Infor BI OLAP Server Rules Engine User Guide

Release 11.0.x

Publication Information

Release: Infor BI 11.0.x
Publication Date: January 17, 2017
Document code: bi_11.0.x_biosreug__en-us

Contents

About this guide.....5

Related documents.....7

Contacting Infor.....9

Chapter 1: Overview of the Infor BI OLAP Server Rules Engine.....11

Objectives of the new Infor BI OLAP Server Rules Engine.....11

Chapter 2: Old Infor BI OLAP Server Rules Engine.....13

Rules, accelerators, and calculation.....13

Restrictions.....13

Chapter 3: New Infor BI OLAP Server Rules Engine.....15

Rules and calculation.....15

Best Practices related to rules.....16

Additional best practices information.....20

Configuration.....23

Chapter 4: Formulas.....25

Limitations.....27

About this guide

In MIS Alea 5.x and Infor BI OLAP Server 10.x there are a number of improvements to the Rules Engine - the component that carries out the business logic calculations in an OLAP data model. In particular, non-accelerated calculation became much faster. This document is intended for users who already have a working knowledge of OLAP and the rules engine and want to know how they can make use of the improvements in the enhanced rules engine.

The document will provide experienced users with an overview of the changes that have been introduced into the software. They will get a quick introduction to the new system, so they can judge what changes, if any, they might want to make in their system, or how the new features will affect the way in which they create OLAP cubes in the future.

The document is not intended for beginners. It is not a tutorial and will not provide any information on how to create OLAP rules. Due to the complexity of the subject it is not possible to give any simple definitive statement on how the changes in the rules engine will affect any given model.

Intended audience

Administrators and users with a working knowledge of the OLAP Server and rules are the intended audience of this document.

The information given herein is related to the release of Infor BI. Older versions might not provide the full functionality.

Organization

This guide contains these chapters:

Chapter	Description
Overview of the Infor BI OLAP Server Rules Engine	Provides a brief overview of the objectives of the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).
Old Infor BI OLAP Server Rules Engine	Provides a comparison of features between the old and the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).
New Infor BI OLAP Server Rules Engine	Details the improved functionality of the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).

Performance warranties

Infor has prepared this guide thoroughly but cannot guarantee optimal performance because there are many variables in the modeling procedure. Infor assumes no liability for any damages or extra costs that result from recommendations in this guide.

Terms and definitions

Term	Definition
BI	Business Intelligence
OLAP	Online Analytical Processing; In-Memory database management system
MIS Alea	Former name of Infor BI OLAP Server
Rules	Definition of the business logic calculations in the OLAP Server
Accelerators	A technique in former OLAP versions to speed up rule calculation. No longer needed with the new rules engine
Old rules engine	The calculation engine used for the calculation of accelerated data Models
New rules engine	Improved rules calculation engine with much better performance on non-accelerated OLAP data models.

Related documents

You can find related documents in the product documentation section of the Infor Xtreme Support portal, as described in 'Contacting Infor'.

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal.

If we update this document after the product release, we will post the new version on this website. We recommend that you check this website periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Overview of the Infor BI OLAP Server Rules Engine

1

This chapter provides a brief overview of the objectives of the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).

Objectives of the new Infor BI OLAP Server Rules Engine

Accelerators were introduced early in the development of the OLAP Server, because rule calculations were dramatically slower than aggregations. Based on the accelerators, the OLAP Server generates, at cube load time or while adding a value to a cell, one or more flags to mark those cells that need to be calculated at runtime. Under the old OLAP Server Rules Engine, the administrator of the model is responsible for writing accelerator definitions to flag the cells to be calculated. This is a complex and error prone process.

The goal of the new OLAP Server Rules Engine is to speed up the non-accelerated rule calculations in such a way that the performance is comparable to the performance of an accelerated cube. As a consequence, the creation of rules is much easier and the error prone process of creating accelerators is no longer required.

Additionally, omitting the accelerators avoids the requirement to create accelerator flags at load time. This speeds up the cube loading dramatically.

The new OLAP Server Rules Engine works with the same rules syntax as the old OLAP Server Rules Engine. The intention is to make it easier for users to switch to it.

For compatibility reasons existing accelerated rules continue to work, meaning that customers are not forced to change their rules. Nevertheless the performance without accelerators can be better than with accelerators, i.e. please try to use your rules with and without accelerators to optimize your implementation.

Note: There is a mistaken belief that the purpose of the new OLAP Server Rules Engine is to "auto-generated accelerators". This is incorrect. The new OLAP Server Rules Engine calculates in a much different way. Accelerators and accelerator flags are no longer required.

Old Infor BI OLAP Server Rules Engine 2

This chapter provides a comparison of features between the old and the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).

Rules, accelerators, and calculation

To understand how the new, improved, Rules Engine works, it is important to have a basic understanding of what the accelerators do, and why they make rules faster. What the accelerators do is to check if the target cell of an accelerated rule needs to be calculated. Usually a target cell is not required to be calculated if the source cells are empty. The key issue is that the old Rules Engine, without accelerators, has to check if cells are empty. Since, in general, cubes tend to be fairly empty, this can take a long time in production environments.

When the OLAP cube is loaded, the server checks the accelerators and marks, with an accelerator flag, the target cells of the accelerator that have non-empty source cells. This operation is repeated each time the cube is loaded,. Every time a value is written into a cell, the server checks if the cell is the source of an accelerator. If it is, then the target cells are marked with accelerator flags. When the rules are calculated, only the flagged cells are calculated. This reduces the time spent in checking if the source cells are empty.

Enabling acceleration means that the responsibility for identifying the base level cells of a calculation is with the writer of the rules and accelerators, not with the OLAP Server. If the accelerators are set incompletely, or incorrectly, this can lead to incorrect calculations by the server, with some values not being taken into account during calculation.

Restrictions

The calculation algorithm of the old rules engine focuses on single cell calculations. The fetches are made individually during the calculation process. In other words: the calculation starts, the Rules Engine interprets the rule, sees that data from a source cell is required, fetches the data, and restarts the

calculation where it left off. When it discovers that a second source cell is required, it interrupts the process, and does another fetch operation.

New Infor BI OLAP Server Rules Engine 3

This chapter details the improved functionality of the new Infor BI OLAP Server Rules Engine (OLAP Server Rules Engine).

Rules and calculation

The new Rules Engine has a completely different method of calculating rules. Instead of fetching each data cell individually as the requirement arises, the new Rules Engine calculates the entire range of cells that the rule is required to execute. This data is fetched as a block in as few operations as possible. The question of whether a given cell is empty is simply ignored. The fetch requests all the cells, whether empty or not. Only the non-empty cells return data, and only this data is processed.

In a sense, the difference in fetching behavior between the old and the new rules engines is similar to the difference between the behavior of DBGet and DBGetC formulas in Office Plus. DBGet get individual data cells from the server. Unlike DBGet, DBGetC collects all the database accesses and sends them to the server in as few individual accesses as possible. Because the turnaround time for individual accesses is high, this is a time saver, even though, from the user perspective, the same amount of data is transferred.

Sometimes it is not clear where the data comes from without executing the rule. This is the case with conditional rules; that is, with rules containing the IF function. For example, if the location of a target cell depends on whether the evaluation of a condition is True or False, then it cannot be clear where the data is supposed to come from. But the OLAP Server solves this problem by fetching the data for the evaluation of all conditions first. Then it continues with two sub-queries, one for the cases where the condition is true and the other one for cases where the condition is false.

Note: Starting with OLAP Server 10.5.0 some default values in the `DB.ini` were increased to speed up calculations. It is recommended to adjust these settings in existing data models accordingly.

File\Section	Parameter	Change	Old value	New value
DB.INI\MEMORY	CacheSize	Enabled by default	0	1
DB.INI\INIT	MaxTileSize	Higher value, can cause issues to costumers without	1e6	1E80

File\Section	Parameter	Change	Old value	New value
		acceleration and not optimized rules		
DB.INI\INIT	MaxExportTileSize		1e6	1E80
DB.INI\MEMORY	CSDSlots	Higher number for more calculations of more users	1024	8192

Best Practices related to rules

This section provides some best practice guidelines for using the new rules engine.

Rules comments

Rules can have comments. There are two formats:

- The Single line format is //
- The multiline format is /* */

We recommend that you always use comments for long rules.

You can include a short description of the rule in the description field.

Avoid constant values in rules

Previously, a rule like this was often used to suppress consolidations on consolidated cells:

```
['Price']=C:0;
```

The problem with this construct is that a constant value is used. For users, the value 0 often equates to 'no value'. But the OLAP Server treats the 0 as a value, that is, it allocates memory for the value and starts calculating with it.

A better solution is this:

```
['Price']=C:#NA;
```

In this case the result is the same - you still sees no values, but the OLAP Server understands that there are no values and does not start to work with them.

Avoid error results

Avoid all invalid references which will return an `#ERROR` result. To remember that a cell contains an `#ERROR` value, the OLAP Server allocates memory. This memory should be preserved.

You can do this by checking dimension boundaries when you are using DE.PREV, DE.NEXT, or DE.SIBLING functions in DB or GETATTR formulas. Additionally, check the existence of elements in the target dimension when you are mapping one dimension into another.

Here are some examples of how a check can be implemented in cases where you must access a value from a previous element:

```
IF( DE.INDEX( 'SOURCEDIM', !SOURCEDIM )=0, #NA , DB(…,DE.PREV( 'dim' , !dim),…))
```

```
IF( DE.INDEX( 'SOURCEDIM', !SOURCEDIM )>=1, DB(…,DE.PREV( 'dim' , !dim),…), #NA )
```

These rules apply:

```
[SOURCEDIM:'1st element',…]= # […] = DB(…,DE.PREV( 'dim' , !dim),…)
```

Avoid dynamic references

For best performance, do not use:

- Variable hypercube names in external cube data references
- Variable dimension names in attribute data references
- Variable field names in attribute references
- Variable table IDs in attribute references

Always use constant strings or constant numeric IDs.

Avoid unnecessary fixed members in cell references

The definition of a rule is always evaluated in the context of the cell that should be calculated. Therefore, you should omit in internal cell references all dimensions, except the ones where the context should change.

This is an example of bad practice:

```
['Actual','Q1','Starting Balance'] = ['Actual','January','Starting Balance'];
```

Instead, the rule should look like this:

```
['Actual','Q1','Starting Balance'] = ['January'];
```

‘Actual’ and ‘Starting Balance’ should be omitted, as they are taken from the context.

Use single-dimensional conditions

The determination of the exact source area is important for caching values for faster calculation. When you use IF-statements, the source area is typically dynamic , dependent on the result of a logical expression. The more dimensions the IF-condition includes, the more complex becomes the determination of the exact source area. Therefore, the goal is to keep the conditional expression of an IF-statement one-dimensional.

In IF-Statements, it is normal to check data based on multiple dimensions: IF the year is equal to 2011 AND the month is equal to January, then one value shall be returned, or else another value. The AND operator creates a multi-dimensional condition.

To avoid this multi-dimensional condition, you can split an IF statement into multiple IF statements. For example:

```
[] = IF( !year = '2011' AND !month = 'January' , [Value1] , [Value2] );
```

This is not optimal because of the multi-dimensional AND condition. It should be rewritten as:

```
[] = IF( !year = '2011' , IF( !month = 'January' , [Value1] , [Value2] ) ,
[Value2] );
```

This is more effective as it has two one-dimensional conditions.

Simplify the rules

Rules can be simplified by mathematical transformations.

Example 1:

```
['Revenue'] = ['Nominal Revenue']-['Discount Percent']/100*['Nominal
Revenue']
```

Here the value [‘Nominal Revenue’] is listed twice. You can transform this into:

```
['Revenue'] = ['Nominal Revenue'] * (1 -['Discount Percent']/100)
```

Where the [‘Nominal Revenue’] is moved in front of the brackets to be multiplied with each of the parts inside the brackets. Because of this change, the [‘Nominal Revenue’] is used only once in the formula.

Example 2:

```
['Payment'] = IF(!Months='December', ['Salary']*['Bonus'], ['Salary'])
```

Again, one element is used twice in the definition. Move it out of the IF statement to get the same result:

```
['Payment'] = ['Salary'] * IF(!Months='December', ['Bonus'], 1)
```

Note: It is more efficient to write several basic rules than one complex rule.

Help OLAP Server to pre-fetch values

In certain situations OLAP Server cannot determine in advance which values need to be retrieved. The creator of rules can help OLAP Server to pre-fetch those values that might be needed in advance of the calculation.

For example, if you have this rule:

```
['Account 1'] = B:
IF(['Account 2'] < 1000],
  ['Account 3a'],
  IF(['Account 2'] < 1000000,
    ['Account 3b'],
    ['Account 3c']
  )
)
)
+ #NA * (['Account 3a']+['Account 3b']+['Account 3c'])
```

A base level cell on Account 1 will receive either the value of Account 3a, Account 3b or Account 3c based on the value of Account 2.

If OLAP Server must calculate a billion base values using this rule, then it should try to get all necessary the source values for the calculation in one step. But this is not easily possible, as different cells should be taken dependent on the value in Account 2. Today, OLAP Server fetches all values for Account 2 in one step. Based on the evaluation of Account 2, it will fetch the values either from Account 3a, Account 3b or Account 3c one by one. This might be inefficient, dependant on the kind of cells behind Account 3a-c. Basically, there can be two situations, or a mixture of both:

1. Account 3a-c are low-cost calculated cells, and base cells, which require no calculation. In this case the cost of returning the values is low in terms of calculation time needed.
2. Account 3a-c are calculated cells (complex calculations, expensive in terms of calculation time needed). In this case it depends on the underlying calculation if the calculation is expensive or low-cost.

If the retrieval of the cells is low-cost, then it would be efficient to retrieve the values for Account 3a-c together with Account 2 in one quick step. This can easily be done be adding a dummy operation that does not influence the result:

```
['Account 1'] = B:
IF(['Account 2'] < 1000],
  ['Account 3a'],
  IF(['Account 2'] < 1000000,
    ['Account 3b'],
    ['Account 3c']
  )
)
+ #NA * (['Account 3a']+['Account 3b']+['Account 3c'])
```

The only difference to the initial rule is in the last line. Here the values Account 3a, Account 3b and Account 3c are added up and then multiplied by #NA. This does not change the overall result of the cell. But it helps especially in the case that the formula needs to calculate with values that are stored in the OLAP Server cube, as OLAP Server has a fast algorithm to fetch the values from the cube.

The same pre-fetching idea might speed up the system in the case of low-cost rule calculation, while it might decrease the performance in case of complicated that got referenced.

This also applies to the rules function STET:

```
'Account 1']=B:
IF(
  ['Account 2']>0,
  ['Account 2']/12,
  STET
)
+ #NA*STET
```

Currently, OLAP Server is unable to detect this situation and do this optimization by itself. We might find later a way to implement this or give the creator of the model a way to define which values should be pre-fetched.

Use Consolidations instead of Rules

Calculations can be defined in the OLAP Server in different ways, as consolidations inside a dimension or as a rule. It is more efficient to define a calculation as a consolidation inside a dimension than as a rule.

Where possible use consolidations instead of rules. For example:

Use the consolidation inside the dimension:

```
Q1
   January
   February
   March
```

Instead of:

```
Q1 = January + February + March
```

Additional best practices information

This section describes some best practices which - although not specifically concerned with rule syntax, will help improve performance.

Use mass data calculation

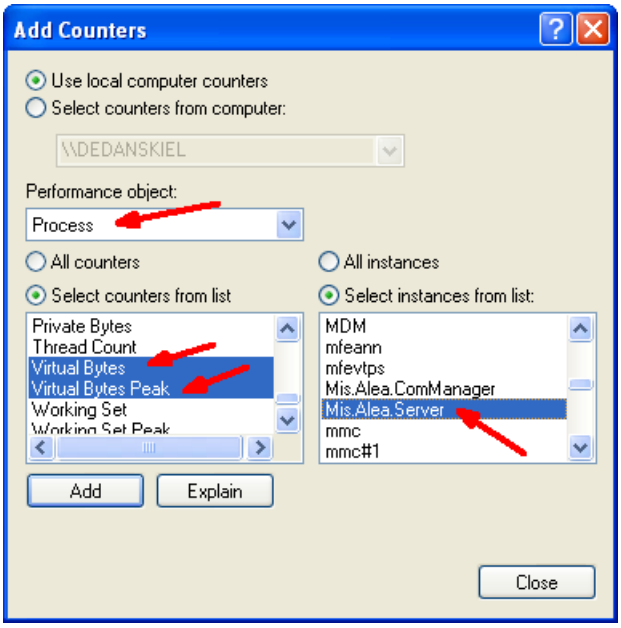
As described, the new rules engine internally retrieves the data for calculation in big blocks. If values can be reused (for example, currency conversion factors are the same for all products), then they are retrieved only once. Therefore, it is important to use mass operations instead of single cell operation.

There is a second reason to prefer mass operations over single cell operations. When working in a network environment, each single cell request travels on its own over the network. In the case of 100 cells, the network latency between the client and the server is multiplied 100 times. With mass operations, network latency has only minimal effect. For example:

- Use DBGetC instead of DBGet
- Use CellGetC or MDXExecute instead of CellGet
- Use ad-hoc reports or Alea ad-hoc reports
- Do not loop and retrieve single cell values. Instead, create a data area and loop over its results.

Monitor memory consumption

The new OLAP Server calculates with big blocks of data, there might be peaks in the memory consumption based on the calculations of the user. It is suggested that you monitor the virtual bytes and their peak on the MIS.Alea.Server process.



The performance counters might be named differently in different versions of the Windows operating system.

The goal is to ensure that the OLAP Server does not hit the 2 or 3 GB limit per process on a 32 bit Windows system, and that Windows does not start swapping.

You can influence the memory needed in the calculation. See 'Configuration' for more details.

Rule debugging support

The OLAP Server can provide some information on how a given cell is calculated. This information can be retrieved through cell notes. If a cell note contains the content "value?", then the OLAP Server returns calculation information in the cell note.

To try this out, open an Alea ad-hoc report in Office Plus showing the cell you want to research. Open a cell note and add the string "value?". Do not add any other characters or new lines. Save the cell note and reopen it.

On a consolidated cell, you might see information like this:

```
value?
---Cell Coordinates ------------------------------
YEARS 2002
ACTVSBUD Actual
REGIONS World
PRODUCTS Total
MONTHS Year
MEASURES Units
Internal Key: 1 1 34 61 13 1
External Key: 1 1 1 1 1 1
------------------------------------------------------
Cell Type: Calculated
```

```
Consolidated by:
REGIONS:'World' 33 - Base Descendants
PRODUCTS:'Total' 60 - Base Descendants
MONTHS:'Year' 12 - Base Descendants
Total consolidated cells: 23760
```

The first part gives you the coordinates of the cell, with the dimension and element names. It also contains an internal and an external key. The external key relates to the id of the element in the dimension.

The second part indicates that this is a calculated element. Moreover it states from how many base descendants it is calculated. This cell value is consolidated from 23760 base cells.

For a cell calculated by rule, the information has some differences:

```
value?
---Cell Coordinates -------------------------------
YEARS 2002
ACTVSBUD Actual
REGIONS World
PRODUCTS Total
MONTHS Year
MEASURES Average price
Internal Key: 1 1 34 61 13 2
External Key: 1 1 1 1 1 2
-----------------------------------------------------------
Cell Type: Calculated
Rule: ['Average price']=['Sales']/['Units'];
---Optimized Rule References----------------------
['Sales']
['Units']
-----------------------------------------------------------
```

Again, the second part indicates that the value is calculated, but this time by rule. The rule is displayed.

General advice on performance testing

An important part of improving the performance of your rules is to implement reliable performance measurement. Only if this is available can the performance or the improvements be judged. This topic highlights some steps in performance testing:

- Make the performance reproducible.

  This is the most important step. Ensure that you always measure performance by using the same report with the same selections, running under the same user credentials.

- Ensure that the values are not cached.

  If the OLAP Server has calculated the values previously, then they are cached and the server responds much faster. Consider emptying the cache before you do the measurement. But do not disable the cache, as this will change the calculation behavior.

- Identify relevant rules for your report.

To work on the correct rule, use the "value?" functionality in order to understand how a value is calculated. Sometimes it helps to disable rules entirely to understand if they have a significant effect on the calculation time.

- Record implemented changes and their effects.

  This will help you to understand how your changes affect performance.

- Implement one change after another.

  By implementing the changes one by one and testing their effects, you can understand the results of your changes.

- Check the results.

  After changing your rules, compare the results on your report with the calculation from the original rules to ensure that you have not accidentally introduced an error into the rule definition.

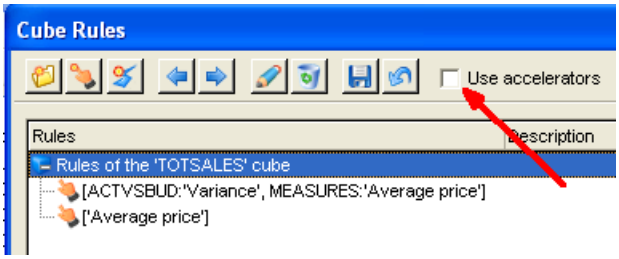
- Be patient.

  Performance testing might take some time. But the more experience you gain, the easier it will be.

Configuration

Activation of the new OLAP Server Rules Engine

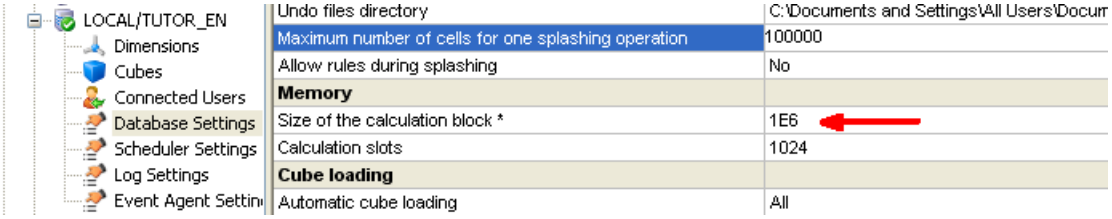
The OLAP Server will, by default, use the old rules engine for calculations on cubes where acceleration is enabled. If you disable acceleration, then the new rules engine will be used.



Which rules engine the server uses is not influenced by the existence of accelerator definitions. I.e. even if there are accelerator definitions that create accelerator flags, they will not be used if the "Use accelerators" checkbox is unchecked.

Size of the calculation blocks

As described in the section Rules and Calculation, the OLAP Server retrieves and calculates the values in blocks. The maximum size of those blocks is defined in the database settings of OLAP Administration, in the section Memory, Size of calculation block:



The default value is 1e6 (equals 106 = 1.000.000) cells. i.e. the OLAP Server will create internally blocks with no more than 1e6 cells. This limit is conservative and too low for many data models due to their sparseness. If it is too low, then calculations can be split into too many blocks, slowing the calculation.

It is recommended to try the server behavior with values like 1e50 or 1e80. Usually they will speed up the systems without problems. But for safety, we recommend that you monitor the memory usage as described above. If you run into a memory problem, avoid especially fixed results in rules as described in 'Best practices' related to rules.

Formulas

4

Alea formulas fall into the categories described in this topic.

See the Infor BI Office Plus for Microsoft Office Excel online help for information on the formulas.

- Attributes formulas
- Cell notes formulas
- Cube properties formulas
- Dimension properties formulas
- Element properties formulas
- Element selection formulas
- Special formulas
- Subset properties formulas
- Values formulas
- Writeback formulas

Attributes formulas

- AT.COUNT
- AT.FIELD
- AT.FIELDCOUNT
- AT.FIELDINDEX
- AT.FIELDTYPE
- AT.GET
- AT.HEAD
- AT.NAME
- AT.SET
- AT.SUB.HEAD

Cell notes formulas

- DIM.MASK.NOTE.DELETE
- DIM.MASK.NOTE.GET
- DIM.MASK.NOTE.SET

- NGET
- NSET

Cube properties formulas

- STABLE.COUNT
- STABLE.NAME
- TABLE.GET.INFO
- TDIM.COUNT
- TDIM.NAME

Dimension properties formulas

- AT.COUNT
- DE.COUNT
- DE.FIRST
- DE.NEXT
- DIM.TOPLEVEL
- DIMENSION.GET.INFO

Element properties formulas

- DE.CHILDNAME
- DE.CHILDRENCOUNT
- DE.COUNT
- DE.FIRST
- DE.INDEX
- DE.ISCHILD
- DE.LEVEL
- DE.NEXT
- DE.PARENTNAME
- DE.PARENTSCOUNT
- DE.PREV
- DE.SIBLING
- DE.TOPCOUNT
- DE.TOPNAME
- DE.TYPE
- DE.WEIGHT

Element selection formulas

- DE.ADD
- DE.NAME

- SE.NAME

Special formulas

- MDS.ERROR
- MDS.ISNA
- MDS.LAST.ERROR
- MDS.NA
- MDS.SET.LAST.ERROR
- SERVER.GET.INFO
- VALUE.BUFFER.ANYTHING.CHANGED
- VALUE.BUFFER.GET.CALC.STATE
- WORKSPACE.GET.PARAMETER
- WORKSPACE.MAKE
- WORKSPACE.RELEASE
- WORKSPACE.SUBRELEASE
- WORKSPACE.VERSION

Subset properties formulas

- SE.COUNT
- SE.INDEX

Values formulas

- DBGET
- DBGETC
- DBGETC.FIRST
- DBGETC.NEXT

Writeback formulas

- DBGET
- DBGETC
- DBSET
- DBSET2
- DBSET3

Limitations

Rule functions used to navigate dimensions and extract information about elements have some limitations when working with hierarchies.

You can use these functions with no limitations:

- DE.TYPE
- DE.PARENTSCOUNT

You can use these functions can be used when the referenced dimension has a single hierarchy.

- DE.COUNT
- DE.INDEX
- DIMIX
- DIM.TOPLEVEL
- DIMSIZ

You can use these functions when both elements belong to the same hierarchy. If the child is a base element, the hierarchy reference is ignored.

- DE.WEIGHT
- DE.ISCHILD
- DE.ISDESCENDANT

You can use these functions on a dimension level. For example, if a dimension has more than one hierarchy, DE.PREV() called on an element belonging to a certain hierarchy returns an element belonging to a different hierarchy.

- HCLOSINGPERIOD
- HOPENINGPERIOD
- CLOSINGPERIOD
- OPENINGPERIOD
- DE.HSIBLING
- DE.NEXT
- DE.PREV
- DE.SIBLING
- DE.PARENTNAME
- DE.LEAFCHILDRENCOUNT
- DE.LEAFDESCENDANTSCOUNT
- DE.LEVEL
- DE.CHILDNAME
- DE.CHILDRENCOUNT
- DE.DESCENDANTSCOUNT
- DE.NAME
- DE.FIRST
- DE.INDEX