Know	vledge Manager Window Tools	Test Help								
🖞 । 🛄 🛄 🔏 🗋 🖺 । 🤈 🔿 🧑	🖕 🚟 👑 🔪 🕨 .NET Environme	ent - Release	- 📮 Android 1	7.9-SNAPSHOT 📮 🔬 🖕						
穿 KB Explorer 🛛 🤻 🗙	Attraction X 📰 Attraction	×				$\sim$	Properties			₽ ×
open: Name or Pattern	Structure Indexes						2 🙀 Fil	ter		>
GXExpert3	Name	Type		Description	Formula		Attributor			
V 😭 Root Module	Attraction Structure	1,100		Attraction	1 official		Accumute.	cityiu		
GeneXus	P AttractionId	Id		Attraction Id						
> Common	AttractionName	Name		Attraction Name						
> 🗂 Web	CountryId	Id		Country Id						
> Temp	CityId	Id		City Id						
> Airline	<ul> <li>AttractionPhoto</li> </ul>	Image		Attraction Photo						
Elight										
Gy0010										
Gx0021										
CH0020										
Pamaina										
Domains										
images	Collectry X					$\checkmark$				
Country		le du cu	lucia de							
Associated Tables	Structure to Web Layout   Rul	es   Events   Variable	es   Help   Documentatio	n   Patterns						
Country	Name	Redundant	Туре	Description	Formula	Nullable				
CountryCity	Country		Country	Country						
Attraction	CountryId		Id	Country Id		No				
Associated Tables	CountryName		Name	Country Name		NO				
Attraction	P CitvId		Id	City Id		No				
PrintAttractions	CityName		Name	City Name		No				
Gx0040										
Gx0051										
Gx0060										
CountryUpdateProc										
TestRedundancyByProc										
> of References										
> 🏅 Customization	_									
> 📄 Documentation	E Output					ф ×				
	Show : Build	<mark>×</mark>  Q				合 🛄 🔸 Autoscroll				
	Web config file: appsettin	gs.json				10.55 <sup>-</sup>				
	Updating web config					and the second s				
	Success: Web contig update	ed				No. of Concession, Name				
	Success: Execution						L .			
B KB Explorer	Success: Run Developer Men	u					Properties	Toolbox 🕞 Tests Expl	or H Resp	onsive
:\Models\GX17StableForu9\GXExpert3								5/0/	0/5 6/6	

Let's see in GeneXus how to declare redundancy and its effects.

We have the Attractions transaction that infers, as usual, the values of CountryName and CityName from the foreign keys. In fact, if we look at the structure of the attractions table we can see that these attributes are not present, and only the foreign keys are there.

GXExpert3 - GeneXus 17						-	J ×
File Edit View Layout Build Kno	wledge Manager Window Tools Test	Help	<b>A C X D I</b>			· · · − □ ×	
ご 🔲 🖻 👗 🗅 🗂 ! • • • • • • •	🚛 🚟 🚟 🔛 🕨 .NET Environment	<ul> <li>Release</li> <li>Android</li> <li>17.9-SNAPS</li> </ul>	Genexus Develop	Country	× Attraction ×	·   +	
👂 KB Explorer 🛛 🤗 🗙	Attraction X 📰 Attraction X		$\leftarrow \rightarrow \mathbf{G} \mathbf{\nabla}$	Iocalhost:808	2/GXExpert3.NETEnvironment/country.a	aspx 🖻 🖈 🔲 🏰 Update 🚦	) 7 ×
pen: Name or Pattern	Structure Indexes		GYSung Web	Angular application	Travel Agency Sam	Other bookmarks	×
GXExpert3	Name	Type		Angular application	Intranet		
✓	Attraction Structure	-16-					<b>^</b>
√ ─ GeneXus	AttractionId	Id	Appli	cation Na	ame	by GeneXuš	-
> 🚞 Common	AttractionName	Name				Sy Concrete	
> 🚞 Web	CountryId	ld Td	Pasanta	Attraction - Co	ountry		
> Temp	AttractionPhoto	 Image	Recents		, ,		
> III Airline							
> iii Flight				Country			
GX0010							
Gx0030							
L Domains					≪ < > ≫ SELECT		
Images							
✓	Country X 📰 Country X 📰	CountryCity X		ld	1		
Associated Tables     Associated Table     Associated Table     Associated Table     Associated Table     Associated Table     Associated Table	Structure Indexes						
Country	Name	Туре		Nome	France		
CountryCity	CountryCity Structure			Name	France		
Attraction	CountryId	Id					
Associated Tables	Cityla	10 Name		City			
Attraction				ony	$\searrow$		
PrintAttractions				lc	I Name		
Gx0051							
Gx0060				× 1	Paris		
CountryUpdateProc					Niao		
TestRedundancyByProc				~ 4	Nice		
> neferences				0			
> T Customization	-						
Documentation	- Output			0			
	Show : Build	Q =		0			
	Web config file: appsettings.js	son		U			
	Updating web config Success: Web config update			0			
	Execution started						
KB Explorer 111 Preferences	Success: Execution			0			esponsive
Models\GX17StableForu9\GXExpert3	Success, Kun Severoper Held					5/0/0/5	6/6

We already have some data loaded at runtime: two countries with their cities. Let's pay special attention to 1, France, with its two cities Paris and Nice. Then we see that we have 3 attractions in France: two in Paris and one in Nice.



If we look for the data of the tables in SQL Server... we see that in the Attractions table there are only the foreign keys.



Well, now in Attraction we want to define the two inferred attributes as redundant. To do so, in the editor of the transaction structure, we add this column...



... which provides check boxes to indicate which attributes of the transaction structure we want to make redundant. It only offers us to make the inferred attributes redundant. If there were formula attributes, it would also offer us to make them redundant, as we will see.

We are going to select both, because we want both to be redundant in the Attraction table.



Let's save. We see that the attributes have been added to the table structure and it is indicated in this way that they are redundant.



If we now execute it, clearly we will have to reorganize the Attraction table to add those attributes that so far were inferred but now we want them to be stored. Note that it is indicated where their values will be taken from. Well, we expected this. But, what are these three internal procedures that GeneXus will create?

Database needs to be reorganized.								
This report describes Database of Please select Reorganize to proc	his report describes Database changes and how they will be handled by reorganization programs. lease select Reorganize to proceed or Cancel.							
Reorganize Cancel								
Pattern:	Table Attraction load rec	lundancy procedure		\$				
Country	Redundant attributes:		CountryName, CityName					
Attraction	Procedure Name:		AttractionLoadRedundancy					
	For Each Attraction (Line: 2) Order: AttractionId Index: IATTI Navigation filters: Loop while: Loop while:	RACTION FirstRecord NotEndOfTable		*				
	Join location: Server	d) ntryld) Countryld, Cityld) CountryName)						

Let's start by examining the last one. Its name is AttractionLoadRedundancy. Here it says that it is the procedure for loading the redundancies of the Attraction table. And it tells us which redundant attributes it has to load. What this procedure will do is to run through the entire table of attractions. For each one, it will go to Country to find the value of the CountryName attribute and store it in the redundant attribute of this Attraction table. Also, it will go to the cities table to find the value of the CityName attribute and store it in the new redundant CityName attribute in this Attraction table.

In sum, it must automatically execute this procedure after reorganizing the Attraction table so that the redundant attributes are loaded with the corresponding values.



Ok, but what about these two procedures?

If we look at the first one, we see that it is a procedure to update the redundancies of the Country table. We know that the CountryName attribute of this table is redundant in Attraction. This means that if we execute the transaction and change the name of a country, this change should also be made for all the attractions that correspond to that country in order to keep the redundancy up to date. This is what this procedure will do, which will be invoked transparently every time the user changes the value of CountryName for a CountryId through the Country transaction (or its business component). Here we can see that the procedure receives as parameter the value of the primary key to instantiate that record; then it runs through the table of attractions filtering by that value, and for each attraction it updates the CountryName in that table.



What about this other procedure? The same goes, but to keep the redundancies relative to the cities table updated. That is to say, when the name of a city in a country is modified from the Country transaction, this procedure will be automatically and transparently executed. The country ID and the city ID will be sent to it by parameter, this table record will be accessed, and for each matching Attraction record, the value of the redundant attribute CityName will be modified in Attraction.

These procedures will then be created in this reorganization, and the logic we just explained will be added to the transactions.

We reorganize.



Now let's see the table in SQL Server. The redundant attributes were added and assigned their corresponding value.

We are now going to change the name of the country France to its name in Spanish, Francia. We will do it through the transaction.

Let's look at the table data... the name in the transaction table has changed... and let's see what happened to the redundant attribute... just as we expected, it was updated.

Likewise, if we are going to modify the name of a city, for example Nice, writing it now in Spanish (Niza)... we see that in the cities table it has been changed obviously, and now in the Attractions table... too.

That was due to the modification made through the Country transaction.



UNIN	150(SQLEAFILE	55 ub0.Attiat		NIN SUISQUEN	T(L3DD - 000	.country		
	AttractionId	Attraction	Countryld	Cityld	AttractionP	AttractionP	CountryNa	CityName
•	1	Eiffel Tower	1	1	<binary dat<="" th=""><th>gxdbfile:Eiff</th><th>Francia</th><th>Paris</th></binary>	gxdbfile:Eiff	Francia	Paris
	2	Smithsonian	2	1	<binary dat<="" th=""><th>gxdbfile:Sm</th><th>United Stat</th><th>Washington</th></binary>	gxdbfile:Sm	United Stat	Washington
	3	Matisse Mus	1	2	<binary dat<="" th=""><th>gxdbfile:Mu</th><th>Francia</th><th>Nice</th></binary>	gxdbfile:Mu	Francia	Nice
	4	Louvre Mus	1	1	<binary dat<="" th=""><th>gxdbfile:Lo</th><th>Francia</th><th>Paris</th></binary>	gxdbfile:Lo	Francia	Paris
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL &	NULL

Let's see what happens if we do it through a procedure... We have a web panel for the user to enter a country ID, and when pressing the button we invoke a procedure that receives that ID and goes to the Country table to change the value of the CountryName attribute for the country with that ID to "Something." Let's run it to test it.

We choose the country with ID 1, which is France. In the Country table, its value was indeed changed to the one assigned by the procedure. But now let's see if the procedure kept the redundancy in Attraction up to date. No, it did not.



Business Components
 Country
 Country.City

CountryUpdateProc

GeneXus invokes those procedures that we saw whose names ended in UpdateRedundancy only when the modifications are made through the transaction (or the business component, logically). It doesn't invoke them when the modifications are made in any other way.

So, we must be careful. In case of modifying attributes that are redundant in other tables through another way than the transaction or the business component, the developer will be responsible for keeping the redundant attributes up to date. GeneXus won't do it.



The other case of redundancies that we had already mentioned was that of formulas. In this example, we see that in addition to offering redundancy of the inferred attributes, it also offers redundancy of the two formulas defined for the transaction: the horizontal one that applies a discount to the price of the flight according to the discount percentage set by the airline, and the aggregate one that counts the number of seats on the flight.

Defining these formulas as redundant will add the attributes to the table. In addition, obviously, in the reorganization GeneXus must also create a procedure to load them with the corresponding values.



That is, a procedure where for each record in the Flight table it will trigger the calculation of each formula... for FlightFinalPrice it will have to go to Airline to find the value of AirlineDiscountPercentage, and store its value in the redundant attribute; for FlightCapacity it will have to count the associated records in the FlightSeat table, and store the result in the redundant attribute.



If we also define, for example, AirlineName as redundant—that is, a referential redundancy—its value would be loaded into the table in this same procedure. That's why the name of the procedure is always the concatenation of the name of the table where the redundancies are, in this case Flight, and LoadRedundancy. That is to say, in this procedure all the redundancies of the table will be loaded: the referential ones and those of formulas.

And it will be the procedure automatically invoked in the reorganization, after the new attributes are added to the database table.



Let's focus only on the formulas again. Of course, once they have been made redundant, when information on flight price or capacity is needed, the formula will not be triggered again, but the stored value will be retrieved, and that is exactly the point.

Hypothetically, if we had to run this list millions of times, and if there were thousands of seats for each flight, calculating the FlightCapacity formula every time could become a performance problem. Having it redundant saves us from having to make the calculation for each query. There is only the cost of the calculation to load the redundant attribute the first time, and to keep it up to date afterwards.

For the horizontal formula of the example, the usefulness of defining it as redundant doesn't seem too clear, at least in terms of performance, unless we need, for example, a panel or web panel where the user wants to filter the flights shown in a grid by flight price.

The case would become more interesting if the horizontal calculation involved many tables of the extended one. Or, even more so, if it were a horizontal formula of the kind that is solved by invoking a procedure that does perform a complex calculation.



If there is a clear advantage to redundant formula attributes, what is the disadvantage?

That they should always be kept up to date, and that update has a cost. If the value of FlightPrice is changed through the Flight transaction (or its business component) the horizontal formula involving it will be triggered again as usual. The same happens if a line is added or deleted: the FlightCapacity formula is triggered again. In both cases, its value is stored in the redundant attributes and the developer doesn't have to worry about doing it.

However, as in the case of referential redundancies, if the modification is done with a procedure, as in these examples, GeneXus will do nothing. Here the developer will have to worry about updating the redundant attributes by adding code.



Table Airline update redundancy procedure				
Redundant attributes to update:	FlightFinalPrice			
From attributes to update:	<u>AirlineDiscountPercentage</u>			
Procedure Name:	AirlineUpdateRedundancy			

What happens if the user enters the Airline transaction and modifies the discount percentage for an airline? (or does it through the business component).

In Flight the redundant formula FlightFinalPrice depends on that value, so the redundant value stored for all Flight records belonging to that airline should be updated.

GeneXus will create in the reorganization the procedure whose name is the concatenation of the table name, in this case Airline, and UpdateRedundancy, as it did with referential redundancies.



This procedure will be invoked from the Airline transaction by passing it the ID and the procedure will access the corresponding record. Next, it will run through the Flight table filtering by that airline and trigger the FlightFinalPrice formula again, storing its result in the table.

All this implies a performance cost. Therefore, a very careful assessment should be made of when it is convenient to make a formula redundant and when it is not.



For the moment, the maintenance of redundant formulas has some limitations.

For example, when we have these three related transactions, where the Flight transaction records the generic information of a flight, such as its price, but the FlightInstance transaction is the one corresponding to the actual flight on a given date. In it, the price of the actual flight is obtained with a formula that takes into account the list price of the flight according to the date.

And then we have a transaction to record flight invoicing. The lines record the flights for which tickets are being purchased, and how many tickets for each one. Then this formula calculates the price for each line, using the inferred formula attribute of FlightInstance, which in turn is a formula, as we said.

Suppose we want to make this formula redundant in the InvoiceFlightInstance table. If we only make it redundant...



...the procedure for loading the redundancy will be handled correctly, by going to the InvoiceFlightInstance table and triggering the horizontal formula for each record, and storing it.

However, redundancy update procedures will not be created.



What do we mean?

That if the user executes the FlightInstance transaction and changes, for example, the value of FlightInstanceDate (suppose that from a lower date than this to a later one), since the horizontal formula will be triggered again and its value will be changed, we would expect the transaction to invoke a procedure– FlightInstanceUpdateRedundancy–that will go to the FlightInstance table and update the corresponding redundancies. But it won't.

Similarly, we would expect that if the user changes the value of the FlightPrice attribute through the Flight transaction there would also be a FlightUpdateRedundancy procedure that goes to the InvoiceFlightInstance table to recalculate and re-store the redundancies of the formula attribute that should be changed when the FlightInstancePrice is changed, when applicable. It won't do it either.



But what if we also make the involved horizontal formula redundant?



We will see when reorganizing that in addition to reporting the changes in both tables to add the two formulas as redundant, and the two procedures for loading the redundancies, two UpdateRedundancy procedures will be created.

The first one that is triggered from the Flight transaction when the price is changed and will update the first redundant formula in FlightInstance. Then, from it, the second one in InvoiceFlightInstance.

GeneXus



The second procedure will be executed when the flight date is changed from the FlightInstance transaction, and will update the redundant attribute in InvoiceFlightInstance.



So, if we have a formula whose calculation involves another one, in order to maintain its redundancy automatically, we need to define the inner formula as redundant as well.

	ght FlightId Price P FlightPriceDate P FlightPriceValue		FlightInstance         FlightInstanceId         FlightInstanceDate         FlightInstancePrice         max(FlightPriceDate, FlightPriceDate <= FlightInstanceDate, , FlightPriceValue)
FlightId	FlightPriceDate	FlightPriceValue	FlightId: 1
			FlightInstanceDate: 03/03/2022
1	01/01/2022	800	FlightInstancePrice: ?
1	02/02/2022	1000	
1	04/04/2022	950	

But there are more limitations. For example, in most cases aggregate/select formulas cannot be maintained through these update procedures.

If we maintain in Flight a list of flight prices by date, for example, in FlightInstance we will have to calculate the price of a particular flight according to the price that corresponds to the actual flight date. That is, we calculate the price according to a max formula.

So, with this data, if we are entering an instance of flight 1, with this date, the max formula will be left with this record...

Fligh	it lightId rice FlightPriceDate FlightPriceValue		FlightInstance         P       FlightInstanceId         P       FlightInstanceDate         P       FlightInstanceDate         P       FlightInstancePrice         Image: State of the sta	ntPriceValue)
FlightId	FlightPriceDate	FlightPriceValue	FlightId: 1	
			FlightInstanceDate: 03/03/2022	
1	01/01/2022	800	FlightInstancePrice: 1000	
1	02/02/2022	1000		
1	04/04/2022	950		

... so it will return the value 1000 for the flight price. If it were being printed in a list, 1000 would be displayed.

Flight Flightd Frice FlightPriceDate FlightPriceValue				max( FlightPriceD	ate, FlightPriceDate <= FlightInstanceDate, , FlightPriceValue)
FlightId	FlightPriceDate	FlightPriceValue	Flightld: 1		
1	01/01/2022	800	FlightInstanceDate: 03	3/03/2022 1100000	LIST
1	02/02/2022	1500	ngittiistancernce.		
1	04/04/2022	950			

#### FlightPriceUpdateRedundancy?

If later on the user goes to the Flight transaction and changes 1000 to 1500 in the line corresponding to this record, since the formula is virtual, when the list is executed again, the max will be triggered and 1500 will be listed.

Flight Flightd Flightd FlightPriceDate			FlightInstance         FlightInstanceId         FlightInstanceDate         FlightInstancePrice         FlightInstancePrice         FlightInstancePrice
FlightId	FlightPriceDate	FlightPriceValue	FlightId: 1
			FlightInstanceDate: 03/03/2022
1	01/01/2022	800	FlightInstancePrice: 1500
1	02/02/2022	1500	
1	04/04/2022	950	

FlightPriceUpdateRedundancy?

But what if we make the max formula redundant? We would expect that a redundancy update procedure would be created for the FlightPrice table, so when the value of FlightPriceValue is modified through the Flight transaction, the FlightInstance table would be accessed looking for which records would be affected by the change to modify the value of FlightInstancePrice. But when trying to figure out which records would be affected, we see how difficult it is to determine this.

Therefore, in this case, GeneXus will not create this redundancy maintenance program. We see this clearly when we define the attribute as redundant and look at the reorg report.



It will only create the redundancy load procedure, but not the update procedure.

So we recommend to always read this impact analysis report to find out which redundant formulas will be maintained and which ones will not.

04/04/2022

950

Fli	ght FlightId Price P FlightPriceDate P FlightPriceValue		FlightInstance	V	max( FlightPriceDa	ate, FlightPriceDate <= FlightInstanceDate, , FlightPriceV	alue)
FlightId	FlightPriceDate	FlightPriceValue	FlightId: 1 FlightInstancel	Date: 03	/03/2022	LIST	
1	01/01/2022	800	FlightInstance	Price:	1500		
1	02/02/2022	1500					

As a side note: if the user accesses the FlightInstance transaction and changes the value of FlightInstanceDate, of course the redundant attribute will be kept up to date. The reason is that inside the transaction the formula will be triggered as usual and its value will be stored. The problem occurs when an attribute involved in the formula calculation is modified from another transaction, not the one of the formula. In this case, from the Flight transaction.

## Rebuild redundancy



If we needed that attribute to be redundant anyway, knowing that the redundancy will not be automatically kept up to date from Flight, we can always invoke the redundancy loading procedure created by GeneXus.

If we look for the files in the environment directory, we will find the redundancy loading procedure for each table with redundant attributes. The one that ends with LoadRedundancy; table name: load redundancy. In the last case, the flightinstanceloadredundancy, which we can invoke as a call to an external procedure.

In addition, even though it is not listed in the impact analysis report, GeneXus creates a procedure named gxlred, for GeneXus Load Redundancy, which invokes each of the LoadRedundancy procedures of each table with redundancies.

So if at any time we want to make sure that all redundancies defined in the KB are updated, we can, for example, invoke this program from an event.

Other limitations in defining redundancies

To define as redundant a formula that is already a formula, we must first define that other formula as redundant

Redundant aggregate formulas that add more than one level of nested redundant formulas will not be properly maintained

Subtypes cannot be defined as redundant

To change the definition of a formula that is redundant, you must first remove the redundancy

There are other limitations for defining redundant attributes that must be considered.

Some of them are listed here and you can read more about them in the wiki.



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