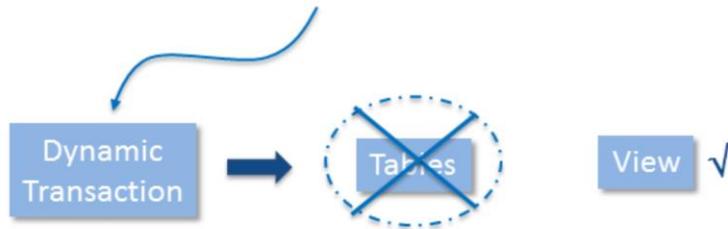


# Dynamic Transactions

*GeneXus* 15

## Data

Data Provider	True
Used to	Retrieve data



- In the DP, we must specify which data we want to retrieve...:
  - when the Transaction form is executed.
  - when using the attributes.
- A Dynamic Transaction can be referenced as Base Transaction

Use case #1 : Data union

**Sale**

- SaleId Id
- SaleDate Date
- ShowId Id
- ShowName Name
- SectorId Id
- SectorName Name
- SectorPrice Price
- CurrencyName Name
- SaleAmount Price

**CreditNote**

- CreditNoteId Id
- CreditNoteDate Date
- SaleId Id
- SaleDate Date
- SaleAmount Price
- ShowId
- ShowName
- SectorId
- SectorName
- SectorPrice

**Movement**

- MovementType Character(1)
- MovementId Id
- MovementDate Date
- MovementAmount Price
- MovementCurrencyName Name

Data Provider	<b>True</b>
Used to	<b>Retrieve data</b>

**Movement\_DataProvider**

Source Rules Variables Help Documentation

```

1 MovementCollection
2 {
3   Movement from Sale
4   {
5     MovementType = "S"
6     MovementId = SaleId
7     MovementDate = SaleDate
8     MovementAmount = SaleAmount
9     MovementCurrencyName = CurrencyName
10  }
11  Movement from CreditNote
12  {
13    MovementType = "C"
14    MovementId = CreditNoteId
15    MovementDate = CreditNoteDate
16    MovementAmount = SaleAmount
17    MovementCurrencyName = CurrencyName
18  }
19 }
```

Use case #1 : Data union

Movement	
MovementType	Character(1)
MovementId	Id
MovementDate	Date
MovementAmount	Price
MovementCurrencyName	Name

Movements 09/27/17

Movement type	Movement id	Movement Amount	Currency
S	1	200	US Dolar
S	2	3500	Uruguayan Peso
S	3	150	US Dolar
C	1	200	US Dolar

```

PrintTodayMovements X
Source | Layout | Rules | Conditions | Variables | Help | Documentation |
Subroutines
1 Print Title
2 For each Movement order (MovementType), MovementDate
3   where MovementDate=&Today
4   Print Movement Movement
5 Endfor

```

MovementType	MovementId	MovementAmount	MovementCurrencyName
--------------	------------	----------------	----------------------

Use case #1 : Data union

Movement	
MovementType	Character(1)
MovementId	Id
MovementDate	Date
MovementAmount	Price
MovementCurrencyName	Name

Data	
Data Provider	True
Used to	Retrieve data

Work With for Web

Apply this pattern on save 

**X HIDE FILTERS**

**Movements**

Q // 25

MOVEMENT TYPE

MOVEMENT AMOUNT

MOVEMENT CURRENCY NAI

Type	Id	Date	Amount	Currency Name
S	1	09/27/17	200	US Dolar
S	2	09/27/17	3500	Uruguayan Peso
S	3	09/27/17	150	US Dolar
C	1	09/27/17	200	US Dolar

Use case #1 : Data union

Movement	
MovementType	Character(1)
MovementId	Id
MovementDate	Date
MovementAmount	Price
MovementCurrencyName	Name

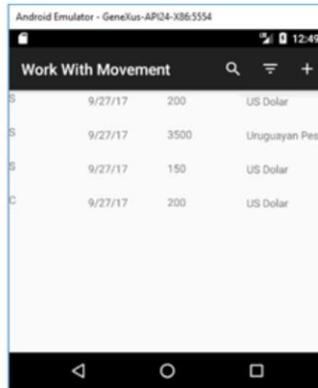
▼ Data

Data Provider	True
Used to	Retrieve data

Work With for Smart Devices

Apply this pattern on save

- Level (Movement)
  - List
  - Detail
  - Section (General)



Use case #2 : Modeling a reality



- ✓ Shows to be presented within 5 days or less
- ✓ SectorAvailableQuantity >= 100

Promotion X

Structure Web Form Win Form Rules

Name	Type
Promotion	Promotion
PromotionShowId	Id
PromotionSectorId	Id
PromotionShowName	Name
PromotionDate	Date
PromotionPrice	Price
PromotionCurrencyName	Name
PromotionVenueName	Name

Data

Data Provider	True
Used to	Retrieve data

Promotion\_DataProvider X

Source Rules Variables Help Documentation

```

1 PromotionCollection
2 {
3   Promotion from Show.Sector
4     where (ShowDate-ServerDate()) <= 5 and (ShowDate-ServerDate()) >=0
5     where SectorAvailableQuantity >=100
6 }
7 {
8   PromotionShowId = ShowId
9   PromotionSectorId = SectorId
10  PromotionShowName = ShowName
11  PromotionDate = ShowDate
12  PromotionPrice = SectorPrice/2
13  PromotionCurrencyName = CurrencyName
14  PromotionVenueName = VenueName
15 }
                
```

Use case #2 : Modeling a reality

GRID

Promotion Date	Promotion Show Id	Promotion Show Name	Promotion Sector Id	Sector Name	Promotion Price	Promotion Currency Name	Promotion Venue Name
PromotionDate	PromotionShowId	PromotionShowName	PromotionSectorId	SectorName	PromotionPrice	PromotionCurrencyName	PromotionVenueName

Use case #2 : Modeling a reality



Promotion Date	Promotion Show Id	Promotion Show Name	Promotion Sector Id	Sector Name	Promotion Venue Name	Promotion Price	Promotion Currency Name
10/03/17	1	Madonna in Concert	1	Orchestra Platinum	Madison Square Garden	100	US Dollar
10/03/17	1	Madonna in Concert	2	Orchestra Gold	Madison Square Garden	75	US Dollar
10/06/17	2	David Bisbal - Tour 2017	1	America Rosstrum	Estadio Centenario	3500	Uruguayan Peso

## Use case #3 : Solving statistics

- For a given period, to know the number of sales per day.
- For a given date, to know the number of sales.
- Best day of the year.

The screenshot displays the GeneXus IDE interface. On the left, the 'Structure' view shows a data provider named 'Statistics' with two attributes: 'StatisticsDate' (Date) and 'StatisticsSalesQuantity' (Numeric(10.0)). Below this, the 'Data' section shows 'Data Provider' set to 'True' and 'Used to' set to 'Retrieve data'.

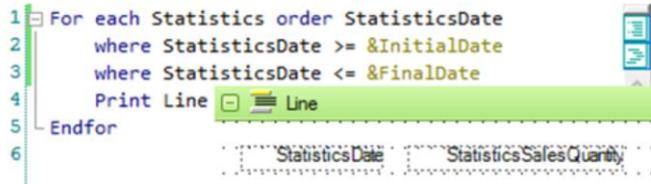
On the right, the 'Source' view shows the following code for the 'StatisticsCollection' class:

```
1 StatisticsCollection
2 {
3   Statistics from Sale unique SaleDate
4   {
5     StatisticsDate = SaleDate
6     StatisticsSalesQuantity = Count(SaleAmount)
7   }
8 }
```

## Use case #3 : Solving statistics

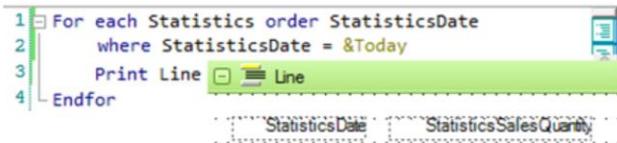
- For a given period, to know the number of sales per day

```
1 For each Statistics order StatisticsDate
2   where StatisticsDate >= &InitialDate
3   where StatisticsDate <= &FinalDate
4   Print Line Line
5 Endfor
```



- For a given date, to know the number of sales.

```
1 For each Statistics order StatisticsDate
2   where StatisticsDate = &Today
3   Print Line Line
4 Endfor
```





## Use case #4 : Making Evolution Easier

```
Person
{
  PersonId*
  PersonName
  GenderId
  GenderName
}
```

```
Gender
{
  GenderId*
  GenderName
  GenderMembers = count(PersonName)
}
```

```
WeightLog
{
  PersonId*
  WeightLogDate*
  WeightLogKilos
}
```

The objective of use case #4 is to show how defining dynamic transactions helps us to easily evolve.

We have a GeneXus Knowledge Base for tracking body weight, with the transactions shown on the slide.

- Person: Allows registering people and for each of them, their gender.
- Gender: Allows registering genders and has a defined formula to know how many people are registered in each gender.
- WeightLog: Allows registering, for each person, in each data, his/her weight.

Now suppose that, with the system already up and running, the developer is asked to track not only the weights but also other body measurements (like chest or waist circumference).

The database model needs to be redesigned in order to store this new data. Of course, it's possible to create a new Transaction object for each new measurement to be tracked, but a better (and more extensible) design is to have just one Transaction for any kind of measurements, as the following slide shows.

## Use case #4 : Making Evolution Easier

The screenshot displays the GeneXus IDE interface with several transaction definitions and data providers. The transactions are:

- Person**: Attributes: PersonId\*, PersonName, GenderId, GenderName.
- Measure**: Attributes: MeasureId\*, MeasureName.
- MeasureLog**: Attributes: PersonId\*, MeasureId\*, MeasureLogDate\*, MeasureLogValue.
- Gender**: Attributes: GenderId\*, GenderName; Calculation: GenderMembers = count(PersonName).
- WeightLog**: Attributes: PersonId\*, WeightLogDate\*, WeightLogKilos.

Two data providers are shown:

- Measure\_DataProvider**: Data Provider: True; Used to: Retrieve data. The associated code snippet is:
 

```

1 MeasureCollection
2 {
3   Measure
4   {
5     MeasureId = 1
6     MeasureName = 'Weight'
7   }
8   Measure
9   {
10    MeasureId = 2
11    MeasureName = 'Chest'
12  }
13  //other...
14 }
```
- WeightLog\_DataProvider**: Data Provider: True; Used to: Retrieve data. The associated code snippet is:
 

```

1 WeightLogCollection
2 {
3   WeightLog from MeasureLog
4   where MeasureId=1
5   {
6     PersonId = PersonId
7     WeightLogDate = MeasureLogDate
8     WeightLogKilos = MeasureLogValue
9   }
10 }
```

As you can see, we have defined two new transactions.

- Measure: Allows registering measurements (for example, “Weight”, “Chest”, etc.) appropriately.
- MeasureLog: Allows registering, for each person, on each data, for a specific measurement, the value measured.

Note that the Measure transaction has been defined as dynamic (take a look at its properties). You should also note that in the associated Data Provider, the data assigned to the Measure attributes is not obtained from tables, but rather it's fixed values. This is because the developer does not want users to edit, delete, or enter new measurements. The developer has assigned specific values. A physical table associated with the Measure transaction will not be created, and note that the MeasureId attribute is used as a foreign key in the MeasureLog transaction.

The WeightLog transaction is not needed anymore since all measurements will be stored in the physical table associated with the new MeasureLog transaction. However, the application code still references it as base transaction in many places, such as For Each statements. So, instead of removing the WeightLog transaction and having to modify wherever it is referenced, it's a good idea to change it into a dynamic Transaction.

**The developer must not forget that if he defines a transaction as Dynamic, the associated physical tables will no longer exist. So, before proceeding with this proposal, he/she has to move the data (in this case, weights from WeightLog to MeasureLog table).**

**GeneXus**

**Use case #4 : Making Evolution Easier**

```

MeasureLog
{
  PersonId*
  MeasureId*
  MeasureLogDate*
  MeasureLogValue
}

WeightLog
{
  PersonId*
  WeightLogDate*
  WeightLogKilos
}

```

```

Event Insert(&Messages)
&MeasureLog = new()
&MeasureLog.PersonId      = PersonId
&MeasureLog.MeasureId    = 1
&MeasureLog.MeasureLogDate = WeightLogDate
&MeasureLog.MeasureLogValue = WeightLogKilos
&MeasureLog.Insert()
&Messages = &MeasureLog.GetMessages()
Endevent

Event Update(&Messages)
&MeasureLog.Load(PersonId, 1, WeightLogDate)
&MeasureLog.MeasureLogValue = WeightLogKilos
&MeasureLog.Update()
&Messages = &MeasureLog.GetMessages()
Endevent

Event Delete(&Messages)
&MeasureLog.Load(PersonId, 1, WeightLogDate)
&MeasureLog.Delete()
&Messages = &MeasureLog.GetMessages()
Endevent

```

**WeightLog\_DataProvider**

Source \* Rules Variables Help Documentation

```

1 WeightLogCollection
2 {
3   WeightLog from MeasureLog
4     where MeasureId=1
5   {
6     PersonId = PersonId
7     WeightLogDate = MeasureLogDate
8     WeightLogKilos = MeasureLogValue
9   }
10 }

```

Data Provider	True
Used to	Retrieve data
Update Policy	Updatable

By default, the form of dynamic Transactions shows data with read-only behavior.

Now, let's suppose that users are used to executing the WeightLog Transaction, and they ask us to still be able to edit it through its form. They also use the MeasureLog transaction, but they want to use both.

It's possible to complete these request, since transactions offer another property under the "Data" group. Its name is "Update Policy" and its possible values are: Read Only / Updatable.

So, by setting the WeightLog Transaction "Update Policy" property = Updatable, its form will allow the users to edit the data; **but in which physical table will the updates be stored?**

The developer has to codify the Insert, Update and Delete events in the WeightLog Transaction Events section, in order to specify his intention. In this example, the logical solution is to store the data in the MeasureLog physical table, using the Business Component concept as shown.

Note that after applying the Insert(), Update() and Delete() methods respectively to the &MeasureLog business component variable, the developer obtains the messages and/or errors triggered (in the &Messages collection variable). By declaring the &Messages variable as a parameter in each event (as shown), those messages are displayed in the WeightLog Dynamic Transaction in a transparent way, like its own messages.

In this way, the WeightLog Dynamic Transaction can be used in **exactly the same way as**

**before** and no changes are necessary to dependent programs. This also applies if the transaction is used as a Business Component, because it is a Dynamic Transaction that allows updates and the corresponding events to store the data are codified.

## ADVANTAGES

Describe realities and intentions flexibly

Systems evolve more easily

Simplify developing

Easier for new developers