
Referenz

instantOLAP

Version 2.1.2



instantOLAP[®]

18.04.2005

Copyright 2004 Thomas Behrends Softwareentwicklung e.K. Alle Rechte vorbehalten.

Dieses Handbuch und die darin beschriebene Software werden unter einer Lizenzvereinbarung zur Verfügung gestellt und dürfen ausschließlich nach Maßgabe dieser Lizenzvereinbarung genutzt oder kopiert werden. Dieses Handbuch dient lediglich der Information, es kann jederzeit ohne Ankündigung geändert werden und stellt keine Zusicherung irgend einer Art seitens Thomas Behrends dar. Thomas Behrends übernimmt keine Verantwortung und Haftung für eventuelle Fehler oder Ungenauigkeiten in diesem Handbuch.

Kein Teil dieses Handbuchs darf anders als in den Lizenzvereinbarungen ausdrücklich gestattet ohne vorherige schriftliche Zustimmung von Thomas Behrends elektronisch, mechanisch oder auf irgend eine andere Weise vervielfältigt, gespeichert oder übertragen werden.

instantOLAP ist ein registriertes von Thomas Behrends. Windows ist ein registriertes Warenzeichen der Microsoft Corporation. Alle anderen Produkt- und Firmennamen sind Warenzeichen oder eingetragene Warenzeichen ihrer jeweiligen Eigentümer.

Inhalt

Konventionen 15

Berichts-Eigenschaften 17

Bericht	18
Author	18
CSS	18
Date	19
Description	20
Filter	20
Formats	21
Icon	21
Logo	22
Model	23
Name	24
Print Height	24
Print Logo	24
Print Width	25
Stylesheet	26
Title	26
Visible	27
Select	28
Default	28
Height	28
Name	29
Options	30
Submit	30
Text	31
Title	32
Type	32
Use Filter	33
Width	34
Äußere Blöcke	35
Filter	35
Iteration	35
Orientation	36
Innere Blöcke	38
Background	38
Border	38
Color	39
Filter	40
Font	40
Font Weight	41
Font Size	42
Format	42
Height	45
Iteration	45
Link	46

Link-Icon	47
Link-Keys	48
Link-Target	48
Link-Name	49
Padding	49
Title	50
Width	51
Abfragen	52
Corner-Align	52
Corner-Background	52
Corner-Bottom-Color	53
Corner-Font	53
Corner-Font-Weight	54
Corner-Font-Size	54
Corner-Foreground	55
Corner-Left-Color	55
Corner-Right-Color	56
Corner-Text	57
Corner-Top-Color	58
Filter	58
Subcube	59
Supress-Cols	60
Supress-Rows	61
Überschriften	63
Align	63
Assertion	63
Background	64
Bottom-Color	65
Bottom-Padding	66
Cell-Align	66
Cell-Background	67
Cell-Bottom-Color	68
Cell-Bottom-Padding	68
Cell-Font	69
Cell-Font-Size	69
Cell-Font-Weight	70
Cell-Foreground	70
Cell-Left-Color	71
Cell-Left-Padding	71
Cell-Link	72
Cell-Link-Icon	73
Cell-Link-Keys	73
Cell-Link-Name	74
Cell-Link-Target	75
Cell-Right-Color	75
Cell-Right-Padding	76
Cell-Top-Color	77
Cell-Top-Padding	77
Cell-Vertical-Align	78
Drilldown	78
Drilldown-Encapsulate	79
Drilldown-Iteration	80
Drilldown-Prefetch	81
Filter	81
Font	82
Font-Size	83

Font-Weight	83
Foreground	84
Format	84
Formula	85
Height	86
Input	87
Iteration	87
Left-Color	88
Left-Padding	89
Link	90
Link-Icon	91
Link-Keys	91
Link-Name	92
Link-Target	93
Right-Color	93
Right-Padding	94
Rotate	94
Sort	95
Sort-Descending	95
Text	96
Title	97
Top-Color	97
Top-Padding	98
Vertical-Align	98
Visible	99
Width	100
Z-Order	100
Kommentare	102
Author	102
Copy-To-Result	102
Date	103
Export	103
Formula	104
Text	104

Diagramm-Eigenschaften 107

Liniendiagramm-Eigenschaften	108
3DDepth	108
3DModeOn	108
AutoLabelSpacingOn	109
Background	109
ChartBackground	110
ChartForeground	110
ChartTitle	111
ConnectedLinesOn	111
DefaultGridLinesColor	111
DefaultGridLinesOn	112
FloatingLabelFont	112
FloatingOnLegendOff	113
Font	113
Foreground	114
GraphInsets	114
GridAdjustmentOn	115
GridImage	115
GridLineColors	115

GridLines	116
GridLinesColor	116
Label_0	117
LabelUrl_0	117
LabelUrlTarget_0	118
LegendColors	118
LegendColumns	119
LegendFont	119
LegendImage	120
LegendOn	120
LegendPosition	121
LegendReverseOn	121
LineStyle	122
LineWidth	122
LowerRange	123
MaxValueLineCount	123
Range	124
RangeAdjusted	124
RangeAdjusterOn	124
RangeAdjusterPosition	125
RangeAxisLabel	125
RangeAxisLabelAngle	126
RangeAxisLabelFont	126
RangeColor	127
RangeDecimalCount	127
RangeLabelFont	127
RangeLabelPostfix	128
RangeLabelPrefix	128
RangeLabelsOff	129
RangeOn	129
RangePosition	129
RangeStep	130
SampleAxisLabel	130
SampleAxisLabelFont	131
SampleAxisRange	131
SampleColors	132
SampleDecimalCount	132
SampleHighlightImage	132
SampleHighlightOn	133
SampleHighlightSize	133
SampleHighlightStyle	134
SampleLabelAngle	134
SampleLabelColors	135
SampleLabelFont	135
SampleLabelsOn	135
SampleLabelStyle	136
SampleScrollerOn	136
SeriesLabelColors	137
SeriesLabelsOn	137
SeriesLabelStyle	138
SeriesLineOff	138
SingleClickURLOn	139
StackedOn	139
TargetLabelsPosition	140
TargetValueLine	140
ThousandsDelimiter	141

TitleFont	141
UrlTarget	141
ValueLabelAngle	142
ValueLabelFont	142
ValueLabelPostfix	143
ValueLabelPrefix	143
ValueLabelsOn	143
ValueLabelStyle	144
ValueLinesColor	144
ValueLinesOn	145
VisibleSamples	145
ZoomOn	146
Balkendiagramm-Eigenschaften	147
3DDepth	147
3DModeOn	147
AutoLabelSpacingOn	148
Background	148
BarAlignment	149
BarLabelAngle	149
BarLabelColors	150
BarLabelFont	150
BarLabelsOn	150
BarLabelStyle	151
BarOutlineColor	151
BarOutlineOff	152
BarType	152
BarWidth	153
ChartBackground	153
ChartForeground	154
ChartTitle	154
DefaultGridLinesColor	154
DefaultGridLinesOn	155
FloatingLabelFont	155
FloatingOnLegendOff	156
Font	156
Foreground	157
GraphInsets	157
GridAdjustmentOn	158
GridImage	158
GridLineColors	158
GridLines	159
GridLinesColor	159
Label_0	160
LabelUrl_0	160
LabelUrlTarget_0	161
LegendColors	161
LegendColumns	162
LegendFont	162
LegendImage	163
LegendOn	163
LegendPosition	164
LegendReverseOn	164
LowerRange	165
MaxValueLineCount	165
MultiColorOn	165
Range	166

RangeAdjusted	166
RangeAdjusterOn	167
RangeAdjusterPosition	167
RangeAxisLabel	168
RangeAxisLabelAngle	168
RangeAxisLabelFont	168
RangeColor	169
RangeDecimalCount	169
RangeLabelFont	170
RangeLabelPostfix	170
RangeLabelPrefix	170
RangeLabelsOff	171
RangeOn	171
RangePosition	172
RangeStep	172
SampleAxisLabel	173
SampleAxisLabelAngle	173
SampleAxisLabelFont	173
SampleAxisRange	174
SampleColors	174
SampleDecimalCount	175
SampleLabelAngle	175
SampleLabelColors	175
SampleLabelFont	176
SampleLabelSelectionColor	176
SampleLabelsOn	177
SampleLabelStyle	177
SampleScrollerOn	178
SeriesLabelColors	178
SeriesLabelFont	179
SeriesLabelsOn	179
SeriesLabelStyle	179
SingleClickURLOn	180
TargetValueLine	180
ThousandsDelimiter	181
TitleFont	181
UrlTarget	182
ValueLabelAngle	182
ValueLabelFont	183
ValueLabelPostfix	183
ValueLabelPrefix	183
ValueLabelsOn	184
ValueLabelStyle	184
ValueLinesColor	185
ValueLinesOn	185
ValueLinesOn	186
ZoomOn	186
Tortendiagramm-Eigenschaften	187
3DModeOn	187
Angle	187
Background	188
ChartTitle	188
Depth	189
DetachedDistance	189
DetachedSlices	189
FloatingLabelFont	190

Font	190
Foreground	191
GraphInsets	191
InsideLabelColor	192
InsideLabelFont	192
Label_0	192
LabelUrl_0	193
LabelUrlTarget_0	194
LegendColors	194
LegendColumns	195
LegendFont	195
LegendImage	195
LegendOn	196
LegendPosition	196
LegendReverseOn	197
OutsiteLabelColor	197
OutsideLabelFont	198
PercentDecimalCount	198
PercentLabelsOn	198
PercentLabelStyle	199
PieLabelFont	199
PieLabelsOn	200
PieRotationOn	200
PointingLabelColor	201
PointingLabelFont	201
SampleColors	202
SampleDecimalCount	202
SampleLabelColors.....	202
SampleLabelsOn	203
SampleLabelStyle	203
SelectionMode	204
SeriesLabelColors	204
SeriesLabelsOn.....	205
SingleClickURLOn.....	205
SliceSeperatorColor.....	206
SliceSeperatorOn	206
ThousandsDelimiter.....	206
TitleFont	207
UrlTarget	207
ValueLabelPrefix	208
ValueLabelsOn.....	208
ValueLabelStyle.....	209

Konfigurations-Eigenschaften

Datenbank.....	212
Catalog.....	212
Charset	212
Column-end bracket.....	213
Column-start bracket	213
Concat-Operator	214
Datasource	214
Drop IN with NULL	215
Escape Character	215
Group By Index	216
JDBC-Driver.....	217

Limit Syntax.....	217
Load Links	218
Load Table-Sizes	219
Max connection age	219
Max connection count.....	220
Max IN-Count.....	220
Name	221
Schema	221
Password	222
Single IN Operator	223
Table-end bracket.....	223
Table-start bracket	224
Table Names	224
Table Types	225
Timeout.....	226
URL.....	226
Use Aliases	227
User	228
Tabelle	230
Name	230
Spalte	231
Name	231
Type	231
Alias	233
Name	233
SQL-Where	233
Table.....	234
Link	235
Direction.....	235
Name	236
Link-Expression	237
Operator.....	237
Source Expression	237
Target Expression	238
CSV-Source	239
Delimiter	239
First Line As Names	239
Name	240
Quote.....	241
Startline	241
Trim	242
URL.....	242
CSV-Column	243
Column	243
Name	243
Type	244
Dimension	245
Cron Pattern	245
Default Text-Attribute	245
Name	246
Storage	247
Key.....	248
Format	248
ID.....	248
Type	249
Unit	250

Key-Attribute	251
Dimension	251
Name	251
Unique	252
Value	253
SQL-Keyloader	254
Database	254
Distinct.....	254
Format	255
Mode.....	255
Null-Value	256
Parent Format	257
Parent Search-Attribute	257
Parent SQL-Expression	259
Parent Trim.....	259
Recursive.....	260
SQL-Check.....	261
SQL-Expression	262
SQL-Order	263
SQL-Where	263
Trim.....	264
SQL-Attribute	265
Dimension	265
Format	265
Ignore Missing Targets	266
Name	267
Null-Value	267
Relink-Attribute	267
SQL-Expression	268
Target Attribute.....	269
Trim.....	270
Unique	270
CSV-Keyloader	272
Column	272
CSV-Source	272
Format	273
Levelname	273
Mode.....	274
Null-Value	274
Parent-Column	275
Parent-Format	276
CSV-Attribute.....	277
Column	277
Dimension	277
Format	278
Name	279
Null-Value	279
Unique	280
Time-Keyloader	281
End	281
Endshift.....	282
Exclude Pattern	282
Exclude Values	283
Locale Format	284
Mode.....	284
Parent Pattern	285

Pattern	286
Start	287
Startshift	288
Time-Attribute.....	289
Dimension	289
Name	289
Pattern	290
Relink-Attribute	290
Unique	291
Number-Keyloader.....	293
Max	293
Min	293
SQL-Cube	295
Complete	295
Database	296
Enable Load	296
Enable Store	297
Line-Dimension	298
Match	298
Name	299
Span All Dimensions.....	299
SQL-Order	300
SQL-Where	301
SQL-Fact	302
Fact	302
Match	302
SQL-Expression	303
SQL-Dimension	304
Attribute	304
Dimension	305
Format	306
Null-ID	306
Omit-Percentage.....	307
SQL-Expression	308
CSV-Cube	309
Complete	309
CSV-Source	310
Line-Dimension	310
Match	311
Span All Dimensions.....	311
CSV-Fact	313
Column	313
Fact	313
Match	313
CSV-Dimension	315
Attribute	315
Column	315
Dimension	316
Format	317
Null-Value	317
Formel	318
Expression	318
Fact	318
Match	319
Memory-Cache	321
Cron-Pattern	321

Match	321
Max Age	322
Max Size	323
File-Cache	324
Cron-Pattern	324
Filename	324
Match	325
Max Age	326
Include	327
Model	327

Die Formelsprache

329

Das Typsystem	330
All	331
Any	331
Boolean	331
Double	331
Integer	331
Key	332
Number	332
String	332
Value	332
Syntax	333
Dimensionen und Selektionen	333
Dimensions-Ebenen	333
Operatoren	334
Klammern	336
Zugriff auf Attribute	337
Zugriff auf Variablen	337
Funktionsaufrufe	340
Ebenen-Funktionen	340
Kennzahl-Funktionen	341
Konstanten	342
Boolean-Konstanten	342
Integer-Konstanten	342
Double-Konstanten	343
NULL	343
String-Konstanten	344
Key-Konstanten	345
Funktionen	346
ABC	346
ABS	346
ADD	347
ALL	348
ANCESTORS	349
AND	350
ATTRIBUTENAMES	351
ATTRIBUTENV	352
ATTRIBUTES	352
AVG	353
BEAUTIFY	354
CEIL	354
CHILDREN	355
CLUSTER	356
CONCAT	357

CONTAINS	358
COUNT	359
CUBE	359
DEPTH	362
DIMENSIONATTRIBUTENAMES	362
DIMENSIONNAME	363
DIMENSIONNAMES	363
DISTINCT	364
DIV	365
DLOOKUP	366
DRILLLEVEL	367
EMPTY	367
ENDSWITH	368
EQUAL	369
EVAL	369
EXISTS	370
EXP	371
FACTROOT	371
FAMILY	372
FILTER	373
FIND	373
FIRST	374
FLOOR	375
FOREACH	376
GREATER	376
GREATER_OR_EQUAL	377
HASKEYS	378
HASLEVEL	379
HASPOSITION	380
HASROLES	380
HASUSER	381
IIF	382
IN	383
INTERSECT	383
ISCHILD OF	384
ISNULL	385
ISPARENT OF	386
JOIN	386
LAST	387
LEAFS	388
LEFT	389
LESS	390
LESS_OR_EQUAL	391
LEVEL	392
LEVELNAMES	393
LEVEL OF	393
LIMIT	394
LOOKUP	395
MATCH	396
MATRIX	397
MAX	398
MAX_X	398
MAX_Y	399
MIN	400
MIN_X	400
MIN_Y	401

MOD	402
MUL	402
NEIGHBOURS	403
NEXT	404
NONFACTROOTS	405
NONLEAFS	406
NOT	407
NOW	408
OR	408
PARENT	409
PEDIGREE	410
POSITIONOF	411
PREV	412
REPLACE	413
REVERSE	413
RIGHT	414
ROUND	415
SORT	415
SPLIT	417
STARTSWITH	417
STRLEN	418
SUB	419
SUBSTR	419
SUM	420
TIMESTAMP	421
TOLOWER	422
TONUMBER	422
TOSTRING	423
TOUPPER	424
UNEQUAL	425
UPPERNEXT	426
UPPERPREV	427
USER	428
WITHOUT	428
X	429
XHEADER	430
Y	431
YHEADER	431
ZERO	432

Formate **435**

Cron-Patterns.....	436
Datums-Formate	437
Farben	439
Zahlen-Formate.....	440

SQL-Ausdrücke **441**

Tabellen und Spalten	443
Konstanten	444
Operatoren	445
Funktionen	447
Aggregations-Funktionen	448
SQL Passthrough	450
Klammern.....	452

Index **453**

KAPITEL 1

Konventionen

Bevor Sie diese Dokumentation lesen, sollten Sie die folgenden Hervorhebungen und Konventionen verstehen. Folgende Konventionen werden verwendet:

Konvention	Art der Informationen
Dreieck (➤)	Schritt-für-Schritt Anleitung. Sie können diesen Anweisungen folgen um eine spezielle Aufgabe zu erledigen.
Fettdruck	Oberflächen-Element die Sie auswählen müssen, z.B. Menüpunkte, Buttons oder Elemente in einer Liste.
Klammern (<>)	Wird verwendet, um Ausdrücke und Parameter zu kennzeichnen.
Monospace-Schrift	Kennzeichnet Code-Beispiele oder die Syntax eines Ausdrucks.
<i>Kursivdruck</i>	Hervorhebung eines besonders wichtigen Punkts.
GROSSSCHRIFT	Namen von Tasten. Z.B. SHIFT, CTRL, oder ALT.
TASTE+TASTE	Tastenkombinationen, bei denen der Benutzer eine Taste gedrückt halten und danach die andere drücken muss, z.B. CTRL+P, oder ALT+F4.

KAPITEL 2

Berichts-Eigenschaften

In diesem Kapitel

Bericht	18
Select.....	28
Äußere Blöcke	35
Innere Blöcke	38
Abfragen.....	52
Überschriften.....	63
Kommentare	102

Bericht

Author

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Dieses Feld speichert den Namen des Berichts-Autors. Dieses Feld kann einen beliebigen Text enthalten und erwartet keine Ausdruck (Formel). Wenn über den Berichts-Wizard aus dem Web-Frontend oder der Workbench ein neuer Bericht erstellt wird, dann wird dieses Feld automatisch befüllt.

Der Autor eines Berichts wird in der Index-Seite des Web-Frontend unterhalb des Berichts-Titels angezeigt.

Beispiele

```
Author = "admin"
```

Siehe auch

Date (auf Seite 19)

CSS

Typ

String

Seit

1.2

Beschreibung

Wenn das Standard STX-Stylesheet (welches über die Eigenschaft Stylesheet definiert wird) verwendet wird, dann bestimmt die CSS-Datei "/iolap/stylesheets/iolap.css" die Farben, Zeichenstätze und Masse des Web-Frontends. Über diese Eigenschaft können Sie ein anderes CSS-Stylesheet für diesen Bericht verwenden und damit das Aussehen des Berichtes verändern.

Beispiele

```
CSS = "'/iolap/stylesheets/green.css'"
```

Siehe auch

Stylesheet (auf Seite 26)

Date

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Dieses Feld enthält den Zeitpunkt der Berichtserstellung. Das Feld erwartet eine einfache Zeichenkette (keine Formel) und kann jeden beliebigen Wert enthalten. Wenn Sie einen neuen Bericht mit dem Berichts-Wizard über das Web-Frontend oder über die Workbench erstellen, dann wird dieses Feld automatisch mit dem aktuellen Datum vorbelegt.

Das Erstellungsdatum eines Berichts wird in der Index-Seite des Web-Frontend unterhalb des Berichts-Titels angezeigt.

Beispiele

```
Date = "01.04.2004 10:30:00"
```

Siehe auch

Author (auf Seite 18)

Description

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Dieses Feld kann eine Beschreibung des Berichtes enthalten. Die Beschreibung eines Berichtes wird auf der Index-Seite im Web-Frontend unter dem Titel dargestellt.

Beispiele

```
Description = "Umsatzbericht für alle Produkte"
```

Siehe auch

Date (auf Seite 19), Author (auf Seite 18)

Filter

Typ

Key

Seit

1.2

Beschreibung

Diese Eigenschaft setzt den globalen Filter für eine Abfrage. Ein Filter besteht aus einer Menge von Schlüsseln (mit keinem, einem oder mehreren Schlüsseln per Dimension). Alle folgenden Formeln innerhalb der Abfrage, die sich auf die aktuelle Selektion (Filterung) einer Dimension beziehen, werden die in diesem Feld oder späteren Filtern definierten Schlüssel ergeben.

Wenn eine Dimension gefiltert wird, bedeutet das nicht automatisch, dass Formeln innerhalb des Berichts nicht auf andere Schlüssel zugreifen können. Z.B. würde bei einem Filter "Produkt:ProduktA" die Formel "Produkt" genau dieses gefilterte Produkt "ProduktA" ergeben. Eine Formel "NEXT(Produkt)" würde jedoch auch ein Ergebnis haben (ProduktB), obwohl dieses Produkt nicht im Filter enthalten ist.

Beispiele

```
Filter = "Produkt:ProduktA"
```

Setzt den Filter auf Produkt:A

Formats

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Die Eigenschaft "Formats" enthält eine (mit Kommata getrennte) Liste der möglichen Export-Formate für diesen Bericht. Über dieses Feld können Sie beeinflussen, welche Export-Formate dem Benutzer nach der erfolgreichen Ausführung eines Berichtes zur Verfügung stehen.

Mögliche Formate sind:

- HTML
- PDF
- EXCEL
- CSV

Beispiele

```
Formats = "HTML,PDF"
```

```
Formats = "HTML,EXCEL,CSV"
```

Icon

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Setzen Sie diese Eigenschaft um ein anderes Symbol für diesen Bericht in der Index-Seite des Web-Frontends anzuzeigen.

Das Icon kann sowohl als relativ (z.B. "mylogo.gif") oder absolut (z.B. "/iolap/mydemo/icon.gif") angegeben werden. Bedenken Sie, dass relative Angaben evtl. nicht mehr funktionieren, wenn ein Bericht gespeichert wurde und dadurch seine Lage auf dem Server ändert.

Im Gegensatz zum Logo eines Berichtes erwartet diese Eigenschaft eine einfache Zeichenkette (keine Formeln), denn das Icon wird bereits vor dem Ausführen des Berichtes ausgelesen und dargestellt.

Beispiele

```
Icon = "icon.gif"
```

Siehe auch

Logo (auf Seite 22)

Logo

Typ

String

Seit

1.2

Beschreibung

Um ein anderes Logo für Ihren Bericht zu verwenden müssen Sie diese Eigenschaft verwenden. Das Standard-Logo eines Berichtes ist "/iolap/logo.gif".

Das Logo kann sowohl als relativ (z.B. "mylogo.gif") oder absolut (z.B. "/iolap/mydemo/icon.gif") angegeben werden. Bedenken Sie, dass relative Angaben evtl. nicht mehr funktionieren, wenn ein Bericht gespeichert wurde und dadurch seine Lage auf dem Server ändert.

Beispiele

```
Logo = "'mylogo.gif'"
```

```
Logo = "IIF( HASROLE( 'manager' ), 'manager.gif',  
'logo.gif' )"
```

Siehe auch

Icon (auf Seite 21)

Model

Typ

Konstante Zeichenkette (Name des Modells)

Beschreibung

Die notwendige Eigenschaft "Model" bestimmt das Modell einer Abfrage. Alle Dimension und Kennzahlen, die von einer Abfrage verwendet werden können, werden über das Modell bestimmt - deswegen ist dies die wichtigste Eigenschaft einer Abfrage.

Jedes Modell wird durch eine Konfiguration (eine Datei mit der Endung ".config") definiert und die Lage der Konfiguration im Repository bestimmt den genauen Modell-Namen. Z.B. generiert eine Konfiguration namens "demo.config" in einem Verzeichnis "demo" das Modell "/demo/demo".

Sie können Modell von Ihrem Bericht aus absolut oder relativ referenzieren. Absolute Pfade beginnen mit einem "/", relative mit einem "../" (für das darüberliegende Verzeichnis) oder einem Modell- bzw. Verzeichnisnamen. Z.B. würde der Wert "/demo/demo" das Modell "demo" aus dem Wurzelverzeichnis "demo" verwenden und ein Wert "config/demo" das Modell "demo" aus einem Unterverzeichnis "config".

Beispiele

Model = "demo"

Verweist auf das Modell "demo" aus dem selben Verzeichnis

Model = "/demo/demo"

Verweist auf das Modell "demo" aus dem Wurzelverzeichnis "demo"

Model = "../demo"

Verweist auf das Modell "demo" aus dem Verzeichnis oberhalb des Berichtes

Model = "config/demo"

Verweist auf das Modell "demo" aus dem Unterverzeichnis "config"

Name

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Die "Name" Eigenschaft setzt den Namen eines Berichtes für die Anzeige in der Navigation und Index-Seite des Web-Frontends. Im Gegensatz zum Titel eines Berichtes erwartet der Namen nur eine beliebige Zeichenkette (keine Formel), da er bereits vor der Ausführung des Berichtes angezeigt wird.

Beispiele

```
Name = "Sales overview"
```

PrintHeight

Typ

String

Beschreibung

Beim Export eines Berichtes in das PDF-Format bestimmt diese Eigenschaft die Seitenhöhe des generierten PDF-Dokumentes. Diese Eigenschaft erwartet eine Formel, die die Seitenhöhe als Zeichenkette im Format `<zahl><format>` zurückgibt. Gültige Formate sind "cm", "in", "mm" oder "px" für Pixel.

Beispiele

```
Print Height = "'50cm'"
```

Siehe auch

Print Logo (auf Seite 24), Print Width (auf Seite 25)

PrintLogo

Typ

String

Seit

2.1

Beschreibung

Wenn Sie einen Bericht in das druckbare PDF-Format exporting bestimmt diese Eigenschaft das Logo, dass in der generierten PDF-Datei dargestellt wird. Das Logo kann sowohl als relativer (z.B. "mylogo.gif") oder absoluter (z.B. "/iolap/mydemo/logo.gif") Pfad angegeben werden. Bedenken Sie, dass relative Angaben evtl. nicht mehr funktionieren, wenn ein Bericht gespeichert wurde und dadurch seine Lage auf dem Server ändert.

Wenn keine "Print Logo" definiert wurde verwendet der Bericht sein Standard-Logo (definiert über die Eigenschaft Logo) für den Export.

Beispiele

```
Print Logo = "'mylogo.gif'"
```

Siehe auch

Logo (auf Seite 22), Print Height (auf Seite 24), Print Width (auf Seite 25)

PrintWidth

Typ

String

Beschreibung

Beim Export eines Berichtes in das PDF-Format bestimmt diese Eigenschaft die Seitenbreite des generierten PDF-Dokumentes. Diese Eigenschaft erwartet eine Formel, die die Seitenhöhe als Zeichenkette im Format <zahl><format> zurückgibt. Gültige Formate sind "cm", "in", "mm" oder "px" für Pixel.

Beispiele

```
Print Width = "'40cm'"
```

Siehe auch

Print Height (auf Seite 24), Print Logo (auf Seite 24)

Stylesheet

Typ

String

Seit

1.2

Beschreibung

Die Eigenschaft "Stylesheet" bestimmt das STX-Stylesheet, das für die Transformation des Berichtes in eine HTML-Seite (oder andere Format) verwendet wird. Das Standard-Stylesheet für die Transformation ist "html.stx".

Wenn Sie diese Eigenschaft ändern können Sie Ihre eigenes Stylesheet zur Darstellung des Berichtes verwenden und ihm ein eigenes "Look&Feel" verwenden. Das Stylesheet erwartet einen relativen Pfad zum Verzeichnis "stylesheets" des Repositories, in dem alle Stylesheets des Systems abgelegt werden.

Beispiele

```
Stylesheet = "'small.stx"
```

Siehe auch

CSS (auf Seite 18)

Title

Typ

String

Seit

1.0

Beschreibung

Der Titel eines Berichtes wird über eine Formel bestimmt, die in dieser Eigenschaft hinterlegt wird. Im Moment der Berichtsausführung wird diese Formel berechnet und das Ergebnis als Überschrift für das Ergebnis verwendet.

Im Titel können Sie aktuelle Filter usw. verwendet und so lange, sprechende Titel erzeugen die sich auf die Auswahl des Benutzers beziehen. Wenn ein Bericht gespeichert wird, dann wird dieser Titel als Standardname für das gespeicherte Ergebnis verwendet.

Beispiele

```
Title = "'Sales report for ' + TOSTRING( Time )"
```

Siehe auch

Name (auf Seite 24)

Visible

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

1.2

Beschreibung

Die Eigenschaft "Visible" bestimmt, ob ein Bericht für den Benutzer in der Navigation sichtbar ist oder nicht. Als Standard-Einstellung ist jeder Bericht sichtbar.

Wenn Sie diese Eigenschaft auf "false" setzen wird der Bericht in der Navigation für den Benutzer unsichtbar, kann aber dennoch über andere Berichte aufgerufen werden, die mit dem unsichtbaren Bericht verknüpft sind. Setzen Sie z.B. Berichte auf unsichtbar, die Sie als Detailberichte hinter andere Berichte schalten möchten.

Beispiele

```
Visible = "false"
```

Select

Default

Typ

Value

Seit

1.2

Beschreibung

Mit diesen Eigenschaften können Sie die Standard-Auswahl eines Selektors bestimmen. Alle Schlüssel, die von der in dieser Eigenschaft definierten Formel als Ergebnis zurückgegeben werden, sollten Teil der Optionen sein, ansonsten wird vom System eine Fehlermeldung produziert.

Wenn der Benutzer mit dem Selektor nur einen Schlüssel auswählen kann (z.B. bei einem Single- oder Radio-Selektor), dann sollte die Default-Formel nur einen Schlüssel zurückgeben. Ansonsten können Sie mehrere Schlüssel als Default definieren, die dann alle im Selektor vorselektiert werden.

Wenn Sie den Default für einen Interval-Selektor bestimmen wollen, dann muss Ihre Formel zwei Schlüssel als Ergebnis haben. Der erste bestimmt dann den Von- und der zweite den Bis-Wert.

Beschreibung

```
Default = "Product:ProductA"
```

```
Default = "LAST( LEVEL( Time, 2 ) )"
```

```
Default = "5"
```

Siehe auch

Options (auf Seite 30)

Height

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt die Höhe des Selektors in Pixeln (die Eigenschaft erwartet eine Formeln, die eine Zahl als Ergebnis hat). Wenn Sie keine Höhe über diese Eigenschaft definieren, dann wird diese automatisch vom Frontend berechnet.

Beispiele

```
Height = "30"
```

Siehe auch

Width (auf Seite 34)

Name**Typ**

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt den Namen eines Selektor. Der Name eines Selektors bestimmt den Namen der Variablen, unter dem Sie innerhalb des Berichtes den ausgewählten Wert (z.B. die gefilterte Dimension) verwenden können.

Wenn Sie eine Dimension durch einen Selektor filtern möchten, dann müssen Sie keinen Namen für den Selektor vergeben, denn dieser wird automatisch vergeben (der Name der Dimension der ersten Option wird als Selektor-Name verwendet).

Beispiele

```
Name = "Y-Dimension"
```

Siehe auch

Options (auf Seite 30)

Options

Typ

Value

Seit

1.0

Beschreibung

Jeder Selektor besitzt eine bietet dem Benutzer eine Menge von Optionen, aus denen er sich eine oder mehrere aussuchen (bzw. darin suchen) kann. Mit dieser Eigenschaft können Sie die Optionen bestimmen, in dem Sie eine Formel hinterlegen.

Wenn die in dieser Eigenschaft hinterlegte Formel Dimension-Schlüssel als Ergebnis hat, dann wird der Name des Selektor automatisch gesetzt und die entsprechende Dimension durch diesen Selektor gefiltert. Wenn Sie andere Werte als Schlüssel in den Optionen bestimmen, dann müssen Sie den Namen zusätzlich angeben, unter dem der Bericht die Auswahl dann als Variable referenzieren kann.

Beispiele

```
Options = "LEVEL( Time, 2 )"
```

```
Options = "10 | 20 | 30"
```

Siehe auch

Default (auf Seite 28), Name (auf Seite 29)

Submit

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft auf "true" gesetzt wird (bzw. wenn die in dieser Eigenschaft hinterlegte Formel true ergibt), dann wird der Bericht sofort und automatisch aktualisiert, nachdem die Auswahl dieses Selektors verändert wurde. Ansonsten muss der Benutzer den Button "Aktualisieren" verwenden, um die veränderten Selektionen sichtbar zu machen.

Beispiele

```
Submit = "true"
```

```
Submit = "false"
```

Text

Typ

String

Seit

1.1

Beschreibung

Normalerweise werden die Optionen eines Selektors mit ihrem Standard-Text dargestellt (z.B. wird für Dimensions-Schlüssel die ID der Schlüssel dargestellt). Wenn Sie andere Texte anzeigen möchten, dann können Sie in dieser Eigenschaft eine Formel hinterlegen, die den Text für jede Option berechnet (wenn in den Optionen Schlüssel vorkommen, wird ein entsprechender Filter an die Formel übergeben, d.h. Sie können sich in der Formel auf den aktuellen Schlüssel über seinen Dimensionsnamen beziehen).

Beachten Sie, dass durch diese Eigenschaft nur der Text im Selektor verändert wird, jedoch nicht der ausgewählte Wert!

Beispiele

```
Text = "LIMIT( Product, 10 )"
```

```
Text = "Product.ProductID + ':' + TOSTRING( Product )"
```

Siehe auch

Title (auf Seite 32)

Title

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie die Überschrift eines Selektors (der Text oberhalb des Selektors) ändern. Wenn Sie in dieser Eigenschaft keine Formel angeben, dann wird der Name des Selektors bzw. der Name der gefilterten Dimension als Titel verwendet.

Beispiele

```
Title = "'Prod. No'"
```

Siehe auch

Name (auf Seite 29), Text (auf Seite 31)

Type

Typ

Konstante Zeichenkette ('single', 'multiple', 'radio', 'checkbox', 'hierarchy', 'button', 'input' oder 'constant')

Seit

1.1

Beschreibung

Es gibt verschiedene Arten von Selektoren, von der Sie eine über diese Eigenschaft für den Selektor setzen können:

- **single:** Der Selektor ist eine einfache Drop-Down Box (bekannt aus den Browsers). Der Benutzer kann nur eine Option auswählen.
- **multiple:** Der Selektor wird als Liste dargestellt, aus dem der Benutzer mehrere Optionen auswählen kann.
- **radio:** Die Optionen werden als Radio-Buttons dargestellt und der Benutzer kann nur eine Option auswählen.
- **checkbox:** Die Optionen werden als Checkboxes (Häkchen) dargestellt aus denen der Benutzer mehrere auswählen kann.

- **input:** Der Benutzer kann über ein Suchfeld mit Hilfe von Wildcards (* und ?) innerhalb der Optionen suchen und ggf. auch mehrere finden.
- **constant:** Der Selektor ist unveränderlich und wird als konstanter Text dargestellt.
- **hierarchy:** Der Benutzer kann innerhalb einer Dimensions-Hierarchy navigieren und sich einen Schlüssel auswählen.
- **button:** Der Benutzer kann eine Option über einen Button direkt auswählen.

Beispiele

Type = "single"

Type = "multiple"

Type = "radio"

Type = "checkbox"

Type = "input"

Type = "constant"

Type = "button"

Type = "hierarchy"

Use Filter

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Siehe auch

2.1

Beschreibung

Wenn die Eigenschaft "Use Filter" auf "true" gesetzt wird, dann verwenden alle Formeln des Selektors (z.B. Options oder Default) den Filter der Abfrage (der z.B. von den vorhergehenden Selektoren oder den Abfrageparametern beeinflusst wird). Ansonsten verwenden alle Eigenschaften des Selektor des Root-Filter (alle Dimensionen sind auf Ihre Root-Keys gesetzt).

In der Standardeinstellung verwenden Selektoren den Root-Filter.

Beispiele

Use Filter = "true"

```
Use Filter = "false"
```

Width

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt die Breite des Selektors in Pixeln (die Eigenschaft erwartet eine Formeln, die eine Zahl als Ergebnis hat). Wenn Sie keine Breite über diese Eigenschaft definieren, dann wird diese automatisch vom Frontend berechnet.

Beispiele

```
Width = "100"
```

Siehe auch

Height (auf Seite 28)

Äußere Blöcke

Filter

Typ

Key

Seit

1.2

Beschreibung

Jedes Element einer Abfrage besitzt einen Filter, der alle Formeln in den darunter enthaltenen Elementen beeinflusst. Ein Filter ist eine Sammlung von Schlüsseln mit jeweils keinem, einem oder mehreren Schlüsseln per Dimension. Alle Formeln von verschachtelten Elementen, die sich auf eine gefilterte Dimension beziehen, werden die entsprechenden Schlüssel aus dem Filter als Ergebnis haben.

Mit der Eigenschaft "Filter" eines äußeren Blocks können Sie diesen Filter beeinflussen. Die Eigenschaft erwartet eine Formel vom Ergebnistyp "Key", dessen Ergebnis auf den Filter angewandt werden. "Angewandt" heisst, dass für alle Dimensionen die im Ergebnis vorkommen die Schlüssel gesetzt werden. Alle anderen Dimensionen bleiben unbeeinflusst.

Beispiele

```
Filter = "Produkt:ProduktA"
```

Jede Formel, die sich auf "Produkt" bezieht, wird jetzt "ProduktA" als (Teil-) Ergebnis haben

Iteration

Typ

Key

Seit

1.1

Beschreibung

Die meisten Elemente eines Berichtes können durch das Anwenden einer "Iteration" automatisch wiederholt werden. Um ein Element zu wiederholen, müssen Sie eine Formel vom Rückgabebetyp "Key" in der Eigenschaft "Iteration" angeben. Dann wird das Element für jeden Schlüssel aus dem Ergebnis der Formeln einmal wiederholt (wenn das Ergebnis z.B. 5 Schlüssel enthält, dann wird der Block 5 mal wiederholt). Wenn die Formel kein Ergebnis oder NULL zurückgibt, dann wird Block nicht dargestellt.

Durch eine Iteration wird ein Block nicht nur wiederholt, sie beeinflusst ausserdem auch den Filter jedes einzelnen generierten Blocks, Der Filter eines wiederholten Blocks setzt sich zusammen durch den äusseren Filter und dem einen Schlüssen aus der Wiederholung des Blocks. Alle Dimensionen aus dem äusseren Filter werden übernommen bis auf die Dimension des wiederholten Schlüssels.

Innerhalb des Wiederholten Elements können Sie sich dann über den Namen dieser Dimension auf den entsprechenden Schlüssel beziehen.

Beispiele

```
Iteration = "CHILDREN( Product )"
```

Wiederholt den Block für jedes Produkt und setzt den Filter für jeden Block. Verschachtelte Element können sich auf das aktuelle Produkt beziehen.

Orientation

Typ

Konstante Zeichenkette ('horizontal', 'vertical' oder 'tabbed')

Seit

2.0

Beschreibung

Die Eigenschaft "Orientation" eines äusseren Blocks definiert, wie seine inneren Blöcke arrangiert werden. Es gibt drei verschiedene Möglichkeiten:

- **horizontal:** Alle inneren Blöcke werden nebeneinander dargestellt (Standardeinstellung)
- **vertical:** Alle inneren Blöcke werden untereinander dargestellt
- **tabbed:** Alle inneren Blöcke werden, jeweils mit einem eigenen Reiter versehen, hintereinander dargestellt und nur einer ist zur Zeit sichtbar

Beispiele

Orientation = "horizontal"

Orientation = "vertical"

Orientation = "tabbed"

Innere Blöcke

Background

Typ

String

Seit

2.1

Beschreibung

Die Eigenschaft "Background" bestimmt die Hintergrundfarbe des inneren Blocks. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Background = "'red'"
```

Block title	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Border (auf Seite 38), Color (auf Seite 39)

Border

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt die Farbe des Rahmens und des Titel-Hintergrundes für den inneren Block. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Border = "'#FF0000'"
```

Block title	
	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Background (auf Seite 38), Color (auf Seite 39)

Color

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft setzt die Zeichenfarbe des Block-Titels. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Color = "'red'"
```

Block title	
	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Background (auf Seite 38), Border (auf Seite 38)

Filter

Typ

Key

Seit

1.2

Beschreibung

Jedes Element einer Abfrage besitzt einen Filter, der alle Formeln in den darunter enthaltenen Elementen beeinflusst. Ein Filter ist eine Sammlung von Schlüsseln mit jeweils keinem, einem oder mehreren Schlüsseln per Dimension. Alle Formeln von verschachtelten Elementen, die sich auf eine gefilterte Dimension beziehen, werden die entsprechenden Schlüssel aus dem Filter als (Teil-) Ergebnis haben.

Mit der Eigenschaft "Filter" eines inneren Blocks können Sie diesen Filter beeinflussen. Die Eigenschaft erwartet eine Formel vom Ergebnistyp "Key", dessen Ergebnis auf den Filter angewandt werden. "Angewandt" heisst, dass für alle Dimensionen die im Ergebnis vorkommen die Schlüssel gesetzt werden. Alle anderen Dimensionen bleiben unbeeinflusst.

Beispiele

```
Filter = "Produkt:ProduktA"
```

Jede Formel, die sich auf "Produkt" bezieht, wird jetzt "ProduktA" als (Teil-) Ergebnis haben

Font

Typ

String

Seit

2.1

Beschreibung

Die Eigenschaft "Font" setzt den Zeichensatz der Block-Überschrift. Die Eigenschaft muss eine Formel enthalten, die den Namen des Zeichensatzes als Zeichenkette zurückgibt. Wenn kein Zeichensatz angegeben wurde oder diese Formel NULL zurückgibt, dann wird der Standard-Zeichensatz für Überschriften verwendet.

Beispiele

Font = "'Courier New'"

Block title	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Font Size (auf Seite 42), Font Weight (auf Seite 41)

FontWeight

Typ

String

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie Fettdruck für die Überschrift ein- oder ausschalten. Mögliche Rückgabewerte für eine hier enthaltene Formel sind 'bold' und 'lighter'.

Beispiele

Font Width = "'lighter'"

Stellt die Überschrift normal dar

Font Width = "'bold'"

Stellt die Überschrift in Fettdruck dar:

Block title	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Font (auf Seite 40), Font Size (auf Seite 42)

FontSize

Typ

Integer

Seit

1.1

Beschreibung

Mit dieser Eigenschaft bestimmen Sie die Grösse des Titel-Zeichensatzes in Pixeln. Wenn Sie keine Font Size bestimmen oder die Formel NULL ergibt, dann wird die Standardgrösse für Überschriften verwendet.

Beispiele

```
Font Size = "20"
```

Setzt die Grösse des Titel-Fonts auf 20 Pixel:

Block title	
	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Siehe auch

Font (auf Seite 40), Font Weight (auf Seite 41)

Format

Typ

Konstante Zeichenkette ('table', 'easychart', 'easychartservlet', 'newsticker' oder eigenes Format)

Seit

1.1

Beschreibung

Die Eigenschaft "Format" bestimmt das Ausgabe-Format für die in diesem Block enthaltenen Abfragen. Die folgenden Formate stehen zur Verfügung:

- **table:** Die Ergebnisse werden als Pivot-Tabelle dargestellt

- **easychart:** Die Ergebnisse werden als Diagramm dargestellt (unter Verwendung des Diagramm-Applets)
- **easychartservlet:** Die Ergebnisse werden als Diagramm dargestellt (als PNG-Grafik)
- **newsticker:** Die Ergebnisse werden in Form eines News-Tickers dargestellt

table

Das Format "table" stellt die Ergebnisse in Form einer Pivot-Tabelle dar. Jede Zeile des Ergebnisses wird zu einer Zeile in der Pivot-Tabelle und jede Spalte wird zu einer Spalte. Für Pivot-Tabellen müssen Sie keine Grössenangaben im inneren Block machen, da die Grösse der Tabelle automatisch berechnet werden kann.

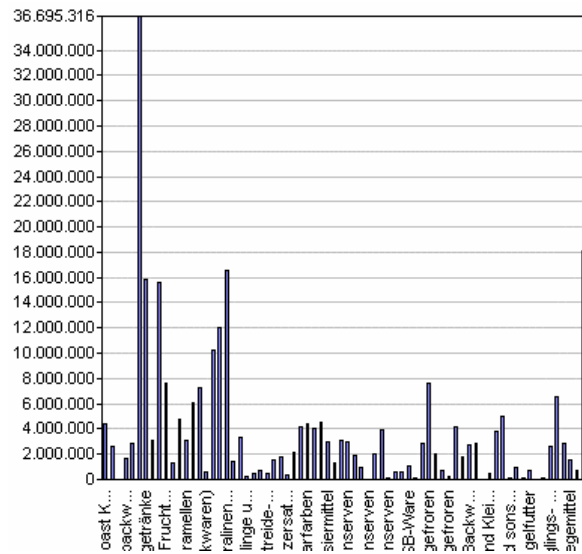
	Absatz
Brotlaibe - Selbstbedienungsware (ohne Toast, K...	4.369.619,449
Schnittbrot - Selbstbedienungsware	2.600.070,619
Toastbrot, Knäckebrot und sonst. Trockenflachbr...	970,614
Torten, Kuchen, Teilchen und sonstige Feinbackw...	1.636.342,142
\Toastbrot, Knäckebrot u. sonstiges Trockenflac...	2.906.375,93
Biere	36.695.316,28
Erfrischungsgetränke	15.902.631,217
Fruchtsaftgetränke	3.052.072,704

Pivot-Tabllen können folgende Elemente von Abfragen darstellen:

- Texte und Werte
- Drilldown
- Links und Zellen-Links
- Ampelsignale (assertions)
- Grafiken in Zellen
- Rotierte Überschriften

easychart and easychartservlet

Die Diagramm-Formate "easychart" und "easychartservlet" stellen das Ergebnis in Form eines Diagramms dar. Das Ergebnis wird automatisch in eine Grafik umgesetzt, jede Spalte aus dem Ergebnis wird ein Wert und jede Zeile aus dem Ergebnis wird eine eigene Zahlenreihe. Da Diagramme immer eine Grössenangabe benötigen, müssen Sie die Höhe und Breite des inneren Blocks bestimmen, ansonsten wird das Diagramm unsichtbar sein.



Es gibt zwei verschiedene Formen der Diagramm-Darstellung: Als Java-Applet (easychart) oder als vom Server generierte PNG-Grafik (easychartservlet). Server-seitig generierte Grafiken sind etwas langsamer als Applet, da das Bild zusätzlich vom Server zum Browser übertragen werden muss. Für die Applet-Darstellung benötigt der Betrachter einen Java 1.1 fähigen Browser (was für die meisten Browser zutrifft).

Für Diagramme werden die folgenden Elemente der Ergebnisse umgesetzt:

- Texte und Zahlen
- Links und Zellen-Links (die Links in Diagrammen können für die Umsetzung eines Drill-Downs verwendet werden)

newsticker

Das "newsticker" Format stellt das Ergebnis in Form eines News-Tickers dar. Jede Zeile des Ergebnisses wird (ohne die Überschriften) in eine Nachricht umgesetzt. Wenn das Ergebnis mehrere Zeilen enthält, dann werden daraus entsprechend viele Nachrichten.

Top 5

Offreie Getränke, Saftbar in Bedienung 7.969.450,85 DEM +++ KW 52/19

Für News-Ticker werden die folgenden Elemente der Ergebnisse umgesetzt:

- Texte und Zahlen
- Zellen-Links

Beispiele

```
Format = "table"
```

```
Format = "easychart"
```

```
Format = "easychartservlet"
```

```
Format = "Newsticker"
```

Siehe auch

Height (auf Seite 45), Width

Height

Typ

Integer

Seit

1.1

Beschreibung

Mit dieser Eigenschaft bestimmen Sie die Höhe des Blocks in Pixeln. Beachten Sie, dass Sie für einige Block-Formate (Diagramme) die Masse des Blocks definieren müssen, da diese sonst nicht dargestellt werden.

Beispiele

```
Height = "300"
```

Siehe auch

Format (auf Seite 42), Width (auf Seite 51)

Iteration

Typ

Key

Seit

1.1

Beschreibung

Die meisten Elemente eines Berichtes können durch das Anwenden einer "Iteration" automatisch wiederholt werden. Um ein Element zu wiederholen, müssen Sie eine Formel vom Rückgabebetyp "Key" in der Eigenschaft "Iteration" angeben. Dann wird das Element für jeden Schlüssel aus dem Ergebnis der Formeln einmal wiederholt (wenn das Ergebnis z.B. 5 Schlüssel enthält, dann wird der Block 5 mal wiederholt). Wenn die Formel kein Ergebnis oder NULL zurückgibt, dann wird Block nicht dargestellt.

Durch eine Iteration wird ein Block nicht nur wiederholt, sie beeinflusst ausserdem auch den Filter jedes einzelnen generierten Blocks, Der Filter eines wiederholten Blocks setzt sich zusammen durch den äusseren Filter und dem einen Schlüssen aus der Wiederholung des Blocks. Alle Dimensionen aus dem äusseren Filter werden übernommen - bis auf die Dimension des wiederholten Schlüssels.

Innerhalb des Wiederholten Elements können Sie sich dann über den Namen dieser Dimension auf den entsprechenden Schlüssel beziehen.

Beispiele

```
Iteration = "LEVEL( Product, 1 )"
```

Wiederholt den Block für jedes Produkt und setzt den Filter für jeden Block. Verschachtelte Element können sich auf das aktuelle Produkt beziehen.

Link

Typ

String

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie einen Link für den inneren Block definieren. Der Link wird auf die Adresse gesetzt, die von der in der Eigenschaft angegebenen Formel zurückgegeben wird. Die Adresse muss eine gültige URL sein (es sind sowohl relative als auch absolute URLs erlaubt). Sie können ausserdem Parameter an den Link (getrennt mit einem ? und &) anhängen.

Der Link wird automatisch vom System durch alle Schlüssel des aktuellen Filters als Parameter erweitert. Wenn Sie dies nicht wünschen und stattdessen selber definieren möchten, welche Schlüssel als Parameter verwendet werden sollen, dann können Sie die Eigenschaft Link Keys dazu verwenden.

Beispiele

```
Link = "'details.html'"
```

Suppen, Suppeneinlagen, Soßen, Brühen, Würzmitt...	3.464,67
Wasch-, Putz- und Reinigungsmittel und Hilfsmittel	7.231,03
Weine, Schaumweine, Spirituosen	33.292,18

[details.html](#)

Siehe auch

Link-Icon (auf Seite 47), Link-Keys (auf Seite 48), Link-Name (auf Seite 49), Link-Target (auf Seite 48)

Link-Icon

Typ

String

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie ein Icon für den Link definieren, das dann neben dem Link dargestellt wird. Das Ergebnis der Formel muss eine gültige URL für eine Grafik (entweder im JPEG, GIF oder PNG Format) sein.

Beispiele

```
Link-Icon = "'/iolap/icons/icon_detail.gif'"
```

Suppen, Suppeneinlagen, Soßen, Brühen, Würzmitt...	3.464,67
Wasch-, Putz- und Reinigungsmittel und Hilfsmittel	7.231,03
Weine, Schaumweine, Spirituosen	33.292,18

 [SHOW DETAILS](#)

Siehe auch

Link (auf Seite 46), Link-Keys (auf Seite 48), Link-Name (auf Seite 49), Link-Target (auf Seite 48)

Link-Keys

Typ

Key

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie die Generierung von Block-Links beeinflussen (indem Sie die an den Link angehängten Parameter bestimmen). Wenn Sie keine Link Keys bestimmen, dann werden alle Dimension-Schlüssel aus dem aktuellen Block-Filter als Parameter an den Link gehängt. Z.b. würde eine Block, der "Produkt:A" und "Zeit:2004" anzeigt, diese als Parameter an seinen Link hängen.

Wenn Sie nicht möchten, dass der exakte Filter übergeben wird, dann können Sie eine Key-Expression für diese Eigenschaft definieren, dessen Ergebnis als Parameter an den Link angefügt werden. Diese Expression kann ebenfalls auf den aktuellen Filter zugreifen, jedoch können auch Sie Schlüssel auslassen oder komplexere Berechnungen durchführen.

Beispiele

```
Link Keys = "Produkt | NEXT( Zeit )"
```

Generiert einen Link für das aktuelle Produkt, jedoch für den folgenden Tag / Monat / Jahr

Siehe auch

Link (auf Seite 46), Link-Icon (auf Seite 47), Link-Name (auf Seite 49), Link-Target (auf Seite 48)

Link-Target

Typ

String

Seit

2.1

Beschreibung

In der Standardeinstellung wird das Link-Ziel im selben Browser-Fenster geöffnet wird der aktuelle Bericht. Über diese Eigenschaft können Sie einen beliebigen Fenster-Namen angeben. Der Link wird dann in einem neuen Browser-Fenster (mit diesem Namen) geöffnet.

Beispiele

```
Link Target = "'new_window'"
```

Siehe auch

Link (auf Seite 46), Link-Icon (auf Seite 47), Link-Icon (auf Seite 47), Link-Name (auf Seite 49)

Link-Name

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt den Namen für den Block-Link. Der name wird anstelle der Link-Adresse dargestellt.

Beispiele

```
Link Name = "'SHOW DETAILS'"
```

Suppen, Suppeneinlagen, Soßen, Brühen, Würzmitt...	3.464,67
Wasch-, Putz- und Reinigungsmittel und Hilfsmittel	7.231,03
Weine, Schaumweine, Spirituosen	33.292,18

[SHOW DETAILS](#)

Siehe auch

Link (auf Seite 46), Link-Icon (auf Seite 47), Link-Keys (auf Seite 48), Link-Target (auf Seite 48)

Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt den Innenabstand (das Padding) des Blocks in Pixeln. Der Innenabstand ist der Abstand zwischen dem Block-Rahmen und dem Titel bzw. dem Blockinhalt. Wenn Sie kein Padding bestimmen oder die Formel NULL ergibt, dann wird der Standard-Abstand verwendet.

Beispiele

`Padding = "10"`

Setzt den Innenabstand auf 10 Pixel:

Block title	
	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Title

Typ

String

Seit

1.1

Beschreibung

Die Eigenschaft "Title" setzt die Überschrift für einen inneren Block. Wenn Sie hier eine Formel definieren und die Formel nicht NULL ergibt, dann wird das Ergebnis der Formel als Überschrift über der Tabelle bzw. Grafik angezeigt.

Beispiele

`Title = "'Sales for product ' + TOSTRING(Product)"`

Sales for product All	
	Absatz (DEM)
Backwaren - Bedienungs- und SB-Ware (Brot, Klei...	7.617,76
Biere, alkoholfreie Getränke, Saftbar in Bedienung	27.101,82

Width

Typ

Integer

Seit

1.1

Beschreibung

Mit dieser Eigenschaft bestimmen Sie die Breite des Blocks in Pixeln. Beachten Sie, dass Sie für einige Block-Formate (Diagramme) die Masse des Blocks definieren müssen, da diese sonst nicht dargestellt werden.

Beispiele

```
Width = "500"
```

Siehe auch

Format (auf Seite 42), Height (auf Seite 45)

Abfragen

Corner-Align

Typ

String ('left', 'center' oder 'right')

Seit

1.2

Beschreibung

Mit dieser Ausrichtung können Sie die Text-Ausrichtung für den Text in der oberen, linken Ecke der Pivot-Tabelle bestimmen. Die folgenden Werte darf ein Formel für diese Eigenschaft zurückgeben:

- 'left': Linksbündig
- 'center': Mittig
- 'right': Rechtsbündig

Beispiele

```
Corner-Align = "'left'"
```

```
Corner-Align = "'center'"
```

```
Corner-Align = "'right'"
```

Corner-Background

Typ

String

Seit

1.2

Beschreibung

Die Eigenschaft "Corner-Background" bestimmt die Hintergrundfarbe für die obere, linke Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Corner-Background = "'red'"
```

```
Corner-Background = "'#FFFF00'"
```

Siehe auch

Corner-Foreground (auf Seite 55)

Corner-Bottom-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die unterere Rahmenfarbe für die linke, obere Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Corner-Bottom-Color = "'black'"
```

Siehe auch

Corner-Left-Color (auf Seite 55), Corner-Right-Color (auf Seite 56), Corner-Top-Color (auf Seite 58)

Corner-Font

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie den Zeichensatz für die obere, linke Ecke der Pivot-Tabelle bestimmen.

Beispiele

```
Corner-Font = "'Arial'"
```

Siehe auch

Corner-Font-Size (auf Seite 54), Corner-Font-Weight (auf Seite 54)

Corner-Font-Weight

Typ

String ('bold' oder 'lighter')

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie Fettdruck für die obere, linke Ecke der Pivor-Tabelle ein bzw. ausschalten. Die Formel in dieser Eigenschaft muss eine der folgenden Zeichenketten als Ergebnis haben:

- 'lighter': Fettdruck aus
- 'bold': Fettdruck ein

Beispiele

```
Corner-Font-Weight = "'lighter'"
```

```
Corner-Font-Weight = "'bold'"
```

Siehe auch

Corner-Font (auf Seite 53), Corner-Font-Size (auf Seite 54)

Corner-Font-Size

Typ

Integer

Seit

1.2

Beschreibung

Diese Eigenschaft bestimme die Zeichengröße (in Pixeln) für die obere, linke Ecke der Pivot-Tabelle.

Beispiele

```
Corner-Font-Size = "10"
```

Siehe auch

Corner-Font (auf Seite 53), Corner-Font-Weight (auf Seite 54)

Corner-Foreground

Typ

String

Seit

1.2

Beschreibung

Die Eigenschaft "Corner-Foreground" bestimmt die Textfarbe für die obere, linke Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Corner-Foreground = "'red'"
```

```
Corner-Foreground = "'#FFFF00'"
```

Siehe auch

Corner-Background (auf Seite 52)

Corner-LeftColor

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die linke Rahmenfarbe für die linke, obere Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Corner-Left-Color = "'black'"
```

	Absatz
KW 40/1999	35.274.934,068
KW 41/1999	34.923.782,952
KW 42/1999	35.311.994,678
KW 43/1999	38.562.901,979

Siehe auch

Corner-Bottom-Color (auf Seite 53), Corner-Right-Color (auf Seite 56), Corner-Top-Color (auf Seite 58)

Corner-Right-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die rechte Rahmenfarbe für die linke, obere Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Corner-Right-Color = "'black'"
```

	Absatz
KW 40/1999	35.274.934,068
KW 41/1999	34.923.782,952
KW 42/1999	35.311.994,678
KW 43/1999	38.562.901,979

Siehe auch

Corner-Bottom-Color (auf Seite 53), Corner-Left-Color (auf Seite 55), Corner-Top-Color (auf Seite 58)

Corner-Text

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie einen Text bestimmen, der in der oberen linken Ecke der Pivot-Tabelle dargestellt werden soll. Das Ergebnis der Formel muss eine Zeichenkette sein und wird dargestellt. Wenn Sie keinen Text bestimmen oder die Formel NULL ergibt, dann werden die Titel (wenn definiert) der Tabellen-Überschriften dargestellt. Wenn keine Titel definiert wurden, dann bleibt die Ecke leer.

Sie können in der Ecke ausserdem auch Grafiken darstellen. Dazu muss die Formel einfach eine gültige URL zu einer Grafik (die mit ".png", ".jpg" oder ".gif" endet) ergeben. Beachten Sie, dass in der Workbench (im Preview) nur Grafiken angezeigt werden, die sich im Repository von instantOLAP befinden.

Beispiele

```
Corner-Text = "NULL"
```

Generiert eine leere Ecke


```
Corner-Text = "'All Amounts in EURO'"
```

Stellt den Text "All amounts in EURO" in der oberen, linken Ecke dar:

All Amounts in EURO	Absatz
KW 40/1999	35.274.934,068
KW 41/1999	34.923.782,952
KW 42/1999	35.311.994,678
KW 43/1999	38.562.901,979
KW 44/1999	37.112.028,385
KW 45/1999	38.699.783,859
KW 46/1999	39.254.478,116

```
Corner-Text = "'/iolap/logo.gif'"
```

Stellt die Grafik "legend.gif" in der oberen, linken Ecke dar:

	Absatz
KW 40/1999	35.274.934,068
KW 41/1999	34.923.782,952
KW 42/1999	35.311.994,678
KW 43/1999	38.562.901,979

Corner-Top-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die obere Rahmenfarbe für die linke, obere Ecke der Pivot-Tabelle. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

Corner-Top-Color = "'black'"

	Absatz
KW 40/1999	35.274.934,068
KW 41/1999	34.923.782,952
KW 42/1999	35.311.994,678
KW 43/1999	38.562.901.979

Siehe auch

Corner-Bottom-Color (auf Seite 53), Corner-Left-Color (auf Seite 55), Corner-Right-Color (auf Seite 56)

Filter

Typ

Key

Seit

1.2

Beschreibung

Diese Eigenschaft setzt einen Filter für eine Abfrage. Eine Abfrage ist eine Sammlung von Dimension-Schlüsseln mit keinem, einem oder mehreren Schlüssel je Dimension. Alle Formeln, die aus verschachtelten Elementen auf die aktuelle Selektion einer Dimension zugreifen werden die hier eingestellten Schlüssel als (Teil-) Ergebnis haben.

Das Filtern von Dimensionen verbietet den verschachtelten Element nicht, auf andere Schlüssel als die gefilterten zuzugreifen, es wird nur eine aktuelle Selektion bestimmt. Wenn z.B. ein Filter "Produkt:A" gesetzt wird, dann ergibt ein Ausdruck "Produkt" in einem verschachtelten Element "Produkt:A", "NEXT(Produkt)" würde aber auch etwas ergeben (z.B. "Produkt:B"), obwohl das Ergebnis nicht Teil des Filters ist.

Wenn Sie Dimension so filtern möchten, dass kein Ausdruck Schlüssel zurückgibt, die nicht Teil des Filters sind, dann müssen Sie die Eigenschaft Subcube verwenden.

Beispiele

Filter = „Produkt:ProduktA“

Setzt den Filter auf "Produkt:A"

Siehe auch

Subcube (auf Seite 59)

Subcube

Typ

Key

Seit

2.0

Beispiele

Die Eigenschaft "Subcube" ermöglicht es der Abfrage, Dimensionen auf eine Teilmenge ihrer Schlüssel einzuschränken. Alle verschachtelten Elemente der Abfrage (z.B. die Tabellen-Überschriften), die sich auf die eingeschränkten Dimensionen beziehen, werden nur noch diese Teilmenge sehen.

Die Eigenschaft erwartet eine Formeln vom Rückgabebetyp "Key". Das Ergebnis der Formel bestimmt den "Subcube" - die Dimensionen der Subcubes bestehen dann nur noch aus den entsprechenden Schlüsseln des Ergebnisses. Beachten Sie, dass Dimensionen komplett leer sein werden, wenn die Formeln keine Schlüssel für diese ergeben, dass gilt auch für die Kennzahl-Dimension.

Die Verwendung von Subcubes kann Ihre Abfragen deutlich vereinfachen und ggf. auch beschleunigen, besonders wenn Sie Drilldown verwenden. Sie können z.B. mit der Subcube-Eigenschaft alle Schlüssel ermitteln, für die eine bestimmte Kennzahl Werte enthält und die entsprechende Dimension dann auf diese Schlüssel reduzieren. Nach dem Setzen des Subcubes wird der Drilldown nur noch Schlüssel mit Daten darstellen. Das ist einfacher, als Schlüssel mit Werten im Drilldown zu suchen. Und, was noch wichtiger ist, die Subcube-Formel wird nur ein einziges Mal je Abfrage ausgeführt, beim Drilldown werden die Formeln ggf. einmal je Zeile ausgeführt.

Verwechseln Sie das Generieren von Subcubes nicht mit Filtern. Ein Filter setzt nur die Selektion für Dimensionen, erlaubt aber immer noch das Verwenden von anderen Schlüsseln als den Selektieren. Nur der Subcube schränkt Dimensionen wirklich ein.

Beispiele

```
Subcube = "LOOKUP( LEVEL( Produkt, 1, 2 ), Fact:Betrag ) | Fact:Betrag"
```

Diese Formel generiert einen Subcube mit allen Produkten, zu denen die Kennzahl "Betrag" existiert und der Kennzahl selbst. Wenn eine Abfrage z.B. eine Iteration wie "LEVEL(Produkt, 1)" inkl. Drilldown für die Y-Achse verwendet, dann werden nur noch Produkt mit Werten angezeigt, auch wenn der Header bei einem Drilldown geöffnet wird. Da auch die Kennzahl "Betrag" im Subcube enthalten ist, kann die zusätzlich Kennzahl auf der X-Achse verwendet werden.

Siehe auch

Filter (auf Seite 58)

Supress-Cols

Typ

Boolean

Seit

1.1

Beschreibung

Wenn die Formel aus "Suppress-Cols" true zurückgibt, dann werden alle leeren Spalten aus dem Abfrageergebnis gelöscht und es bleiben nur Spalten mit mindestens einem Wert sichtbar. Beim Suchen nach leeren Spalten werden nur die Daten-Zellen der Pivot-Tabelle beachtet, nicht die Überschriften.

Beachten Sie dass, auch wenn leere Spalten unterdrückt werden sollen, immer noch die komplette Pivot-Tabelle im Speicher des Systems erstellt wird und die Spalten erst nachträglich (!) gelöscht werden. Um evtl. Performance-Probleme oder entsprechende Fehlermeldungen zu vermeiden, sollten Sie bei sehr grossen Tabellen die Funktion LOOKUP verwenden, um Schlüssel mit existierenden Werten zu finden.

Beispiele

```
Supress-Cols = "true"
```

Siehe auch

Supress-Rows (auf Seite 61)

Supress-Rows

Typ

Boolean

Seit

1.1

Beschreibung

Wenn die Formel aus "Suppress-Rows" true zurückgibt, dann werden alle leeren Zeilen aus dem Abfrageergebnis gelöscht und es bleiben nur Zeilen mit mindestens einem Wert sichtbar. Beim Suchen nach leeren Zeilen werden nur die Daten-Zellen der Pivot-Tabelle beachtet, nicht die Überschriften.

Beachten Sie dass, auch wenn leere Zeilen unterdrückt werden sollen, immer noch die komplette Pivot-Tabelle im Speicher des Systems erstellt wird und die Zeilen erst nachträglich (!) gelöscht werden. Um evtl. Performance-Probleme oder entsprechende Fehlermeldungen zu vermeiden, sollten Sie bei sehr grossen Tabellen die Funktion LOOKUP verwenden, um Schlüssel mit existierenden Werten zu finden.

Beispiele

Supress-Rows = "true"

Siehe auch

Supress-Cols (auf Seite 60)

Überschriften

Align

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Text-Ausrichtung für die Tabellenüberschrift. Mit dieser Eigenschaft beeinflussen Sie nicht die Text-Ausrichtung der Zellen, die zu dieser Überschrift gehören, verwenden Sie dazu die Eigenschaft "Cell-Align". Die Formel muss eine der folgenden Zeichenketten zurückgeben:

- 'left': Linksbündig
- 'center': Mittig
- 'right': Rechtsbündig

Beispiele

```
Align = "'left'"
```

```
Align = "'center'"
```

```
Align = "'right'"
```

Siehe auch

Cell-Align (auf Seite 66)

Assertion

Typ

Double

Seit

1.2

Beschreibung

Mit der Eigenschaft "Assertion" können Sie eine Ampelanalyse für Ihren Bericht implementieren. Diese Eigenschaft muss eine Formel enthalten, die für jede Zelle einen Wert von Typ Double berechnet (da die Formel für jede Zelle angewandt wird, kann auch jede Zelle eine andere Ampel-Farbe erhalten). Abhängig von Ergebnis wird eine Zelle anders eingefärbt:

- Ergebnis ≤ -1 : Die Zelle wird rot eingefärbt
- $-1 < \text{Ergebnis} < 0$: Die Zelle wird gelb eingefärbt
- Ergebnis = 0: Die Zelle wird nicht eingefärbt
- $0 < \text{Ergebnis}$: Die Zelle wird grün eingefärbt
- Ergebnis = NULL: Die Zelle wird nicht eingefärbt

Diese Eigenschaft wird nicht nur zum Einfärben von Zellen in der Ergebnisanzeige verwendet, Sie kann auch Auswirkungen auf einige Automations-Aufgaben (wie z.B. EMail-Versand) haben. Z.B. könnten Sie nur dann EMails mit einem Berichtsergebnis versenden wollen, wenn eine Assertion im Ergebnis vorkommt (also mindest eine Zelle grün, gelb oder rot ist). Dazu können Sie im Bericht eine Assertion definieren und diese dann in der Automation verwenden.

Beispiele

`Assertion = "-1"`

Erzeugt eine rote Einfärbung jeder Zelle

`Assertion = "-0.5"`

Erzeugt eine gelbe Einfärbung jeder Zelle

`Assertion = "1"`

Erzeugt eine grüne Einfärbung jeder Zelle

`Assertion = "0"`

Erzeugt keine Einfärbung

`Assertion = "NULL"`

Erzeugt keine Einfärbung

Background

Typ

String

Seit

1.2

Beschreibung

Mit der Eigenschaft "Background" bestimmen Sie die Hintergrundfarbe für Überschriften. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Diese Eigenschaft setzt nur die Hintergrundfarbe der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Background.

Beispiele

```
Background = "'red'"
```

Setzt die Hintergrundfarbe auf rot

```
Background = "'#FFFF00'"
```

Setzt die Hintergrundfarbe auf gelb

Siehe auch

Cell-Background (auf Seite 67), Foreground (auf Seite 84)

Bottom-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des unteren Rahmens für alle Überschriften, die von diesem Element erzeugt werden. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Diese Eigenschaft setzt nur die Rahmenfarbe der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Bottom-Color.

Beispiele

```
Bottom-Color = "'black'"
```

Siehe auch

Cell-Bottom-Color (auf Seite 68), Left-Color (auf Seite 88), Right-Color (auf Seite 93), Top-Color (auf Seite 97)

Bottom-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den unteren Innenabstand (Padding) in Pixeln für alle von diesem Element generierten Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Diese Eigenschaft setzt nur das Padding der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Bottom-Padding.

Beispiele

```
Bottom-Padding = "10"
```

Siehe auch

Cell-Bottom-Padding (auf Seite 68), Left-Padding (auf Seite 89), Right-Padding (auf Seite 94), Top-Padding (auf Seite 98)

Cell-Align

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Text-Ausrichtung für die Zellen, die zur Überschrift gehören. Die Formel muss eine der folgenden Zeichenketten als Ergebnis haben:

- 'left': Linksbündig
- 'center': Mittig
- 'right': Rechtsbündig

Beispiele

```
Cell-Align = "'left'"
```

```
Cell-Align = "'center'"
```

```
Cell-Align = "'right'"
```

Siehe auch

Align (auf Seite 63), Cell-Vertical-Align (auf Seite 78)

Cell-Background

Typ

String

Seit

1.2

Beschreibung

Mit der Eigenschaft "Background" bestimmen Sie die Hintergrundfarbe für die Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Cell-Background = "'red'"
```

```
Cell-Background = "'#FFFF00'"
```

Siehe auch

Background (auf Seite 64), Cell-Foreground (auf Seite 70)

Cell-Bottom-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des unteren Rahmens für alle Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn

Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Beispiele

```
Cell-Bottom-Color = "'black'"
```

Siehe auch

Bottom-Color (auf Seite 65), Cell-Left-Color (auf Seite 71), Cell-Right-Color (auf Seite 75), Cell-Top-Color (auf Seite 77)

Cell-Bottom-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den unteren Innenabstand (Padding) in Pixeln für alle zu dieser Überschrift gehörenden Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Beispiele

Cell-Bottom-Padding = "10"

Siehe auch

Bottom-Padding (auf Seite 66), Cell-Left-Padding (auf Seite 71), Cell-Right-Padding (auf Seite 76), Cell-Top-Padding (auf Seite 77)

Cell-Font

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Zeichensatz für alle Zellen, die zu dieser Überschrift gehören. Die Formel muss einen gültigen Zeichensatznamen als Zeichenkette zurückgeben.

Beispiele

Cell-Font = "'Arial'"

Siehe auch

Cell-Font-Size (auf Seite 69), Cell-Font-Weight (auf Seite 70), Font (auf Seite 82)

Cell-Font-Size

Typ

Integer

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Zeichengröße (in Pixeln) für alle Zellen, die zu diesem Header gehören.

Beispiele

```
Cell-Font-Size = "10"
```

Siehe auch

Cell-Font (auf Seite 69), Cell-Font-Weight (auf Seite 70), Font-Size (auf Seite 83)

Cell-Font-Weight

Typ

String ('bold' oder 'lighter')

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie für die Zellen, die zu dieser Überschrift gehören, Fettdruck ein- bzw. ausschalten. Die Formel muss eine der folgenden Zeichenketten zurückgeben:

- 'lighter': Normal
- 'bold': Fettdruck

Beispiele

```
Cell-Font-Weight = "'lighter'"
```

```
Cell-Font-Weight = "'bold'"
```

Siehe auch

Cell-Font (auf Seite 69), Cell-Font-Size (auf Seite 69), Font-Weight (auf Seite 83)

Cell-Foreground

Typ

String

Seit

1.2

Beschreibung

Mit der Eigenschaft "Cell-Foreground" bestimmen Sie die Textfarbe aller Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beispiele

```
Cell-Foreground = "'red'"
```

```
Cell-Foreground = "'#FFFF00'"
```

Siehe auch

Cell-Background (auf Seite 67), Foreground (auf Seite 84)

Cell-Left-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des linken Rahmens für alle Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Beispiele

```
Cell-Left-Color = "'black'"
```

Siehe auch

Cell-Bottom-Color, Cell-Right-Color, Cell-Top-Color, Left-Color

Cell-Left-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den linken Innenabstand (Padding) in Pixeln für alle zu dieser Überschrift gehörenden Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Beispiele

```
Cell-Left-Padding = "10"
```

Siehe auch

Cell-Bottom-Padding, Cell-Right-Padding, Cell-Top-Padding, Left-Padding

Cell-Link

Typ

String

Seit

1.1

Beschreibung

Die Eigenschaft wird dazu verwendet, Links für alle Zellen, die zu dieser Überschrift gehören, zu generieren. Das Ergebnis dieser Formel muss eine gültige URL sein (sowohl relative also auch absolute URLs sind erlaubt). Sie können an den Link auch Parameter anfügen (mit den Trennzeichen ? und &).

Der Link wird automatisch um den aktuellen Filter (welcher bei Zellen sowohl von den Überschriften aus der X- als auch Y-Achse beeinflusst wird) in Form von Parametern erweitert. Wenn Sie nicht automatisch den kompletten Filter als Parameterliste möchten können Sie dies mit der Eigenschaft Cell-Link-Keys kontrollieren.

Beispiele

```
Cell-Link = "'otherquery.html'"
```

Erzeugt einen Link zum Bericht "otherquery" im HTML-format

```
Cell-Link = "'otherquery.pdf'"
```

Erzeugt einen Link zum Bericht "otherquery" im PDF-Format

```
Cell-Link = "'otherquery.html?limit=10'"
```

Erzeugt einen Link zum Bericht "otherquery" im HTML-Format mit dem Parameter "limit" und dem Wert 10

Siehe auch

Cell-Link-Icon (auf Seite 73), Cell-Link-Keys (auf Seite 73), Cell-Link-Name (auf Seite 74), Link (auf Seite 90)

Cell-Link-Icon

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie ein Icon angeben, das innerhalb zu dieser Überschrift gehörenden Zellen dargestellt wird. Wenn Sie ein solches Icon verwenden, wird dieses anstelle des Überschrift-Textes mit dem Link versehen. Das Ergebnis dieser Formel muss eine gültige URL auf eine Grafik (im JPEG, GIF oder PNG-Format) sein.

Beispiele

```
Cell-Link-Icon = "'/iolap/icons/icon_chart.gif'"
```

Siehe auch

Cell-Link (auf Seite 72), Cell-Link-Keys (auf Seite 73), Cell-Link-Name (auf Seite 74), Cell-Link-Target (auf Seite 75), Link-Icon (auf Seite 91)

Cell-Link-Keys

Typ

Key

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie die Generierung von Cell-Links beeinflussen (indem Sie die an den Link angehängten Parameter bestimmen). Wenn Sie keine Link Keys bestimmen, dann werden alle Dimension-Schlüssel aus dem aktuellen Block-Filter als Parameter an den Link gehängt. Z.b. würde eine Block, der "Produkt:A" und "Zeit:2004" anzeigt, diese als Parameter an seinen Link hängen.

Wenn Sie nicht möchten, dass der exakte Filter übergeben wird, dann können Sie eine Key-Expression für diese Eigenschaft definieren, dessen Ergebnis als Parameter an den Link angefügt werden. Diese Expression kann ebenfalls auf den aktuellen Filter zugreifen, jedoch können auch Sie Schlüssel auslassen oder komplexere Berechnungen durchführen.

Beispiele

```
Cell-Link-Keys = "Product | NEXT( Time )"
```

Mit dem aktuellen Produkt verknüpfen, jedoch mit dem folgenden Jahr / Monat / Tag

Siehe auch

Cell-Link (auf Seite 72), Cell-Link-Icon (auf Seite 73), Cell-Link-Name (auf Seite 74), Cell-Link-Target (auf Seite 75), Link-Keys (auf Seite 91)

Cell-Link-Name

Typ

String

Seit

1.2

Beschreibung

Die Eigenschaft bestimmt den Namen für die Links der Zellen. Der Name wird als Popup-Window dargestellt wenn der Benutzer mit der Mouse über den Link-Text oder ggf. über das Link-Icon (wenn vorhanden) fährt.

Beispiele

```
Cell-Link-Name = "'Zeige Details für ' + {Produkt}'"
```

Siehe auch

Cell-Link (auf Seite 72), Cell-Link-Icon (auf Seite 73), Cell-Link-Keys (auf Seite 73), Cell-Link-Target (auf Seite 75), Link-Name (auf Seite 92)

Cell-Link-Target

Typ

String

Seit

2.1

Beschreibung

In der Standardeinstellung wird das Cell-Link-Ziel im selben Browser-Fenster geöffnet wird der aktuelle Bericht. Über diese Eigenschaft können Sie einen beliebigen Fenster-Namen angeben. Der Link wird dann in einem neuen Browser-Fenster (mit diesem Namen) geöffnet.

Beispiele

```
Cell-Link-Target = "'window1'"
```

Siehe auch

Cell-Link (auf Seite 72), Cell-Link-Icon (auf Seite 73), Cell-Link-Keys (auf Seite 73), Cell-Link-Name (auf Seite 74), Link-Target (auf Seite 93)

Cell-Right-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des rechten Rahmens für alle Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Beispiele

```
Cell-Right-Color = "'black'"
```

Siehe auch

Cell-Bottom-Color (auf Seite 68), Cell-Left-Color (auf Seite 71), Cell-Top-Color (auf Seite 77), Right-Color (auf Seite 93)

Cell-Right-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den rechten Innenabstand (Padding) in Pixeln für alle zu dieser Überschrift gehörenden Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Beispiele

```
Cell-Right-Padding = "10"
```

Siehe auch

Cell-Bottom-Padding (auf Seite 68), Cell-Left-Padding (auf Seite 71), Cell-Top-Padding (auf Seite 77), Right-Padding (auf Seite 94)

Cell-Top-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des oberen Rahmens für alle Zellen, die zu dieser Überschrift gehören. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Beispiele

```
Cell-Top-Color = "'black'"
```

Siehe auch

Cell-Bottom-Color (auf Seite 68), Cell-Left-Color (auf Seite 71), Cell-Right-Color (auf Seite 75), Top-Color (auf Seite 97)

Cell-Top-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den oberen Innenabstand (Padding) in Pixeln für alle zu dieser Überschrift gehörenden Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Beispiele

```
Cell-Top-Padding = "10"
```

Siehe auch

Cell-Bottom-Padding (auf Seite 68), Cell-Left-Padding (auf Seite 71), Cell-Right-Padding (auf Seite 76), Top-Padding (auf Seite 98)

Cell-Vertical-Align

Typ

String

Seit

2.1

Beschreibung

Die Eigenschaft bestimmt die vertikale Ausrichtung für die alle Zellen, die zu dieser Überschrift gehören. Die Formel muss eine der folgenden Zeichenketten ergeben:

- 'top': Oberer Rand
- 'middle': Mittig
- 'bottom': Unterer Rand

Beispiele

```
Cell-Vertical-Align = "'top'"
```

```
Cell-Vertical-Align = "'middle'"
```

```
Cell-Vertical-Align = "'bottom'"
```

Siehe auch

Cell-Align (auf Seite 66), Vertical-Align (auf Seite 98)

Drilldown

Typ

Boolean

Seit

1.0

Beschreibung

Mit dieser Eigenschaft können Sie das Drilldown für diese Überschrift aktivieren (die Formel muss dazu "true" zurückgeben).

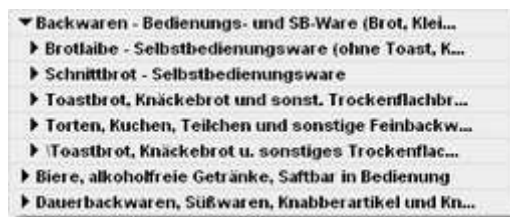
In der Standardeinstellung werden dann bei einem Drilldown alle Kind-element des aktuellen Schlüssels dargestellt. Dieses Verhalten können Sie jedoch über die Eigenschaft Drilldown-Iteration ändern.

Ausserdem gibt es zwei verschiedene Arten der Darstellung für Drilldown: Die vertikale (vertical) und die verschachtelte (encapsulated). Verwenden Sie die Eigenschaft Drilldown-Encapsulate, um von der vertikalen (die Standardeinstellung) auf die verschachtelte Darstellung umzuschalten.

Beispiele

```
Drilldown = "true"
```

Schaltet Drilldown für die Überschrift ein:



Siehe auch

Drilldown-Encapsulate (auf Seite 79), Drilldown-Iteration (auf Seite 80), Drilldown-Prefetch (auf Seite 81)

Drilldown-Encapsulate

Typ

Boolean

Seit

1.2

Beschreibung

Wenn Sie mit der Drilldown-Eigenschaft das Drilldown für eine Überschrift ermöglichen, dann kann der Benutzer die Überschrift "aufklappen" um dessen Kind-Elemente und deren Daten zu sehen. Beim Aufklappen werden die neuen Elemente unterhalb (oder rechts davon, wenn die Überschrift sich auf der X-Achse befinden) ihres Vater-Elements dargestellt.

Alternativ können Sie auch eine andere Darstellung wählen, indem Sie diese Eigenschaft auf "true" setzen. In diesem Fall werden die Kind-Elemente als verschachtelte Elemente der Original-Überschrift dargestellt. Das Verschachteln von Drilldown gibt etwas mehr Überblick, benötigt aber auch mehr Platz auf dem Bildschirm. Ausserdem fällt bei dieser Darstellungsart beim Drilldown die Zeile für das geöffnete Element weg, es gibt also keine Summendarstellung etc.

Beispiele

```
Drilldown-Encapsulate = "false"
```

```
Drilldown-Encapsulate = "true"
```

Siehe auch

Drilldown (auf Seite 78), Drilldown-Iteration (auf Seite 80), Drilldown-Prefetch (auf Seite 81)

Drilldown-Iteration

Typ

Key

Seit

1.2

Beschreibung

Mit der Eigenschaft "Drilldown-Iteration" können sie bestimmen, welche Dimensions-Schlüssel bei einem Drilldown dargestellt werden. In der Standard-Einstellung zeigt eine Überschrift die Kind-Elemente des Schlüssels an, der zur Überschrift gehört. Meistens entspricht das genau dem gewünschten Verhalten, jedoch sind manchmal komplexere Drilldowns gefordert. Z.B. wenn beim Aufklappen Schlüssel aus einer anderen Dimension (z.B. alle Kunden zu einem Produkt) dargestellt werden sollen oder wenn nur Kinder mit bestimmten Daten dargestellt werden sollen.

Die Drilldown-Iteration erwartet eine Formel von Rückgabebetyp "Key". Beim Drilldown wird diese Formel dann für den entsprechende Überschrift ausgeführt und jeder Schlüssel aus dem Ergebnis wird zu einer neuen Überschrift beim Drilldown. Die neuen Überschriften besitzen dann die selbe Drilldown-Iteration wie ihr Vater.

Beispiele

```
Drilldown-Iteration = "Product.Customers"
```

Siehe auch

Drilldown (auf Seite 78), Drilldown-Encapsulate (auf Seite 79),
Drilldown-Prefetch (auf Seite 81)

Drilldown-Prefetch

Typ

Boolean

Seit

1.2

Beschreibung

Vor allen Überschriften mit Drilldown wird ein entsprechendes Icon dargestellt, wenn die Überschrift aufklappbar ist bzw. wenn es Kind-Elemente gibt. Um herauszufinden, ob solche Kind-Elemente vorhanden sind, muss das System dazu für jede Überschrift die Anzahl der Kinder zählen bzw die Drilldown-Iteration (wenn sie für diese Überschrift definiert wurde) ausführen.

Besonders, wenn eine komplexe Drilldown-Iteration definiert wurde, kann das Zeit kosten und System-Ressourcen beanspruchen (z.B. wenn die Iteration Datenbankabfragen durchführen muss). In diesem Fall können Sie diese Eigenschaft auf "false" setzen und die Vorrasschau damit abschalten. In diesem Fall zeigen alle Überschriften das Icon an, unabhängig ob der Drilldown zum Erfolg führen wird.

Beispiele

```
Drilldown-Prefetch = "false"
```

Siehe auch

Drilldown (auf Seite 78), Drilldown-Encapsulate (auf Seite 79),
Drilldown-Iteration (auf Seite 80)

Filter

Typ

Key

Seit

1.2

Beschreibung

Mit dieser Optionalen Eigenschaft können Sie einen Filter auf die aktuelle Überschrift anwenden. Die Anwendung des Filters passiert vor der Berechnung aller anderen Eigenschaften für diese Überschrift und wird z.B. die Text-, Gestaltungs- oder Formel-Eigenschaften der Überschrift und der dazugehörenden Zellen beeinflussen.

Die Eigenschaft erwartet eine Formel vom Ergebnistyp "Key", dessen Ergebnis auf den Filter angewandt werden. "Angewandt" heisst, dass für alle Dimensionen die im Ergebnis vorkommen die Schlüssel gesetzt werden. Alle anderen Dimensionen bleiben unbeeinflusst.

Beispiele

```
Filter = "Product:A"
```

Wende den Schlüssel "Product:A" auf den Filter der Überschrift an

Siehe auch

Iteration (auf Seite 87)

Font

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Zeichensatz für diese Überschrift. Die Formel muss einen gültigen Zeichensatznamen als Zeichenkette zurückgeben.

Beachten Sie, dass Sie mit dieser Eigenschaft nicht den Zeichensatz der Zellen beeinflussen können, die zu dieser Überschrift gehören. Verwenden Sie dazu die stattdessen die Eigenschaft Cell-Font.

Beispiele

```
Font = "'Arial'"
```

Siehe auch

Cell-Font (auf Seite 69), Font-Size (auf Seite 83), Font-Weight (auf Seite 83)

Font-Size

Typ

Integer

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Zeichengröße (in Pixeln) für diese Überschrift.

Beachten Sie, dass Sie mit dieser Eigenschaft nicht den Zeichengröße der Zellen beeinflussen können, die zu dieser Überschrift gehören. Verwenden Sie dazu die stattdessen die Eigenschaft Cell-Font-Size.

Beispiele

```
Font-Size = "10"
```

Siehe auch

Cell-Font-Size (auf Seite 69), Font (auf Seite 82), Font-Weight (auf Seite 83)

Font-Weight

Typ

String ('bold' oder 'lighter')

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie für diese Überschrift Fettdruck ein- bzw. ausschalten. Die Formel muss eine der folgenden Zeichenketten zurückgeben:

- 'lighter': Normal
- 'bold': Fettdruck

Beachten Sie, dass Sie mit dieser Eigenschaft nicht den Fettdruck der Zellen beeinflussen können, die zu dieser Überschrift gehören. Verwenden Sie dazu die stattdessen die Eigenschaft Cell-Font-Weight.

Beispiele

```
Font-Weight = "'lighter'"
```

```
Font-Weight = "'bold'"
```

Siehe auch

Cell-Font-Weight (auf Seite 70), Font (auf Seite 82), Font-Size (auf Seite 83)

Foreground

Typ

String

Seit

1.2

Beschreibung

Mit der Eigenschaft "Foreground" bestimmen Sie die Textfarbe für diese Überschrift. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält.

Beachten Sie, dass Sie mit dieser Eigenschaft nicht den Textfarbe der Zellen beeinflussen können, die zu dieser Überschrift gehören. Verwenden Sie dazu die stattdessen die Eigenschaft Cell-Foreground.

Beispiele

```
Foreground = "'red'"
```

```
Foreground = "'#FFFF00'"
```

Siehe auch

Background (auf Seite 64), Cell-Foreground (auf Seite 70)

Format

Typ

String

Seit

1.1

Beschreibung

Mit dieser Eigenschaft können Sie das Darstellungsformat für Zahlen oder Datums/Zeitangaben aller Zellen bestimmen, die zum dieser Überschrift gehören. Abhängig vom Rückgabebetyp der Eigenschaft Formula oder von der aktuellen Kennzahl erwartet diese Eigenschaft ein Datums- oder Zahlenformat. Die Formel in dieser Eigenschaft muss eine entsprechende Zeichenkette zurückgeben.

Wenn Sie kein Format für die Zellen definieren, wird das System das Standardformat der verwendeten Kennzahl verwenden. Wenn die Kennzahl kein Standardformat besitzt oder Sie anstelle einer Kennzahl eine Formel verwenden, wird ein Standardformat verwendet.

Beachten Sie, dass Sie auch Zahlen als Zeit/Datum darstellen können, indem Sie ein Datumsformat angeben. In diesem Fall wird die Zahl als Anzahl Millisekunden interpretiert und muss ggf. von Ihnen noch umberechnet werden.

Beispiele

```
Format = "'#,###,##0.00'"
```

Verwendet dieses Zahlenformat für die Zellen

```
Format = "'HH:mm:ss'"
```

Verwendet dieses Zeit-/Datumsformat für die Zellen

Siehe auch

Formula (auf Seite 85)

Formula

Typ

Value

Seit

1.1

Beschreibung

Es gibt zwei verschiedene Wege, den Inhalt der Zellen zu bestimmen: Über Kennzahlen in den Überschriften oder über die Verwendung von Formeln über diese Eigenschaft:

- Die Verwendung von Kennzahlen in Überschriften ist der üblichere Weg: Durch die Angabe einer Kennzahl in einer Überschrift (bzw. in dessen Iteration) wird die entsprechende Zeile oder Spalte für diese Kennzahl gefiltert (üblicherweise passiert das automatisch, wenn Sie in der Workbench die Kennzahl auf die Tabelle ziehen). Jede Zelle ohne Formel wird jetzt diese Kennzahl ausgegeben.
- Formeln müssen Sie dann definieren, wenn Sie komplexere Berechnungen durchführen möchten, für die es keine eigene Kennzahl gibt oder wenn Sie Werte aggregieren möchten (z.B. wenn der Benutzer einen Zeitraum auswählen kann).

Um eine Formel für eine Überschrift zu definieren, müssen Sie eine Formel vom Rückgabebetyp "Value" in dieser Eigenschaft angeben. Diese Formel wird dann für einzelne Zelle ausgeführt.

Beispiele

```
Formula = "Betrag() * 2"
```

Stellt die Kennzahl "Betrag" multipliziert mit 2 dar

Siehe auch

Format (auf Seite 84)

Height

Typ

Integer

Seit

1.2

Beschreibung

Die Eigenschaft "Height" bestimmt die Höhe der Überschrift in Pixeln. Wenn keine Höhe gesetzt wird, bestimmt das System diese automatisch. Die Formel in dieser Eigenschaft muss eine Zahl vom Typ "Integer" zurückgeben.

Beispiele

```
Height = "50"
```

Siehe auch

Width (auf Seite 100)

Input

Typ

Boolean

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um Zellen eingabefähig zu machen. Diese Eigenschaft erwartet eine Formel mit dem Rückgabebetyp "Boolean", die für jede Zelle (die zu dieser Überschrift gehört) ausgeführt wird. Wenn das Ergebnis "true" ist (und der Benutzer das Recht hat, die entsprechenden Schlüssel und Kennzahlen zu bearbeiten), wird die jeweilige Zelle eingabefähig.

Wenn Sie Zellen eingabefähig machen möchten, dürfen Sie keine Formel für diese Zellen verwenden, nur Iterationen. Wenn Sie eine Formel verwenden, dann wird das System nicht mehr in der Lage sein zu berechnen, welche Kennzahl wie geändert werden muss.

Beachten Sie auch, dass die reine Eingabefähigkeit für Zellen noch nicht ausreicht, um Daten in die Datenquellen zurückzuschreiben. Sie müssen ausserdem auch die entsprechenden Cubes in der Konfiguration so einstellen, dass sie Daten speichern können (siehe die Eigenschaft "Enable Store" der SQL-Cubes).

Beispiele

```
Edit = "true"
```

Macht die komplette Zeile / Spalte editierbar

```
Edit = "X() = 5"
```

Macht nur Spalte 5 editierbar

Iteration

Typ

Key

Seit

1.0

Beschreibung

Diese Eigenschaft erlaubt die Wiederholung (Iteration) bzw. Generierung von Überschriften für eine Menge von Dimensions-Schlüsseln. Die Iteration von Überschriften wird dazu verwendet, grössere Mengen von Überschriften durch eine einzige Definition erzeugen zu lassen, anstatt jede Überschrift einzeln manuell anzulegen. Ausserdem ändern sich generierte Überschriften automatisch mit, wenn sich das dahinterliegende Modell ändert (z.B. wenn ein neues Produkt eingeführt wird etc.).

Die Iteration ist eine Formel vom Rückgabebetyp "Key", die eine beliebige Menge von Schlüssel zurückgeben kann. Für jeden Schlüssel wird genau eine Kopie der original Überschrift erstellt und an dessen Stelle eingesetzt. Der Filter jeder generierten Überschriften ist eine Kopie des umgebenden Filters, jedoch mit dem jeweiligen Schlüssel. Dadurch hat jede generierte Überschrift (und damit auch Zeile bzw. Spalte) einen anderen Filter und alle Eigenschaften bzw. Formeln beziehen sich auf den jeweiligen Schlüssel und ergeben verschiedene Werte.

Wenn die Iteration NULL ergibt, wird keine Überschrift erzeugt. Wenn keine Iteration angegeben wird, dann erzeugt die Überschrift genau eine Zeile bzw. Spalte oder weitere Filterung.

Beispiele

```
Iteration = "LEVEL( Produkt, 1 )"
```

Generiere eine Überschrift je Produkt

```
Iteration = "Fact:Betrag"
```

Zeige eine Überschrift für die Kennzahl "Betrag"

Siehe auch

Filter (auf Seite 81)

Left-Color**Typ**

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des linken Rahmens für alle Überschriften, die von diesem Element erzeugt werden. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Diese Eigenschaft setzt nur die Rahmenfarbe der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Left-Color.

Beispiele

```
Left-Color = "'black'"
```

Siehe auch

Bottom-Color (auf Seite 65), Cell-Left-Color (auf Seite 71), Right-Color (auf Seite 93), Top-Color (auf Seite 97)

Left-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den linken Innenabstand (Padding) in Pixeln für alle von diesem Element generierten Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Diese Eigenschaft setzt nur das Padding der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Left-Padding.

Beispiele

```
Left-Padding = "10"
```

Siehe auch

Bottom-Padding (auf Seite 66), Cell-Left-Padding (auf Seite 71), Right-Padding (auf Seite 94), Top-Padding (auf Seite 98)

Link

Typ

String

Seit

1.1

Beschreibung

Die Eigenschaft wird dazu verwendet einen Link für diese Überschrift zu generieren. Das Ergebnis dieser Formel muss eine gültige URL sein (sowohl relative also auch absolute URLs sind erlaubt). Sie können an den Link auch Parameter anfügen (mit den Trennzeichen ? und &).

Der Link wird automatisch um den aktuellen Filter in Form von Parametern erweitert. Wenn Sie nicht automatisch den kompletten Filter als Parameterliste möchten können Sie dies mit der Eigenschaft Link-Keys kontrollieren.

Diese Eigenschaft bestimmt nur den Link der Überschriften, nicht der dazugehörenden Zellen. Verwenden Sie dazu die Eigenschaft Cell-Link.

Beispiele

```
Cell-Link = "'otherquery.html' "
```

Link zum Bericht "otherquery" im HTML-format

```
Cell-Link = "'otherquery.pdf' "
```

Link zum Bericht "otherquery" im PDF-format

```
Cell-Link = "'otherquery.html?limit=10' "
```

Link zum Bericht "otherquery" im HTML-format mit dem Parameter "limit" und dem Wert 10

Siehe auch

Cell-Link (auf Seite 72), Link-Icon (auf Seite 91), Link-Keys (auf Seite 91), Link-Name (auf Seite 92), Link-Target (auf Seite 93)

Link-Icon

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie ein Icon angeben, das innerhalb der Überschrift dargestellt wird. Wenn Sie ein solches Icon verwenden, wird dieses anstelle des Überschrift-Textes mit dem Link versehen. Das Ergebnis dieser Formel muss eine gültige URL auf eine Grafik (im JPEG, GIF oder PNG-Format) sein.

Diese Eigenschaft bestimmt nur das Icon der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Link-Icon.

Beispiele

```
Link-Icon = "'/iolap/icons/icon_chart.gif'"
```

Siehe auch

Cell-Link-Icon (auf Seite 73), Link (auf Seite 90), Link-Keys (auf Seite 91), Link-Name (auf Seite 92), Link-Target (auf Seite 93)

Link-Keys

Typ

Key

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie die Generierung von Links beeinflussen (indem Sie die an den Link angehängten Parameter bestimmen). Wenn Sie keine Link Keys bestimmen, dann werden alle Dimension-Schlüssel aus dem aktuellen Block-Filter als Parameter an den Link gehängt. Z.B. würde eine Block, der "Produkt:A" und "Zeit:2004" anzeigt, diese als Parameter an seinen Link hängen.

Wenn Sie nicht möchten, dass der exakte Filter übergeben wird, dann können Sie eine Key-Expression für diese Eigenschaft definieren, dessen Ergebnis als Parameter an den Link angefügt werden. Diese Expression kann ebenfalls auf den aktuellen Filter zugreifen, jedoch können auch Sie Schlüssel auslassen oder komplexere Berechnungen durchführen.

Beispiele

```
Link-Keys = "Product | NEXT( Time )"
```

Verknüpfung mit dem aktuellen Produkt, jedoch mit dem nächsten Jahr / Monat / Tag

Siehe auch

Cell-Link-Keys (auf Seite 73), Link (auf Seite 90), Link-Icon (auf Seite 91), Link-Name (auf Seite 92), Link-Target (auf Seite 93)

Link-Name

Typ

String

Seit

1.2

Beschreibung

Die Eigenschaft bestimmt den Namen für den Link dieser Überschrift. Der Name wird als Popup-Window dargestellt wenn der Benutzer mit der Mouse über den Link-Text oder ggf. über das Link-Icon (wenn vorhanden) fährt.

Diese Eigenschaft bestimmt nur den Link-Namen für diese Überschrift, nicht der dazugehörenden Zellen. Verwenden Sie dazu die Eigenschaft Cell-Link-Name.

Beispiele

```
Link-Name = "'Zeige Details für ' + {Produkt}"
```

Siehe auch

Cell-Link-Name (auf Seite 74), Link (auf Seite 90), Link-Icon (auf Seite 91), Link-Keys (auf Seite 91), Link-Target (auf Seite 93)

Link-Target

Typ

String

Seit

2.1

Beschreibung

In der Standardeinstellung wird das Link-Ziel im selben Browser-Fenster geöffnet wird der aktuelle Bericht. Über diese Eigenschaft können Sie einen beliebigen Fenster-Namen angeben. Der Link wird dann in einem neuen Browser-Fenster (mit diesem Namen) geöffnet.

Beispiele

```
Link-Target = "'window1'"
```

Siehe auch

Cell-Link-Target (auf Seite 75), Link (auf Seite 90), Link-Icon (auf Seite 91), Link-Keys (auf Seite 91), Link-Name (auf Seite 92)

Right-Color

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Farbe des rechten Rahmens für alle Überschriften, die von diesem Element erzeugt werden. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Diese Eigenschaft setzt nur die Rahmenfarbe der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Right-Color.

Beispiele

```
Right-Color = "'black'"
```

Siehe auch

Bottom-Color (auf Seite 65), Cell-Right-Color (auf Seite 75), Left-Color (auf Seite 88), Top-Color (auf Seite 97)

Right-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den rechten Innenabstand (Padding) in Pixeln für alle von diesem Element generierten Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Diese Eigenschaft setzt nur das Padding der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Right-Padding.

Beispiele

```
Right-Padding = "10"
```

Siehe auch

Bottom-Padding (auf Seite 66), Cell-Right-Padding (auf Seite 76), Left-Padding (auf Seite 89), Top-Padding (auf Seite 98)

Rotate

Typ

Boolean

Seit

1.2

Beschreibung

Wenn die Formel in dieser Eigenschaft "true" ergibt wird der Text der Überschrift um 90 Grad gedreht angezeigt.

Beispiele

```
Rotate = "true"
```

Sort

Typ

Value

Seit

2.1

Beschreibung

Die "Sort" Eigenschaft ermöglicht es, durch eine Iteration generierte Überschriften nach einer beliebigen Formel zu sortieren. Diese Sortier-Formel wird für jede generierte Überschrift (bzw. den jeweiligen Dimension-Schlüssel) berechnet und die Überschriften dann nach diesem Ergebnis sortiert. Wenn zwei oder mehr Überschriften den selben Sortier-Wert besitzen, dann bleibt die ursprüngliche Reihenfolge zwischen diesen erhalten.

Die Überschriften werden in der Standardeinstellung aufsteigend sortiert, d.h. vom kleinsten Sortier-Wert zum grössten hin. Verwenden Sie die Eigenschaft Sort-Descending, um die Überschrift absteigend zu sortieren.

Beispiele

```
Sort = "Betrag()"
```

Sortiere die generierten Überschriften nach der Kennzahl "Betrag"

Siehe auch

Sort-Descending (auf Seite 95)

Sort-Descending

Typ

Boolean

Seit

2.1

Beschreibung

In der Standardeinstellung werden Überschriften durch die Eigenschaft "Sort" aufsteigend (also von den kleinsten zu den grössten Sortier-Werten) sortiert. Setzen Sie diese Eigenschaft auf "true" (bzw. geben Sie hier eine Boolean-Expression an, die "true" ergibt), um die Überschriften stattdessen absteigend zu sortieren.

Beispiele

```
Sort-Descending = "true"
```

Sortiert die Überschriften absteigend

```
Sort-Descending = "false"
```

Sortiert die Überschriften aufsteigend

Siehe auch

Sort (auf Seite 95)

Text

Typ

String

Seit

1.1

Beschreibung

Die Formel in dieser Eigenschaft bestimmt den Text, der innerhalb der Überschrift dargestellt wird. Wenn Sie keine Text-Formel definieren, dann werden im Fall einer Iteration die IDs der jeweiligen Dimension-Schlüssel. Wenn Sie keine Iteration und keine Text-Formel angeben, dann bleiben die Überschriften leer.

Beispiele

```
Text = "TOSTRING( Product )"
```

```
Text = "LIMIT( TOSTRING( Product ), 50 )"
```

```
Text = "Product.ShortText"
```

Siehe auch

Iteration (auf Seite 87), Title (auf Seite 97)

Title

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Titel einer Überschrift. Der Titel wird in der oberen, linken Ecke der Pivot-Tabelle (in der selben Zeile bzw. Spalte wie die Überschrift) dargestellt. Wenn Sie keinen Titel angeben, dann bleibt die Ecke leer (ausser Sie haben den Text für die Ecke über die Corner-Text Eigenschaft der Abfrage gesetzt).

Beispiele

```
Title = "'Product'"
```

Siehe auch

Text (auf Seite 96)

Top-Color

Typ

String

Beschreibung

Diese Eigenschaft setzt die Farbe des oberen Rahmens für alle Überschriften, die von diesem Element erzeugt werden. Die Formel muss eine Zeichenkette zurückgeben, die den Namen der Farbe oder die entsprechende HEX-Kodierung (mit einem führenden #) enthält. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, wird kein Rahmen erzeugt.

Diese Eigenschaft setzt nur die Rahmenfarbe der Überschriften, nicht der dazugehörenden Zellen. Verwenden Sie dazu die Eigenschaft Cell-Top-Color.

Beispiele

```
Top-Color = "'black'"
```

Siehe auch

Bottom-Color (auf Seite 65), Cell-Top-Color (auf Seite 77), Left-Color (auf Seite 88), Right-Color (auf Seite 93)

Top-Padding

Typ

Integer

Seit

2.1

Beschreibung

Diese Eigenschaft setzt den oberen Innenabstand (Padding) in Pixeln für alle von diesem Element generierten Überschriften. Das Padding ist der Abstand zwischen dem Text und dem Rahmen der Überschrift und wird als Integer angegeben. Wenn Sie diese Eigenschaft leer lassen oder die Formel NULL ergibt, dann wird das Standard-Padding verwendet.

Diese Eigenschaft setzt nur das Padding der Überschriften, nicht der dazugehörigen Zellen. Verwenden Sie dazu die Eigenschaft Cell-Top-Padding.

Beispiele

```
Top-Padding = "10"
```

Siehe auch

Bottom-Padding (auf Seite 66), Cell-Top-Padding (auf Seite 77), Left-Padding (auf Seite 89), Right-Padding (auf Seite 94)

Vertical-Align

Typ

String

Seit

2.1

Beschreibung

Die Eigenschaft bestimmt die vertikale Ausrichtung für diese Überschrift. Die Formel muss eine der folgenden Zeichenketten ergeben:

- 'top': Oberer Rand
- 'middle': Mittig
- 'bottom': Unterer Rand

Beachten Sie, dass Sie mit dieser Eigenschaft nicht die Ausrichtung der Zellen beeinflussen können, die zu dieser Überschrift gehören. Verwenden Sie dazu die stattdessen die Eigenschaft Cell-Vertical-Align.

Beispiele

```
Vertical-Align = "'top'"
```

```
Vertical-Align = "'middle'"
```

```
Vertical-Align = "'bottom'"
```

Siehe auch

Align (auf Seite 63), Cell-Vertical-Align (auf Seite 78)

Visible

Typ

Boolean

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt, ob eine Überschrift und die dazugehörigen Zellen im Ergebnis sichtbar sein werden, oder nicht. Wenn die Formel "true" ergibt (die für jede Überschrift und Zelle einzeln durchgeführt wird), dann wird die entsprechende Überschrift bzw. Zelle sichtbar. Wenn alle Zellen einer Zeile oder Spalte unsichtbar sind, verschwindet diese komplett.

Diese Eigenschaft gilt sowohl für die Überschrift als auch für deren Zellen. Verwenden Sie die X- und Y-Funktionen, um gezielt bestimmte Zellen oder Zeilen / Spalten unsichtbar zu machen.

Beispiele

`Visible = "false"`

Verstecke komplette Zeile oder Spalte

`Visible = "X() > 1"`

Verstecke nur die erste Spalte

Width

Typ

Integer

Seit

1.2

Beschreibung

Die Eigenschaft "Width" bestimmt die Breite der Überschrift in Pixeln. Wenn keine Breite gesetzt wird, bestimmt das System diese automatisch. Die Formel in dieser Eigenschaft muss eine Zahl vom Typ "Integer" zurückgeben.

Beispiele

`Width = "200"`

Siehe auch

Height (auf Seite 86)

Z-Order

Typ

Integer

Seit

2.0

Beschreibung

Viele verschiedene Eigenschaften einer Überschriften beeinflussen die zu dieser Überschrift gehörenden Zellen. Jede Zelle wird jedoch sowohl von den Überschriften aus der X- als auch von denen aus der Y-Achse beeinflusst. Wenn von beiden Überschriften die selbe Eigenschaft einer Zelle (z.B. die Hintergrundfarbe oder die Formel) auf verschiedene Werte gesetzt wird, dann entscheidet diese Eigenschaft "Z-Order", von welcher Überschrift die Zelle die Eigenschaft übernimmt: Die Überschrift mit der höheren Z-Order gewinnt die Kontrolle über die Zelle.

Die Standard "Z-Order" für eine Überschrift ist "0". Es sind sowohl positive als auch negative Werte erlaubt.

Beispiele

Z-Order = "1"

Überschrift hat eine höhere Priorität als Standard

Kommentare

Author

Typ

Konstante Zeichenkette

Seit

1.1

Beschreibung

Diese (optionale) Eigenschaft enthält den Namen des Kommentar-Autors. Diese Eigenschaft erwartet einen einfachen Text und keine Formel!

Beispiele

```
Author = "admin"
```

Siehe auch

Date (auf Seite 103)

Copy-To-Result

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

1.2

Beschreibung

Wenn diese Eigenschaft den Wert "true" enthält, dann wird der Kommentar bei einem Snapshot des mit exportiert. Ansonsten wird der Kommentar nicht mit übernommen.

Beispiele

```
Copy-To-Result = "true"
```

```
Copy-To-Result = "false"
```

Siehe auch

Export (auf Seite 103)

Date

Typ

Konstante Zeichenkette

Seit

1.1

Beschreibung

Diese (optionale) Eigenschaft enthält das Datum, an dem der Kommentare erstellt wurden. Es gibt keine Formatvorgabe für das Datum, Sie können jedes beliebige verwenden.

Beispiele

```
Date = "01.01.2004"
```

Siehe auch

Author (auf Seite 102)

Export

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

1.2

Beschreibung

Wenn diese Eigenschaft auf "false" gesetzt wird, dann wird der Kommentar beim Exportieren des Berichtes (nach PDF oder Excel) nicht mit übertragen.

Beispiele

```
Export = "true"
```

```
Export = "false"
```

Siehe auch

Copy-To-Result (auf Seite 102)

Formul

Typ

String

Seit

1.2

Beschreibung

Der Text für einen Kommentar kann auf zwei verschiedene Art und Weisen bestimmt werden: Als statischer Text in der Eigenschaft "Text" oder als Formel in dieser Eigenschaft. Wenn Sie diese Eigenschaft verwenden, können Sie den Kommentar-Text mit einer Formel (die einen Text zurückgeben muss) berechnen. In diesem Fall wird der statische Text aus der Eigenschaft "Text" ignoriert.

Beispiele

```
Formula = "'Hello ' + $USER"
```

Siehe auch

Text (auf Seite 104)

Text

Typ

Konstante Zeichenkette

Seit

1.1

Beschreibung

Diese Eigenschaft enthält den (statischen) Text des Kommentar. Wenn Sie den Text Ihres Kommentars mit einer Formel berechnen wollen, dann lassen Sie diese Eigenschaft leer und verwenden stattdessen die Eigenschaft "Formul".

Beispiele

Text = "Dies ist ein Kommentar"

Siehe auch

Formula (auf Seite 104)

KAPITEL 3

Diagramm-Eigenschaften

In diesem Kapitel

Liniendiagramm-Eigenschaften	108
Balkendiagramm-Eigenschaften	147
Tortendiagramm-Eigenschaften	187

Liniendiagramm- Eigenschaften

3DDepth

Typ

Integer

Seit

1.2

Beschreibung

Setzt die Tiefe des 3D-Effekts in Pixeln.

Beispiele

```
3DDepth = "10"
```

Siehe auch

3DModeOn

3DModeOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet den 3D-Modus an oder aus.

Beispiele

```
3DModeOn = "true"
```

```
3DModeOn = "false"
```

Siehe auch

3DDepth

AutoLabelSpacingOn

Typ

Boolean-Expression

Seit

1.2

Beschreibung

Normalerweise werden alle Kategorien dargestellt, auch wenn nicht genug Platz für alle Beschriftungen existiert (dann überlappen sie). Wenn diese Eigenschaft auf "true" gesetzt werden, dann werden nur die Beschriftungen dargestellt, für die genug Platz ist - alle anderen werden ausgelassen.

Beispiele

```
AutoLabelSpacingOn = "true"
```

```
AutoLabelSpacingOn = "false"
```

Background

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Hintergrundfarbe ausserhalb des eigentlichen Diagramm-Bereichs. Die Formel muss ein gültiges Farb-Format als Ergebnis haben.

Beispiele

```
Background = "'green'"
```

```
Background = "'#FFFF00'"
```

Siehe auch

ChartBackground, ChartForeground, Foreground

ChartBackground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Hintergrundfarbe des eigentlichen Diagramm-Bereichs. Das Ergebnis der Formel muss ein gültiges Farbformat sein.

Beispiele

```
ChartBackground = "'blue'"
```

```
ChartBackground = "'#FFFF00'"
```

ChartForeground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Rahmenfarbe des eigentlichen Diagramm-Bereichs. Das Ergebnis der Formel muss ein gültiges Farbformat sein.

Beispiele

```
ChartForeground = "'black'"
```

```
ChartForeground = "'#FFFF00'"
```

ChartTitle

Typ

String

Seit

1.2

Beschreibung

Das Ergebnis der Formel in dieser Eigenschaft bestimmt den Titel des Diagramms.

Beispiele

```
ChartTitle = "'Turnover'"
```

ConnectedLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Normalerweise wird eine Linie mit fehlenden Werte mit Lücken dargestellt. Mit dieser Eigenschaft können Sie diese Lücken überbrücken und eine Linie zum nächsten Wert in der Linie zeichnen lassen. Das funktioniert nicht in der gestapelten (Stacked) Darstellung von Linien-Diagrammen.

Beispiele

```
ConnectedLinesOn = "true"
```

```
ConnectedLinesOn = "false"
```

DefaultGridLinesCdor

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Standard-Gitterfarbe.

Beispiele

```
DefaultGridLinesColor = "'black'"
```

```
DefaultGridLinesColor = "'#FFFF00'"
```

DefaultGridLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Diese Eigenschaft schaltet die vertikalen Gitterlinien ein bzw. aus.

Beispiele

```
DefaultGridLinesOn = "true"
```

```
DefaultGridLinesOn = "false"
```

FloatingLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Werte- und Kategorien-Beschriftungen, die beim Überfahren mit der Maus erscheinen.

Beispiele

```
FloatingLabelFont = "'Arial'"
```

FloatingOnLegendOff

Typ

Boolean

Seit

1.2

Beschreibung

Normalerweise werden für alle Werte Beschriftungen dargestellt, wenn der Benutzer mit der Maus über einen Serien-Namen in der Legende fährt. Diese Eigenschaft schaltet dieses Verhalten aus.

Beispiele

```
FloatingOnLegendOff = "true"
```

```
FloatingOnLegendOff = "false"
```

Font

Typ

String

Seit

1.2

Beschreibung

Setzt den Standard-Zeichensatz für die Diagramm-Beschriftungen.

Beispiele

```
Font = "'Arial'"
```

Foreground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Farbe des Titels, der Legenden-Beschriftungen, der Werte- und Kategorien-Beschriftungen und der Skala. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
Foreground = "'blue'"
```

```
Foreground = "'#ffff00'"
```

GraphInsets

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um den Abstand zwischen dem eigentlichen Diagramm und den äußeren Rahmen zu bestimmen. Der Abstand wird bestimmt mit 4 verschiedenen Parametern in der folgenden Reihenfolge bestehen: Oben, links, unten und rechts. Ein Wert von -1 steht für den jeweiligen Standard-Abstand. Das Ergebnis muss eine Zeichenkette sein, die diese 4 Wert (mit Kommata getrennt) enthält.

Beispiele

```
GraphInsets = "'-1, 50, -1, -1'"
```

GridAdjustmentOn

Typ

Boolean

Seit

1.2

Beschreibung

Der Benutzer kann (wenn diese Formel true ergibt) das Gitter beim Betrachten durch das Greifen und Verschieben der Gitter-Kante mit der Maus verändern. Ein Doppelklick auf die Kante setzt das Gitter wieder in die Normalposition zurück.

Beispiele

```
GridAdjustmentOn = "true"
```

```
GridAdjustmentOn = "false"
```

GridImage

Typ

String

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie ein Hintergrundbild für das Diagramm-Gitter bestimmen.

Beispiele

```
GridImage = "'/iolap/images/chartbg.gif'"
```

```
GridImage = "'http://myserver/images/chartbg.gif'"
```

GridLineColors

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft kann die Farbe einzelner vertikal Gitterlinien gesetzt werden.

Beispiele

```
GridLinesColors = "'black'"
```

```
GridLinesColors = "'#FFFF00'"
```

GridLines

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Position der vertikalen Gitterlinien. Geben Sie einen Wert je Gitterlinie an. Der Wert muss jeweils relativ zur gesamten Axen-Breite (zwischen 0 und 100) angegeben werden. Alle Werte werden in For einer einzelnen Zeichenkette, durch Kommata getrennt, erwartet.

Beispiele

```
GridLines = "'10,20,30,40,50,60,70,80,90'"
```

GridLinesCobr

Typ

String

Seit

1.2

Beschreibung

Setzt die Farbe für alle Gitterlinien ausser den Standard-Gitterlinien. Die Formel muss eine Farbe im gültigen Farbformat zurückgeben.

Beispiele

```
GridLinesColor = "'black'"
```

```
GridLinesColor = "'#FFFF00'"
```

Label_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine Beschriftung an beliebiger Stelle des Diagramms darzustellen. Der erste Teil der Parameter ist der eigentliche Text, der zweite und dritte sind jeweils die X- und Y-Position. Wenn X oder Y zwischen 0 und 1 liegen, dann bestimmen sie die Position der Beschriftung relativ zur Diagrammgrösse. Ansonsten werden die Angaben als absolute Angaben (in Pixeln) interpretiert. Ausserdem gibt es optionale 4. und 5. Parameter, die die Nummer und Serie einer bestehenden Beschriftung bestimmen, auf die Sie zeigen möchten.

Beispiele

```
Label = "'orange sales,100,100'"
```

```
Label = "'apple sales,0.4,0.5'"
```

```
Label = "'banana sales,200,200,4,0'"
```

Siehe auch

LabelUrl_0 (auf Seite 117), LabelUrlTarget_0 (auf Seite 118)

LabelUrl_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine URL an eine Beschriftung zu hängen. Die URL wird dann geöffnet, wenn der Benutzer auf die Beschriftung klickt.

Beispiele

```
LabelURL_0 = "'details.html'"
```

Siehe auch

Label_0 (auf Seite 117), LabelUrlTarget_0 (auf Seite 118)

LabelUrlTarget_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kontrolliert, auf welcher HTML-Seite eine URL geöffnet wird, wenn der Benutzer auf eine Beschriftung klickt.

Beispiele

```
LabelURLTarget_0 = "'_blank'"
```

Siehe auch

Label_0 (auf Seite 117), LabelUrl_0 (auf Seite 117)

LegendColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der Kästchen für die Legende. Wenn keine Farben definiert werden, dann werden die der SampleColors Eigenschaft verwendet. Die Formel muss eine Zeichenkette mit der Liste der Farben (durch Kommata getrennt) zurückgeben.

Beispiele

```
LegendColors = "red,green,blue"
```

LegendColumns

Typ

Integer

Seit

2.1

Beschreibung

Bestimmt die Anzahl der Spalten, die zur Darstellung der Legende verwendet werden sollen.

Beispiele

```
LegendColumns = "2"
```

LegendFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Legende.

Beispiele

```
LegendFont = "'Arial'"
```

```
LegendFont = "'Arial, bold, 12'"
```

LegendImage

Typ

String

Seit

1.2

Beschreibung

Bestimmt eine Grafik, die vor den Legenden-Beschriftungen dargestellt werden soll (anstelle der einfachen Legenden-Kästchen).

Beispiele

```
LegendImage = "'/iolap/images/legend.gif'"
```

LegendOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Darstellung der Legende ein bzw. aus

Beispiele

```
LegendOn = "true"
```

```
LegendOn = "false"
```

LegendPosition

Typ

String ('right', 'left', 'top' oder 'bottom')

Seit

1.2

Beschreibung

Bestimmt die Position der Legende. Mögliche Ergebnisse der Formel sind 'right' (rechts), 'left' (links), 'top' (oben) oder 'bottom' (unten).

Beispiele

```
LegendPosition = "'right'"
```

```
LegendPosition = "'left'"
```

```
LegendPosition = "'top'"
```

```
LegendPosition = "'bottom'"
```

LegendReverseOn

Typ

Boolean

Seit

2.1

Beschreibung

Schaltet die Rückwärtsdarstellung der Legende ein bzw. aus. Normalerweise wird die Legende von links nach rechts bzw. von oben nach unten dargestellt.

Beispiele

```
LegendReverseOn = "true"
```

```
LegendReverseOn = "false"
```

LineStyle

Typ

String

Seit

2.1

Beschreibung

Verwenden Sie diese Eigenschaft, um gestrichelte oder gepunktete Linien darzustellen. Erwartet wird eine Liste von Werten (getrennt durch das | Zeichen) von (sich wiederholenden) Strich- und Leerraum-Längen. Geben Sie nur einen Wert an, wenn die Striche und Leerräume die selbe Länge haben sollen. Z.B. erzeugt `LineStyle = "3"` einen Strich von 3 Pixeln Länge mit einem folgenden Leerraum gleicher Länge und `LineStyle = "3|6"` einen Strich der Länge 3 gefolgt von einem Leerraum der Länge 6. Sie können ausserdem mehrere (durch Kommata getrennte) Strokes definieren, die dann für die verschiedenen Serien gelten.

Beispiele

```
LineStyle = "3"
```

```
LineStyle = "3|6"
```

```
LineStyle = "3|6,2|4"
```

LineWidth

Typ

String

Seit

1.2

Beschreibung

Setzt die Liniendicke für die Serien. Sie können die Dicken für die verschiedenen Serien angeben, in dem Sie eine Komma-separierte Liste angeben. Wenn Sie mehr als eine Serie im Diagramm darstellen, können Sie auch nur einen Wert angeben, der dann für alle gilt. Die Standard-Dicke für Linien ist 2.

Beispiele

```
LineWidth = "'1'"
```

```
LineWidth = "'1,2'"
```

LowerRange

Typ

Number

Seit

1.2

Beschreibung

Setzt die untere Grenze der Werte-Skala (Y-Achse). Wenn keine Grenze gesetzt ist, wird diese automatisch an den kleinsten Wert aus dem Diagramm angepasst (oder auf 0 gesetzt, wenn kein negativer Wert im Diagramm vorkommt).

Beispiele

```
LowerRange = "40"
```

MaxValueLineCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die maximale Anzahl an vertikalen Gitternetzlinien für dieses Diagramm. Sie können (bis zu einem gewissen Grad) diese Eigenschaft dazu verwenden, um den Absatz zwischen den Linien zu bestimmen.

Beispiele

```
MaxValueLineCount = "5"
```

Range

Typ

Number

Seit

1.2

Beschreibung

Setzt die obere Grenze für die Wertskala (Y-Achse). Wenn keine Grenze definiert wird, dann wird diese automatisch an den grössten Wert des Diagramms angepasst.

Beispiele

```
Range = "100"
```

RangeAdjusted

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt, welche Werte-Skalen durch welchen Regler verändert werden. Normalerweise kontrolliert der Regler 1 die Skala 1 usw.

Beispiele

```
RangeAdjusted = "1"
```

```
RangeAdjusted = "2"
```

RangeAdjusterOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet den Regler für die Werte-Skala ein. Der Regler ermöglicht es dem Benutzer, die obere und untere Grenze der Skala mit der Maus einzustellen. Normalerweise wird der Regler an der rechten Seite des Diagramms dargestellt. Das kann mit der Eigenschaft `RangeAdjusterPosition` geändert werden.

Beispiele

```
RangeAdjusterOn = "true"
```

```
RangeAdjusterOn = "false"
```

RangeAdjusterPosition

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Position des Reglers für die Werte-Skala. 0 steht für links, 1 für rechts. Normalerweise wird der Regler rechts dargestellt.

Beispiele

```
RangeAdjusterPosition = "1"
```

RangeAxisLabel

Typ

String

Seit

1.2

Beschreibung

Setzt eine Beschriftung neben der Werte-Skala (Y-Achse). Normalerweise wird diese Beschriftung horizontal dargestellt, Sie können den Drehwinkel der Beschriftung jedoch mit der Eigenschaft `RangeAxisLabelAngle` ändern.

Beispiele

```
RangeAxisLabel = "'Amount'"
```

RangeAxisLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Setzt den Drehwinkel der Beschriftung (im Uhrzeigersinn) für die Werte-Skala (Y-Achse) in Grad.

Beispiele

```
RangeAxisLabelAngle = "270"
```

RangeAxisLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die optionale Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeAxisLabelFont = "'Arial'"
```

RangeColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe der Werte-Skala (Y-Achse) und Skalen-Markierungen. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
RangeColor = "'green'"
```

```
RangeColor = "'#FFFF00'"
```

RangeDecimalCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl der Nachkommastellen für die Darstellung der Werte auf der Werte-Skala (Y-Achse).

Beispiele

```
RangeDecimalCount = "10"
```

RangeLabelFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Beschriftungen der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelFont = "'Arial'"
```

RangeLabelPostfix

Typ

String

Seit

1.2

Beschreibung

Definiert einen Postfix (einen Text hinter den einzelnen Werten) für die Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelPostfix = "'ms'"
```

RangeLabelPrefix

Typ

String

Seit

1.2

Beschreibung

Definiert einen Prefix (einen Text vor den einzelnen Werten) für die Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelPrefix = "'$'"
```

RangeLabelsOff

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Beschriftung der Werte-Skala (Y-Achse) ab, wenn die Formel true ergibt.

Beispiele

```
RangeLabelsOff = "true"
```

RangeOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Werte-Skala (Y-Achse) ein bzw. aus.

Beispiele

```
RangeOn = "true"
```

RangePosition

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Position der Werte-Skala (Y-Achse). 0 steht für links, 1 für rechts. Die Standard-Position der ersten Skala ist links, die der zweiten rechts.

Beispiele

```
RangePosition = "2"
```

RangeStep

Typ

Number

Seit

1.2

Beschreibung

Wenn Sie die obere Grenze der Werte-Skala (den Range) nicht selber bestimmen, wird diese automatisch auf den grössten Wert oder 0 gesetzt. Wenn der "RangeStep" mit dieser Eigenschaft gesetzt wird, dann wird die obere Grenze auf den nächsten durch diesen Wert teilbaren Wert gesetzt.

Wenn der grösste Wert im Diagramm z.B. 325 und ist und der RangeStep auf 100 gesetzt wird, dann wird die obere Grenze automatisch auf 400 gesetzt.

Beispiele

```
RangeStep = "100"
```

SampleAxisLabel

Typ

String

Seit

1.2

Beschreibung

Setzt eine Beschriftung unterhalb der X-Achse.

Beispiele

```
SampleAxisLabel = "'Date'"
```

SampleAxisLabelFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Kategorien (Sample-Achse) (normalerweise die X-Achse).

Beispiele

```
SampleAxisLabelFont = "'Arial'"
```

SampleAxisRange

Typ

Number

Seit

1.2

Beschreibung

Bestimmt den Darstellungsbereich für die Kategorien (Sample-Achse). Der Bereich wird dazu verwendet, um zu berechnen wo die vertikalen Gitternetzlinien dargestellt werden sollen.

Beispiele

```
SampleAxisRange = "1000"
```

SampleColors

Typ

String

Seit

1.2

Beschreibung

Setzt die Farben der Linien im Diagramm. Diese Eigenschaft erwartet eine Formel, die die Farben (durch Kommata getrennt) in einer Zeichenkette zurückgibt.

Beispiele

```
SampleColors = "'red,green,blue'"
```

SampleDecimalCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl der Nachkommastellen, die bei der Darstellung der Werte für die einzelnen Balken angezeigt werden.

Beispiele

```
SampleDecimalCount = "2"
```

SampleHighlightImage

Typ

String

Seit

2.1

Beschreibung

Verwenden Sie diese Eigenschaft, um Grafiken für alle Werte-Punkte der Linien darzustellen.

Beispiele

```
SampleHightlightImage = "'/iolap/images/dot.gif'"
```

SampleHightlightOn

Typ

String

Seit

1.2

Beschreibung

Jede Datenserie kann optional verschiedene Punkte-Markierungen (Highlights) für ihre Linie darstellen, entweder einen Kreis, ein Quadrat oder eine Raute. Verwenden Sie diese Eigenschaft, um die Darstellung von Hightlights für die Serien ein bzw. auszuschalten.

Diese Formel muss eine Liste von Werten (jeweils true "für" an oder "false" für aus) in einer Zeichenkette und durch Kommata getrennt zurückgeben. Jeder Wert steht für eine Datenserie.

Beispiele

```
SampleHightlightOn = "'true, false'"
```

SampleHightlightSize

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um die Grösse die einzelnen Punkte-Markierungen (Highlights) für jede Daten-Serie zu setzen. Die Standard-Grösse ist 6.

Diese Eigenschaft muss die Grössen als Zeichenkette, in der die Grössen durch Kommata getrennt angegeben werden, zurückgeben.

Beispiele

```
SampleHighlightSize = "'2, 4, 2'"
```

SampleHighlightStyle

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft um die Art der Punkte-Markierungen (Highlights) für jede Daten-Serie zu setzen. Die Möglichen Arten sind Kreis ('circle', 'circle_opaque' oder 'circle_filled'), ein Quadrat ('square', 'square_opaque' oder 'square_filled') oder eine Raute ('diamond', 'diamond_opaque' oder 'diamond_filled').

Diese Eigenschaft muss eine Zeichenkette, in der die Arten durch Kommata getrennt angegeben werden, zurückgeben.

Beispiele

```
SampleHighlightStyle = "'diamond, circle'"
```

SampleLabelAngle

Typ

Integer

Seit

2.1

Beschreibung

Setzt den Drehwinkel (im Uhrzeigersinn) der statischen Kategorie-Beschriftungen in Grad.

Beispiele

```
SampleLabelAngle = "270"
```

SampleLabelColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben für die Beschriftung der Kategorien (Sample- bzw. X-Achse).

Beispiele

```
SampleLabelColors = "'red,green,blue'"
```

SampleLabelFont

Typ

String

Seit

2.1

Beschreibung

Setzt den Zeichensatz für die statischen Kategorie-Beschriftungen.

Beispiele

```
SampleLabelFont = "'Arial'"
```

```
SampleLabelFont = "'Dialog, plain, 12'"
```

SampleLabelsOn

Typ

String

Seit

2.1

Beschreibung

Macht die Beschriftung der Kategorien (X-Achse) sichtbar.

Beispiele

```
SampleLabelsOn = "true"
```

```
SampleLabelsOn = "false"
```

SampleLabelStyle

Typ

String ('below', 'floating', 'outside' oder 'below_and_floating')

Seit

1.2

Beschreibung

Bestimmt, wie Kategorie-Beschriftungen dargestellt werden sollten. Diese können entweder unterhalb des eigentlichen Diagramm-Bereichs ('below'), neben den Punkten den Linien ('outside'), als "Tool Top" über dem jeweiligen Punkt (der sichtbar wird, wenn der Benutzer mit der Maus über den Punkt fährt) oder als Kombination ('below_and_floating') dargestellt werden.

Beispiele

```
SampleLabelStyle = "'below'"
```

```
SampleLabelStyle = "'floating'"
```

```
SampleLabelStyle = "'below_and_floating'"
```

SampleScrdlerOn

Typ

Boolean

Seit

1.2

Beschreibung

Der Benutzer kann (mit der X-Achse) den sichtbaren Bereich des Diagramms wählen und vergrössern, wenn Sie den "Sample Scroller" mit dieser Eigenschaft einschalten.

Beispiele

```
SampleScrollerOn = "true"
```

```
SampleScrollerOn = "false"
```

SeriesLabelColors

Typ

String

Seit

1.2

Beschreibung

Setzt die Farben der Serienbezeichnungen in der Legende. Diese Eigenschaft erwartet eine Formel, die die Farben (mit Kommata getrennt) in einer Liste zurückgibt.

Beispiele

```
SeriesLabelColors = "'red, green, blue'"
```

SeriesLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Serienbezeichnungen als "Tool Top" ein. Die Bezeichnungen werden dann sichtbar, wenn der Benutzer mit der Maus über die Serie (Linie) fährt.

Beispiele

```
SeriesLabelsOn = "true"
```

```
SeriesLabelsOn = "false"
```

SeriesLabelStyle

Typ

String ('inside' or 'outside')

Seit

1.2

Beschreibung

Legt fest, wie die Bezeichnungen der Datenserien dargestellt werden. Die Bezeichnung kann entweder neben den Punkten ('outside') auf der Linie oder als "Tool Tip" (der erscheint, wenn der Benutzer mit der Maus über den Punkt fährt) dargestellt werden ('inside').

Beispiele

```
SampleLabelStyle = "'inside'"
```

```
SampleLabelStyle = "'outside'"
```

SeriesLineOff

Typ

Value

Seit

1.2

Beschreibung

Macht die Linien aller oder einzelner Datenserien unsichtbar. Zusammen mit den optionalen Punkt-Markierungen können so Plotter-Charts (die nur aus Punkten bestehen) dargestellt werden.

Beispiele

```
SeriesLineOff = "false"
```

Macht alle Linien unsichtbar

```
SeriesLineOff = "'1,3'"
```

Macht die Linien der Datenserien 1 und 3 unsichtbar

SingleClickURLOn

Typ

Boolean

Seit

1.2

Beschreibung

Setzen Sie diese Eigenschaft auf true, damit Links (z.B. in Drilldown-fähigen Diagrammen) mit einem einzelnen Klick statt mit einem Doppelklick angewählt werden können.

Beispiele

```
SingleClickURLOn = "true"
```

```
SingleClickURLOn = "false"
```

StackedOn

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft true ergibt, wird das Liniendiagramm mit ausgefüllten und aufeinander "gestapelten" Bereichen dargestellt (Area Chart).

Beispiele

```
StackedOn = "true"
```

```
StackedOn = "false"
```

TargetLabelsPosition

Typ

String

Seit

2.1

Beschreibung

Bestimmt die Position der Beschriftung für die TargetValueLine. Kann entweder links ('left') oder rechts ('right') sein. Im Normalfall werden die Beschriftungen auf der selben Seite dargestellt wie die Werteskala.

Beispiele

```
TargetLabelsPosition = "'left'"
```

```
TargetLabelsPosition = "'right'"
```

TargetValueLine

Typ

String

Seit

1.2

Beschreibung

Sie können einzelne Gitternetzlinien mit einer spezifischen Farbe und einem frei definierbaren Text zum Diagramm hinzufügen. Der erste Teil des Parameters ist der Text, der zweite der Wert (an dem die Linie auf der Werte-Achse dargestellt werden soll) und der dritte die Farbe der Linie. Es gibt ausserdem einen optionalen vierten Teil, der bestimmt ob nur der Text oder auch der Wert dargestellt werden soll (das ist der Normalfall). Alle Teile müssen (durch Kommata getrennt) in einer Zeichenkette angegeben werden.

Beispiele

```
TargetValueLine = "break even, 150, green"
```

ThousandsDelimiter

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um das Tausender-Trennzeichen für alle Werte-Darstellungen zu definieren.

Beispiele

```
ThousandsDelimiter = "','"
```

TitleFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz der Diagramm-Überschrift.

Beispiele

```
TitleFont = "'Arial'"
```

UrlTarget

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt, in welchem Browser-Fenster neue URLs beim Anklicken von Links geöffnet werden:

- `_self`: Öffnet das Ziel im selben Fenster.
- `_blank`: Öffnet das Ziel in einem neuen, leeren Fenster.
- `name`: Öffnet das Ziel in einem Fenster mit bestimmten Namen.

Beispiele

```
UrlTarget = "'_self'"
```

```
UrlTarget = "'window1'"
```

ValueLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt den Drehwinkel (im Uhrzeigersinn) für die Werte-Beschriftungen der Linien.

Beispiele

```
ValueLabelAngle = "270"
```

ValueLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Werte-Beschriftungen der Linien.

Beispiele

```
ValueLabelFont = "'Arial'"
```

ValueLabelPostfix

Typ

String

Seit

1.2

Beschreibung

Bestimmt einen Text (Postfix), der hinter den Werte-Beschriftungen der Linien angefügt wird.

Beispiele

```
ValueLabelPostfix = "'ms'"
```

ValueLabelPrefix

Typ

String

Seit

1.2

Beschreibung

Bestimmt einen Text (Prefix), der vor den Werte-Beschriftungen der Linien dargestellt wird.

Beispiele

```
ValueLabelPrefix = "'$'"
```

ValueLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Darstellung der Werte innerhalb der Linien ein.

Beispiele

```
ValueLabelsOn = "true"
```

```
ValueLabelsOn = "false"
```

ValueLabelStyle

Typ

String ('inside', 'outside' or 'floating')

Seit

1.2

Beschreibung

Bestimmt die Darstellungsart der Werte innerhalb der Linien. Die Werte können neben dem Punkt ('outside'), direkt auf dem Punkt ('inside') oder als "Tool Tip" ('floating') dargestellt werden.

Beispiele

```
ValueLabelStyle = "'inside'"
```

```
ValueLabelStyle = "'outside'"
```

```
ValueLabelStyle = "'floating'"
```

ValueLinesColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe der vertikalen Gitternetzlinien. Die Formel muss eine gültige Farbe als Zeichenkette zurückgeben.

Beispiele

```
ValueLinesColor = "'red'"
```

```
ValueLinesColor = "'#FFFF00'"
```

ValueLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Aktiviert die horizontalen Gitternetzlinien im Diagramm-Hintergrund.

Beispiele

```
ValueLinesOn = "true"
```

```
ValueLinesOn = "false"
```

VisibleSamples

Typ

String

Seit

1.2

Beschreibung

Bestimmt, welcher Teil der Kategorien (einzelnen Punkte) sichtbar sein soll. Der erste Teil der Zeichenkette ist die Nummer des ersten Balkens, der zweite (durch ein Komma getrennt) die Anzahl der darzustellenden Punkte. Wenn Sie einen "Sample Scroller" verwenden, bestimmt diese Eigenschaft die Starteinstellung des Scrollers.

Beispiele

```
VisibleSamples = "10,25"
```

ZoomOn

Typ

Boolean

Seit

2.1

Beschreibung

Wenn die Eigenschaft "ZoomOn" true ergibt, kann der Benutzer mit der Maus Ausschnitte des Diagramms vergrößern.

Beispiele

```
ZoomOn = "true"
```

```
ZoomOn = "false"
```

Balkendiagramm-Eigenschaften

3DDepth

Typ

Integer

Seit

1.2

Beschreibung

Setzt die Tiefe des 3D-Effekts in Pixeln.

Beispiele

3DDepth = "10"

Siehe auch

3DModeOn (auf Seite 147)

3DModeOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet den 3D-Modus an oder aus.

Beispiele

3DModeOn = "true"

3DModeOn = "false"

Siehe auch

3DDepth (auf Seite 147)

AutoLabelSpacingOn

Typ

Boolean

Seit

1.2

Beschreibung

Normalerweise werden alle Kategorien dargestellt, auch wenn nicht genug Platz für alle Beschriftungen existiert (dann überlappen sie). Wenn diese Eigenschaft auf "true" gesetzt werden, dann werden nur die Beschriftungen dargestellt, für die genug Platz ist - alle anderen werden ausgelassen.

Beispiele

```
AutoLabelSpacingOn = "true"
```

```
AutoLabelSpacingOn = "false"
```

Background

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Hintergrundfarbe ausserhalb des eigentlichen Diagramm-Bereichs. Die Formel muss ein gültiges Farb-Format als Ergebnis haben.

Beispiele

```
Background = "'green'"
```

```
Background = "'#FFFF00'"
```

Siehe auch

ChartBackground (auf Seite 153), ChartForeground (auf Seite 154),
Foreground (auf Seite 157)

BarAlignment

Typ

String ('vertical' oder 'horizontal')

Seit

1.2

Beispiele

Setzt die Ausrichtung der Balken auf vertikal (Standardeinstellung)
oder horizontal.

Beispiele

```
BarAlignment = "'vertical'"
```

```
BarAlignment = "'horizontal'"
```

BarLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Setzt den Drehwinkel (im Uhrzeigersinn) der Balken-Beschriftungen.

Beispiele

```
BarLabelAngle = "270"
```

BarLabelColors

Typ

String

Seit

2.1

Beschreibung

Setzt die Farbe der Balken-Beschriftungen. Diese Eigenschaft betrifft nur die Beschriftungen unterhalb des eigentlichen Diagramm-Bereichs.

Beispiele

```
BarLabelColors = "'red, #cc00cc, blue'"
```

BarLabelFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die statischen Balken-Beschriftungen.

Beispiele

```
BarLabelFont = "'Arial'"
```

```
BarLabelFont = "'Arial, bold, 12'"
```

BarLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Balken-Beschriftungen ein bzw. aus.

Beispiele

```
BarLabelsOn = "true"
```

```
BarLabelsOn = "false"
```

BarLabelStyle

Typ

String ('below', 'floating' oder 'below_and_floating')

Seit

1.2

Beschreibung

Setzt die Darstellungsart für Balken-Beschriftungen:

- 'below': unterhalb der Balken
- 'floating': sichtbar, wenn der Benutzer mit der Maus über einen Balken fährt
- 'below_and_floating': Kombination aus beidem

Beispiele

```
BarLabelStyle = "'below'"
```

```
BarLabelStyle = "'floating'"
```

```
BarLabelStyle = "'below_and_floating'"
```

BarOutlineColor

Typ

String

Seit

1.2

Beschreibung

Die Farbe der Balken-Umrahmung. Die Formel muss eine gültige Farbe als Ergebnis haben.

Beispiele

```
BarOutlineColor = "'red'"
```

```
BarOutlineColor = "'#FFFF00'"
```

BarOutlineOff

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet (wenn die Formel true ergibt) die Rahmenfarbe der Balken ab.

Beispiele

```
BarOutlineOff = "true"
```

```
BarOutlineOff = "false"
```

BarType

Typ

String ('stacked' oder 'side')

Seit

1.2

Beschreibung

Setzt die Darstellungsart der Balken für den Fall, das mehrere Datenserien vorhanden sind: Entweder werden die Balken der verschiedenen Serien aufeinander gestapelt ('stacked') oder nebeneinander ('side' = Standardeinstellung) dargestellt.

Beispiele

```
BarType = "'stacked'"
```

```
BarType = "'side'"
```

BarWidth

Typ

Double

Seit

1.2

Beschreibung

Setzt die relative Breite jedes Balkens (von 0.0 bis 1.0). Wenn die Formel 1.0 ergibt, dann wird kein Platz mehr zwischen den Balken freigelassen.

Beispiele

```
BarWidth = "1.0"
```

```
BarWidth = "1.5"
```

ChartBackground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Hintergrundfarbe des eigentlichen Diagramm-Bereichs. Das Ergebnis der Formel muss ein gültiges Farbformat sein.

Beispiele

```
ChartBackground = "'blue'"
```

```
ChartBackground = "'#FFFF00'"
```

ChartForeground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Rahmenfarbe des eigentlichen Diagramm-Bereichs. Das Ergebnis der Formel muss ein gültiges Farbformat sein.

Beispiele

```
ChartForeground = "'black'"
```

```
ChartForeground = "'#FFFF00'"
```

ChartTitle

Typ

String

Seit

1.2

Beschreibung

Das Ergebnis der Formel in dieser Eigenschaft bestimmt den Titel des Diagramms.

Beispiele

```
ChartTitle = "'Turnover'"
```

DefaultGridLinesCdor

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Standard-Gitterfarbe.

Beispiele

```
DefaultGridLinesColor = "'black'"
```

```
DefaultGridLinesColor = "'#FFFF00'"
```

DefaultGridLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Diese Eigenschaft schaltet die vertikalen Gitterlinien ein bzw. aus.

Beispiele

```
DefaultGridLinesOn = "true"
```

```
DefaultGridLinesOn = "false"
```

FloatingLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Werte- und Balkenbeschriftungen, die beim Überfahren mit der Maus erscheinen.

Beispiele

```
FloatingLabelFont = "'Arial'"
```

FloatingOnLegendOff

Typ

Boolean

Seit

1.2

Beschreibung

Normalerweise werden für alle Werte Beschriftungen dargestellt, wenn der Benutzer mit der Maus über einen Serien-Namen in der Legende fährt. Diese Eigenschaft schaltet dieses Verhalten aus.

Beispiele

```
FloatingOnLegendOff = "true"
```

```
FloatingOnLegendOff = "false"
```

Font

Typ

String

Seit

1.2

Beschreibung

Setzt den Standard-Zeichensatz für die Diagramm-Beschriftungen.

Beispiele

```
Font = "'Arial'"
```

Foreground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Farbe des Titels, der Legenden-Beschriftungen, der Werte-Beschriftungen, der Skala und der Balken-Rahmen. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
Foreground = "'blue'"
```

```
Foreground = "'#ffff00'"
```

GraphInsets

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um den Abstand zwischen dem eigentlichen Diagramm und den äußeren Rahmen zu bestimmen. Der Abstand wird bestimmt mit 4 verschiedenen Parametern in der folgenden Reihenfolge bestehen: Oben, links, unten und rechts. Ein Wert von -1 steht für den jeweiligen Standard-Abstand. Das Ergebnis muss eine Zeichenkette sein, die diese 4 Wert (mit Kommata getrennt) enthält.

Beispiele

```
GraphInsets = "'-1, 50, -1, -1'"
```

GridAdjustmentOn

Typ

Boolean

Seit

1.2

Beschreibung

Der Benutzer kann (wenn diese Formel true ergibt) das Gitter beim Betrachten durch das Greifen und Verschieben der Gitter-Kante mit der Maus verändern. Ein Doppelklick auf die Kante setzt das Gitter wieder in die Normalposition zurück.

Beispiele

```
GridAdjustmentOn = "true"
```

```
GridAdjustmentOn = "false"
```

GridImage

Typ

String

Seit

2.1

Beschreibung

Mit dieser Eigenschaft können Sie ein Hintergrundbild für das Diagramm-Gitter bestimmen.

Beispiele

```
GridImage = "'/iolap/images/chartbg.gif'"
```

```
GridImage = "'http://myserver/images/chartbg.gif'"
```

GridLineColors

Typ

String

Seit

1.2

Beschreibung

Mit dieser Eigenschaft kann die Farbe einzelner vertikal Gitterlinien gesetzt werden.

Beispiele

```
GridLinesColors = "'black'"
```

```
GridLinesColors = "'#FFFF00'"
```

GridLines

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Position der vertikalen Gitterlinien. Geben Sie einen Wert je Gitterlinie an. Der Wert muss jeweils relativ zur gesamten Axen-Breite (zwischen 0 und 100) angegeben werden. Alle Werte werden in For einer einzelnen Zeichenkette, durch Kommata getrennt, erwartet.

Beispiele

```
GridLines = "'10,20,30,40,50,60,70,80,90'"
```

GridLinesCobr

Typ

String

Seit

1.2

Beschreibung

Setzt die Farbe für alle Gitterlinien ausser den Standard-Gitterlinien. Die Formelmuss eine Farbe im gültigen Farbformat zurückgeben.

Beispiele

```
GridLinesColor = "'black'"
```

```
GridLinesColor = "'#FFFF00'"
```

Label_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine Beschriftung an beliebiger Stelle des Diagramms darzustellen. Der erste Teil der Parameter ist der eigentliche Text, der zweite und dritte sind jeweils die X- und Y-Position. Wenn X oder Y zwischen 0 und 1 liegen, dann bestimmen sie die Position der Beschriftung relativ zur Diagrammgrösse. Ansonsten werden die Angaben als absolute Angaben (in Pixeln) interpretiert. Ausserdem gibt es optionale 4. und 5. Parameter, die die Nummer und Serie einer bestehenden Beschriftung bestimmen, auf die Sie zeigen möchten.

Beispiele

```
Label = "'orange sales,100,100'"
```

```
Label = "'apple sales,0.4,0.5'"
```

```
Label = "'banana sales,200,200,4,0'"
```

Siehe auch

LabelUrl_0 (auf Seite 160), LabelUrlTarget_0 (auf Seite 161)

LabelUrl_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine URL an eine Beschriftung zu hängen. Die URL wird dann geöffnet, wenn der Benutzer auf die Beschriftung klickt.

Beispiele

```
LabelURL_0 = "'details.html'"
```

Siehe auch

Label_0 (auf Seite 160), LabelUrlTarget_0 (auf Seite 161)

LabelUrlTarget_0

Typ

String-Expression

Seit

2.1

Beschreibung

Diese Eigenschaft kontrolliert, auf welcher HTML-Seite eine URL geöffnet wird, wenn der Benutzer auf eine Beschriftung klickt.

Beispiele

```
LabelURLTarget_0 = "'_blank'"
```

Siehe auch

Label_0 (auf Seite 160), LabelUrl_0 (auf Seite 160)

LegendColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der Kästchen für die Legende. Wenn keine Farben definiert werden, dann werden die der SampleColors Eigenschaft verwendet. Die Formel muss eine Zeichenkette mit der Liste der Farben (durch Kommata getrennt) zurückgeben.

Beispiele

```
LegendColors = " 'red,green,blue' "
```

LegendColumns

Typ

Integer

Seit

2.1

Beschreibung

Bestimmt die Anzahl der Spalten, die zur Darstellung der Legende verwendet werden sollen.

Beispiele

```
LegendColumns = "2"
```

LegendFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Legende.

Beispiele

```
LegendFont = "'Arial'"
```

```
LegendFont = "'Arial, bold, 12'"
```

LegendImage

Typ

String

Seit

1.2

Beschreibung

Bestimmt eine Grafik, die vor den Legenden-Beschriftungen dargestellt werden soll (anstelle der einfachen Legenden-Kästchen).

Beispiele

```
LegendImage = "'/iolap/images/legend.gif'"
```

LegendOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Darstellung der Legende ein bzw. aus.

Beispiele

```
LegendOn = "true"
```

```
LegendOn = "false"
```

LegendPosition

Typ

String ('right', 'left', 'top' oder 'bottom')

Seit

1.2

Beschreibung

Bestimmt die Position der Legende. Mögliche Ergebnisse der Formel sind 'right' (rechts), 'left' (links), 'top' (oben) oder 'bottom' (unten).

Beispiele

```
LegendPosition = "'right'"
```

```
LegendPosition = "'left'"
```

```
LegendPosition = "'top'"
```

```
LegendPosition = "'bottom'"
```

LegendReverseOn

Typ

Boolean

Seit

2.1

Beschreibung

Schaltet die Rückwärtsdarstellung der Legende ein bzw. aus. Normalerweise wird die Legende von links nach rechts bzw. von oben nach unten dargestellt.

Beispiele

```
LegendReverseOn = "true"
```

```
LegendReverseOn = "false"
```

LowerRange

Typ

Number

Seit

1.2

Beschreibung

Setzt die untere Grenze der Werte-Skala (Y-Achse). Wenn keine Grenze gesetzt ist, wird diese automatisch an den kleinsten Wert aus dem Diagramm angepasst (oder auf 0 gesetzt, wenn kein negativer Wert im Diagramm vorkommt).

Beispiele

```
LowerRange = "40"
```

MaxValueLineCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die maximale Anzahl an vertikalen Gitternetzlinien für dieses Diagramm. Sie können (bis zu einem gewissen Grad) diese Eigenschaft dazu verwenden, um den Absatz zwischen den Linien zu bestimmen.

Beispiele

```
MaxValueLineCount = "5"
```

MultiColorOn

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft true ergibt, erhält jeder Balken eine eigene Farbe. Verwenden Sie die Eigenschaft SampleColors, um die Farbe der Balken zu bestimmen.

Beispiele

```
MultiColorOn = "true"
```

```
MultiColorOn = "false"
```

Range

Typ

Number

Seit

1.2

Beschreibung

Setzt die obere Grenze für die Wertskala (Y-Achse). Wenn keine Grenze definiert wird, dann wird diese automatisch an den grössten Wert des Diagramms angepasst.

Beispiele

```
Range = "100"
```

RangeAdjusted

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt, welche Werte-Skalen durch welchen Regler verändert werden. Normalerweise kontrolliert der Regler 1 die Skala 1 usw.

Beispiele

```
RangeAdjusted = "1"
```

```
RangeAdjusted = "2"
```

RangeAdjusterOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet den Regler für die Werte-Skala ein. Der Regler ermöglicht es dem Benutzer, die obere und untere Grenze der Skala mit der Maus einzustellen. Normalerweise wird der Regler an der rechten Seite des Diagramms dargestellt. Das kann mit der Eigenschaft `RangeAdjusterPosition` geändert werden.

Beispiele

```
RangeAdjusterOn = "true"
```

```
RangeAdjusterOn = "false"
```

RangeAdjusterPosition

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Position des Reglers für die Werte-Skala. 0 steht für links, 1 für rechts. Normalerweise wird der Regler rechts dargestellt.

Beispiele

```
RangeAdjusterPosition = "1"
```

RangeAxisLabel

Typ

String

Seit

1.2

Beschreibung

Setzt eine Beschriftung neben der Werte-Skala (Y-Achse). Normalerweise wird diese Beschriftung horizontal dargestellt, Sie können den Drehwinkel der Beschriftung jedoch mit der Eigenschaft `RangeAxisLabelAngle` ändern.

Beispiele

```
RangeAxisLabel = "'Amount'"
```

RangeAxisLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Setzt den Drehwinkel der Beschriftung (im Uhrzeigersinn) für die Werte-Skala (Y-Achse) in Grad.

Beispiele

```
RangeAxisLabelAngle = "270"
```

RangeAxisLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die optionale Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeAxisLabelFont = "'Arial'"
```

RangeColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe der Werte-Skala (Y-Achse) und Skalen-Markierungen. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
RangeColor = "'green'"
```

```
RangeColor = "'#FFFF00'"
```

RangeDecimalCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl der Nachkommastellen für die Darstellung der Werte auf der Werte-Skala (Y-Achse).

Beispiele

```
RangeDecimalCount = "10"
```

RangeLabelFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Beschriftungen der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelFont = "'Arial'"
```

RangeLabelPostfix

Typ

String

Seit

1.2

Beschreibung

Definiert einen Postfix (einen Text hinter den einzelnen Werten) für die Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelPostfix = "'ms'"
```

RangeLabelPrefix

Typ

String

Seit

1.2

Beschreibung

Definiert einen Prefix (einen Text vor den einzelnen Werten) für die Beschriftung der Werte-Skala (Y-Achse).

Beispiele

```
RangeLabelPrefix = "'$'"
```

RangeLabelsOff

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Beschriftung der Werte-Skala (Y-Achse) ab, wenn die Formel true ergibt.

Beispiele

```
RangeLabelsOff = "true"
```

RangeOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Werte-Skala (Y-Achse) ein bzw. aus.

Beispiele

```
RangeOn = "true"
```

RangePosition

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Position der Werte-Skala (Y-Achse). 0 steht für links, 1 für rechts. Die Standard-Position der ersten Skala ist links, die der zweiten rechts.

Beispiele

```
RangePosition = "2"
```

RangeStep

Typ

Number

Seit

1.2

Beschreibung

Wenn Sie die obere Grenze der Werte-Skala (den Range) nicht selber bestimmen, wird diese automatisch auf den grössten Wert oder 0 gesetzt. Wenn der "RangeStep" mit dieser Eigenschaft gesetzt wird, dann wird die obere Grenze auf den nächsten durch diesen Wert teilbaren Wert gesetzt.

Wenn der grösste Wert im Diagramm z.B. 325 und ist und der RangeStep auf 100 gesetzt wird, dann wird die obere Grenze automatisch auf 400 gesetzt.

Beispiele

```
RangeStep = "100"
```

SampleAxisLabel

Typ

String

Seit

1.2

Beschreibung

Setzt eine Beschriftung unterhalb der X-Achse.

Beispiele

```
SampleAxisLabel = "'Date'"
```

SampleAxisLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt den Drehwinkel für die Beschriftung der Kategorien (Sample-Achse). Diese Eigenschaft ist u.A. nützlich, wenn Sie die Balken horizontal darstellen und die Kategorien sich dadurch am linken Rand befinden.

Beispiele

```
SampleAxisLabelAngle = "270"
```

SampleAxisLabelFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Kategorien (Sample-Achse) (normalerweise die X-Achse).

Beispiele

```
SampleAxisLabelFont = "'Arial'"
```

SampleAxisRange

Typ

Number

Seit

1.2

Beschreibung

Bestimmt den Darstellungsbereich für die Kategorien (Sample-Achse). Der Bereich wird dazu verwendet, um zu berechnen wo die vertikalen Gitternetzlinien dargestellt werden sollen.

Beispiele

```
SampleAxisRange = "1000"
```

SampleColors

Typ

String

Seit

1.2

Beschreibung

Setzt die Farben der Balken. Normalerweise wird das Diagramm im Einfarb-Modus dargestellt und alle Balken haben die selbe Farbe (die erste Farbe aus dieser Eigenschaft). Wenn Sie die Eigenschaft MultiColorOn auf true setzen erhält jeder Balken eine eigene Farbe. Diese Eigenschaft erwartet eine Liste von Farben die durch Kommata getrennt sind.

Beispiele

```
SampleColors = "'red,green,blue'"
```

SampleDecimalCount**Typ**

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl der Nachkommastellen, die bei der Darstellung der Werte für die einzelnen Balken angezeigt werden.

Beispiele

```
SampleDecimalCount = "2"
```

SampleLabelAngle**Typ**

Integer

Seit

2.1

Beschreibung

Setzt den Drehwinkel (im Uhrzeigersinn) der statischen Kategorie-Beschriftungen in Grad.

Beispiele

```
SampleLabelAngle = "270"
```

SampleLabelColors**Typ**

String

Seit

1.2

Beschreibung

Bestimmt die Farben für die Beschriftung der Kategorien (Sample- bzw. X-Achse). Dies betrifft sowohl die Texte für die einzelnen Balken als auch die in der Legende. Die Formel muss eine Liste von Farben als Zeichenkette zurückgeben, die durch Kommata getrennt sind.

Beispiele

```
SampleLabelColors = "'red,green,blue'"
```

SampleLabelFont

Typ

String

Seit

2.1

Beschreibung

Setzt den Zeichensatz für die statischen Kategorie-Beschriftungen.

Beispiele

```
SampleLabelFont = "'Arial'"
```

```
SampleLabelFont = "'Dialog, plain, 12'"
```

SampleLabelSelectionColor

Typ

String

Seit

1.2

Beschreibung

Bestimmen Sie mit dieser Eigenschaft die Farbe für von Kategorien (Balken) für den Fall, dass sie vom Benutzer angewählt wurden.

Beispiele

```
SampleLabelSelectionColor = "'red'"
```

```
SampleLabelSelectionColor = "'#FFFF00'"
```

SampleLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft true ergibt, werden die Bezeichnungen der Balken innerhalb oder ausserhalb jedes Balkens dargestellt. Die Darstellungsposition kann mit der Eigenschaft SampleLabelStyle bestimmt werden.

Beispiele

```
SampleLabelsOn = "true"
```

```
SampleLabelsOn = "false"
```

Siehe auch

SampleLabelStyle

SampleLabelStyle

Typ

String ('inside' oder 'outside')

Seit

1.2

Beschreibung

Bestimmt, ob die Beschriftungen der Balken innerhalb oder ausserhalb der Balken positioniert werden. Der Standardwert ist "outside".

Beispiele

```
SampleLabelStyle = "'inside'"
```

```
SampleLabelStyle = "'outside'"
```

SampleScrollerOn

Typ

Boolean

Seit

1.2

Beschreibung

Der Benutzer kann (mit der X-Achse) die Balken vergrössern und den darzustellenden Ausschnitt wählen, wenn Sie den "Sample Scroller" mit dieser Eigenschaft einschalten.

Beispiele

```
SampleScrollerOn = "true"
```

```
SampleScrollerOn = "false"
```

SeriesLabelColors

Typ

String

Seit

1.2

Beschreibung

Setzt die Farben der Serienbezeichnungen in der Legende. Diese Eigenschaft erwartet eine Formel, die die Farben (mit Kommata getrennt) in einer Liste zurückgibt.

Beispiele

```
SeriesLabelColors = "'red, green, blue'"
```

SeriesLabelFont

Typ

String

Seit

2.1

Beschreibung

Setzt den Zeichensatz für die Serien-Beschriftungen.

Beispiele

```
SeriesLabelFont = "'Arial'"
```

```
SeriesLabelFont = "'Dialog, plain, 12'"
```

SeriesLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Stellt die Bezeichnungen der Serien innerhalb oder ausserhalb jedes Balkens dar. Normalerweise werden die Bezeichnungen ausserhalb dargestellt, das kann mit der Eigenschaft SeriesLabelStyle geändert werden.

Beispiele

```
SeriesLabelsOn = "true"
```

```
SeriesLabelsOn = "false"
```

SeriesLabelStyle

Typ

String ('inside' oder 'outside')

Seit

1.2

Beschreibung

Bestimmt, ob die Bezeichner der Balken-Serien innerhalb oder ausserhalb der Balken dargestellt werden. Die Standardeinstellung ist "outside". Mit Hilfe der Eigenschaft "SeriesLabelOn" können Sie die Darstellung der Bezeichner aktivieren. In gestapelten Balkendiagrammen können die Bezeichner nur ausserhalb der Balken dargestellt werden.

Beispiele

```
SeriesLabelStyle = "'inside'"
```

```
SeriesLabelStyle = "'outside'"
```

SingleClickURLOn

Typ

Boolean

Seit

1.2

Beschreibung

Setzen Sie diese Eigenschaft auf true, damit Links (z.B. in Drilldown-fähigen Diagrammen) mit einem einzelnen Klick statt mit einem Doppelklick angewählt werden können.

Beispiele

```
SingleClickURLOn = "true"
```

```
SingleClickURLOn = "false"
```

TargetValueLine

Typ

String

Seit

1.2

Beschreibung

Sie können einzelne Gitternetzlinien mit einer spezifischen Farbe und einem frei definierbaren Text zum Diagramm hinzufügen. Der erste Teil des Parameters ist der Text, der zweite der Wert (an dem die Linie auf der Werte-Achse dargestellt werden soll) und der dritte die Farbe der Linie. Es gibt ausserdem einen optionalen vierten Teil, der bestimmt ob nur der Text oder auch der Wert dargestellt werden soll (das ist der Normalfall). Alle Teile müssen (durch Kommata getrennt) in einer Zeichenkette angegeben werden.

Beispiele

```
TargetValueLine = "break even, 150, green"
```

ThousandsDelimiter

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um das Tausender-Trennzeichen für alle Werte-Darstellungen zu definieren.

Beispiele

```
ThousandsDelimiter = "'.'"'
```

TitleFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz der Diagramm-Überschrift.

Beispiele

```
TitleFont = "'Arial'"
```

UrlTarget

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt, in welchem Browser-Fenster neue URLs beim Anklicken von Links geöffnet werden:

- `_self`: Öffnet das Ziel im selben Fenster.
- `_blank`: Öffnet das Ziel in einem neuen, leeren Fenster.
- `name`: Öffnet das Ziel in einem Fenster mit bestimmten Namen.

Beispiele

```
UrlTarget = "'_self'"
```

```
UrlTarget = "'window1'"
```

ValueLabelAngle

Typ

Integer

Seit

1.2

Beschreibung

Setzt den Drehwinkel (im Uhrzeigersinn) für die Werte-Beschriftungen der Balken.

Beispiele

```
ValueLabelAngle = "270"
```

ValueLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Werte-Beschriftungen der Balken.

Beispiele

```
ValueLabelFont = "'Arial'"
```

ValueLabelPostfix

Typ

String

Seit

1.2

Beschreibung

Bestimmt einen Text (Postfix), der hinter den Werte-Beschriftungen der Balken angefügt wird.

Beispiele

```
ValueLabelPostfix = "'ms'"
```

ValueLabelPrefix

Typ

String

Seit

1.2

Beschreibung

Bestimmt einen Text (Prefix), der vor den Werte-Beschriftungen der Balken dargestellt wird.

Beispiele

```
ValueLabelPrefix = "'$'"
```

ValueLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Stellt die Werte für die einzelnen Balken dar. Die Werte können ausserhalb der Balken, innerhalb der Balken oder aus "Tool Tip" dargestellt werden, der beim Überfahren mit der Maus erscheint. Die Art der Darstellung können Sie mit der Eigenschaft ValueLabelStyle bestimmen.

Beispiele

```
ValueLabelsOn = "true"
```

```
ValueLabelsOn = "false"
```

ValueLabelStyle

Typ

String ('inside', 'outside' oder 'floating')

Seit

1.2

Beschreibung

Bestimmt die Art der Darstellung der Wert für die einzelnen Balken. Die Werte können innerhalb (inside) der Balken, ausserhalb (outside) der Balken oder als "Tool Tip" (der beim Überfahren mit der Maus erscheint) dargestellt werden (floating). Die Normaldarstellung ist "ausserhalb" (outside).

Beispiele

```
ValueLabelStyle = "'inside'"
```

```
ValueLabelStyle = "'outside'"
```

```
ValueLabelStyle = "'floating'"
```

ValueLinesColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe der vertikalen Gitternetzlinien. Die Formel muss eine gültige Farbe als Zeichenkette zurückgeben.

Beispiele

```
ValueLinesColor = "'red'"
```

```
ValueLinesColor = "'#FFFF00'"
```

ValueLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die horizontalen Gitternetzlinien ein. Wenn Sie die Balken horizontal darstellen, schaltet diese Eigenschaft stattdessen die vertikalen Gitterlinien ein.

Beispiele

```
ValueLinesOn = "true"
```

```
ValueLinesOn = "false"
```

ValueLinesOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die horizontalen Gitternetzlinien ein. Wenn Sie die Balken horizontal darstellen, schaltet diese Eigenschaft stattdessen die vertikalen Gitterlinien ein.

Beispiele

```
ValueLinesOn = "true"
```

```
ValueLinesOn = "false"
```

ZoomOn

Typ

Boolean

Seit

2.1

Beschreibung

Wenn die Eigenschaft "ZoomOn" true ergibt, kann der Benutzer mit der Maus Ausschnitte des Diagramms vergrößern.

Beispiele

```
ZoomOn = "true"
```

```
ZoomOn = "false"
```

Tortendiagramm-Eigenschaften

3DModeOn

Typ

Integer

Seit

1.2

Beschreibung

Schaltet den 3D-Modus an oder aus.

Beispiele

```
3DModeOn = "true"
```

```
3DModeOn = "false"
```

Angle

Typ

Double

Seit

1.2

Beschreibung

Bestimmt die Neigung einer Torte in der 3D Darstellung. Der Wert muss zwischen 10 und 90 liegen, der Standardwert ist 30.

Beispiel

```
Angle = "10"
```

Background

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft setzt die Hintergrundfarbe ausserhalb des eigentlichen Diagramm-Bereichs. Die Formel muss ein gültiges Farb-Format als Ergebnis haben.

Beispiele

```
Background = "'green'"
```

```
Background = "'#FFFF00'"
```

Siehe auch

Foreground (auf Seite 191)

ChartTitle

Typ

String

Seit

1.2

Beschreibung

Das Ergebnis der Formel in dieser Eigenschaft bestimmt den Titel des Diagramms.

Beispiele

```
ChartTitle = "'Turnover'"
```

Depth

Typ

Double

Seit

2.1

Beschreibung

Setzt die Tiefe einer Torte in der 3D-Darstellung. Die Tiefe muss relativ zur Breite der Torte angegeben werden und zwischen 0.0 und 1.0 liegen. Die Standard-Tiefe ist 0.4.

Beispiele`Depth = "0.1"``Depth = "0.6"`

DetachedDistance

Typ

Double

Seit

1.2

Beschreibung

Bestimmt den Abstand eines abgetrennten Tortenstückes vom Mittelpunkt der Torte. Der Abstand wird als relativer Faktor zum Radius der Torte angegeben, so dass z.B. ein Wert von 0.5 das Tortenstück um die Hälfte der Tortenbreite verschiebt.

Beispiele`DetachedDistance = "0.75"`

DetachedSlices

Typ

String

Seit

1.2

Beschreibung

Trennt einzelne Tortenstücke von der Torte ab. Die Nummern der Stücke werden als (Komma-separierte) Liste in einer Zeichenkette erwartet.

Beispiele

```
DetachedSlices = "'1,3,5'"
```

FloatingLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Werte- und Kategorien-Beschriftungen, die beim Überfahren mit der Maus erscheinen.

Beispiele

```
FloatingLabelFont = "'Arial'"
```

Font

Typ

String

Seit

1.2

Beschreibung

Setzt den Standard-Zeichensatz für die Diagramm-Beschriftungen.

Beispiele

```
Font = "'Arial'"
```

Foreground

Typ

String

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Farbe des Titels und der Legenden-Beschriftungen. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
Foreground = "'blue'"
```

```
Foreground = "'#ffff00'"
```

GraphInsets

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um den Abstand zwischen dem eigentlichen Diagramm und den äußeren Rahmen zu bestimmen. Der Abstand wird bestimmt mit 4 verschiedenen Parametern in der folgenden Reihenfolge bestehen: Oben, links, unten und rechts. Ein Wert von -1 steht für den jeweiligen Standard-Abstand. Das Ergebnis muss eine Zeichenkette sein, die diese 4 Wert (mit Kommata getrennt) enthält.

Beispiele

```
GraphInsets = "'-1, 50, -1, -1'"
```

InsideLabelColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe für alle inneren Beschriftungen der Torten. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
InsideLabelColor = "'green'"
```

InsideLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Beschriftung der Kategorien, Werte und Prozentanzeigen innerhalb der Tortenstücke.

Beispiele

```
InsideLabelFont = "'Arial'"
```

Label_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine Beschriftung an beliebiger Stelle des Diagramms darzustellen. Der erste Teil der Parameter ist der eigentliche Text, der zweite und dritte sind jeweils die X- und Y-Position. Wenn X oder Y zwischen 0 und 1 liegen, dann bestimmen sie die Position der Beschriftung relativ zur Diagrammgrösse. Ansonsten werden die Angaben als absolute Angaben (in Pixeln) interpretiert. Ausserdem gibt es optionale 4. und 5. Parameter, die die Nummer und Serie einer bestehenden Beschriftung bestimmen, auf die Sie zeigen möchten.

Beispiele

```
Label = "'orange sales,100,100'"
```

```
Label = "'apple sales,0.4,0.5'"
```

```
Label = "'banana sales,200,200,4,0'"
```

Siehe auch

LabelUrl_0 (auf Seite 193), LabelUrlTarget_0 (auf Seite 194)

LabelUrl_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kann dazu verwendet werden, eine URL an eine Beschriftung zu hängen. Die URL wird dann geöffnet, wenn der Benutzer auf die Beschriftung klickt.

Beispiele

```
LabelURL_0 = "'details.html'"
```

Siehe auch

Label_0 (auf Seite 192), LabelUrlTarget_0 (auf Seite 194)

LabelUrlTarget_0

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft kontrolliert, auf welcher HTML-Seite eine URL geöffnet wird, wenn der Benutzer auf eine Beschriftung klickt.

Beispiele

```
LabelUrlTarget_0 = "'_blank'"
```

Siehe auch

Label_0 (auf Seite 192), LabelUrl_0 (auf Seite 193)

LegendColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der Kästchen für die Legende. Wenn keine Farben definiert werden, dann werden die der SampleColors Eigenschaft verwendet. Die Formel muss eine Zeichenkette mit der Liste der Farben (durch Kommata getrennt) zurückgeben.

Beispiele

```
LegendColors = "'red,green,blue'"
```

LegendColumns

Typ

Integer

Seit

2.1

Beschreibung

Bestimmt die Anzahl der Spalten, die zur Darstellung der Legende verwendet werden sollen.

Beispiele

```
LegendColumns = "2"
```

LegendFont

Typ

String

Seit

1.2

Beschreibung

Setzt den Zeichensatz für die Legende.

Beispiele

```
LegendFont = "'Arial'"
```

```
LegendFont = "'Arial, bold, 12'"
```

LegendImage

Typ

String

Seit

1.2

Beschreibung

Bestimmt eine Grafik, die vor den Legenden-Beschriftungen dargestellt werden soll (anstelle der einfachen Legenden-Kästchen).

Beispiele

```
LegendImage = "/iolap/images/legend.gif"
```

LegendOn

Typ

Boolean

Seit

1.2

Beschreibung

Schaltet die Darstellung der Legende ein bzw. aus.

Beispiele

```
LegendOn = "true"
```

```
LegendOn = "false"
```

LegendPosition

Typ

String ('right', 'left', 'top' oder 'bottom')

Seit

1.2

Beschreibung

Bestimmt die Position der Legende. Mögliche Ergebnisse der Formel sind 'right' (rechts), 'left' (links), 'top' (oben) oder 'bottom' (unten).

Beispiele

```
LegendPosition = "right"
```

```
LegendPosition = "left"
```

```
LegendPosition = "'top'"
```

```
LegendPosition = "'bottom'"
```

LegendReverseOn

Typ

Boolean

Seit

2.1

Beschreibung

Schaltet die Rückwärtsdarstellung der Legende ein bzw. aus. Normalerweise wird die Legende von links nach rechts bzw. von oben nach unten dargestellt.

Beispiele

```
LegendReverseOn = "true"
```

```
LegendReverseOn = "false"
```

OutsideLabelColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe für alle äußeren Beschriftungen der Torten. Die Formel muss ein gültiges Farbformat als Ergebnis haben.

Beispiele

```
InsideLabelColor = "'green'"
```

OutsideLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für die Beschriftung der Kategorien, Werte und Prozentanzeigen ausserhalb der Tortenstücke.

Beispiele

```
OutsideLabelFont = "'Arial'"
```

PercentDecimalCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl Ziffern für Prozentdarstellungen.

Beispiele

```
PercentDecimalCount = "2"
```

PercentLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Durch das Aktivieren der Prozent-Beschriftungen werden die Prozentwerte für die einzelnen Tortenstücke sichtbar, wenn der Benutzer mit der Maus über das Tortenstück fährt. Alternativ können die Prozentwerte durch Setzen des PercentLabelStyle auch statisch dargestellt werden.

Beispiele

```
PercentLabelsOn = "true"
```

```
PercentLabelsOn = "false"
```

PercentLabelStyle

Typ

String ('floating' oder 'inside')

Seit

1.2

Beschreibung

Die Prozentwerte für Tortenstücke können entweder als "Tool Tip" ('floating') oder statisch ('inside') dargestellt werden.

Beispiele

```
PercentLabelStyle = "'floating'"
```

```
PercentLabelStyle = "'inside'"
```

PieLabelFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz für alle Tortenbeschriftungen. Die Tortenbeschriftungen werden unterhalb der einzelnen Torten dargestellt, wenn mehr als eine Torte dargestellt wird und die Eigenschaft PieLabelsOn auf true gesetzt wurde.

Beispiele

```
PieLabelFont = "'Arial'"
```

PieLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Die Tortenbeschriftungen werden unter jeder einzelnen Torte dargestellt wenn mehrere Datenserien / Torten dargestellt werden.

Beispiele

```
PieLabelsOn = "true"
```

```
PieLabelsOn = "false"
```

PieRotationOn

Typ

Boolean

Seit

2.1

Beschreibung

Bestimmt ob der Benutzer die Torte mit der Maus drehen kann. Wenn die Formel in dieser Eigenschaft true ergibt, kann der Benutzer die Torte mit der Maus festhalten und drehen.

Beispiele

```
PieRotationOn = "true"
```

PointingLabelColor**Typ**

String

Seit

2.1

Beschreibung

Bestimmt die Farben für die äußeren Markierungen aller Torten.

Beispiele

```
PointingLabelColor = "'red'"
```

```
PointingLabelColor = "'#FFFF00'"
```

PointingLabelFont**Typ**

String

Seit

2.1

Beschreibung

Bestimmt den Zeichensatz für alle äußeren Markierungen der Tortenstücke.

Beispiele

```
PointLabelFont = "'Arial'"
```

```
PointLabelFont = "'Verdana, bold, 12'"
```

SampleColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der einzelnen Tortenstücke. Die Eigenschaft muss eine Liste von Farben in einer Zeichenkette und durch Kommata getrennt zurückgeben.

Beispiele

```
SampleColors = "'red,green,blue'"
```

SampleDecimalCount

Typ

Integer

Seit

1.2

Beschreibung

Bestimmt die Anzahl der Nachkommastellen für Werte-Beschriftungen.

Beispiele

```
SampleDecimalCount = "2"
```

SampleLabelColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der Beschriftungen innerhalb der Legende. Die Formel muss eine Liste von Formel (durch Kommata separiert) als Zeichenkette zurückgeben.

Beispiele

```
SampleLabelColors = "'red,green,blue'"
```

SampleLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Durch das Einschalten der Kategorie-Beschriftungen (Sample Labels) mit dieser Eigenschaft wird diese beim Überfahren eines Tortenstücks mit der Maus sichtbar. Sie können die Beschriftungen ausserdem auch statisch darstellen, in dem Sie die Eigenschaft SampleLabelStyle entsprechend einstellen.

Beispiele

```
SampleLabelsOn = "true"
```

```
SampleLabelsOn = "false"
```

SampleLabelStyle

Typ

String ('inside' oder 'floating')

Seit

1.2

Beschreibung

Die Kategorie-Beschriftungen für die Tortenstücke können entweder als "Tool Tip" ('floating') dargestellt werden, der beim Überfahren des Tortenstücks mit der Maus sichtbar wird, oder als statischer Text ('inside') innerhalb der Torte.

Beispiele

```
SampleLabelStyle = "'inside'"
```

```
SampleLabelStyle = "'floating'"
```

SelectionMode

Typ

String ('detached','triangle' oder 'circle')

Seit

1.2

Beschreibung

Bestimmt den Effekt für das Selektieren eines Tortenstücks oder einer Daten-Serie. Entweder können selektierte Stück von der Torte abgetrennt ('detached') werden oder es wird ein Dreieck ('triangle') oder Kreis ('circle') innerhalb der Stücke dargestellt.

Beispiele

```
SelectionMode = "'detached'"
```

```
SelectionMode = "'triangle'"
```

```
SelectionMode = "'circle'"
```

SeriesLabelColors

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farben der Serienbezeichnungen innerhalb der Legende. Diese Eigenschaft erwartet eine Formel, die die Farben (mit Kommata getrennt) in einer Liste zurückgibt.

Beispiele

```
SeriesLabelColors = "'red,green,blue'"
```

SeriesLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Durch das Aktivieren der Serien-Beschriftung mit dieser Eigenschaft wird diese beim Überfahren des Tortenstücks (oder des Legenden-Textes) mit der Maus dargestellt. Die Beschriftung kann ausserdem auch als statischer Text innerhalb des Stücks dargestellt werden, wenn Sie dies entsprechend mit der Eigenschaft SeriesLabelStyle definieren.

Beispiele

```
SeriesLabelsOn = "true"
```

```
SeriesLabelsOn = "false"
```

SingleClickURLOn

Typ

Boolean

Seit

1.2

Beschreibung

Setzen Sie diese Eigenschaft auf true, damit Links (z.B. in Drilldown-fähigen Diagrammen) mit einem einzelnen Klick statt mit einem Doppelklick angewählt werden können.

Beispiele

```
SingleClickURLOn = "true"
```

```
SingleClickURLOn = "false"
```

SliceSeperatorColor

Typ

String

Seit

1.2

Beschreibung

Bestimmt die Farbe der Trennlinie zwischen den Tortenstücken. Wenn Sie keine Farbe bestimmen, erhält die Linie die gleiche Farbe wie das Tortenstück selbst, nur etwas dunkler. Die Formel in dieser Eigenschaft muss eine gültige Farbe als Zeichenkette zurückgeben.

Beispiele

```
SliceSeperatorColor = "'black'"
```

SliceSeperatorOn

Typ

Boolean

Seit

1.2

Beschreibung

Stellt eine Linie zwischen den Tortenstücken dar. Sie können die Farbe der Linie mit der Eigenschaft SliceSeperatorColor bestimmen.

Beispiele

```
SliceSeperatorOn = "true"
```

```
SliceSeperatorOn = "false"
```

ThousandsDelimiter

Typ

String

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um das Tausender-Trennzeichen für alle Werte-Darstellungen zu definieren.

Beispiele

```
ThousandsDelimiter = "','"
```

TitleFont

Typ

String

Seit

1.2

Beschreibung

Bestimmt den Zeichensatz der Diagramm-Überschrift.

Beispiele

```
TitleFont = "'Arial'"
```

UrlTarget

Typ

String

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt, in welchem Browser-Fenster neue URLs beim Anklicken von Links geöffnet werden:

- `_self`: Öffnet das Ziel im selben Fenster.
- `_blank`: Öffnet das Ziel in einem neuen, leeren Fenster.
- `name`: Öffnet das Ziel in einem Fenster mit bestimmten Namen.

Beispiele

```
UrlTaget = "'_self'"
```

```
UrlTaget = "'window1'"
```

ValueLabelPrefix

Typ

String

Seit

1.2

Beschreibung

Bestimmt einen Text (Prefix), der vor den Werte-Beschriftungen dargestellt wird.

Beispiele

```
ValueLabelPrefix = "'$'"
```

ValueLabelsOn

Typ

Boolean

Seit

1.2

Beschreibung

Durch das Aktivieren der Value-Label wird der Wert für jedes Tortenstück dargestellt, wenn der Benutzer mit der Maus über das Tortenstück oder dem entsprechenden Eintrag in der Legende fährt. Die Werte können alternativ durch das Anpassen der Eigenschaft "ValueLabelStyle" auch als feststehende Texte dargestellt werden.

Beispiele

```
ValueLabelsOn = "true"
```

```
ValueLabelsOn = "false"
```

ValueLabelStyle

Typ

String ('inside' oder 'floating')

Seit

1.2

Beschreibung

Die Werte können als statische innerhalb der Tortenstücke oder als dynamische Labels, die beim Überfahren mit der Maus sichtbar werden, dargestellt werden. Für statische Labels müssen Sie diese Eigenschaft auf "inside", für dynamische auf "floating" setzen.

Beispiele

```
ValueLabelStyle = "'inside'"
```

```
ValueLabelStyle = "'floating'"
```


Konfigurations-Eigenschaften

In diesem Kapitel

Datenbank	212
Tabelle	230
Spalte	231
Alias.....	233
Link.....	235
Link-Expression	237
CSV-Source.....	239
CSV-Column.....	243
Dimension.....	245
Key	248
Key-Attribute	251
SQL-Keyloader	254
SQL-Attribute	265
CSV-Keyloader	272
CSV-Attribute	277
Time-Keyloader	281
Time-Attribute.....	289
Number-Keyloader	293
SQL-Cube	295
SQL-Fact.....	302
SQL-Dimension	304
CSV-Cube	309
CSV-Fact	313
CSV-Dimension.....	315
Formel.....	318
Memory-Cache	321
File-Cache.....	324
Include.....	327

Datenbank

Catalog

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Die Eigenschaft "Catalog" bestimmt, auf welchen Datenbank-Katalog die Verbindung zugreifen wird. Wenn kein Katalog angegeben wird, dann wird der Standard-Katalog des verbindenden Benutzers verwendet. Nicht alle Datenbanken unterstützen die Unterteilung in Kataloge (bzw. besitzen nur einen einzigen Katalog). In diesem Fall wird diese Eigenschaft ignoriert.

Beispiele

`Catalog = "Northwind"`

Charset

Typ

Konstante Zeichenkette

Seit

2.0.3

Beschreibung

Diese Eigenschaft bestimmt den Zeichensatz für eingehende (Fehler-) Meldungen von der Datenbank. Wenn ein Zeichensatz definiert wird, dann konvertiert instantOLAP alle Meldungen anhand dieses Zeichensatzes.

Normalerweise benötigen Sie diese Eigenschaft nicht, jedoch senden einige Datenbanken (z.B. SAS-Datenbanken auf Host-Systemen) ihre Meldungen nicht immer in ASCII. In diesem Fall können mit dieser Eigenschaft die Meldungen in ein lesbare Format umgesetzt.

Beispiele

```
Charset = "ISO-8859-1"
```

Column-end bracket

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Begrenzer (nach dem Spaltennamen) für den SQL-Generator, mit dem er Spaltennamen in den generierten Statements einschliesst. Zusammen mit der Eigenschaft Column-start bracket können Sie so bestimmen, wie Spaltennamen (die z.B. Sonderzeichen oder Leerzeichen enthalten) an die Datenbank übergeben werden sollen, ohne eine Fehlermeldung von der Datenbank zu erhalten.

Für die gängigen Datenbanken sind die Begrenzer bereits vordefiniert und müssen nicht von Ihnen angegeben werden.

Beispiele

```
Column-end bracket = "]"
```

Column-startbracket

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Begrenzer (vor dem Spaltennamen) für den SQL-Generator, mit dem er Spaltennamen in den generierten Statements einschliesst. Zusammen mit der Eigenschaft Column-end bracket können Sie so bestimmen, wie Spaltennamen (die z.B. Sonderzeichen oder Leerzeichen enthalten) an die Datenbank übergeben werden sollen, ohne eine Fehlermeldung von der Datenbank zu erhalten.

Für die gängigen Datenbanken sind die Begrenzer bereits vordefiniert und müssen nicht von Ihnen angegeben werden.

Beispiele

Column-start bracket = "["

Concat-Operator

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

Diese Eigenschaft bestimmt die Zeichenfolge für den Concat-Operator, die vom SQL-Generator für diese Datenbank verwendet wird. In ANSI/92 entspricht der Operator normalerweise der Zeichenfolge "||", jedoch erwarten einige Datenbanken (z.B. MS Access) abweichen davon einen anderen Operator (z.B. "+").

Beispiele

Concat-Operator = "+"

Datasource

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Die Eigenschaft "Datasource" ermöglicht es Ihnen, eine Datenbank über eine bereits im Application-Server vordefinierte J2EE-Datasource anzubinden. Wenn der Administrator des Application-Server eine solche Datasource für Sie bereitgestellt hat, dann müssen Sie keine URL, JDBC-Treiber, Benutzernamen und Passwort für die Verbindung mehr angeben. Das einzige, was Sie benötigen, ist der JNDI-Name der Datasource, den Sie in dieser Eigenschaft angeben müssen.

Beispiele

```
Data source = "java:comp/env/jdbc/salesDS"
```

DropIN with NULL

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

2.0

Beschreibung

instantOLAP generiert SQL-Statements mit IN-Listen zum Filtern von Schlüsseln aus Dimensionen. Wenn ein Schlüssel auf den Wert "NULL" gemappt wurde, dann wird diese NULL auch in der IN-Liste der generierten Statements vorkommen (z.B. ... WHERE ProductId IN (1, 2, NULL)). Einige wenige Datenbanken unterstützen jedoch keine NULL-Werte in IN-Listen (z.B. MS Access) und generieren eine Fehlermeldung in solchen Fällen. In diesem Fall muss diese Eigenschaft auf NULL gesetzt werden und die NULL wird bei der Generierung der Liste weggelassen.

Diese Eigenschaft ist für die meisten Datenbanksysteme bereits vordefiniert.

Beispiele

```
Drop IN with NULL = "false":
```

```
SELECT Amount, ProductID FROM SALES WHERE ProductID IN (
NULL, '1', '2' )
```

```
Drop IN with NULL = "true":
```

```
SELECT Amount, ProductID FROM SALES WHERE ProductID IN (
'1', '2' )
```

EscapeCharacter

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt das "Escape-Zeichen", das in generierten SQL-Statements vor möglichen Sonderzeichen gesetzt wird. Wenn Sie kein Escape Character definieren, dann werden Sonderzeichen 1:1 in die Statements übernommen und können ggf. zu Fehlermeldungen von den Datenbanken führen.

Diese Eigenschaft ist für die meisten Datenbanksysteme bereits vordefiniert.

Beispiele

```
Escape character = "":
```

```
Select ... WHERE ProductName IN ( 'Hühnereier' )
```

```
Escape character = "\":
```

```
Select ... WHERE ProductName IN ( 'H\ühnereier' )
```

GroupBy Index

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.1

Beschreibung

Einige Datenbanken erlauben in den GROUP BY Clauses der SELECT-Statements nur die Angabe der Spaltennummer (anstelle des Ausdrucks oder eines Spaltennamens bzw. Aliases), nachdem gruppiert werden soll. Wenn Sie diese Eigenschaft auf "true" setzen, dann generiert instantOLAP die GROUP BY Clause entsprechend mit solchen Nummern.

Diese Eigenschaft ist für die meisten Datenbanksysteme bereits vordefiniert.

Beispiele

```
Group By Index = "false":
```

```
SELECT SUM( SALES.AMOUNT ), SALES.CUSTOMER_ID, SUBSTR(  
SALES.DATE, 1, 4 ) FROM SALES GROUP BY CUSTOMER_ID, SUBSTR(  
SALES.DATE, 1, 4 )
```

```
Group By Index = "true":
```

```
SELECT SUM( SALES.AMOUNT ), SALES.CUSTOMER_ID, SUBSTR(
SALES.DATE, 1, 4 ) FROM SALES GROUP BY 2, 3
```

JDBC-Driver

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft definiert den JDBC-Treiber (bzw. den genauen Pfad der Treiber-Klasse), der für eine Verbindung zu Datenbank verwendet wird.

Für jeden Datenbank-Typ benötigen Sie einen anderen JDBC-Treiber, den Sie vor seiner Benutzung im Application-Server installieren müssen. Um herauszufinden, welchen Treiber Sie für Ihre Datenbank benötigen, sollten Sie die Dokumentation Ihrer Datenbank lesen oder auf der instantOLAP-Webseite nachschauen.

Für die bekanntesten Datenbanken können Sie sich die Treiber-Klassen in der instantOLAP-Workbench vorschlagen lassen, Sie müssen den Treiber dennoch vorher installieren (die meisten Treiber sind Copyright-geschützt und dürfen nicht mit instantOLAP ausgeliefert werden). Der einzige Treiber, den Sie nicht installieren müssen, ist der JDBC-ODBC Connector, der Teil des Java-Systems ist.

Wenn Sie planen, eine Verbindung zur Datenbank über eine, im Application-Server bereits vordefiniert, J2EE-Datasource durchzuführen, dann benötigen Sie diese Eigenschaft nicht.

Beispiele

```
JDBC Driver = "com.sun.jdbc.JdbcOdbcDriver"
```

LimitSyntax

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

Die Eigenschaft definiert die vom SQL-Generator verwendete Syntax für Top-10 Abfragen. Einige (jedoch nicht alle) Datenbanken bieten eine spezielle Syntax, um das Ergebnis einer Abfrage auf die ersten n Zeilen zu beschränken.

Die Limit-Syntax wird als Zeichenkette definiert, in der zwei Platzhalter \$1 und \$2 vorkommen müssen. Der Platzhalter \$1 wird durch das eigentliche SELECT-Statement (ohne das Wort SELECT) ersetzt, der Platzhalter \$2 durch die Anzahl zu limitierender Zeilen.

Wenn Sie keine Limit-Syntax definieren, dann wird instantOLAP ein normales SELECT-Statement generieren und nur die ersten n Zeilen des Ergebnisses einlesen. Dies kann u.U. langsamer sein oder die Datenbank mehr belasten.

Für die gängigsten Datenbanken-Typen ist die Limit-Syntax bereits vordefiniert.

Beispiele

```
Limit Syntax = "SELECT $1 LIMIT $2" (MySQL Limit-Syntax)
```

```
Limit Syntax = "SELECT * FROM ( SELECT $1 ) WHERE ROWNUM <= $2" (Oracle Limit-Syntax)
```

LoadLinks

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.1

Beschreibung

Wenn Sie diese Eigenschaft auf "true" setzen wird instantOLAP versuchen, alle innerhalb der Datenbank definierten Foreign Keys zu laden und als Links für die Datenbank-Konfiguration verwenden. Nicht alle Datenbanken können Foreign Keys speichern.

Beispiele

```
Load Links = "true"
```

```
Load Links = "false"
```

LoadTable-Sizes

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.1

Beschreibung

instantOLAP fragt die Grösse jeder Tabelle (in Datensätzen) ab, wenn diese Eigenschaft auf "true" gesetzt wird und die generierten SELECT-Statements anhand dieser Information optimieren. Der SQL-Generator sortiert dazu die Tabellen in der FROM-Clause der generierten SELECT-Statements von den grossen hin zu den kleinen (einige Datenbank wie z.B. Oracle können dadurch erheblich schneller werden).

Beispiele

```
Load Table-Sizes = "true"
```

```
Load Table-Sizes = "false"
```

Max connectionage

Typ

Konstanter Integer-Wert

Seit

1.2

Beschreibung

Wenn Sie eine Verbindung zu einer Datenbank über den internen Connection-Pool von instantOLAP aufbauen möchten (d.h. keine vordefinierte Data-Source aus dem Application-Server verwenden), dann definiert diese Eigenschaft das maximale Alter ein aufgebauten Verbindung zur Datenbank in Sekunden. Jede Verbindung, die älter ist als durch diese Eigenschaft erlaubt, wird automatisch geschlossen und erneut aufgebaut. Das ist sinnvoll zur Vermeidung ungültiger Verbindungen, die z.B. durch eine Neustart der Datenbank entstehen können.

Beispiele

`Max connection age = "300"`

Baut eine Datenbankverbindung nach max. 5 Minuten erneut auf

Max connectioncount

Typ

Konstanter Integer-Wert

Seit

1.2

Beschreibung

Wenn Sie eine Verbindung zu einer Datenbank über den internen Connection-Pool von instantOLAP aufbauen möchten (d.h. keine vordefinierte Data-Source aus dem Application-Server verwenden), dann definiert diese Eigenschaft die maximale Anzahl gleichzeitiger Verbindungen zur Datenbank, die im Connection-Pool gehalten wird. Alle Benutzer von instantOLAP teilen sich die Verbindungen zu einer Datenbank, d.h. Sie sollten eine ausreichende Menge an Verbindungen bereitstellen. Es ist aber möglich, das mehr Benutzer mit einer Datenbank arbeiten, als Verbindungen offen sind.

Beispiele

`Max connection count = "5"`

Öffnet 5 maximal gleichzeitige Verbindungen zur Datenbank

Max IN-Count

Typ

Konstanter Integer-Wert

Seit

1.2

Beschreibung

Der SQL-Generator von instantOLAP erzeugt IN-Listen in den SELECT-Statements, um Dimensionen zu filtern. Diese Listen können recht lang werden und es gibt je nach Datenbank eine mehr oder wenige grosse Beschränkung in der maximalen Länge. Normalerweise dürfen solche Listen 1000 oder mehr Elemente besitzen, aber einige Datenbank erlauben nur kleinere Listen.

Der Standard-Wert für diese Eigenschaft beträgt 500, durch das Ändern dieser Eigenschaft können Sie entsprechend längere oder kürzere Listen generieren lassen. Wenn ein SELECT-Statement zu lang wird und die Anzahl der Elemente diese Grenze überschneidet, dann wird das Statement automatisch vom SQL-Generator in zwei oder mehr Statments aufgeteilt.

Für die gängigsten Datenbanken-Typen ist diese Eigenschaft bereits vordefiniert.

Beispiele

```
Max In-Count = "100"
```

Name

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den logischen Namen der Datenbank-Verbindung. Alle weiteren Elemente einer Konfiguration (z.B. SQL-Keyloader oder SQL-Cubes) verwenden eine Datenbank-Verbindung anhand ihres logischen Namens, den Sie zwingend definieren müssen.

Beispiele

```
Name = "myDB"
```

Schema

Typ

Konstante Zeichenkette

Seit

1.1

Beschreibung

Die Eigenschaft "Schema" definiert das Datenbank-Schema auf das Sie mit dieser Datenbank-Verbindung zugreifen möchten. Wenn Sie kein Schema angeben, dann wird das Standard-Schema des Datenbank-Benutzers verwendet. Nicht alle Datenbanken unterstützen Schematas, für diese wird eine Schema-Angabe dann ignoriert.

Alle SQL-Ausdrücke in Ihrer Konfiguration, die mit dieser Datenbank arbeiten, werden automatisch auf die Tabellen des hier definierten Schemas zugreifen. Sie können dennoch auf andere Tabellen anderer Schemata zugreifen indem Sie den Schema-Namen, durch einen Punkt '.' getrennt, vor den Tabellennamen setzen.

Beispiele

```
Schema = "Northwind":
```

Der Ausdruck "mytable.mycolumn" greift auf die Tabelle "mytable" des aktuellen Schemas "Northwind" zu

Der Ausdruck "otherschema.othertable.othercolumn" greift auf die Tabelle "othertable" des Schemas "otherschema" zu

Password

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft definiert das Datenbank-Passwort für den Fall, dass über den instantOLAP Connection-Pool eine Verbindung zur Datenbank aufbauen (d.h. keine vordefinierte Datasource aus dem Application-Server verwenden).

Wenn Sie die Datenbank über eine Data-Source ansprechen wollen, dann benötigen Sie diese Eigenschaft nicht.

Beispiele

```
Password = "tiger"
```

Siehe auch

User (auf Seite 228)

SingleIN Operator

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

instantOLAP verwendet den SQL IN-Operator, um Dimensionen in den generierten SELECT-Statements zu filtern. Der IN Operator erwartet eine Liste von Werten als Parameter, z.B. CUSTOMER_ID IN (1, 2, 7). Wenn es nur einen Wert gibt, nach dem gefiltert werden soll, dann können Sie den SQL-Generator dazu zwingen einen anderen Operator zu verwenden, z.B. "=" oder "LIKE".

Wenn diese Eigenschaft nicht gesetzt wird, dann erzeugt der Generator immer IN-Operatoren. Für die gängigsten Datenbank-Typen ist diese Eigenschaft bereits vordefiniert.

Beispiele

Single IN Operator = NULL:

```
SELECT ... WHERE ... CUSTOMER_ID IN ( 1 ) ...
```

Single IN Operator = "=":

```
SELECT ... WHERE ... CUSTOMER_ID = 1 ...
```

Table-end bracket

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Begrenzer (nach dem Tabellennamen) für den SQL-Generator, mit dem er Tabellennamen in den generierten Statements einschliesst. Zusammen mit der Eigenschaft Table-start bracket können Sie so bestimmen, wie Tabellennamen (die z.B. Sonderzeichen oder Leerzeichen enthalten) an die Datenbank übergeben werden sollen, ohne eine Fehlermeldung von der Datenbank zu erhalten.

Für die gängigen Datenbanken sind die Begrenzer bereits vordefiniert und müssen nicht von Ihnen angegeben werden.

Beispiele

```
Table-end bracket = "]"
```

Table-startbracket

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Begrenzer (vor dem Tabellennamen) für den SQL-Generator, mit dem er Tabellennamen in den generierten Statements einschliesst. Zusammen mit der Eigenschaft Table-end bracket können Sie so bestimmen, wie Tabellennamen (die z.B. Sonderzeichen oder Leerzeichen enthalten) an die Datenbank übergeben werden sollen, ohne eine Fehlermeldung von der Datenbank zu erhalten.

Für die gängigen Datenbanken sind die Begrenzer bereits vordefiniert und müssen nicht von Ihnen angegeben werden.

Beispiele

```
Table-start bracket = "["
```

Table Names

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

In der Standard-Einstellung stellt die Workbench alle Tabellen mit ihren Spalten dar, wenn Sie eine Datenbank im Datenbank-Explorer öffnen. Abhängig von der Anzahl der Tabellen in Ihrer Datenbank kann dies Zeit kosten oder sogar zu einer Fehlermeldung führen, wenn die maximale Anzahl darstellbarer Tabellen (250) überschritten wurde.

Anstatt alle Tabellen anzuzeigen können Sie in diesem Fall eine Liste von Tabellennamen (mit Kommata getrennt) in dieser Eigenschaft angeben. Dann wird die Workbench nur noch diese Tabellen anzeigen. Der Standard-Wert "*" für diese Eigenschaft wird alle Tabellen anzeigen.

Beispiele

```
Table Names = "SALES, CUSTOMER, PRODUCT"
```

```
Table Names = "*" 
```

Siehe auch

Table Types (auf Seite 225)

Table Types

Typ

Konstante Zeichenkette

Seite

2.0

Beschreibung

Diese Eigenschaft bestimmt, welche Arten von Tabellen im Database-Explorer der Workbench angezeigt werden sollen. Sie müssen eine, mit Kommata getrennte, List von Typen in dieser Eigenschaft angeben. Die Standard-Typen bzw. der Standard-Wert für diese Eigenschaft lauten "TABLE,VIEW". In einigen Datenbanken existieren weitere Typen, z.B. "SNAPSHOT". Wenn Sie möchten, dass die Workbench auch solche Tabellen im Explorer darstellt, dann müssen Sie diese mit in die Eigenschaft aufnehmen.

Beispiele

```
Table-Types = "TABLE,VIEW,SNAPSHOT"
```

Siehe auch

Table Names (auf Seite 224)

Timeout

Typ

Konstanter Integer-Wert

Seit

1.2

Beschreibung

Mit dieser Eigenschaft bestimmen Sie eine Timeout für alle Datenbank-Abfragen, die über diese Verbindung abgesetzt werden. Wenn diese Eigenschaft leer gelassen wird, dann wird kein Timeout für die Abfragen definiert. Nicht alle Datenbank-Treiber unterstützen einen Timeout, deshalb hat diese Eigenschaft ggf. keine Auswirkung.

Beachten Sie, dass der konkrete Timeout für eine SQL-Abfrage sich aus dem Minimum aus diesem Wert und dem restlichen Timeout der Berichtsausführung definiert.

Beispiele

```
Timeout = "90":
```

Setzt den Timeout auf 1,5 Minuten

URL

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft definiert die URL für den Fall, dass über den instantOLAP Connection-Pool eine Verbindung zur Datenbank aufbauen (d.h. keine vordefinierte Datasource aus dem Application-Server verwenden). Die URL definiert die Verbindung zum Datenbank-Server, d.h. den verwendeten Datenbank-Typ, den Server und ggf. die Datenbank-Instanz sowie optionale Zusatzparameter.

Das Format für eine URL lautet "jdbc:<Treiber-Name>:<Verbindungs-Optionen>". Da jede Datenbank eine unterschiedliche URL im unterschiedlichen Format erwartet und unterschiedliche Optionen erlaubt, sollten Sie die Dokumentation Ihrer Datenbank bzw. des JDBC-Treibers zur Hand nehmen. Beachten Sie, dass eine URL nur im Zusammenhang mit dem dazugehörigen Treiber eingestellt werden kann, den Sie über die Eigenschaft JDBC-Driver setzen müssen.

Für die üblichen Datenbanken existieren bereits voreingestellte Standard-URLs, die in der Workbench beim Anlegen einer Datenquelle als Vorauswahl angezeigt werden. In diesen Standard-URLs müssen Sie dann die Teile in den spitzen Klammern durch Ihre konkreten Einstellungen ersetzen.

Sollten Sie eine vordefinierte Datasource aus dem Application-Server verwenden, dann ist diese Eigenschaft ohne Bedeutung.

Beispiele

```
URL = "jdbc:mysql://myserver/SALES":
```

Erstellt eine Verbindung zur Datenbank "SALES" auf einem MySQL-Server

UseAliases

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Der SQL-Generator von instantOLAP kann SQL-Statements auf zwei verschiedene Arten erzeugt, mit oder ohne die Verwendung von Aliases. Aliase sind logische Namen, die für Spalten oder komplexe Ausdrücke vergeben werden und in der WHERE-, SORT- oder GROUP-BY Clause eines SQL-Statements anstelle des Ausdrucks selbst verwendet werden. Einige Datenbank erlauben keine komplexen Ausdrücke in diesen Elemente, dann müssen Sie Aliase verwenden. Andere hingegen erlauben keine Aliase und werfen bei ihrer Verwendung eine entsprechende Fehlermeldung "Unbekannt Spalte.." - für diese Datenbank müssen Sie ohne die Verwendung von Aliases arbeiten.

Für die üblichen Datenbanken ist diese Eigenschaft bereits voreingestellt.

Beispiele

```
Use aliases = "true":
```

```
SELECT SUM( amount ), SUBSTR( date, 1, 4 ) A1 FROM sales  
GROUP BY A1
```

```
Use aliases = "false":
```

```
SELECT SUM( amount ), SUBSTR( date, 1, 4 ) FROM sales GROUP  
BY SUBSTR( date, 1, 4 )
```

User

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft definiert den Datenbank-Benutzer für den Fall, dass über den instantOLAP Connection-Pool eine Verbindung zur Datenbank aufbauen (d.h. keine vordefinierte Datasource aus dem Application-Server verwenden).

Wenn Sie die Datenbank über eine Data-Source ansprechen wollen, dann benötigen Sie diese Eigenschaft nicht.

Beispiele

```
User = "scott"
```

Siehe auch

Password

Tabelle

Name

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Normalerweise werden alle Tabellen automatisch aus der verwendeten Datenbank ausgelesen und in das Repository von instantOLAP übernommen. In äusserst seltenen Fällen könnten Sie aber gezwungen sein, eine Tabellen-Definition manuell vorzunehmen, z.B. wenn der Datenbank-Treiber nicht in der Lage ist den Datenbank-Katalog auszulesen oder wenn eine Tabelle unbekannte Spaltentypen enthält (Spalten mit unbekanntem Typen werden nicht mit übernommen).

In diesem Fall können Sie eine manuelle Tabellen-Definition vornehmen. Diese Eigenschaft bestimmt den Namen der Tabelle, welcher exakt mit dem Namen in der Datenbank übereinstimmen muss. Manuell definierte Tabellen können genauso verwendet werden wie automatisch eingelesene.

Beispiele

Name = "Sales"

Spalte

Name

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Für manuell definierte Tabellen (d.h. Tabellen die nicht automatisch aus der Datenbank ermittelt wurden) müssen Sie alle Spalten und deren Datentypen angeben. Diese Eigenschaft bestimmt den Namen einer Spalte. Der Name muss mit dem tatsächlichen Namen aus der Tabelle übereinstimmen, ansonsten werden die generierten SQL-Statements zur Laufzeit Fehlermeldungen produzieren.

Beispiele

Name = "Amount"

Type

Typ

Konstante Zeichenkette ('string','integer','double' oder 'date')

Seit

1.0

Beschreibung

Für manuell definierte Tabellen (d.h. Tabellen die nicht automatisch aus der Datenbank ermittelt wurden) müssen Sie alle Spalten und deren Datentypen angeben. Diese Eigenschaft definiert den Datentyp einer Spalte und muss entweder auf 'string', 'integer', 'double' oder 'date' gesetzt werden. In instantOLAP wird nur zwischen wenigen Datentypen für Spalten unterschieden, Datenbanken besitzen in der Regel weitaus mehr Typen. Daher werden die Spalten vom Original Datenbank-Spaltentyp in den instantOLAP-Typ konvertiert, wenn das Ergebnis einer Abfrage ausgelesen wird.

Beispiele

Type = "integer"

Alias

Name

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Namen für einen Alias. Der Alias, der mit diesem Element generiert wird, kann wie eine normale Tabelle (bzw. wie eine Kopie der Originaltabelle) in allen SQL-Statements verwendet werden, die ggf. noch um den Filter (bzw. durch die Where-Eigenschaft) eingeschränkt wurde.

Beispiele

Name = "ActiveProducts"

SQL-Where

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.2

Beschreibung

Wenn Sie einen Alias für Ihre Datenbank definieren, können Sie diesem optional einen Filter mitgeben. Jedesmal, wenn Sie den Alias in einem SQL-Statement verwenden, wird dann der SQL-Generator automatisch diesen Filter mit an das Statement hängen. Dadurch sehen Sie immer nur eine Teilmenge der Originaltabelle für Ihr Alias.

Beachten Sie unbedingt, dass Sie bei der Definition des Where-Filters immer den Namen des Alias und nicht den der Original-Tabelle verwenden! Wenn Sie z.B. einen Alias namens "ActiveProducts" auf Basis der Original-Tabelle "Products" definieren, dann muss der Where-Filter "ActiveProducts" verwenden (z.B. "ActiveProducts.active = 1"). Ansonsten würde durch die Verwendung des Aliases die Original-Tabelle eingeschränkt, was zu einer Fehlermeldung oder sogar falschen Ergebnissen führen würde.

Beispiele

```
SQL-Where = "ActiveProducts.active = 1"
```

Table

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Original-Tabelle, auf den der Alias sich bezieht.

Beispiele

```
Table = "Products"
```

Link

Direction

Typ

Konstante Zeichenkette ('left', 'right' oder 'both')

Seit

2.0

Beschreibung

Mit dieser Eigenschaft können Sie die Richtung eines Links bestimmen. Die Richtung eines Link definiert, von welcher Tabelle aus ein Link für den SQL-Generator sichtbar und verwendbar sein wird.

Wenn Sie z.B. drei Tabellen A, B und C mit den folgenden beiden Links verwenden:

- A --> B (Richtung: Rechts)
- B <-- C (Richtung: Links)

dann wird der SQL-Generator nicht dazu in der Lage sein, ein SQL-Statement zu generieren, bei dem die Kennzahl aus der Tabelle A und die Dimensionen in den Tabellen B und C liegen, da es keinen gültigen Pfad von der Tabelle A nach C gibt (die Links werden immer von den Tabellen mit den Kennzahlen aus gebildet), da der zweite Link die falsche Richtung besitzt. In diesem Fall müssten Sie die Richtung des zweiten Links auf "left" oder "both" setzen.

Seien Sie vorsichtig mit dem Anlegen von Links und bei der Bestimmung der Link-Richtung, ansonsten könnte der SQL-Generator Statements mit zu grossen Ergebnissen erzeugen, weil mehr als eine Kennzahl-Tabelle in das Statement mit aufgenommen wurde. Versuchen Sie, nur Links mit den Richtungen "left" oder "right" zu definieren und "both" zu vermeiden.

Die Richtung eines Links wird ausserdem im ERM-Diagramm sichtbar sein.

Beispiele

```
Direction = "left"
```

```
Direction = "right"
```

```
Direction = "both"
```

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese optionale Eigenschaft definiert den Namen eines Links, unter dem der Link später in der Datenbank-Definition und im ERM-Diagramm (innerhalb der Workbench) sichtbar sein wird.

Beispiele

```
Name = "sales_customer"
```

Link-Expression

Operator

Typ

Konstante Zeichenkette ('<', '<=', '=', '>=', '>' oder '<>')

Seit

2.0.1

Beschreibung

Diese Eigenschaft definiert den logischen Operator für eine Link-Expression. Der Operator kann '<', '<=', '=', '>=', '>' oder '<>' sein. Dieser Operator wird vom SQL-Generator verwendet.

Beispiele

```
Operator = "="
```

SourceExpression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

2.0.1

Beschreibung

Mit dieser Eigenschaft können sie den linken SQL-Ausdruck für den Link definieren, der vom SQL-Generator verwendet wird. Der Ausdruck kann entweder ein einfacher Tabellen/Spalten-Ausdruck sein oder etwas komplexeres, z.B. der Aufruf einer Funktion.

Beachten Sie, dass alle linken SQL-Ausdrücke eines Links (falls Sie mehrere verwenden) die selbe Tabelle verwenden müssen, ansonsten wird ein Fehler vom SQL-Generator geworfen.

Beispiele

```
Source Expression = "sales.productID"
```

```
Source Expression = "@SUBSTR( sales.date, 1, 4 )"
```

Target Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

2.0.1

Beschreibung

Mit dieser Eigenschaft können sie den rechten SQL-Ausdruck für den Link definieren, der vom SQL-Generator verwendet wird. Der Ausdruck kann entweder ein einfacher Tabellen/Spalten-Ausdruck sein oder etwas komplexeres, z.B. der Aufruf einer Funktion.

Beachten Sie, dass alle rechten SQL-Ausdrücke eines Links (falls Sie mehrere verwenden) die selbe Tabelle verwenden müssen, ansonsten wird ein Fehler vom SQL-Generator geworfen.

Beispiele

```
Target Expression = "sales.productID"
```

```
Target Expression = "@SUBSTR( sales.date, 1, 4 )"
```

CSV-Source

Delimiter

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Mit dieser Eigenschaft können Sie das Trennzeichen ändern, das beim Einlesen der CSV-Datei verwendet wird. Das Standard-Zeichen ist ",", aber jedes andere Zeichen ist ebenfalls möglich.

Beispiele

```
Delimiter = ";"
```

Siehe auch

Quote (auf Seite 241)

FirstLine As Names

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

2.0

Beschreibung

Einige CSV-Dateien enthalten in der ersten Zeile die Namen der Spalten. Wenn Sie eine solche Datei einlesen, dann sollten Sie diese Eigenschaft auf "true" setzen um automatisch die Spalten mit ihren Namen zu verbinden um später die Namen der Spalten anstelle ihrer Nummern verwenden zu können und um die erste Spalte automatisch zu überspringen und nicht als Datenzeile auszuwerten.

Da durch diese Eigenschaft die erste Zeile beim Einlesen der Daten automatisch übersprungen wird, hat diese auch eine Auswirkung auf die Eigenschaft Startline: Wenn Sie die Startline auf "10" setzen, dann werden die Daten effektiv erst ab Zeile 11 eingelesen (10 Zeilen plus die erste Zeile, die die Namen enthält).

Beispiele

```
First Line As Names = "true":
```

Mit einer CSV-Datei wie:

```
ProductNo,Date,Amount  
1,10.01.2004,1021.23  
2,12.01.2004,2483.10  
...
```

würden die Spalten automatisch wie folgt angebunden werden:

- ProductNo: Spalte 1
- Date: Spalte 2
- Amount: Spalte 3

Siehe auch

Startline (auf Seite 241)

Name

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den logischen Namen einer CSV-Datenquelle. Alle folgenden Elemente der Konfiguration (z.B. Keyloader oder CSV-Cubes) verwenden diese Datenquelle später unter diesen Namen.

Beispiele

```
Name = "SalesCSV"
```

Quote

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Mit dieser Eigenschaft können Sie den Begrenzer für Zeichenketten bestimmen, der beim Auslesen von CSV-Dateien erwartet wird. Das Standard-Zeichen ist ", aber jeder andere Begrenzer ist auch möglich.

Beispiele

```
Quote = "''
```

Siehe auch

Delimiter (auf Seite 239)

Startline

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft ermöglicht es, die ersten n Zeilen einer CSV-Datei beim Einlesen zu überspringen. Wenn diese Eigenschaft auf "0" gesetzt wird (der Standard-Wert), dann liest das System die CSV-Datei ab der ersten Zeile ein. Bei "5" wird die Datei ab der sechsten Zeile eingelesen usw.

Beispiele

```
Startline = "10"
```

Beginnt ab der 11. Zeile mit dem Einlesen der Datei

Trim

Typ

Konstanter Boolean-Wert ('true' oder 'false')

Seit

2.0

Beschreibung

Wenn diese Eigenschaft auf "true" gesetzt wird, dann werden alle Zeichenketten, die aus der Datei gelesen werden, von überflüssigen Leerzeichen am Ende befreit.

Beispiele

```
Trim = "true"
```

URL

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Die URL definiert die Lage der CSV-Datei. Da Sie hier eine URL angeben können, ist möglich sowohl Datei vom lokalen Filesystem als auch über HTTP oder anderen Quellen zu laden. Diese sind die üblichen Formen von URLs:

- `file://<path>` für Dateien aus dem lokalen Filesystem
- `http://<server>:<port>/<location>` für Dateien, die über HTTP ausgelesen werden können

Beispiele

```
URL = "file://c:/data.csv"
```

```
URL = "http://myserver/data.csv"
```

CSV-Column

Column

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Index (die Spaltennummer) einer manuell definierten CSV-Spalte. Der Index der ersten Spalte ist "1".

Beispiele

Index = "4"

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Mit dieser Eigenschaft bestimmen Sie den Namen einer manuell angelegten CSV-Spalte. Die anderen CSV-Elemente in der Konfiguration (die CSV-Keyloader und -Cubes) können auf eine Spalte über den Namen zugreifen, der mit dieser Eigenschaft gesetzt wurde.

Beispiele

Name = "CustomerID"

Type

Typ

Konstante Zeichenkette ('string', 'int', 'double' oder 'date')

Seit

1.2

Beschreibung

Verwenden Sie diese Eigenschaft, um dem System den Typ einer CSV-Spalte mitzuteilen.

Beispiele

```
Type = "string"
```

```
Type = "int"
```

```
Type = "double"
```

```
Type = "date"
```

Dimension

CronPattern

Typ

Konstante Zeichenkette (Cron-Pattern (siehe "Cron-Patterns" auf Seite 436))

Seit

1.1

Beschreibung

Das "Cron Pattern" einer Dimension bestimmt ob und wie oft die Schlüssel einer Dimension vom System synchronisiert werden. Das Cron Pattern ist ein Zeit-Schema (wie aus Unix bekannte Cron Patterns), das genau beschreibt wann das passieren wird.

Wenn das Cron Pattern einer Dimension ausgelöst wird, dann werden alle Keyloader dieser Dimension auf Neuerungen in den Datenquellen überprüft. Wenn eine solche Neuerung entdeckt wird, dann werden alle Schlüssel der Dimension gelöscht und erneut geladen, jetzt mit den neuen Inhalten der Datenquellen.

Das Synchronisieren von Dimension kann zeit-intensiv sein und sollte sehr bedächtig eingesetzt werden. Synchronisieren Sie daher Ihre Dimensionen nicht zu oft und versuchen Sie, das Cron Pattern an die Häufigkeit der Änderungen in der Datenquelle anzupassen. Eine Zeitdimension sollte z.B. nicht öfter als einmal täglich erneuert werden, dann ansonsten keine neuen Tage hinzugefügt würden.

Beispiele

```
Cron pattern = "*" :
```

Die Dimension wird jede Minute synchronisiert

```
Cron pattern = "* 0 0" :
```

Die Dimension wird jeden Tag um 00:00 synchronisiert

Default Text-Attribute

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

Üblicherweise werden die IDs der Schlüssel in Abfragen dargestellt, wenn ein Schlüssel in den Überschriften wird und kein bestimmter Text dargestellt wird. Mit dieser Eigenschaft können Sie dieses Verhalten ändern und ein bestimmtes Attribut der Dimension als Standard-Text definieren.

Wenn dieses Attribute nicht für jeden Schlüssel der Dimension existiert, dann werden die Schlüssel ohne Attribut in den Bericht einen leeren Text anzeigen.

Beispiele

```
Default Text-Attribute = "Text"
```

Name

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den Namen einer Dimension, unter dem diese dann später innerhalb der Konfiguration oder in Berichten verwendet wird.

Der Name einer Dimension sollte in der Regel immer die Einzahl enthalten, also "Produkt" statt "Produkte" oder "Kunde" statt "Kunden". Das verbessert die Anzeige der Berichte, da der Name einer Dimension auch als Überschrift für Selektoren und Spalten verwendet wird.

Beispiele

```
Name = "Product"
```

Storage

Typ

Konstante Zeichenkette ('transient' oder 'persistent')

Seit

2.0

Beschreibung

Wenn eine Dimension erstmal erstellt oder später synchronisiert wird, dann werden die Daten der Dimension (nicht die Kennzahlen) aus der Dimension in das instantOLAP-System übertragen und dort gespeichert. Es gibt zwei verschiedene Arten der Speicherung: Im Hauptspeicher (mit der Einstellung "transient") oder auf Festplatte (mit der Einstellung "persistent"). Sie sollten für jede Dimension einzeln entscheiden, wo diese gespeichert wird.

Die transient Speicherung im Hauptspeicher ist sehr schnell, kann bei sehr grossen Dimensionen aber entsprechende Ressourcen benötigen. In 512 MB Speicher können (als Faustregel) ca. 1.000.000 Schlüssel mit ihren Attributen abgelegt werden.

Die persistente Speicher ist praktisch unbegrenzt (ausser durch die Grösse des freien Festplattenspeichers), ist aber auch signifikant langsamer als die Speicherung im Hauptspeicher. Diese Art der Speicherung ist gut geeignet für Dimensionen, deren Schlüssel nicht oft gemeinsam verwendet werden (z.B. werden Sie keinen Bericht erstellen wollen, in dem eine sehr grosse Anzahl von Kunden gleichzeitig ausgewertet werden).

Alle kleinen Dimensionen, besonders die Kennzahl-Dimension, sollten im Hauptspeicher gehalten werden.

Beispiele

```
Storage = "transient"
```

```
Storage = "persistent"
```

Key

Format

Typ

Konstante Zeichenkette (Zahlen-Format (siehe "Zahlen-Formate" auf Seite 440))

Seit

1.2

Beschreibung

Für Kennzahlen (Schlüssel, die innerhalb der Kennzahl-Dimension definiert oder geladen werden) bestimmt diese Eigenschaft das Standard Zahlen- oder Datumsformat. Wenn die Kennzahl in einem Bericht (als Überschrift) verwendet wird, dann wird dieses Format automatisch zur Formatierung der entsprechenden Zeile oder Spalte angewendet.

Kennzahl-Formate können nicht verwendet werden, wenn Sie den Inhalt einer Zeile oder Spalte im Bericht über eine Formel berechnen lassen. In diesem Fall müssen Sie die Formatierung im Bericht manuell bestimmen.

Die Eigenschaft ist nur für Kennzahlen gültig. Wenn Sie diese Eigenschaft für andere Schlüssel als Kennzahlen angeben wird das System einen entsprechenden Fehler melden.

Beispiele

Format = "###,###,##0.00"

Format = "0%"

ID

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Jeder Schlüssel innerhalb einer Dimension besitzt eine eindeutige ID, die u.A. dazu verwendet wird, den Schlüssel später innerhalb der Konfiguration und Abfragen zu referenzieren. Mit dieser Eigenschaft können Sie die ID von manuell definierten Schlüsseln bestimmen.

Die ID eines Schlüssels wird ausserdem dazu verwendet, um Schlüssel als Überschriften innerhalb von Berichten darzustellen. Wenn Sie dieses Verhalten ändern möchten, können Sie in der Eigenschaft Default Text-Attribute der Dimension bestimmen, welches Attribut der Schlüssel stattdessen angezeigt werden sollen.

Wenn diese ID bereits innerhalb der Dimension existiert und von einem anderen Schlüssel verwendet wird, dann wird das System einen Fehler melden.

Beispiele

ID = "Product A"

ID = "Amount "

Siehe auch

Default Text-Attribute (auf Seite 245)

Type

Typ

Konstante Zeichenkette ('integer', 'double', 'string', 'boolean' oder 'date')

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Typ für eine Kennzahl (ein Schlüssel, der innerhalb der Kennzahl-Dimension angelegt wurde). Der Typ einer Kennzahl ist wichtig bei ihrer Verwendung innerhalb von Ausdrücken und bestimmt den Rückgabetyt für die automatisch vom System erstellen Kennzahl-Funktionen.

Es gibt verschiedene Typen ('integer', 'double', 'string', 'boolean' und 'date'). Der Standardtyp für Kennzahlen ist 'double'.

Beachten Sie, dass die Eigenschaft nur im Zusammenhang mit Kennzahlen verwendet werden kann.

Beispiele

Type = "string"

Type = "integer"

Type = "double"

Type = "date"

Type = "boolean"

Unit

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt die Einheit einer Kennzahl (ein Schlüssel innerhalb der Kennzahl-Dimension). Die Einheit ist eine optionale und frei definierbare Zeichenkette, die innerhalb von Berichten (in den Überschriften) dargestellt werden wenn eine Kennzahl verwendet wird.

Einheiten können nur für Kennzahlen bestimmt werden, ansonsten wird das System einen Fehler melden.

Beispiele

Unit = "EUR"

Key-Attribute

Dimension

Typ

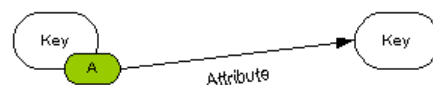
Konstante Zeichenkette

Seit

1.2

Beschreibung

Attribute können entweder Zeichenketten enthalten oder als Zeiger auf einen anderen Schlüssel einer anderen Dimension (dann werden sie "Link" genannt) funktionieren. Um auf eine andere Dimension zu verweisen, müssen Sie in dieser Eigenschaft die Ziel-Dimension bestimmen. Der Loader wird dann das Element in der Ziel-Dimension suchen, dessen ID der Eigenschaft "Value" dieses Attributes entspricht.



Wenn kein Ziel-Schlüssel mit dieser ID existiert, dann wird das Laden des Modells mit einer entsprechenden Fehlermeldung unterbrochen.

Beispiele

Dimension = "Customer"

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Jedes Attribut eines Schlüssels besitzt einen Namen, der über diese Eigenschaft bestimmt wird.

Beispiele

Name = "Color"

Unique

Typ

Konstante Zeichenkette ('true', 'false' oder 'auto')

Seit

2.0

Beschreibung

Ein Attribut kann als "eindeutig" definiert werden. Nur eindeutige Attribute können innerhalb von Cubes verwendet werden, um Dimensionen anzubinden (z.B. kann eine Dimension "Produkt" über ein eindeutiges Attribut "ProduktID" an ein Spalte "PRODUCT_ID" einer Fakten-Tabelle angebunden werden).

Wenn ein Attribute als eindeutig definiert wird, dann darf jeweils nur ein Schlüssel den gleichen Wert in diesem Attribut besitzen, z.B würde es verboten sein, dass zwei oder mehr Element einer Dimension "Produkt" die selbe "ProduktID" besitzen, wenn diese als eindeutiges Attribute definiert ist.

Es gibt zwei verschiedene Wege, ein Attribut als eindeutig zu definieren:

- Setzen Sie diese Eigenschaft auf "true": Das Attribut wird dann als zwingend eindeutig definiert. Wenn der selbe Attribut-Wert in zwei oder mehreren Schlüsseln vorkommt, wird der Loader einen Fehler melden und den Ladeprozess unterbrechen.
- Setzen Sie diese Eigenschaft auf "auto": Der Loader wird automatisch ermitteln, ob ein Attribut eindeutig ist. Dies kann jedoch ggf. zu Problemen führen, da ein Attribute, das zum Zeitpunkt der Modellierung und Anbindung an einen Cube noch eindeutig ist, später nicht mehr eindeutig sein könnten und zu einer Fehlermeldung führt.

Wenn Sie diese Eigenschaft auf "false" setzen, dann wird das Attribute nicht als eindeutig markiert und kann nicht innerhalb von Cubes verwendet werden. Beachten Sie, dass nicht-eindeutige Attribute etwas weniger System-Ressourcen benötigen. Deshalb sollten Sie vermeiden, Attribute als eindeutig zu definieren, solange Sie diese nicht in Cubes verwenden möchten.

Beispiele

Unique = "true"

```
Unique = "false"
```

```
Unique = "auto"
```

Value

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt den Wert eines Attributes. Da diese Eigenschaft ein Attribute für einen konstanten Schlüssel definiert, ist der Wert des Attributes ebenfalls konstant und besteht aus einer einfachen Zeichenketten. Abhängig von Typ des Attributes (es kann eine einfache Zeichenkette oder ein Link sein) muss diese Eigenschaft die Zeichenkette selbst oder die ID des Ziel-Schlüssels aus der Ziel-Dimension enthalten.

Beispiele

```
Value = "ProductA"
```

SQL-Keyloader

Database

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Die Eigenschaft "Database" bestimmt, aus welcher Datenbank der Keyloader seine Schlüssel laden und generieren wird. Der Wert in dieser Eigenschaft muss genau dem Namen einer vorher definierten Datenbank-Anbindung entsprechen.

Beispiele

```
Database = "SalesDB"
```

Distinct

Typ

Boolean

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt, ob die vom SQL-Generator erzeugt Statements die DISTINCT-Clause verwendet werden, um eine eindeutige Liste von Schlüsseln aus der Datenbank zu laden. Die DISTINCT-Clause in SQL eliminiert alle mehrfach vorkommenden Ergebnis-Zeilen. Einige Datenbank unterstützen jedoch das DISTINCT nicht oder nicht in Kombination mit bestimmten Spaltentypen (z.B. BLOBs). In diesem Fall können Sie das DISTINCT wegfällen lassen, indem Sie diese Eigenschaft auf "false" setzen.

Auch wenn diese Eigenschaft auf "false" gesetzt wird liest instantOLAP die Schlüssel nur einmal einlesen, jedoch werden Mehrfachnennungen dann "manuell" entfernt. Das kann das Laden von Schlüsseln erheblich verlangsamen, da evtl ein erheblich grösseres Ergebnis eingelesen werden muss. Deshalb sollten Sie vermeiden, diese Eigenschaft auf "false" zu setzen.

Beispiele

```
Distinct = "true"
```

```
Distinct = "false"
```

Format

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie das Ergebnis der SQL-Expression in Zeichenketten umwandeln. Abhängig vom Ergebnistyp der Expression muss diese Eigenschaft ein Zahlen- oder Datumsformat enthalten. Wenn Sie kein Format angeben, dann wird das Ergebnis der Abfrage mit dem Standard-Format des Ergebnis-Typs in eine Zeichenkette umgewandelt.

Beispiele

```
Format = "#,###,##0"
```

```
Format = "0.00"
```

```
Format = "dd.MM.yyyy"
```

```
Format = "dd.MM.yyyy HH:mm:ss"
```

Mode

Typ

Konstante Zeichenkette ('insert_and_extend', 'insert' oder 'extend')

Seit

2.0

Beschreibung

instantOLAP unterstützt drei verschiedene Modi für Keyloader: Einfügen-und-Erweitern ('insert_and_extend'), Einfügen ('insert') und Erweitern ('extend'). Dieser Modus bestimmt das Verhalten des Loaders für den Fall, dass ein geladener Schlüssel bereits in der Dimension existiert:

- **insert:** Es werden nur Schlüssel geladen, die bisher nicht in der Dimension existieren. Wenn ein Schlüssel bereits existiert, dann wird ein entsprechender Fehler vom System gemeldet und das Laden des Modells unterbrochen.
- **insert_and_extend:** Der Loader wird bereits existierende Schlüssel nicht bemängeln sondern die neuen (durch diesen Loader) definierten Attribute an den bereits bestehenden Schlüssel anhängen. Existiert der Schlüssel noch nicht, dann wird dieser neu angelegt. Sie können diesen Loader sowohl dazu verwenden neue Schlüssel anzulegen also auch bestehende um Attribute zu erweitern.
- **extend:** Der Loader wird nur neue Attribute zu bereits bestehenden Schlüsseln hinzufügen. Es werden keine neuen Schlüssel durch diesen Loader erzeugt. Sie können diesen Modus verwenden, um Schlüssel aus anderen Loadern durch Attribute aus weiteren Datenquellen zu erweitern. Wenn ein Schlüssel nicht existiert, wird dieser übersprungen und kein Fehler gemeldet.

Beispiele

```
Mode = "insert_and_extend"
```

```
Mode = "insert"
```

```
Mode = "extend"
```

Null-Value

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese optionale Eigenschaft bestimmt, welche Zeichenkette als ID verwendet wird, wenn das Ergebnis der SQL-Expression NULL ist. Sie können dadurch NULL-Werte durch eine gültige ID ersetzen. Wenn Sie keinen Null-Value definieren, dann werden leere Ergebnisse übersprungen und kein Wert angelegt.

Beispiele

Null-Value = "Unknown customer"

ParentFormat

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie das Format angeben, mit dem das Ergebnis der Parent SQL-Expression in Zeichenketten umgewandelt wird. Abhängig vom Ergebnistyp der Expression müssen Sie hier ein Zahlen- oder Datumsformat angeben. Wenn Sie kein Format bestimmen wird das jeweilige Standardformat für den Datentyp verwendet.

Beispiele

Parent Format = "0"

ParentSearch-Attribute

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Wenn Sie die Parent SQL-Expression für einen SQL-Keyloader manuell definieren, dann werden die geladenen Schlüssel unter die Vater-Schlüssel eingehängt, deren ID dem Ergebnis der Expression entsprechen. In einigen Fällen ist es aber nicht oder nur umständlich möglich, die ID des Vater zu ermitteln, stattdessen kann aber auf ein Attribute des Vaters (eine technische ID, eine Produkt-Nummer etc). zugegriffen werden.

In diesem Fall können Sie definieren, über welches Attribute der Vater gesucht werden soll. Dann muss das Ergebnis der Parent SQL-Expression nicht mehr der ID des Vaters sondern dem Attribut entsprechen.

Beispiele

Als Beispiel dienen zwei Tabellen CATEGORY und PRODUCT:

CATEGORY_ID	NAME
1	BEER
1	WINE

PRODUCT_ID	CATEGORY_ID	NAME
1	1	LIGHT BEER
2	1	STRONG BEER
3	2	WHITE WINE
4	2	RED WINE

Stellen Sie sich zwei SQL-Keyloader von, von dem der erste die Kategorien wie folgt lädt:

- SQL-Expression = "CATEGORY.NAME"
- SQL-Attribute mit Namen "CategoryId" und SQL-Expression "CATEGORY.CATEGORY_ID"

Ein zweiter Loader lädt die Produkte wie folgt:

- SQL-Expression = "PRODUCT.NAME"
- Parent SQL-Expression = "PRODUCT.CATEGORY_ID"
- Parent Search-Attribute = "CategoryId"

Der zweite Loader wird seine Schlüssel korrekt in die Hierarchie einsortieren, ohne die der Vater-Schlüssel zu kennen. Stattdessen wird der jeweilige Vater über die "CategoryId" (die in der PRODUCT-Tabelle steht) gesucht.

Siehe auch

Parent SQL-Expression (auf Seite 259)

ParentSQL-Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.2

Beschreibung

Diese optionale Eigenschaft bestimmt den SQL-Ausdruck, mit dem die IDs generiert werden, die dann zum Suchen der Vater-Schlüssel und Einhängen der neuen Schlüssel in die Hierarchie verwendet werden.

Das Setzen der Parent SQL-Expression generiert nicht die Väter-Schlüssel sondern bestimmt nur, nach welchem Schlüssel für die neuen Schlüssel als Vater gesucht werden soll. Wenn kein Vater existiert, wird weder der Vater noch der neue Schlüssel generiert.

Wenn Sie keine Parent SQL-Expression angeben, dann wird die SQL-Expression des darüberliegenden SQL-Keyloaders (sofern vorhanden) als Parent SQL-Expression verwendet. Die Verwendung von verschachtelten Keyloader ist der Regelfall, das manuelle setzen dieser Expression eher die Ausnahme.

Wenn Sie den Vater nicht anhand seiner ID suchen möchten sondern über ein Attribut des Vaters, dann können Sie dies über die Eigenschaft Parent Search-Attribute bestimmen.

Beispiele

```
SQL-Parent-Expression = "STAFF.COMPANY":
```

Sucht nach Schlüssel mit der ID aus der Spalte COMPANY als Vater

ParentTrim

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Mit der Eigenschaft "Trim Parent" können Sie den Loader dazu veranlassen, die IDs der Vater-Schlüssel von überflüssigen Leerzeichen am Ende zu befreien bevor damit der Vater-Schlüssel für einen neuen Schlüssel gesucht wird.

Beispiele

```
Trim parent = "true"
```

```
Trim parent = "false"
```

Recursive

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie den SQL-Keyloader in den "rekursiven" Lademodus umschalten. Rekursives Laden von Schlüssel bedeutet, dass Sie sowohl Väter als auch Kinder mit einem Keyloader gleichzeitig laden und eine komplette Hierarchie aufbauen können. Das ist besonders hilfreich, wenn Sie eine Tabelle auslesen möchten, in der mehrere Hierarchieebenen in einer Tabelle liegen.

Dies geschieht über eine Wiederholung des Ladevorgangs, der erst abbricht, wenn keine neuen Schlüssel mehr geladen wurden. Da in einem Durchgang evtl. Schlüssel geladen werden, die erst im nächsten als Väter dienen, kann so eine komplette Hierarchie geladen werden. Im rekursiven Modus müssen Sie immer die Parent SQL-Expression definieren.

Beispiele

Als Beispiel dient eine Tabelle PRODUCT:

PRODUCT_ID	PARENT_ID	NAME
5	2	WHITE WINE
4	2	RED WINE
3	1	BEER

2	1	WINE
1	NULL	ALL PRODUCTS

Mit dieser Tabelle könnten Sie einen rekursiven SQL-Keyloader mit den folgenden Einstellungen definieren:

- `SQL-Expression = "PRODUCT.NAME"`
- `SQL-Parent-Expression = "PRODUCT.PARENT_ID"`
- `Parent-Attribute = "ProductID"` (suche den Vater über seine ProductID)
- `SQL-Attribute "ProductID"` mit der `SQL-Expression "PRODUCT.PRODUCT_ID"`
- `Recursive = "true"`

Jetzt würde der Loader alle Schlüssel in 3 Durchgängen laden:

Erster Durchgang:

- `ALL PRODUCTS` wird geladen

Zweiter Durchgang:

- `WINE` wird geladen (der Vater `ALL PRODUCTS` existiert jetzt)
- `BEER` wird geladen (der Vater `ALL PRODUCTS` existiert jetzt)

Dritter Durchgang:

- `WHITE WINE` wird geladen (Der Vater `WINE` existiert jetzt)
- `RED WINE` wird geladen (Der Vater `WINE` existiert jetzt)

Vierter Durchgang:

- Es kann nichts mehr geladen werden und der Loader bricht ab

Siehe auch

Parent Search-Attribute, Parent SQL-Expression

SQL-Check

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.2

Beschreibung

Jedesmal wenn eine Dimension synchronisiert wird, fragt Sie bei den verschiedenen Keyloadern nach ob eine Änderung in den Quell-Daten stattgefunden hat. Für einen SQL-Loader können Sie dazu in dieser Eigenschaft eine SQL-Expression angeben, die zur Überprüfung einer Änderung verwendet wird. Die Dimension wird zum erneuten Laden der Schlüssel veranlasst, wenn das Ergebnis dieses SQL-Ausdrucks einen anderen Wert ergibt als bei der Abfrage zuvor. Wenn kein "SQL-Check" definiert wurde, dann wird die Dimension ihre Schlüssel jedesmal neu laden.

Beispiele

```
SQL-Check = "MAX( Products.ROWID )":
```

Überprüft (in Oracle-Datenbanken) auf die höchste ROWID

```
SQL-Check = "COUNT( Products.ID )":
```

Wurden Produkte hinzugefügt oder gelöscht?

SQL-Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den SQL-Ausdruck mit dem die IDs für die neuen Schlüssel generiert werden. Der Ausdruck muss ein gültiges SQL-Fragment für die Datenbank sein, die von diesem Loader verwendet wird. Sie können ein einfache Spalte aus einer Tabelle lesen oder komplexere SQL-Funktionen und -Ausdrücke verwenden.

Jeder Schlüssel innerhalb einer Dimension muss eine eindeutige ID besitzen. Deswegen sollte dieser SQL-Ausdruck auch eindeutige IDs generieren.

Beispiele

```
SQL-Expression = "Customer.Name"
```

```
SQL-Expression = "Customer.CustomerID || Customer.Name"
```

```
SQL-Expression = "@SUBSTR( Customer.CusomerName, 1, 10 )"
```

SQL-Order

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.1

Beschreibung

Um die Reihenfolge der vom SQL-Keyloader generierten Schlüssel zu definieren können Sie diese optionale Eigenschaft verwenden. Diese Eigenschaft erwartet einen SQL-Ausdruck nach dem die Datenbank-Abfrage sortiert wird.

Beispiele

```
Order = "Cusomer.ID"
```

SQL-Where

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.1

Beschreibung

Verwenden Sie diese Eigenschaft, um das Ergebnis des SQL-Keyloaders mit einem Filter (einer WHERE-Clause) einzuschränken. Die WHERE-Clause in dieser Eigenschaft wird bei jeder Ausführung des Keyloaders an die generierten SQL-Statements angehängt. Die Eigenschaft erwartet eine SQL-Expression vom Typ "Boolean".

Beispiele

```
SQL-Where = "Products.State = '1'":
```

Lädt nur Produkte mit Status "1"

Trim

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Mit der Eigenschaft "Trim" können Sie den Keyloader dazu bringen, überflüssige Leerzeichen am Ende der ID zu entfernen, nachdem Sie aus der Datenbank gelesen wurden.

Beispiele

```
Trim = "true"
```

```
Trim = "false"
```

SQL-Attribute

Dimension

Typ

Konstante Zeichenkette (Name der anderen Dimension)

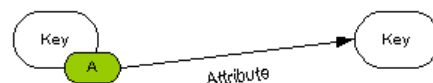
Seit

2.0

Beschreibung

Attribute können entweder Zeichenketten enthalten oder als Zeiger auf einen anderen Schlüssel einer anderen Dimension (dann werden sie "Link" genannt) funktionieren. Um auf eine andere Dimension zu verweisen, müssen Sie in dieser Eigenschaft die Ziel-Dimension bestimmen. Der Loader wird dann das Element in der Ziel-Dimension suchen, dessen ID dem Wert entspricht, der durch die SQL-Expression generiert wurde und darauf verweisen.

Wenn kein Ziel-Schlüssel mit der entsprechenden ID gefunden wird, dann wird das System einen entsprechenden Fehler melden und das Laden des Modells unterbrechen. Dieses Verhalten können Sie mit der Eigenschaft Ignore Missing Targets ändern.



Beispiele

Dimension = "Customer" :

Dieses Attribut wird ein Link zur Dimension "Customer"

Siehe auch

Ignore Missing Targets (auf Seite 266), Relink-Attribute (auf Seite 267)

Format

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Mit dieser Eigenschaft können Sie das Ergebnis der SQL-Expression in Zeichenketten umwandeln. Abhängig vom Ergebnis-Typ der Expression muss diese Eigenschaft ein Zahlen- oder Datums-Format enthalten, das dann auf das Ergebnis der SQL-Abfrage angewandt wird bevor der Wert als Attribut gespeichert (oder ein verknüpfter Schlüssel damit gesucht) wird.

Wenn Sie kein Format angeben, dann wird das Standard-Konvertierungsformat für den jeweiligen Rückabetyp der SQL-Expression verwendet.

Beispiele

```
Format = "#,###,##0"
```

```
Format = "0.00"
```

```
Format = "dd.MM.yyyy"
```

```
Format = "dd.MM.yyyy HH:mm:ss"
```

IgnoreMissingTargets

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.1

Beschreibung

Wenn dieses Attribut als Link zu einer anderen Dimension generiert wurde, dann werden in der Ziel-Dimension die Schlüssel mit der jeweiligen ID gesucht, die dem Ergebnis der SQL-Expression entsprechen. Wenn kein Schlüssel gefunden wird, dann bricht das System das Laden des Modells ab und meldet einen entsprechenden Fehler.

Sie können dieses Verhalten ändern und fehlende Ziele ignorieren, indem Sie diese Eigenschaft auf "true" setzen.

Beispiele

```
Ignore Missing Targets = "true"
```

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Jedes Attribut muss einen Namen besitzen, denn Sie über diese Eigenschaft setzen müssen.

Beispiele

Name = "Color"

Null-Value

Typ

Konstante Zeichenkette

Seit

1.0

Beschreibung

Diese optionale Eigenschaft bestimmt, welcher Wert für Attribute verwendet wird wenn das Ergebnis der SQL-Expression NULL ist. Wenn Sie diese Eigenschaft nicht setzen werden alle Attribute für NULL-Werte übersprungen und nicht eingelesen.

Beispiele

Null-Value = "Unbekannt"

Relink-Attribute

Typ

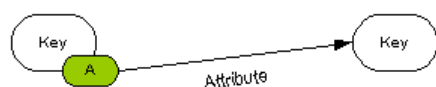
Konstante Zeichenkette

Seit

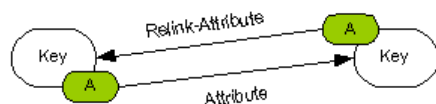
2.0

Beschreibung

Schlüssel einer Dimension können mit denen einer anderen als "Link" verknüpft werden (siehe Eigenschaft Dimension). Wenn ein Schlüssel verknüpft wird, dann können Sie den Link innerhalb von Ausdrücken wie andere Attribute verwenden, jedoch wird dieser dann Elemente vom Typ "Key" als Ergebnis haben. Z.B. würde der Ausdruck "Customer.Product" die Produkte eines Kunden zurückgeben (wenn die Kunden mit einem Link-Attribute namens "Product" geladen wurden).



Normalerweise funktionieren Links nur in eine Richtung. Der erste (Quell-) Schlüssel (in dessen Key-Loader Sie den Link über die Eigenschaft Dimension definiert haben) zeigt auf den Ziel-Schlüssel, jedoch nicht andersherum. Wenn Sie aber ausserdem auch möchten, dass der Ziel-Schlüssel auf den Quell-Schlüssel zurückverweist, dann müssen Sie den Namen des gegenläufigen Links in dieser Eigenschaft angeben. Z.B. können die Eigenschaft "Relink-Attribute" im obige Beispiel auf "Customers" gesetzt werden und dann wäre ein Ausdruck "Product.Customers" ebenfalls zulässig.

**Beispiele**

```
Relink-Attribute = "Customers"
```

Siehe auch

Dimension (auf Seite 265)

SQL-Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.2

Beschreibung

Diese notwendige Eigenschaft bestimmt den SQL-Ausdruck, der für die Generierung des Attribut-Wertes verantwortlich ist. Der Ausdruck muss ein gültiges SQL-Fragment für die Datenbank sein, die vom entsprechenden SQL-Keyloader verwendet wird. Sie können einfache Tabellen/Spalten-Ausdrücke oder komplexere Formeln in der SQL-Expression verwenden.

Beispiele

```
SQL-Expression = "CUSTOMER.NAME" :
```

Lade den Namen des Kunden als Attribut

TargetAttribute

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft ermöglicht eine komplexere und fortgeschrittenere Verknüpfung von zwei verschiedenen Dimensionen.

Die Eigenschaft Dimension ermöglicht eine einfache Verknüpfung bei dem der Ziel-Schlüssel anhand seiner ID gesucht wird. Manchmal ist das jedoch nicht möglich, da der Loader des Quell-Loaders keinen Zugriff auf die ID des Ziels hat (oder es zu aufwendige SQL-Statements benötigen würde, die komplette ID zu generieren).

In diesem Fall können Sie zwei Schlüssel verknüpfen, indem Sie das Ziel anhand eines seiner Attribute suchen. Z.B. könnte ein Loader für Produkte Zugriff auf die Spalte "MANUFACUTER_ID" haben und das Ziel in der Hersteller-Dimension über diese ID (die z.B. als Attribut "ManufacturerID" im Ziel existiert) suchen. Wenn in der Ziel-System kein entsprechendes Attribut existiert, wird das System einen Fehler melden.

Beispiele

```
Target-Attribute = "CustomerID" :
```

Suche den Ziel-Schlüssel anhand seines Attributes "Customer-ID"

Trim

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Wenn diese Eigenschaft "Trim" auf "true" gesetzt wird, dann zwingen Sie den Loader dazu, überflüssige Leerzeichen am Ende des Attributwertes abzuschneiden.

Beispiele

```
Trim = "true"
```

```
Trim = "false"
```

Unique

Typ

Konstante Zeichenkette ('true', 'false' oder 'auto')

Seit

2.0

Beschreibung

Ein Attribut kann als "eindeutig" definiert werden. Nur eindeutige Attribute können innerhalb von Cubes verwendet werden, um Dimensionen anzubinden (z.B. kann eine Dimension "Produkt" über ein eindeutiges Attribut "ProduktID" an ein Spalte "PRODUCT_ID" einer Fakten-Tabelle angebunden werden).

Wenn ein Attribute als eindeutig definiert wird, dann darf jeweils nur ein Schlüssel den gleichen Wert in diesem Attribut besitzen, z.B würde es verboten sein, dass zwei oder mehr Elemente einer Dimension "Produkt" die selbe "ProduktID" besitzen, wenn diese als eindeutiges Attribute definiert ist.

Es gibt zwei verschiedene Wege, ein Attribut als eindeutig zu definieren:

- Setzen Sie diese Eigenschaft auf "true": Das Attribut wird dann als zwingend eindeutig definiert. Wenn der selbe Attribut-Wert in zwei oder mehreren Schlüsseln vorkommt, wird der Loader einen Fehler melden und den Ladeprozess unterbrechen.
- Setzen Sie diese Eigenschaft auf "auto": Der Loader wird automatisch ermitteln, ob ein Attribut eindeutig ist. Dies kann jedoch ggf. zu Problemen führen, da ein Attribute, das zum Zeitpunkt der Modellierung und Anbindung an einen Cube noch eindeutig ist, später nicht mehr eindeutig sein könnten und zu einer Fehlermeldung führt.

Wenn Sie diese Eigenschaft auf "false" setzen, dann wird das Attribute nicht als eindeutig markiert und kann nicht innerhalb von Cubes verwendet werden. Beachten Sie, dass nicht-eindeutige Attribute etwas weniger System-Ressourcen benötigen. Deshalb sollten Sie vermeiden, Attribute als eindeutig zu definieren, solange Sie diese nicht in Cubes verwenden möchten.

Beispiele

```
Unique = "true"
```

```
Unique = "false"
```

```
Unique = "auto"
```

CSV-Keyloader

Column

Typ

Konstante Zeichenkette (Name der Spalte)

Seit

1.2

Beschreibung

Die Eigenschaft "Column" definiert, aus welcher Spalte der CSV-Source die IDs für die zu generierenden Schlüssel ausgelesen werden sollen. Der Wert kann entweder eine Zahl sein, die die Spaltennummer definiert (beginnend mit 1) oder Namen der Spalte (die Spaltennamen werden in der CSV-Source selbst definiert).

Beispiele

```
Column = "1"
```

```
Column = "Product"
```

CSV-Source

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt aus welcher CSV-Source der Keyloader die Schlüssel laden und generieren wird. Diese Eigenschaft muss den genauen Namen einer vorher definierten CSV-Source enthalten.

Beispiele

```
CSV-Source = "customers"
```

Format

Typ

Format (siehe "Formate" auf Seite 435)

Seit

2.0

Beschreibung

Verwenden Sie diese Eigenschaft, um die aus der Spalte geladenen IDs zu formatieren, bevor Schlüssel mit ihr erzeugt werden. Abhängig vom Typ der Spalte müssen Sie hier ein Zahlen- oder Datums-Format angeben.

Beispiele

```
Format = "0"
```

Levelname

Typ

Konstante Zeichenkette

Seit

2.1

Beschreibung

Jeder Level (d.h. jede Ebene) einer Dimension besitzt einen Namen, der innerhalb einer Dimension eindeutig ist. Wenn Sie mit einem CSV-Keyloader Schlüssel aus einer CSV-Source lesen, dann bestimmt diese Eigenschaft den Namen der Ebene, in der die Schlüssel durch diesen Loader angelegt werden.

Wenn kein Name definiert wird, dann wird automatisch einer vom System generiert oder der von anderen Loadern definierte Name verwendet. Beachten Sie, dass ein Level nur einen einzigen Namen besitzen kann und mehrere Namen nicht möglich sind. Wenn verschiedene Loader also verschiedene Namen für die selbe Ebene vergeben, dann wird ein entsprechender vom System gemeldet.

Beispiele

```
Levelname = "Customer"
```

Mode

Typ

Konstante Zeichenkette ('insert_and_extend', 'insert' oder 'extend')

Seit

2.0

Beschreibung

instantOLAP unterstützt drei verschiedene Modi für Keyloader: Einfügen-und-Erweitern ('insert_and_extend'), Einfügen ('insert') und Erweitern ('extend'). Dieser Modus bestimmt das Verhalten des Loaders für den Fall, dass ein geladener Schlüssel bereits in der Dimension existiert:

- **insert:** Es werden nur Schlüssel geladen, die bisher nicht in der Dimension existieren. Wenn ein Schlüssel bereits existiert, dann wird ein entsprechender Fehler vom System gemeldet und das Laden des Modells unterbrochen.
- **insert_and_extend:** Der Loader wird bereits existierende Schlüssel nicht bemängeln sondern die neuen (durch diesen Loader) definierten Attribute an den bereits bestehenden Schlüssel anhängen. Existiert der Schlüssel noch nicht, dann wird dieser neu angelegt. Sie können diesen Loader sowohl dazu verwenden neue Schlüssel anzulegen also auch bestehende um Attribute zu erweitern.
- **extend:** Der Loader wird nur neue Attribute zu bereits bestehenden Schlüsseln hinzufügen. Es werden keine neuen Schlüssel durch diesen Loader erzeugt. Sie können diesen Modus verwenden, um Schlüssel aus anderen Loadern durch Attribute aus weiteren Datenquellen zu erweitern. Wenn ein Schlüssel nicht existiert, wird dieser übersprungen und kein Fehler gemeldet.

Beispiele

```
Mode = "insert_and_extend"
```

```
Mode = "insert"
```

```
Mode = "extend"
```

Null-Value

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese optionale Eigenschaft bestimmt die Zeichenkette, die als ID anstelle von aus der CSV-Datei geladenen NULL-Werte verwendet werden sollen. Mit dieser Eigenschaft können sie also leere Werte aus der CSV-Datei durch sinnvolle Werte ersetzen. Wenn kein Null-Value definiert wird, dann werden leere Werte aus der CSV-Datei übergangen und nicht eingelesen.

Beispiele

```
Null-Value = "Unbekannter Kunde"
```

Parent-Column

Typ

Konstante Zeichenkette (Name der Spalte)

Seit

1.2

Beschreibung

Die optionale Eigenschaft "Parent-Column" bestimmt die Spalte der CSV-Source, aus der die IDs des Vater-Schlüssels ermittelt wird. Das Setzen dieser Eigenschaft erzeugt nicht die Vater-Schlüssel sondern definiert nur, unter welchen (bereits bestehenden) Schlüssel die neuen einsortiert werden. Wenn kein Vater mit der entsprechenden ID existiert, dann wird ein Schlüssel nicht geladen und übersprungen.

Wenn Sie keine Parent-Column angeben, dann wird die ID-Spalte des darüberliegenden CSV-Loaders als Parent-Column verwendet (z.B. wenn Sie in der Workbench eine CSV-Spalte auf einen bestehenden Loader schieben). Verschachtelte Keyloader sind der übliche Weg eine Vater-Kind Beziehung aufzubauen, das manuelle Setzen der Parent-Column ist eher die Ausnahme.

Beispiele

```
Parent-Column = "4"
```

```
Parent-Column = "Category"
```

Parent-Format

Typ

Format (siehe "Formate" auf Seite 435)

Seit

2.0

Beschreibung

Verwenden Sie diese Eigenschaft, um den aus der Parent-Column geladenen Wert zu formatieren, bevor das System damit den Vater-Schlüssel sucht. Abhängig vom Typ der Spalte müssen Sie hier ein Zahlen- oder Datums-Format angeben.

Beispiele

```
Parent-Format = "0"
```

CSV-Attribute

Column

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt, aus welcher Spalte der CSV-Datei der Wert für dieses Attribute geladen wird.

Hier kann entweder die Spaltennummer als Zahl (der Index der ersten ist 1) oder der Name der Spalte angegeben werden (die Namen der Spalten werden innerhalb des CSV-Source Elementes definiert).

Beispiele

```
Column = "2"
```

```
Column = "Name"
```

Dimension

Typ

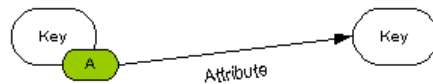
Konstante Zeichenkette

Seit

2.0

Beschreibung

Attribute können entweder Zeichenketten enthalten oder als Zeiger auf einen anderen Schlüssel einer anderen Dimension (dann werden sie "Link" genannt) funktionieren. Um auf eine andere Dimension zu verweisen, müssen Sie in dieser Eigenschaft die Ziel-Dimension bestimmen. Der Loader wird dann das Element in der Ziel-Dimension suchen, dessen ID dem Wert entspricht, der dem aus der (durch die Eigenschaft Column) CSV-Spalte geladenen Wert entspricht und darauf verweisen.



Wenn kein Schlüssel in der Zieldimension mit der entsprechenden ID gefunden wurde, dann meldet das System einen entsprechenden Fehler und bricht den Ladevorgang ab.

Beispiele

`Dimension = "Kunde"` :

Dieses Attribute wird ein Link zur "Kunde"

Format

Typ

Format (siehe "Formate" auf Seite 435)

Seit

1.2

Beschreibung

Mit dieser Eigenschaft können Sie den Inhalt einer CSV-Spalte in Zeichenketten konvertieren. Abhängig vom Datentyp der Spalte müssen Sie hier ein Dezimal- oder Datums-Format angeben. Dann wird das System den Inhalt der Spalte anhand dieses Formates in eine Zeichenkette umwandeln.

Wenn Sie kein Format bestimmen, dann werden die Inhalte der Spalte in Zeichenketten (mit dem jeweiligen Standardformat) konvertiert.

Beispiele

`Format = "#,###,##0"`

`Format = "0.00"`

```
Format = "dd.MM.yyyy"
```

```
Format = "dd.MM.yyyy HH:mm:ss"
```

Name

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Jedes Attribut eines Schlüssels muss einen Namen besitzen. Diese Eigenschaft bestimmt den Namen dieses Attributs.

Beispiele

```
Name = "color"
```

Null-Value

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese optionale Eigenschaft bestimmt, welchen Wert das Attribut erhält wenn die Spalte aus der CSV-Datei leer (NULL) ist. Wenn Sie keinen Null-Value definieren, dann werden leere Werte aus der Spalte nicht übernommen und das Attribut fällt für den jeweiligen Schlüssel weg.

Beispiele

```
Null-Value = "Unnamed"
```

Unique

Typ

Konstante Zeichenkette ('true', 'false' oder 'auto')

Seit

2.0

Beschreibung

Ein Attribut kann als "eindeutig" definiert werden. Nur eindeutige Attribute können innerhalb von Cubes verwendet werden, um Dimensionen anzubinden (z.B. kann eine Dimension "Produkt" über ein eindeutiges Attribut "ProduktID" an ein Spalte "PRODUCT_ID" einer Fakten-Tabelle angebunden werden).

Wenn ein Attribute als eindeutig definiert wird, dann darf jeweils nur ein Schlüssel den gleichen Wert in diesem Attribut besitzen, z.B würde es verboten sein, dass zwei oder mehr Elemente einer Dimension "Produkt" die selbe "ProduktID" besitzen, wenn diese als eindeutiges Attribute definiert ist.

Es gibt zwei verschiedene Wege, ein Attribut als eindeutig zu definieren:

- Setzen Sie diese Eigenschaft auf "true": Das Attribut wird dann als zwingend eindeutig definiert. Wenn der selbe Attribut-Wert in zwei oder mehreren Schlüsseln vorkommt, wird der Loader einen Fehler melden und den Ladeprozess unterbrechen.
- Setzen Sie diese Eigenschaft auf "auto": Der Loader wird automatisch ermitteln, ob ein Attribut eindeutig ist. Dies kann jedoch ggf. zu Problemen führen, da ein Attribute, das zum Zeitpunkt der Modellierung und Anbindung an einen Cube noch eindeutig ist, später nicht mehr eindeutig sein könnten und zu einer Fehlermeldung führt.

Wenn Sie diese Eigenschaft auf "false" setzen, dann wird das Attribute nicht als eindeutig markiert und kann nicht innerhalb von Cubes verwendet werden. Beachten Sie, dass nicht-eindeutige Attribute etwas weniger System-Ressourcen benötigen. Deshalb sollten Sie vermeiden, Attribute als eindeutig zu definieren, solange Sie diese nicht in Cubes verwenden möchten.

Beispiele

```
Unique = "true"
```

```
Unique = "false"
```

```
Unique = "auto"
```

Time-Keyloader

End

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den Zeitpunkt bis zum dem vom Time-Loader Schlüssel generiert werden. Diese Eigenschaft erwartet ein Datums-Format im lokalen Format des Servers. Wenn Sie nicht das lokale Format des Servers verwenden möchten, können Sie das mit der Eigenschaft Locale Format ändern.

Wenn keine Eigenschaft "End" definiert wird, dann wird der Loader das "End" vom umgebenden Time-Keyloader übernehmen. Wenn der Loader unter keinen anderen liegt, dann wird der aktuelle Zeitpunkt als End-Zeitpunkt verwendet.

Beispiele

End = "31.12.2004" :

Der Keyloader generiert Schlüssel bis zum 31.12.2004

End = "31.12.yyyy" :

Der Keyloader generiert Schlüssel bis zum 31.12 des aktuellen Jahres

End = "dd.MM.yyyy" :

Der Keyloader generiert Schlüssel bis heute

End = " " :

Der Keyloader generiert Schlüssel bis heute

Siehe auch

Endshift (auf Seite 282), Locale Format (auf Seite 284), Start (auf Seite 287)

Endshift

Typ

Integer-Konstante

Seit

1.2

Beschreibung

Die Eigenschaft End ermöglicht es Ihnen, den Endzeitpunkt auf einen konstantes Jahr, Monat, Tag oder auf das aktuelle Jahr, Monat etc. zu setzen. Es ist aber nicht möglich, relative Angaben zu machen, z.B. morgen oder der letzte Monat. Diese Eigenschaft erlaubt es Ihnen daher, den Endzeitpunkt zusätzlich vorwärts oder rückwärts zu verschieben. Geben Sie hier die Anzahl der Millisekunden (positiv oder negativ) an, um den Sie den Zeitpunkt verschieben möchten.

Beispiele

```
Stop = "dd.MM.yyyy"
```

```
Endshift = "86400000"
```

Diese Einstellungen lassen des Loader auf den morgigen Tag enden, da das Ende (heute) um $24 * 60 * 60 * 1000 (=86400000)$ Millisekunden verschoben wird.

Siehe auch

End (auf Seite 281), Startshift (auf Seite 288)

Exclude Patterm

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.2

Beschreibung

Ein Time-Keyloader ermöglicht das Überspringen von Schlüssel bei der Generierung, d.h. für bestimmte Wochentage, Tage etc. können optional keine Schlüssel erzeugt werden. Für das Überspringen benötigen Sie zwei Eigenschaften: Das Exclude-Pattern (diese Eigenschaft) und die Exclude Values. Diese Eigenschaft definiert ein weiteres Datums-Format, mit dem für jeden Schlüssel eine Zeichenkette erstellt wird. Wenn diese Zeichenkette in den "Exclude Values" vorkommt, dann wird der Schlüssel nicht zur Dimension hinzugefügt. Die "Exclude Values" enthalten eine, durch Kommata getrennte, Liste von Zeichenketten.

Beispiele

```
Exclude-Pattern = "ee"
```

```
Exclude-Values = "Sa,So"
```

Dieses Beispiel verhindert die Generierung von Tagen, die am Wochenende liegen. Das Pattern "eee" (Wochentag) ergibt am Wochen "Sa" oder "So", die in den Exclude-Values stehen.

Siehe auch

Exclude Values (auf Seite 283)

Exclude Values

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Eine Time-Keyloader ermöglicht das Auslassen von Schlüssel bei der Generierung. Diese Eigenschaft bestimmt, im Zusammenhang mit der Eigenschaft Exclude Pattern, welche Schlüssel ausgelassen werden sollen. Lesen Sie hierzu die Beschreibung der Eigenschaft Exclude Pattern.

Siehe auch

Exclude Pattern (auf Seite 282)

LocaleFormat

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.2

Beschreibung

Einige Eigenschaften eines Time-Keyloaders arbeiten mit Datums-Formaten. Diese Formate werden im lokalen Zeitformat des Server erwartet. Sie können diese Eigenschaft verwenden, um ein anderes Datumsformat zu definieren und den Time-Keyloader somit unabhängig von der Sprache des Servers zu machen.

Beispiele

```
Locale Format = "dd.MM.yyyy" :
```

Der Loader akzeptiert jetzt Angaben im Format "dd.MM.yyyy"

```
Locale Format = "yyyy/dd/MM" :
```

Der Loader akzeptiert jetzt Angaben im Format "yyyy/MM/dd"

Siehe auch

End (auf Seite 281), Exclude Pattern (auf Seite 282), Start (auf Seite 287)

Mode

Type

Konstante Zeichenkette ('insert_and_extend', 'insert' oder 'extend')

Seit

2.0

Beschreibung

instantOLAP unterstützt drei verschiedene Modi für Keyloader: Einfügen-und-Erweitern ('insert_and_extend'), Einfügen ('insert') und Erweitern ('extend'). Dieser Modus bestimmt das Verhalten des Loaders für den Fall, dass ein geladener Schlüssel bereits in der Dimension existiert:

- **insert:** Es werden nur Schlüssel geladen, die bisher nicht in der Dimension existieren. Wenn ein Schlüssel bereits existiert, dann wird ein entsprechender Fehler vom System gemeldet und das Laden des Modells unterbrochen.
- **insert_and_extend:** Der Loader wird bereits existierende Schlüssel nicht bemängeln sondern die neuen (durch diesen Loader) definierten Attribute an den bereits bestehenden Schlüssel anhängen. Existiert der Schlüssel noch nicht, dann wird dieser neu angelegt. Sie können diesen Loader sowohl dazu verwenden neue Schlüssel anzulegen also auch bestehende um Attribute zu erweitern.
- **extend:** Der Loader wird nur neue Attribute zu bereits bestehenden Schlüsseln hinzufügen. Es werden keine neuen Schlüssel durch diesen Loader erzeugt. Sie können diesen Modus verwenden, um Schlüssel aus anderen Loadern durch Attribute aus weiteren Datenquellen zu erweitern. Wenn ein Schlüssel nicht existiert, wird dieser übersprungen und kein Fehler gemeldet.

Beispiele

```
Mode = "insert_and_extend"
```

```
Mode = "insert"
```

```
Mode = "extend"
```

ParentPattern

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.0

Beschreibung

Wenn ein Schlüssel von einem Time-Keyloader generiert wird, dann wird dieser in die Hierarchie der Dimension eingehängt. Zum Einhängen muss der Vater eines neuen Schlüssel mit dieser Eigenschaft, die ein Datums-Format enthalten muss, bestimmt werden. Für jeden generierten Schlüssel wird ebenfalls die ID des Vaters generiert und dieser danach gesucht. Wenn der Vater gefunden wird, dann wird neue Schlüssel als Kind-Element daruntergehängt. Wird keiner gefunden, dann wird der neue Schlüssel übersprungen und nicht zu Dimension hinzugefügt. Beachten Sie, dass der Vater eines Schlüssels nur gesucht, nicht aber generiert wird.

Üblicherweise müssen Sie diese Eigenschaft nicht setzen da ein Time-Keyloader das Pattern eines darüberliegenden Loaders als Parent verwendet. Verschachtelte Loader werden z.B. durch Drag&Drop in der Workbench erstellt.

Wenn kein "Parent" definiert wurde und dieser Loader sich nicht unterhalb eines anderen Time-Keyloaders befindet, dann werden alle Schlüssel direkt unter dem Wurzel-Element der Dimension aufgehängt.

Beispiele

```
Parent = "yyyy" :
```

Suche einen Vater mit dem Jahr als ID

Siehe auch

Pattern (auf Seite 286)

Pattern

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.0

Beschreibung

Ein Time-Keyloader generiert Schlüssel für einen Zeitraum. Jeder Schlüssel muss eine eindeutige ID (ein Zeichenkette) besitzen, die nur ein einziges Mal in seiner Dimension vorkommen darf. Diese Eigenschaft definiert, wie die IDs für die generierten Schlüssel erzeugt werden. Sie erwartet ein Datums-Format, mit dem das jeweilige Datum in eine Zeichenkette als ID umgewandelt wird.

Durch das "Pattern" wird automatisch auch die Schrittweite des Loaders bestimmt. Wenn z.B. das Pattern "yyyy" (die Jahreszahl mit 4 Ziffern) lautet, dann generiert der Loader Schlüssel in Jahres-Schritten. Wenn das Pattern "MMM/yy" (Monat und Jahr) lautet, dann in Monats-Schritten und so weiter.

Beispiele

```
Pattern = "yyyy" :
```

Erzeuge Schlüssel für Jahre (... ,2004,2005,...)

```
Pattern = "MMM/yyy" :
```

Erzeuge Schlüssel für Monate (...Jan/2004,Feb/2004,...)

Pattern = "dd.MM.yyyy" :

Erzeuge Schlüssel für Tage (...01.01.2004,02.01.2004,...)

Siehe auch

End (auf Seite 281), Start (auf Seite 287)

Start

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den Zeitpunkt, abdem vom Time-Loader Schlüssel generiert werden. Diese Eigenschaft erwartet ein Datums-Format im lokalen Format des Servers. Wenn Sie nicht das lokale Format des Servers verwenden möchten, können Sie das mit der Eigenschaft Locale Format ändern.

Wenn keine Eigenschaft "Start" definiert wird, dann wird der Loader das "Start" vom umgebenden Time-Keyloader übernehmen. Wenn der Loader unter keinen anderen liegt, dann wird der aktuelle Zeitpunkt als Start-Zeitpunkt verwendet.

Beispiele

Start = "01.01.2004" :

Der Time-Keyloader generiert ab dem 01.01.2004

Start = "01.01.yyyy" :

Der Time-Keyloader generiert ab dem 01.01 des aktuellen Jahres

Start = "dd.MM.yyyy" :

Der Time-Keyloader generiert ab heute

Start = "" :

Der Time-Keyloader generiert ab heute

Siehe auch

End (auf Seite 281), Locale Format (auf Seite 284), Startshift (auf Seite 288)

Startshift

Typ

Integer-Konstante

Seit

1.2

Beschreibung

Die Eigenschaft Start ermöglicht es Ihnen, den Startzeitpunkt auf einen konstantes Jahr, Monat, Tag oder auf das aktuelle Jahr, Monat etc. zu setzen. Es ist aber nicht möglich, relative Angaben zu machen, z.B. morgen oder der letzte Monat. Diese Eigenschaft erlaubt es Ihnen daher, den Startzeitpunkt zusätzlich vorwärts oder rückwärts zu verschieben. Geben Sie hier die Anzahl der Millisekunden (positiv oder negativ) an, um den Sie den Zeitpunkt verschieben möchten.

Beispiele

```
Start = "dd.MM.yyyy"
```

```
Startshift = "-86400000"
```

Diese Einstellungen lassen des Loader beim gestrigen Tag beginnen da der Start (heute) um $-24 * 60 * 60 * 1000$ ($=-86400000$) Millisekunden verschoben wird.

Siehe auch

Endshift (auf Seite 282), Start (auf Seite 287)

Time-Attribute

Dimension

Typ

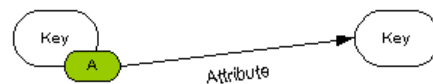
Konstante Zeichenkette

Seit

2.0

Beschreibung

Attribute können entweder Zeichenketten enthalten oder als Zeiger auf einen anderen Schlüssel einer anderen Dimension (dann werden sie "Link" genannt) funktionieren. Um auf eine andere Dimension zu verweisen, müssen Sie in dieser Eigenschaft die Ziel-Dimension bestimmen. Der Loader wird dann das Element in der Ziel-Dimension suchen, dessen ID dem Wert entspricht, der der (durch die Eigenschaft Pattern) generierten ID Wert entspricht und darauf verweisen.



Wenn kein Schlüssel in der Zieldimension mit der entsprechenden ID gefunden wurde, dann meldet das System einen entsprechenden Fehler und bricht den Ladevorgang ab.

Beispiele

Dimension = "Weekday" :

Dieses Attribut wird als Link zur Dimension "Weekday" erstellt

Siehe auch

Pattern (auf Seite 290), Relink-Attribute (auf Seite 290)

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Jedes Attribut muss einen Namen besitzen. Diese Eigenschaft definiert den Namen des Attributes.

Beispiele

Name = "Weekday"

Pattem

Typ

Datums-Format (siehe "Datums-Formate" auf Seite 437)

Seit

1.0

Beschreibung

Diese Eigenschaft setzt das Muster (Datums-Format) mit dem der Wert für dieses Attribut generiert wird.

Beispiele

Pattern = "eee" :

Dieses Attribut enthält den Wochentag

Relink-Attribute

Typ

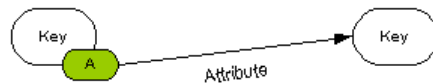
Konstante Zeichenkette

Seit

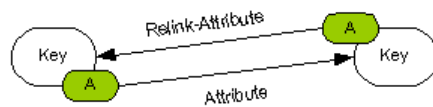
2.0

Beschreibung

Schlüssel einer Dimension können mit denen einer anderen als "Link" verknüpft werden (siehe Eigenschaft Dimension). Wenn ein Schlüssel verknüpft wird, dann können Sie den Link innerhalb von Ausdrücken wie andere Attribute verwenden, jedoch wird dieser dann Elemente vom Typ "Key" als Ergebnis haben. Z.B. würde der Ausdruck "Time.Weekday" die Wochentage z.B. eines Monats zurückgeben (wenn die Zeit mit einem Link-Attribute namens "Weekday" geladen wurden).



Normalerweise funktionieren Links nur in eine Richtung. Der erste (Quell-) Schlüssel (in dessen Key-Loader Sie den Link über die Eigenschaft Dimension definiert haben) zeigt auf den Ziel-Schlüssel, jedoch nicht andersherum. Wenn Sie aber ausserdem auch möchten, dass der Ziel-Schlüssel auf den Quell-Schlüssel zurückverweist, dann müssen Sie den Namen des gegenläufigen Links in dieser Eigenschaft angeben. Z.B. können die Eigenschaft "Relink-Attribute" im obige Beispiel auf "Time" gesetzt werden und dann wäre ein Ausdruck "Weekday.Time" ebenfalls zulässig.



Beispiele

Relink-Attribute = "Days"

Siehe auch

Dimension (auf Seite 289)

Unique

Typ

Konstante Zeichenkette ('true', 'false' oder 'auto')

Seit

2.0

Beschreibung

Ein Attribut kann als "eindeutig" definiert werden. Nur eindeutige Attribute können innerhalb von Cubes verwendet werden, um Dimensionen anzubinden (z.B. kann eine Dimension "Produkt" über ein eindeutiges Attribut "ProduktID" an ein Spalte "PRODUCT_ID" einer Fakten-Tabelle angebunden werden).

Wenn ein Attribute als eindeutig definiert wird, dann darf jeweils nur ein Schlüssel den gleichen Wert in diesem Attribut besitzen, z.B würde es verboten sein, dass zwei oder mehr Elemente einer Dimension "Produkt" die selbe "ProduktID" besitzen, wenn diese als eindeutiges Attribute definiert ist.

Es gibt zwei verschiedene Wege, ein Attribut als eindeutig zu definieren:

- Setzen Sie diese Eigenschaft auf "true": Das Attribut wird dann als zwingend eindeutig definiert. Wenn der selbe Attribut-Wert in zwei oder mehreren Schlüsseln vorkommt, wird der Loader einen Fehler melden und den Ladeprozess unterbrechen.
- Setzen Sie diese Eigenschaft auf "auto": Der Loader wird automatisch ermitteln, ob ein Attribut eindeutig ist. Dies kann jedoch ggf. zu Problemen führen, da ein Attribute, das zum Zeitpunkt der Modellierung und Anbindung an einen Cube noch eindeutig ist, später nicht mehr eindeutig sein könnten und zu einer Fehlermeldung führt.

Wenn Sie diese Eigenschaft auf "false" setzen, dann wird das Attribute nicht als eindeutig markiert und kann nicht innerhalb von Cubes verwendet werden. Beachten Sie, dass nicht-eindeutige Attribute etwas weniger System-Ressourcen benötigen. Deshalb sollten Sie vermeiden, Attribute als eindeutig zu definieren, solange Sie diese nicht in Cubes verwenden möchten.

Beispiele

```
Unique = "true"
```

```
Unique = "false"
```

```
Unique = "auto"
```

Number-Keyloader

Max

Typ

Integer-Konstante

Seit

1.2

Beschreibung

Diese Eigenschaft definiert das Ende des Bereiches der generierten Nummern (Schlüssel). Der letzte generierte Schlüssel wird diese Nummer als ID haben.

Beispiele

```
Max = "1000"
```

Siehe auch

Min (auf Seite 293)

Min

Typ

Integer-Konstante

Seit

1.2

Beschreibung

Diese Eigenschaft definiert den Start des Bereiches der generierten Nummern (Schlüssel). Der erste generierte Schlüssel wird diese Nummer als ID haben.

Beispiele

```
Min = "1"
```

Siehe auch

Max (auf Seite 293)

SQL-Cube

Complete

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt, ob ein Cube als vollständig oder unvollständig gilt. Wenn eine Wert theoretisch aus einem vollständigen Cube geladen werden kann, dann wird aus keinem anderen geladen, auch wenn der Wert nicht gefunden wurde. Eine genauere Beschreibung zu vollständigen und unvollständigen Cubes finden Sie in der Cube-Beschreibung. In der Standard-Einstellung sind Cubes vollständig.

Beispiele

Eine Datensuche mit der Eigenschaft "Complete", die auf "false" gesetzt wurde, könnte wie folgt ablaufen:

- 1 Die Daten werden im 1. Würfel (mit Complete = false) gesucht und nicht gefunden
- 2 Die Daten werden im 2. Würfel (mit Complete = false) gesucht und nicht gefunden
- 3 Die Daten werden über eine passende Formel berechnet

Bei den gleichen Würfeln, in denen jedoch "Complete" auf "true" gesetzt wurde, könnte es stattdessen wie folgt ablaufen:

- 1 Die Daten werden im 1. Würfel (mit Complete = true) gesucht und nicht gefunden
- 2 Die Suche wird aufgegeben und der Wert auf NULL gesetzt

Siehe auch

Span All Dimensions (auf Seite 299)

Database

Typ

Konstante Zeichenkette (Logischer Name der Datenbank)

Seit

1.0

Beschreibung

Diese notwendige Eigenschaft bestimmt, aus welcher Datenbank der Cube seine Daten laden wird. Der Wert muss dem Namen einer zuvor definierten Datenbank-Anbindung entsprechen.

Beispiele

```
Database = "SalesDB"
```

EnableLoad

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

SQL-Cubes können sowohl Daten aus Datenbanken lesen als auch in ihnen speichern. Beide Eigenschaft können separat ein- oder abgeschaltet werden. Die Eigenschaft "Enable Load" bestimmt, ob ein Cube Daten aus der Datenbank lesen wird.

Die Standard-Einstellung für diese Eigenschaft ist "true". Jeder Cube, der Daten lesen kann, wird alle Daten aus seiner Datenquelle auslesen, wenn die angebenen Kennzahlen und Dimensionen der gesuchten Koordinate entsprechen. Wenn diese Eigenschaft auf "false" gesetzt wird, dann wird der Cube keine Daten liefern.

Beispiele

```
Enable Load = "true"
```

```
Enable Load = "false"
```

Siehe auch

Enable Store (auf Seite 297)

EnableStore

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

SQL-Cubes können sowohl Daten aus Datenbanken lesen als auch in ihnen speichern. Beide Eigenschaft können separat ein- oder abgeschaltet werden. Die Eigenschaft "Enable Store" bestimmt, ob ein Cube Daten in die Datenbank schreiben kann.

Die Standard-Einstellung für diese Eigenschaft ist "false". Wenn Sie sie auf "true" setzen, dann wird der Cube neue oder geänderte Daten zurück in die Datenbank schreiben die Kennzahl und alle Dimensionen der zu speichernden Koordinate korrekt angebunden wurden.

Das Schreiben in Datenbanken funktioniert (im Gegensatz zum Lesen) nur dann, wenn ausschließlich einfache SQL-Ausdrücke, die nur Tabellen und Spalten verwenden, im Cube verwendet wurden. Ansonsten ist es dem Cube nicht möglich, den Wert zu berechnen, der in den Spalten der Zieltabellen gespeichert werden muss. Wenn Sie z.B. eine Spalte "PRODUCT.ID" angebunden hätten, könnte diese zum Schreiben werden (das System könnte direkt die ID eines Produkt darin speichern). Ein komplexerer Ausdruck wie "SUBSTR(SALES.DATE, 1, 4)" könnte jedoch nicht zum Schreiben von Daten verwendet werden.

Beispiele

```
Enable Store = "true"
```

```
Enable Store = "false"
```

Siehe auch

Enable Load (auf Seite 296)

Line-Dimension

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft setzt die Line-Dimension für diesen Cube. Die Line-Dimension wird dazu verwendet, um Daten aus einem Cube in Listenform (und nicht aggregiert) auszulesen. Eine detaillierte Beschreibung zum Thema Line-Dimension finden Sie in der Cube-Dokumentation.

Als Wert müssen Sie den genauen Namen der (zuvor definierten) Dimension angeben, in der die Zeilen-Nummern (von 1 aufsteigend) als Element angelegt wurden.

Eine solche Dimension kann mit Hilfe eines Number-Keyloaders generiert werden, existiert aber bereits auch schon als Vorgabe im System. Wenn Sie hier keine Dimension angeben, wird diese Vorgabe-Dimension auch verwendet.

Beispiele

```
Line-Dimension = "Line"
```

Match

Typ

Boolean

Seit

1.2

Beschreibung

Für jeden Cube kann optional eine Filter-Bedingung (Match-Condition) definiert werden, durch die bestimmt wird welche Daten aus einem Cube gelesen werden sollen. Diese Bedingung für alle Koordinaten der zu ladenden Werte überprüft, die Koordinate auf die diese Bedingung zutrifft werden dann ggf. aus dem Cube geladen. Weitere Informationen zur Match-Condition finden Sie in der Cube-Dokumentation.

Beispiele

```
Match = "HASKEYS( Time:'2004' )":
```

Lade nur Daten für das Jahr 2004

```
Match = "HASLEVEL( Time, 2 )":
```

Lade nur Daten für die zweite Ebene der Dimension "Time"

Siehe auch

Span All Dimensions (auf Seite 299)

Name

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft bestimmt den Namen eines Cubes. Die Verwendung der Eigenschaft ist optional und wird nur zur Anzeige innerhalb der Workbench und Log-Ausgaben verwendet.

Beispiele

```
Name = "SALESCUBE"
```

SpanAll Dimensions

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.0

Beschreibung

Diese Eigenschaft definiert, ob ein Cube als "Hypercube" fungiert und alle Dimensionen des Modells umspannt oder ob er (als sogenannter "Multicube") nur einen Teil der Dimensionen umspannt (das ist die Standard-Einstellung). Weitere Informationen zu Hypercubes und Multicubes finden Sie in der Cube-Dokumentation.

Beispiele

```
Span All Dimensions = "true"
```

```
Span All Dimensions = "false"
```

Siehe auch

Complete (auf Seite 295)

SQL-Order

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

2.0

Beschreibung

Mit dieser Eigenschaft können Sie eine "ORDER BY" Clause an alle von diesem Cube generierten SQL-Statements anhängen lassen.

Das Verwenden von Sortierung kann besonders dann von Interesse sein, wenn Sie die Daten aus einem Cube ohne Aggregation und unter Zuhilfenahme einer Line-Dimension auslesen und die einzelnen Datensätze in einer bestimmten Reihenfolge haben möchten. Wenn Sie ohne Line-Dimension arbeiten, hätte diese Eigenschaft keine weitere Auswirkung auf den Cube.

Beispiele

```
SQL-Order = "ORDER.ORDER_NO"
```

Mit dieser Sortierung würde ein Statement wie folgt generiert werden:

```
SELECT ... FROM ORDER, ... WHERE ... ORDER BY ORDER.ORDER_NO
```

Siehe auch

SQL-Where (auf Seite 301)

SQL-Where

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.2

Beschreibung

Wenn Sie möchten, dass der Cube nur einen Teil der verwendeten Tabellen ausliest, können Sie einen Filter (in Form einer SQL-WHERE Clause) in dieser Eigenschaft angeben. Diese Clause wird dann alle von diesem Cube generierten SQL-Statements angehängt.

In einigen Fällen kann es eleganter sein, die Tabellen durch die Verwendung von Aliases mit vordefinierter WHERE-Clause einzuschränken und nur diese Aliase im Cube zu verwenden. Dadurch lässt sich die Einschränkung zentraler verwalten.

Beispiele

```
SQL-Where = "INVOICE.STATE = 1"
```

Mit diesem SQL-Where würde der Cube ein SQL-Statement wie das folgende erzeugen:

```
SELECT ... FROM ORDER, ... WHERE ... AND INVOICE.STATE = 1  
...
```

Siehe auch

SQL-Order (auf Seite 300)

SQL-Fact

Fact

Typ

Konstante Zeichenkette (ID der Kennzahl)

Seit

2.0

Beschreibung

Diese notwendige Eigenschaft bestimmt die Kennzahl, die durch diese Mapping im Cube angebunden wird. Normalerweise müssen Sie diese Eigenschaft nicht ändern, da sie automatisch vorbelegt wird wenn einen Cube in der Workbench erzeugen, z.B. durch das Ziehen einer Spalte aus dem Database-Explorer auf den Cube.

Wenn eine Kennzahl an einen Würfel angebunden wurde, dann wird dieser alle Vorkommnisse der Kennzahl laden (soweit alle Dimensionen für eine zu ladene Koordinate ebenfalls im Würfel angebunden wurden). Wenn Sie nur einen Teil der Kennzahl aus diesem Cube laden möchten oder die selbe Kennzahl aus verschiedenen Spalten (abhängig von bestimmten Bedingungen) laden möchten, dann können Sie mir der Eigenschaft Match eine Einschränkung für diese Anbindung definieren.

Beispiele

```
Fact = "Amount"
```

Siehe auch

Match (auf Seite 302)

Match

Typ

Boolean

Seit

1.0

Beschreibung

Wenn eine Kennzahl an einen SQL-Cube angebunden wird, dann lädt dieser alle Vorkommnisse der Kennzahl, solange auch alle Dimensionen einer gewünschten Koordinate angebunden wurden. Mit der Eigenschaft "Match" können Sie die Anbindung mit einer Bedingung verknüpfen und nur bestimmte Koordinaten aus der angebundenen SQL-Expression laden.

Mit dieser Eigenschaft ist es z.B. möglich, die selbe Kennzahl in einem Cube aus verschiedenen Spalten (SQL-Expression) zu laden, jeweils mit einer anderen Bedingung. Z.B. können Sie eine Kennzahl aus 12 verschiedenen Spalten laden, jeweils für einen anderen Monat des Jahres.

Beispiele

```
Match = "Time.Month = 'Jan' ":
```

Diese Anbindung gilt nur für den Monat "Januar"

Siehe auch

Fact (auf Seite 302)

SQL-Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.0

Beschreibung

Diese Eigenschaft bestimmt den SQL-Ausdruck, der vom SQL-Cube verwendet wird um die Kennzahl aus der Tabelle (aggregiert) auszulesen. Der Ausdruck sollte immer eine Aggregation enthalten (SUM, MIN, MAX, AVG, COUNT oder eine datenbankspezifische Funktion), ansonsten ist der SQL-Generator nicht in der Lage, aggregierte Werte für höhere Hierarchie-Ebenen der Dimensionen zu berechnen.

Beispiele

```
SQL-Expression = "SUM( SALES.AMOUNT )"
```

SQL-Dimension

Attribute

Typ

Konstante Zeichenkette (Name des Attributes)

Seit

2.0

Beschreibung

Dimensionen können auf zwei verschiedene Arten an eine Faktentabelle gebunden werden, mit oder ohne Verwendung von Attribute.

Wenn Sie diese Eigenschaft leer lassen, dann wird die in der Eigenschaft Dimension angegebene Dimension direkt an die Datenbank (bzw. an den in der Eigenschaft SQL-Expression definierten SQL-Ausdruck) gebunden. Eine Dimension direkt anzubinden bedeutet, dass der Würfel beim Einlesen von Ergebnissen die Schlüssel direkt über ihre ID suchen wird. Da jeder Schlüssel eine ID besitzt, wird diese Art der Anbindung alle Schlüssel einer Dimension (inklusive dem obersten Wurzel-Schlüssel) betreffen.

Normalerweise möchten Sie aber nur einen Teil einer Dimension und nicht alle Ebenen mit dem selben SQL-Ausdruck verbinden und dabei den Wurzel-Schlüssel der Dimension auslassen:

Verschiedene Ebenen einer Dimension sollten in der Regel mit verschiedenen SQL-Ausdrücken verbunden werden (um verschiedene Aggregationen zu erzeugen) und der oberste Schlüssel einer Dimension (der Wurzel-Schlüssel) wird normalerweise nicht angebunden (so dass der SQL-Generator die Dimension bei Auslesen weder filtert noch selektiert und alle anderen Dimension komplett über diese aggregiert).

Das beides kann am elegantesten durch die Verwendung von Attributen bei der Anbindung erreicht werden. Anstatt eine komplette Hierarchie an einen SQL-Ausdruck zu binden werden dann nur Schlüssel mit einem bestimmten Attribut (dessen Name in dieser Eigenschaft angegeben wird) angebunden. Der SQL-Generator verbindet dann nur die Schlüssel mit der SQL-Expression, die auch dieses Attribut besitzen und sucht diese Schlüssel dann über das Attribut anstatt über die ID (es ist sehr sinnvoll, technische IDs für Schlüssel einzuführen).

So könnten in einer Dimension "Produkt" z.B. alle Schlüssel der Ebene 1 (in der z.B. die Produktgruppen liegen) ein Attribut "GroupID" und alle der Ebene 2 (in der die eigentlichen Produkte liegen) eine "ProductID" besitzen. Beide Attribute können dann separat an die Datenbank gebunden werden.

Wenn mehrere Anbindungen für eine Dimension in einem SQL-Cube existieren, jede für ein anderes Attribut, dann wird der SQL-Generator ggf. mehrere Statements erzeugen und absenden, jeweils für den Teil der Dimension der betroffen ist.

Beispiele

```
Attribute = "ProductID":
```

Binde nur Schlüssel mit einem existierenden Attribut names "ProductID" an (z.B. an einen SQL-Ausdruck SALES.PRODUCT_ID)

Siehe auch

Dimension (auf Seite 305)

Dimension

Typ

Konstante Zeichenkette (Name der Dimension)

Seit

1.0

Beschreibung

Diese notwendige Eigenschaft bestimmt (über ihren Namen) die Dimension der Spalten-Anbindung.

Dimensionen können auf zwei verschiedene Arten an eine Spalte gebunden werden, entweder über ein Attribut oder direkt. Lesen Sie die Beschreibung der Eigenschaft Attribute für eine genauere Beschreibung dieser Art der Anbindung. Wenn kein Attribut verwendet wird, dann wird die Spalte direkt mit dieser Dimension verbunden und die Ergebnisse der SQL-Expression müssen genau mit den IDs der Dimension übereinstimmen.

Beispiele

```
Dimension = "Product"
```

Siehe auch

Attribute (auf Seite 304)

Format

Typ

Format (siehe "Formate" auf Seite 435)

Seit

1.2

Beschreibung

Die Eigenschaft "Format" erlaubt die Formatierung des SQL-Ergebnisses, nach dem es aus der Datenbank gelesen wurde und bevor damit der entsprechende Schlüssel in der Dimension gesucht wird. Abhängig vom Typ der SQL-Expression muss dies ein Zahlen- oder Datums-Format sein.

Beispiele

```
Format = "0"
```

```
Format = "dd.MM.yyyy"
```

Null-ID

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Diese Eigenschaft definiert, welcher Schlüssel (d.h. welche ID bzw. welcher Attribut-Wert) angebunden wird, wenn die SQL-Expression NULL ergibt. Wenn keine Null-ID definiert wird und die Expression NULL ergibt, dann bleibt der Wert ungebunden.

Beispiele

```
Null-ID = "UnknownProduct"
```

Omit-Percentage

Typ

Double-Konstante (Werte zwischen 0.0 und 1.0)

Seit

1.2

Beschreibung

Die Eigenschaft "Omit-Percentage" wird vom SQL-Generator für die Optimierung der generierten Statements verwendet. Sie kontrolliert, ob er eine WHERE Clause im Statement erzeugt, die die gesuchten Schlüssel einschränkt oder ob Werte für alle Schlüssel einer Dimension eingelesen und nachträglich gefiltert werden.

Das Hinzufügen von Filtern zu SQL-Statement (normalerweise als IN-Listen) ist sehr schnell, wenn nur eine bestimmte Teilmenge der Schlüssel gefiltert werden soll, kann aber sehr lange Statements erzeugen, wenn viele Schlüssel gefiltert werden. Im schlimmsten Fall müssen sogar mehrere Statements erzeugt und abgesetzt werden. Das Weglassen eines Filters würde in diesem Fall ein kürzeres Statement erzeugen, bei dem aber mehr Daten als benötigt geladen werden.

Diese Eigenschaft bestimmt die Prozentzahl von Schlüssel in einer Dimension, ab dem der Generator den Filter wegfällen lässt. Der Standard-Wert ist 0.75 (das entspricht 75%). Wenn Sie z.B. 8 oder mehr Schlüssel aus einer Dimension mit 10 Schlüssel laden, dann wird der Filter weggelassen. Bei weniger wird er generiert.

Diese Eigenschaft kann ggf. auch zur Optimierung von Cubes dienen, da Statements ohne Filter schneller ausgeführt werden können, wenn in der Tabelle kein Index für die gefilterte Spalte existiert.

Beispiele

Stellen Sie sich eine Dimension mit 10 Schlüssel (z.B. Produkten) vor. Jedes Produkt ist mit seinem Attribut "ProduktID" an die SQL-Expression "SALES.PRODUCT_ID" angebunden. Der Benutzer fragt 8 dieser 10 Produkte ab. Ist Eigenschaft auf "0.9" gesetzt, dann erzeugt der Generator folgendes Statement:

```
Omit-Percentage = "0.9":
```

```
SELECT SUM( AMOUNT ), PRODUCT_ID FROM SALES WHERE  
PRODUCT_ID IN ( 1, 2, 3, 4, 5, 6, 7, 8 )
```

Alle selektieren Produkte werden im Filter aufgelistet und die Datenbank gibt nur Werte für diese 8 Produkte zurück.

Wenn die Eigenschaft einen Wert kleiner als 0.8 enthält, dann läßt der Generator den Filter wegfallen:

```
Omit-Percentage = "0.7":
```

```
SELECT SUM( AMOUNT ), PRODUCT_ID FROM SALES
```

Dies lädt die Werte für alle Produkte aus der Tabelle. Der Cube ignoriert aber die nicht gewünschten Produkte beim Einlesen des Ergebnisses. Da in diesem Fall kein Filter auf die Tabelle angewendet wird, kann die Ausführung des Statements ggf. viel schneller sein.

SQL-Expression

Typ

SQL-Ausdruck (siehe "SQL-Ausdrücke" auf Seite 441)

Seit

1.0

Beschreibung

Diese Eigenschaft enthält den SQL-Ausdruck an dem die Dimension (oder ein Attribut der Dimension) gebunden ist. Der SQL-Generator wird diesen Ausdruck dazu verwenden, um die Schlüssel aus der Kennzahl-Tabelle auszulesen und um nach ihnen zu gruppieren und zu filtern. Abhängig von der Art der Dimensions-Anbindung (direkt oder über ein Attribut) muss der Ausdruck Werte ergeben, die entweder den IDs der Dimension-Schlüssel oder dem entsprechenden Attribut entsprechen. Lesen Sie hierzu auch die Beschreibung der Eigenschaften Dimension und Attribute.

Beispiele

```
SQL-Expression = "SALES.PRODUCT_ID"
```

Siehe auch

Attribute (auf Seite 304), Dimension (auf Seite 305)

CSV-Cube

Complete

Typ

Konstanter Boolean-Wert (true oder false)

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt, ob ein Cube als vollständig oder unvollständig gilt. Wenn eine Wert theoretisch aus einem vollständigen Cube geladen werden kann, dann wird aus keinem anderen geladen, auch wenn der Wert nicht gefunden wurde. Eine genauere Beschreibung zu vollständigen und unvollständigen Cubes finden Sie in der Cube-Beschreibung. In der Standard-Einstellung sind Cubes vollständig.

Beispiele

Eine Datensuche mit der Eigenschaft "Complete", die auf "false" gesetzt wurde, könnte wie folgt ablaufen:

- 1 Die Daten werden im 1. Würfel (mit Complete = false) gesucht und nicht gefunden
- 2 Die Daten werden im 2. Würfel (mit Complete = false) gesucht und nicht gefunden
- 3 Die Daten werden über eine passende Formel berechnet

Bei den gleichen Würfeln, in denen jedoch "Complete" auf "true" gesetzt wurde, könnte es stattdessen wie folgt ablaufen:

- 1 Die Daten werden im 1. Würfel (mit Complete = true) gesucht und nicht gefunden
- 2 Die Suche wird aufgegeben und der Wert auf NULL gesetzt

Siehe auch

Span All Dimensions (auf Seite 311)

CSV-Source

Typ

Konstante Zeichenkette (Name der Datenquelle)

Seit

1.0

Beschreibung

Für einen CSV-Cube müssen Sie die CSV-Datenquelle (CSV-Source) angeben, aus dem die Daten geladen werden sollen. Geben Sie den Namen der Datenquelle dazu in dieser Eigenschaft an. Der Name muss exakt dem Namen der CSV-Source entsprechen.

Beispiele

```
CSV-Source = "Sales"
```

Line-Dimension

Typ

Konstante Zeichenkette (Name der Dimension)

Seit

2.0

Beschreibung

Diese Eigenschaft setzt die Line-Dimension für diesen CSV-Cube. Die Line-Dimension wird dazu verwendet, um Daten aus einem Cube in Listenform (und nicht aggregiert) auszulesen. Eine detaillierte Beschreibung zum Thema Line-Dimension finden Sie in der Cube-Dokumentation.

Als Wert müssen Sie den genauen Namen der (zuvor definierten) Dimension angeben, in der die Zeilen-Nummern (von 1 aufsteigend) als Element angelegt wurden.

Eine solche Dimension kann mit Hilfe eines Number-Keyloaders generiert werden, existiert aber bereits auch schon als Vorgabe im System. Wenn Sie hier keine Dimension angeben, wird diese Vorgabe-Dimension auch verwendet. Es wird empfohlen, besonders im Hinblick auf die Kompatibilität mit zukünftigen Versionen, die Verwendung von eigenene Line-Dimensions zu vermeiden.

Beispiele

```
Line-Dimension = "Line"
```

Match

Typ

Boolean

Seit

1.2

Beschreibung

Für jeden Cube kann optional eine Filter-Bedingung (Match-Condition) definiert werden, durch die bestimmt wird welche Daten aus einem Cube gelesen werden sollen. Diese Bedingung für alle Koordinaten der zu ladenden Werte überprüft, die Koordinate auf die diese Bedingung zutrifft werden dann ggf. aus dem Cube geladen. Weitere Informationen zur Match-Condition finden Sie in der Cube-Dokumentation.

Beispiele

```
Match = "ISCHILDOP( Time, Time:'2004 )"
```

SpanAll Dimensions

Typ

Konstanter Boolean-Wert (true oder false)

Seit

2.0

Beschreibung

Diese Eigenschaft definiert, ob ein Cube als "Hypercube" fungiert und alle Dimensionen des Modells umspannt oder ob er (als sogenannter "Multicube") nur einen Teil der Dimensionen umspannt (das ist die Standard-Einstellung). Weitere Informationen zu Hypercubes und Multicubes finden Sie in der Cube-Dokumentation.

Beispiele

```
Span All Dimensions = "true"
```

Span All Dimensions = "false"

Siehe auch

Complete (auf Seite 309)

CSV-Fact

Column

Typ

Konstante Zeichenkette (Name der Spalte)

Seit

1.2

Beschreibung

Geben Sie in dieser Eigenschaft den Namen der Spalte an, der die Kennzahl enthält.

Beispiele

Column = "Sales"

Fact

Typ

Konstante Zeichenkette (ID der Kennzahl)

Seit

1.0

Beschreibung

Geben Sie in dieser Eigenschaft den Namen der Kennzahl an, die Sie aus der entsprechenden Spalte der CSV-Datei lesen möchten.

Beispiele

Fact = "Amount"

Match

Typ

Boolean

Seit

1.2

Beschreibung

Für jede Anbindung einer Kennzahl können Sie eine optionale Bedingung bestimmen. Eine Kennzahl wird dann nur an die CSV-Spalte gebunden, wenn diese Boolean-Expression (die u.A. von alle Dimensionen abhängig sein kann) "true" ergibt.

Über die Match-Expression ist es möglich, mehrere CSV-Spalten an die selbe Kennzahl zu binden, jeweils für einen anderen Teil des Cubes (z.B. können Sie für jeden Monat eine andere Spalte auslesen).

Beispiele

```
Match = "HASKEYS( Time:'2004' )"
```

CSV-Dimension

Attribute

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Wenn diese Eigenschaft einen Attribut-Namen enthält (und nicht leer gelassen wird), dann wird der Inhalt der CSV-Spalte über dieses Attribute and die Dimension angebunden. Für jeden Wert aus der Spalte sucht das System nach dem Dimensions-Schlüssel, bei dem dieses Attribut mit dem Wert aus der Spalte übereinstimmt. Wenn kein Attribut angegeben wird, dann sucht das System direkt nach dem Schlüssel mit der übereinstimmenden ID.

Es können nur eindeutige Attribute an Cubes angebunden werden.

Beispiele

```
Attribute = "ProductID"
```

Siehe auch

Dimension (auf Seite 316)

Column

Typ

Konstante Zeichenkette (Name der Spalte)

Seit

1.0

Beschreibung

Diese Eigenschaft enthält den Namen der CSV-Spalte, an den die Dimension angebunden werden soll. Abhängig von der Art der Anbindung (direkt an die Dimension oder über ein Attribut) müssen die Werte in der Spalte den IDs der Dimensions-Schlüssel oder dem gewählten Attribut entsprechen. Lesen Sie auch die Beschreibungen der Eigenschaften Dimension und Attribute zu diesem Thema.

Beispiele

```
Column = "ProductID"
```

Siehe auch

Attribute (auf Seite 315), Dimension (auf Seite 316)

Dimension

Typ

Konstante Zeichenkette (Name der Dimension)

Seit

1.2

Beschreibung

Diese notwendige Eigenschaft bestimmt (über ihren Namen) die Dimension der Spalten-Anbindung.

Dimensionen können auf zwei verschiedene Arten an eine Spalte gebunden werden, entweder über ein Attribut oder direkt. Lesen Sie die Beschreibung der Eigenschaft Attribute für eine genauere Beschreibung dieser Art der Anbindung. Wenn kein Attribut verwendet wird, dann wird die Spalte direkt mit dieser Dimension verbunden und die Werte in der Spalte müssen genau mit den IDs der Dimension übereinstimmen.

Beispiele

```
Dimension = "Product"
```

Siehe auch

Attribute (auf Seite 315)

Format

Typ

Format (siehe "Formate" auf Seite 435)

Seit

1.2

Beschreibung

Die Eigenschaft "Format" ermöglicht das Formatieren des Spalteninhalten nachdem er aus der Datei ausgelesen und bevor er zur Suche eines Schlüssels in einer Dimension verwendet wird. Abhängig vom Typ der CSV-Spalte (der in der CSV-Source definiert wird) muss das Format entweder ein Zahlen- oder Datums-Format sein.

Beispiele

```
Format = "0"
```

```
Format = "dd.MM.yyyy"
```

Null-Value

Typ

Konstante Zeichenkette

Seit

2.0

Beschreibung

Mit dieser Eigenschaft wird definiert, welcher Schlüssel (mit welcher ID) an die Spalte gebunden wird, wenn die Spalte den Wert NULL enthält, also leer ist. Wenn keine Null-ID angegeben wird, dann werden leere Spalte ignoriert und bleiben ungebunden.

Beispiele

```
Null-ID = "UnknownProduct"
```

Siehe auch

Column (auf Seite 315)

Formel

Expression

Typ

Value

Seit

1.1

Beschreibung

Diese Eigenschaft bestimmt den Ausdruck dieser Formel. Dieser Ausdruck ist die eigentlicher Teil der Formeln und berechnet ihr Ergebnis. Der Ausdruck muss vom Typ "Value" sein, darf also Werte aller Art aber keine Dimensions-Schlüssel zurückgeben.

Der Ausdruck kann jede andere Kennzahl und alle Dimensionen verwenden. Formeln können z.B. verwendet werden, um Aggregationen auszuführen (wenn die Cubes das nicht selber können) oder um fehlende Kennzahlen zu berechnen, die von keinem Cube geliefert werden.

Beispiele

```
Expression = "Quantity() / Price()":
```

Berechne die Kennzahl "Amount" aus zwei anderen Kennzahlen

```
Expression = "SUM( Quantity( CHILDREN( Time ) ) )":
```

Summiere die Kennzahl "Quantity" aus der nächsten Zeit-Ebene

Fact

Typ

Konstante Zeichenkette (ID der Kennzahl)

Seit

2.0

Beschreibung

Jede Formel berechnet eine bestimmte Kennzahl. Diese Eigenschaft muss den Namen der Kennzahl enthalten, die von dieser Formel berechnet wird.

Beispiele

Fact = "Amount" --> Diese Formel berechnet die Kennzahl "Amount"

Siehe auch

Match (auf Seite 319)

Match

Typ

Boolean

Seit

1.2

Beschreibung

Mit der Eigenschaft Fact können Sie bestimmen, welche Kennzahl von dieser Formel berechnet wird. Wenn Sie nur die Kennzahl bestimmen, dann wird diese für den gesamten Cube (für alle Schlüssel aller Dimensionen) berechnet. Wenn Sie die Kennzahl aber nur für einen Teil berechnen lassen möchten (z.B. nur für eine bestimmte Ebene einer Dimension), dann können Sie eine Match-Expression für diese Formel in dieser Eigenschaft bestimmen.

Die Match-Expression schränkt die Verwendung der Formel auf alle Koordinaten ein, für die diese Expression "true" ergibt (die Match-Expression ist ein Ausdruck vom Typ Boolean). Sie könnte z.B. die aktuelle Koordinate auf das Vorkommen eines bestimmten Schlüssels testen. Match-Expressions für Formeln funktionieren ähnlich wie die von Cubes oder Caches.

Beispiele

Match = "HASLEVEL(Time, 1)":

Berechne nur die erste Ebene der Dimension "Zeit"

Match = "HASKEYS(Product:A)":

Berechne nur für das Produkt "A"

Siehe auch

Fact (auf Seite 318)

Memory-Cache

Cron-Pattem

Typ

Cron-Pattern (siehe "Cron-Patterns" auf Seite 436)

Seit

2.0

Beschreibung

Memory-Caches werden regelmäßig vom System auf veraltete Einträge (die älter sind als die Zeitspanne, die über die Eigenschaft Max Age eingestellt wird) und löscht diese. Da dies eine zeitaufwendige Aufgabe sein kann, ist es möglich, die Überprüfung zeitlich zu steuern und mehr oder weniger oft vom System durchführen zu lassen, indem man ein Cron-Pattern in dieser Eigenschaft definiert.

Die Verwendung des Cron-Patterns gibt einem ausserdem die Möglichkeit, den Cache zu einem bestimmten Zeitpunkt komplett löschen zu lassen, indem man die Eigenschaft Max Age auf 0 setzt. Z.B. würde ein Cron-Pattern "* 0 0" kombiniert mit einer Max Age von "0" den Cache um 00:00 jede Nacht löschen.

Beispiele

```
Cron = "* 0 0":
```

Überprüfe the Memory-Cache jede Nacht um 00:00 auf veraltete Einträge

Match

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft leer gelassen wird, dann speichert der Cache alle Einträge (d.h. alle Kennzahlen für alle Koordinaten). Wenn Sie nur einen Teil der Daten speichern möchten (z.B. eine bestimmte Kennzahl oder alle Einträge die älter als einen Monat sind), dann können Sie hier eine Boolean-Expression angeben, die bestimmt, welche Daten von diesem Cache gespeichert werden.

Die Boolean-Expression muss "true" für alle akzeptierten Koordinaten zurückgeben. Sie können dazu u.A. die Funktionen HASKEYS oder HASLEVEL verwenden.

Beispiele

```
Match = "HASKEYS( Fact:Amount )":
```

Nur die Kennzahl "Amount" speichern

```
Match = "EXISTS( NEXT( Time ) )":
```

Speichert nicht das aktuelle Jahr, den aktuellen Monat, Tag etc.

MaxAge

Typ

Integer-Konstante (Anzahl Sekunden)

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt das maximale Alter der Cache-Einträge in Sekunden. Jeder Eintrag dieses Caches wird automatisch gelöscht, wenn er älter als das hier definierte Maximalalter geworden ist. Zusätzlich zum maximalen Alter können Sie ausserdem bestimmen, wann und wie oft die Überprüfung auf veraltete Einträge vorgenommen werden soll. Verwenden Sie dazu die Eigenschaft Cron-Pattern.

Wenn das maximale Alter leer bleibt, dann werden die Einträge niemals veralten und gelöscht. Wenn das maximale Alter auf "0" gesetzt wird, dann veralten die Einträge sofort und werden bei der nächsten Überprüfung komplett gelöscht (das macht nur Sinn, wenn Sie die Frequenz der Überprüfung mit der Eigenschaft Cron-Pattern definieren).

Beispiele

```
Max Age = "900" :
```

Das maximale Alter der Einträge beträgt 15 Minuten

Siehe auch

Cron-Pattern (auf Seite 321), Max Size (auf Seite 323)

Max Size

Typ

Integer-Konstante (Anzahl Einträge)

Seit

1.0

Beschreibung

Jeder Memory-Cache besitzt eine Grössenbegrenzung (die Anzahl der Einträge, die vom Cache gespeichert werden können). Verwenden Sie diese Eigenschaft, um die Grösse zu bestimmen.

Wenn Sie die Grösse des Caches setzen sollten Sie immer den Speicherverbrauch bedenken. Für jeden Eintrag muss der Wert selbst (meistens 8 Bytes) und die Koordinate des Wertes gespeichert werden. Die Koordinate benötigt 4 Bytes je Dimension. Wenn Zeichenketten gespeichert werden benötigt der Cache entsprechend mehr Speicher.

$(\text{Anzahl der Dimensionen} * 4 + 8) * \text{Grösse des Caches}$

Z.B. würde ein Cache mit einer maximalen Grösse von 5000 Einträgen in einem 10-dimensionalen Modell ca. 240 KB benötigen.

Beispiele

```
Max Size = "5000"
```

Siehe auch

Max Age (auf Seite 322)

File-Cache

Cron-Pattem

Typ

Cron-Pattern (siehe "Cron-Patterns" auf Seite 436)

Seit

2.0

Beschreibung

Filecaches werden regelmäßig vom System auf veraltete Einträge (die älter sind als die Zeitspanne, die über die Eigenschaft Max Age eingestellt wird) und löscht diese. Da dies eine zeitaufwendige Aufgabe sein kann, ist es möglich, die Überprüfung zeitlich zu steuern und mehr oder weniger oft vom System durchführen zu lassen, indem man ein Cron-Pattern in dieser Eigenschaft definiert.

Die Verwendung des Cron-Patterns gibt einem ausserdem die Möglichkeit, den Filecache zu einem bestimmten Zeitpunkt komplett löschen zu lassen, indem man die Eigenschaft Max Age auf 0 setzt. Z.B. würde ein Cron-Pattern "* 0 0" kombiniert mit einer Max Age von "0" den Cache um 00:00 jede Nacht löschen.

Beispiele

```
Cron = "* 0 0":
```

Überprüft den Filecache jede Nacht um 00:00 auf veraltete Einträge

Filename

Typ

Konstante Zeichenkette

Seit

1.2

Beschreibung

Jeder Filecache muss einen eindeutigen Dateinamen besitzen, der über diese Eigenschaft definiert wird. Die Dateinamen von Filecache sind relativ, ohne Pfadangaben oder Datei-Endung (alle Filecaches werden im Datenverzeichnis von instantOLAP gespeichert, deshalb besteht keine Notwendigkeit für Pfadangaben).

Beispiele

```
Filename = "salescache1":
```

Dies wird einen Filecache names "salescache1.db" im Datenverzeichnis von instantOLAP (bzw. in einem Unterverzeichnis davon) erstellen.

Match

Typ

Boolean

Seit

1.2

Beschreibung

Wenn diese Eigenschaft leer gelassen wird, dann speichert der Cache alle Einträge (d.h. alle Kennzahlen für alle Koordinaten). Wenn Sie nur einen Teil der Daten speichern möchten (z.B. eine bestimmte Kennzahl oder alle Einträge die älter als einen Monat sind), dann können Sie hier eine Boolean-Expression angeben, die bestimmt, welche Daten von diesem Cache gespeichert werden.

Die Boolean-Expression muss "true" für alle akzeptierten Koordinaten zurückgeben. Sie können dazu u.A. die Funktionen HASKEYS oder HASLEVEL verwenden.

Beispiele

```
Match = "HASKEYS( Fact:Amount )":
```

Speichert nur die Kennzahl "Amount"

```
Match = "EXISTS( NEXT( Time ) )":
```

Speichert nicht das aktuelle Jahr, den aktuellen Monat, Tag etc.

MaxAge

Typ

Integer-Konstante (Anzahl Sekunden)

Seit

1.2

Beschreibung

Diese Eigenschaft bestimmt das maximale Alter der Cache-Einträge in Sekunden. Jeder Eintrag dieses Caches wird automatisch gelöscht, wenn er älter als das hier definierte Maximalalter geworden ist. Zusätzlich zum maximalen Alter können Sie ausserdem bestimmen, wann und wie oft die Überprüfung auf veraltete Einträge vorgenommen werden soll. Verwenden Sie dazu die Eigenschaft Cron-Pattern.

Wenn das maximale Alter leer bleibt, dann werden die Einträge niemals veralten und gelöscht. Wenn das maximale Alter auf "0" gesetzt wird, dann veralten die Einträge sofort und werden bei der nächsten Überprüfung komplett gelöscht (das macht nur Sinn, wenn Sie die Frequenz der Überprüfung mit der Eigenschaft Cron-Pattern definieren).

Beispiele

Max Age = "900":

Das maximale Alter wird auf 15 Minuten gesetzt

Siehe auch

Cron-Pattern (auf Seite 324)

Include

Model

Typ

Konstante Zeichenkette (Name des Modells)

Seit

2.0

Beschreibung

Wenn Sie ein anderes Modell in Ihr Modell importieren möchten, dann müssen Sie den Namen des anderen Modells in dieser Eigenschaft angeben. Beachten Sie, dass Sie hier den Namen des Modells und nicht den Dateinamen der Konfiguration angeben müssen (hängen Sie also keine Endung `.config` an den Modell-Namen).

Beispiele

```
Model = "general/DefaultModel"
```


KAPITEL 5

Die Formelsprache

In diesem Kapitel

Das Typsystem	330
Syntax.....	333
Konstanten	342
Funktionen.....	346

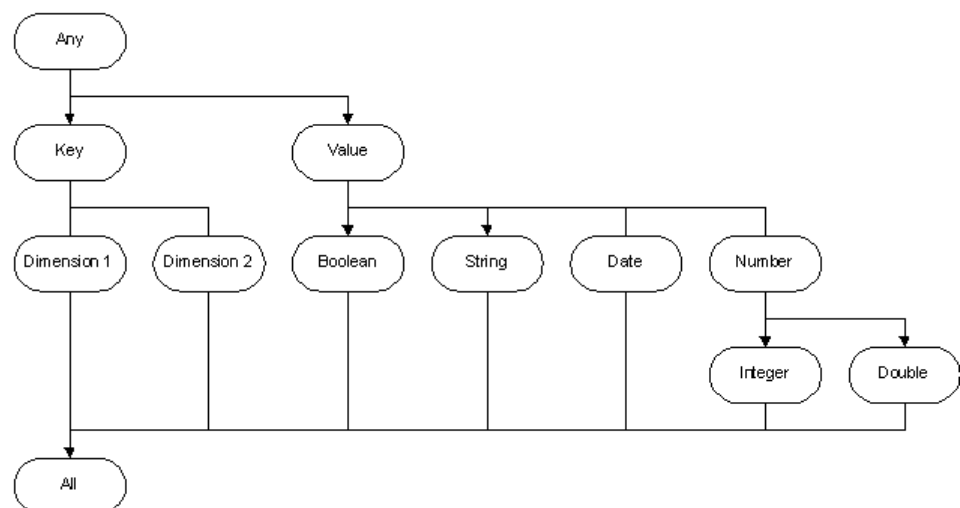
Das Typsystem

Ab der Version 2.0 besitzt instantOLAP ein Typsystem wie in Programmiersprachen üblich, in dem alle Werte einem bestimmten Typ angehören. Durch das Typsystem ist instantOLAP in der Lage, bessere Syntax-Checks durchzuführen und genauere Fehlermeldungen zu geben. Ausserdem können Funktionen jetzt besser miteinander kombiniert werden, da das Typsystem jetzt kontrolliert, welche Funktion wieder als Parameter für andere Funktionen dienen kann.

Das Typsystem ist hierarchisch aufgebaut, mit Typen, Subtypen und Supertypen. Einige Typen sind instanzierbar, d.h. es existieren Werte, die genau diesen besitzen. Andere sind nur "Hilfstopen", die zur Gruppierung von mehreren Typen zu einem Supertypen dienen, von denen es aber nie Instanzen geben kann. Ein solcher Hilfstyp ist z.B. "Value".

Ein Supertyp eines Typs ist ein anderer Typ, der in der Hierarchie oberhalb eines Typs liegt. "Value" ist z.B. ein Supertyp von "Integer". Ein Subtyp ist synonym dazu ein Typ, der unterhalb liegt. Wenn eine Funktion einen Parameter vom Typ X erwartet, dann können Sie hier Werte übergeben die entweder genau vom erwarteten Typ oder vom einem Subtyp davon sind. Eine Funktion, die z.B. einen Parameter vom Typ "Number" erwartet, würde als auch Werte vom Typ "Integer" oder "Double" akzeptieren.

Dies ist das gesamte Typsystem mit allen in instantOLAP existierenden Typen:



Dimensionen werden automatisch zu eigenen Typen und Teil des Typsystems: Jede Dimension ist ein Subtyp von "Key", d.h. alle Funktionen, die Parameter vom Typ "Key" erwarten, akzeptieren automatisch Schlüssel jeder Dimension.

All

Der Typ "All" ist ein sehr spezieller Hilfstyp, der Subtyp von allen anderen Typen ist. Der einzige Wert, der den Typ "All" besitzt, ist NULL. Da "All" Subtyp aller anderen Typen ist, können Sie NULL bei jeder Funktion und jedem Parameter als Wert übergeben!

NULL steht für einen undefinierten Wert.

Any

Der Typ "Any" ist der Basistyp, der automatisch der Supertyp aller anderen Typen ist, da er ganz oben in der Typ-Hierarchie liegt. Wenn eine Funktion einen Parameter vom Typ "Any" erwartet, dann können Sie hier jeden möglichen Wert als Parameter übergeben, da jeder Typ Subtyp von "Any" ist.

Vom Typ selbst kann es keine Instanzen geben, d.h. es gibt keinen Wert, der direkt den Typ "Any" besitzt.

Boolean

Werte vom Typ "Boolean" repräsentieren logische Werte, also "wahr" oder "falsch". Es gibt dementsprechend nur zwei verschiedene Werte, die den Typ "Boolean" besitzen: TRUE und FALSE. "Boolean" ist ein Subtyp von Value und besitzt keine eigenen Subtypen.

Double

Alle reellen Zahlen in instantOLAP haben den Typ "Double", der ein direkter Subtyp von Number ist. D.h. jede Funktion, die einen Parameter vom Typ "Number" erwartet, akzeptiert auch Reelle Zahlen.

Integer

Alle ganzen Zahlen in instantOLAP haben den Typ "Integer", der ein direkter Subtyp von "Number" ist. D.h. jede Funktion, die einen Parameter vom Typ "Number" erwartet, akzeptiert auch Integer-Zahlen.

Key

Alle Elemente einer Dimension (die Schlüssel) besitzen den Typ "Key" (genauer gesagt bildet jede Dimension einen eigenen Typ, der automatisch Subtyp von "Key" ist, und die Schlüssel einer Dimension habe ihre eigene Dimension als Typ). Schlüssel gelten nicht als Werte, deswegen ist dieser Typ ein direkter Subtyp von Any und nicht von Value.

Number

Der Typ "Number" ist ein weiterer Hilfstyp, der die numerischen Typen "Integer" und "Double" zu einer Gruppe zusammenfasst. "Number" ist Subtyp von Value und besitzt diese zwei Typen als Subtypen. Von "Number" selbst kann es keine Instanzen geben.

String

Zeichenketten besitzen in instantOLAP den Typ "String". Dieser Typ ist ein direkter Subtyp von Value und besitzt keine eigenen Subtypen.

Value

Der Typ "Value" ist ein weiterer Hilfstyp, der alle Typen (Zahlen, Datum, Booleans etc.) bis auf "Key" zu einer Gruppe zusammenfasst. Es gibt keine Werte, die direkt dem Typ "Value" besitzen.

Syntax

Dimensionen und Selektionen

Syntax

`<dimension-syntax> := <dimension-name>`

Ergebnis-Typ

Key

Beschreibung

Mit dem Dimensions-Namen selbst kann man auf eine Dimension bzw. auf die aktuelle Selektion (den Filter) einer Dimension zugreifen. Im Filter werden je Dimension keiner, einer oder mehrere Schlüssel vermerkt. Durch den Dimensions-Namen kann man genau auf den Teil des aktuellen Filters zugreifen, der für die entsprechende Dimension gilt.

Der Filter wird durch verschiedene Elemente eines Berichtes beeinflusst, u.A.:

- durch die URL eines Berichtes
- die Selektoren
- den Filter für Berichte, Blöcke, Pivot-Tabellen und Überschriften
- den Iterationen der Blöcke und Überschriften

Beispiele

Time

Aktuelle Selektion der Dimension "Time"

Dimensions-Ebenen

Syntax

`<level-expression> := <dimension-name> '::' <level-name>`

`<level-expression> := <level-name>`

Ergebnis-Typ

Key

Beschreibung

Wie beim Zugriff auf Dimensionen und Selektionen können Sie auch gezielt auf die gefilterten Schlüssel einer bestimmten Ebene einer Dimension zugreifen. Im Gegensatz zu den Dimensionen erhalten Sie hier aber nicht direkt die Schlüssel der Dimension aus dem Filter, sondern alle darüber oder darunter liegenden Schlüssel, abhängig davon, ob diese Ebene oberhalb oder unterhalb der selektierten Schlüssel liegt.

Beispiele

Wenn z.B. der aktuelle Filter den Schlüssel "Feb/2004" für die Dimension "Zeit" enthält, dann ergibt

```
Zeit::Jahr
```

Ergibt z.B. "2004"

```
Zeit::Tag
```

Ergibt z.B. "01.02.2004 + 02.02.2004 + 03.02.2004 + ... + 29.02.2004"

Siehe auch

Dimensionen und Selektionen (auf Seite 333), LEVEL (auf Seite 392)

Operatoren

Verwendung von Operatoren

Einige Funktionen können auch über einen Operator anstelle über die übliche Funktions-Schreibweise aufgerufen werden. Für einige wenige Operatoren gibt es sogar keine Funktion als Gegenstück.

Funktionen besitzen eine Priorität, die die Reihenfolge der Ausführung innerhalb eines Ausdrucks regelt. Das System berechnet als erstes den Operator mit der höchsten Priorität bis runter zu dem Operator mit der kleinsten. Wenn zwei Operatoren die gleiche Priorität besitzen, werden sie von links nach rechts berechnet.

Wenn Sie abweichend von dieser Regel eine andere Reihenfolge der Ausführung festlegen möchten, dann müssen Sie dazu Ihren Ausdruck entsprechend mit Klammer versehen.

Liste der Operatoren

Op.	Pri	Funktion	Beispiel
-----	-----	----------	----------

:	6	Key-operator	Time:'2004'
::	6	Level-operator	Time::Month
.	5	Attribute-Operator	Product.color
[]	5	CUBE (auf Seite 359)	[CHILDREN(Product)]
{ }	5	TOSTRING (auf Seite 423)	{Product}
*	4	MUL (auf Seite 402)	10 * 20
/	4	DIV (auf Seite 365)	10 / 20
%	4	MOD (auf Seite 402)	20 % 10
+	3	ADD (auf Seite 347)	10 + 20
-	3	SUB (auf Seite 419)	10 - 20
	3	JOIN (auf Seite 386)	Product:ProductA Product:ProductB
IN	3	IN (auf Seite 383)	Product:ProductA IN Product
<	2	LESS (auf Seite 390)	10 < 20
>	2	GREATER (auf Seite 376)	10 > 20
=	2	EQUAL (auf Seite 369)	10 = 20
<=	2	LESS_OR_EQUAL (auf Seite 391)	10 <= 20
>=	2	GREATER_OR_EQUAL (auf Seite 377)	10 >= 20
<>	2	UNEQUAL (auf Seite 425)	10 <> 20
?	2	MATCH (auf Seite 396)	Product ? Product.color = 'red'

AND	1	AND (auf Seite 350)	Product.color = 'red' and Amount() > 0
OR	1	OR (auf Seite 408)	Product.color = 'red' or Amount() > 0

Siehe auch

Klammern (auf Seite 336), Funktionsaufrufe (auf Seite 340)

Klammern

Syntax

```
<bracket-expression> := '(' <expression> ')'
```

Beschreibung

Mit Klammern können Sie die Ausführungs-Reihenfolge Ihrer Ausdrücke ändern. Im Normalfall werden Ausdrücke nach der Priorität der in ihr enthaltenen Operatoren, beginnend mit der höchsten, ausgeführt. Operatoren mit der selben Priorität werden von links nach rechts ausgeführt. Durch das Setzen von Klammer zwingen Sie das System dazu, den Inhalt der Klammer zuerst zu berechnen, bevor das Ergebnis der Klammern mit den Rest des Ausdrucks berechnet wird. Sie können Klammern beliebig kombinieren und verschachteln - dann werden die Klammer von innen nach aussen berechnet.

Beachten Sie, dass Funktionen sich wie Klammern verhalten: Alle Parameter eines Funktionsaufrufs werden zuerst berechnet (von links nach rechts), danach wird die Funktion selbst ausgeführt und das Ergebnis der Funktion wird zur Berechnung des restlichen Ausdrucks verwendet. Wenn Sie verschiedene Funktionsaufrufe ineinander verschachteln (d.h. eine Funktion als Parameter für eine andere Funktion verwenden), dann werden wie bei Klammern die innen liegenden Funktionen zuerst berechnet.

Beispiele

```
( 10 + 20 ) * 30 --> 900
```

```
10 + 20 * 30 --> 610
```

Siehe auch

Funktionsaufrufe (auf Seite 340), Operatoren (auf Seite 334)

Zugriff auf Attribute

Syntax

```
<attribute-expression> := <key-expression> '.' <attribute-name>
```

Ergebnis-Typ

Entspricht dem Typ des Attributs.

Beschreibung

Jedes Element einer Dimension (Schlüssel bzw. Keys) kann kein, eins oder mehrere Attribute besitzen. Attribute können einfache Werte (z.B. Zeichenketten, Zahlen oder Logische Werte) sein oder wiederum Elemente (Keys) einer anderen Dimension. Wenn ein Attribut den Typ Key besitzt, dann nennt man diese Verbindung zwischen zwei Dimensionen einen "Link".

Um auf ein Attribut zuzugreifen müssen Sie unmittelbar hinter einem Ausdruck vom Typ Key (Attribute können nur aus Schlüssel ermittelt werden) den Punkt-Operator "." verwenden. Der Ergebnis-Typ des Operators entspricht dem Typ des Attributes. Wenn z.B. ein Attribute den Typ String hat, dann ist der Ergebnis-Typ des Operators ebenfalls String. Wenn der Typ des Attributs Key ist, können Sie den Attribute-Operator erneut auf das Ergebnis anwenden um weitere Attribute auszulesen.

Wenn der Ausdruck vor dem Operator NULL ergibt, dann ist das Ergebnis des Operators ebenfalls NULL.

Beispiele

```
Product.Color
```

Ergibt die Farbe des aktuellen Produktes

```
Product.Vendor.Name
```

Ergibt den Namen des Herstellers des aktuellen Produktes

Siehe auch

ATTRIBUTENAMES (auf Seite 351), ATTRIBUTES (auf Seite 352)

Zugriff auf Variablen

Syntax

```
<variable-expression> := '$' <variable-name>
```

Ergebnis-Typ

String

Beschreibung

instantOLAP ermöglicht die Definition und Verwendung von Variablen innerhalb von Berichten. Variablen können auf zwei verschiedene Arten und Weisen erzeugt werden:

- Durch das Hinzufügen eines Selektors zum Bericht, der den Benutzer den Wert für eine Variable auswählen lässt
- Durch das Übergeben von beliebigen Parameter über die URL eines Berichtes im Browser

Definition von Variablen in Selektoren

Jeder Selektor generiert automatisch eine Variable mit dem gleichen Namen, den auch der Selektor hat. Ein Selektor beeinflusst in der Regel die aktuelle Selektion (d.h. den Filter) eines Berichtes, da der Benutzer in der Regel Schlüssel über Selektoren auswählt, in Wirklichkeit handelt es sich hierbei aber auch um eine Variable, deren Name dem Namen einer Dimension entspricht (ein Selektor für die Dimension "Zeit" erzeugt z.B. auch eine Variable "Zeit"). Wenn ein Selektor eine Dimension nicht beeinflusst weil der Name mit keinem Namen einer Dimension übereinstimmt, dann wird nur die Variable angelegt (z.B. würde ein Selektor namens "X" eine Variable namens "X" erzeugen aber keine Dimension beeinflussen, wenn keine mit einem solchen Namen existiert).

Übergabe von Variablen in der URL

Wenn ein Bericht von ausserhalb aufgerufen wird (z.B. über einen Link aus einer anderen Webseit heraus), dann können sie weitere Parameter an die URL des Berichtes anhängen. Alle Parameter werden automatisch in Variablen umgewandelt.

Typ und Auswertung von Variablen

Variablen sind immer von Typ String. Wenn Sie eine Variable als einen Wert eines anderen Typ interpretieren möchten, z.B. als Zahl, dann können Sie dazu die EVAL-Funktion verwenden. Die EVAL-Funktion erwartet eine Zeichenkette als Parameter, wertet diese als Ausdruck aus (oder meldet einen Fehler, wenn der Ausdruck nicht korrekt ist) und gibt das Ergebnis dieses Ausdrucks als Ergebnis zurück. So können Sie Variablen auf ein sehr flexible Art und Weise nutzen, z.B. um komplett verschiedene Überschriften in einem Bericht anzeigen und durch den Benutzer auswählen zu lassen.

Listen

Variablen können keinen, einen oder mehrere Werte besitzen, abhängig von der Definition der Variable. Wenn eine Variable z.B. durch einen Selektor vom Typ "Single" definiert wurde, kann sie maximal einen Wert haben. Wenn Sie durch einen "Multiple" Selektor definiert wurde, kann sie auch mehrere Werte besitzen. Wenn Sie eine Variable über die URL eines Berichtes übergeben, dann wird aus jeder Nennung der Variable in der URL ein Wert.

Vordefinierte Variablen

`$COUNTRY`: Die Variable COUNTRY enthält den Länder-Code für den aktuellen Benutzer

`$LANGUAGE`: Die Variable LANGUAGE enthält den Sprach-Code für den aktuellen Benutzer

`$USER`: The USER variable contains the account name of the current user.

Beispiele

`$COUNTRY`

Ergibt z.B. 'US' (d.h. der aktuelle Benutzer verwendet 'US' als Land in seinem Browser)

`$LANGUAGE`

Ergibt z.B. 'de' (der aktuelle Benutzer verwendet Deutsch als Sprache in seinem Browser)

`$USER`

Ergibt z.B. 'admin' (der aktuelle Benutzer heisst 'admin')

`EVAL($X)`

Wertet die Variable X als Ausdruck aus

Siehe auch

EVAL (auf Seite 369)

Funktionsaufrufe

Beschreibung

Wie in Programmiersprachen unterstützt instantOLAP Funktionen zur Berechnung von Werten zur Laufzeit eines Berichtes. Jede Funktion besitzt einen Namen, eine Reihe von erwarteten Parameter und einen bestimmten Ergebnis-Typ. Wenn Sie eine Funktion in einem Ausdruck verwenden, dann wird der Teil mit dem Funktionsaufruf wie ein Ausdruck mit genau diesem Typ behandelt und kann an andere Funktionen als Parameter weitergegeben werden, vorausgesetzt dass die Funktion diesen Typ, oder einen Subtyp davon, als Parameter erwartet.

Um eine Funktion aufzurufen, müssen Sie als ersten den Namen der Funktion verwenden, gefolgt von Klammern, in denen (durch Kommata getrennt) die Parameter übergeben werden. Die Parameter selbst sind wieder Ausdrücke, d.h. hier können Sie wieder Konstanten, Funktionen, Dimensionen etc. verwenden.

Beispiele

```
NEXT( Zeit )
```

```
FIRST( LEVEL( Zeit, 1 ) )
```

Ebenen-Funktionen

Syntax

```
<levelfunction-expression> := <level-name> '(' [ <Key> {  
' , ' <Key> } ] ')'
```

Beschreibung

Für jede Ebene jeder Dimension wird im System automatisch eine Funktion angelegt, mit der Sie die Schlüssel dieser Ebene ermitteln können, die unterhalb oder oberhalb der im aktuellen Filter selektierten Schlüssel der entsprechenden Dimension liegen. Damit lassen sich Funktionen wie "Gib mir die Monate des aktuellen Jahres" oder "Gib mit die Produktgruppen der aktuell selektierten Produkte" realisieren.

Zusätzlich können Sie den Funktionen eine Liste von Schlüsseln als Parametern übergeben, mit der Sie abweichend von der aktuellen Selektion der Dimension andere Schlüssel auswerten können.

Beispiele

```
Month()
```

```
Month( NEXT( Time ) )
```

Kennzahl-Funktionen

Syntax

```
<fact-expression> := <fact-name> '(' [ <Key> { ',' <Key> }  
' )'
```

Beschreibung

Für jede Kennzahl wird in instantOLAP automatisch eine Funktion angelegt, die Sie zum Auslesen einer Kennzahl aus den Cubes verwenden können. Wie in der Funktion CUBE können Sie für diese Funktionen ausserdem Schlüssel als Parameter übergeben, mit den Sie bestimmen können für welche, abweichend vom aktuellen Filter, Schlüssel Sie diese Kennzahl auslesen möchten. Lesen Sie die Dokumentation der Funktion CUBE für eine detaillierte Beschreibung.

Beachten Sie, dass nur Kennzahlen mit "wohlgeformten" Namen automatisch in Funktionen umgewandelt werden, d.h. ihre Namen dürfen keine Leer- oder Sonderzeichen enthalten.

Beispiele

```
Amount ( )
```

```
Amount ( NEXT( Time ) )
```

Konstanten

Für die meisten Typen können Konstanten innerhalb von Ausdrücken verwendet werden:

- NULL (auf Seite 343) als Konstante für den Typ All
- TRUE und FALSE (siehe "Boolean-Konstanten" auf Seite 342) als Konstanten für den Typ Boolean
- Zeichenketten (siehe "String-Konstanten" auf Seite 344) als Konstanten für den Typ String
- Ganze Zahlen (siehe "Integer-Konstanten" auf Seite 342) als Konstanten für den Typ Integer
- Reelle Zahlen (siehe "Double-Konstanten" auf Seite 343) als Konstanten für den Typ Double
- Dimension-Schlüssel (siehe "Key-Konstanten" auf Seite 345) für den Typ Key

Boolean-Konstanten

Syntax

```
<Boolean-Constant> := 'TRUE' | 'true' | 'FALSE' | 'false'
```

Beschreibung

Die Werte TRUE und FALSE (jeweils in Gross- und Kleinschreibung erlaubt) sind die einzigen möglichen Werte des Typs Boolean.

Beispiele

```
IIF( TRUE, 10, 20 )
```

Siehe auch

AND (auf Seite 350), Boolean, NOT (auf Seite 407), OR (auf Seite 408)

Integer-Konstanten

Syntax

```
<Integer-Constant> = ['-'] { <digit> }
```

Beschreibung

Konstanten vom Typ Integer repräsentieren eine ganze Zahl.

Beispiele

0
100
-10

Siehe auch

Integer

Double-Konstanten

Syntax

`<Double-Constant> = ['-'] { <digit> } '.' { <digit> }`

Beschreibung

Konstanten vom Typ Double repräsentieren eine reelle Zahl.

Beispiele

0.0
100.10
-10.100

Siehe auch

Double

NULL

Syntax

`<NULL-Constant> := 'NULL'`

Beschreibung

Wie z.B. aus SQL-Datenbanken bekannt repräsentiert die Konstante "NULL" einen undefinierten Wert für alle Typen. NULL ist der einzige Wert, der den Typ All besitzt (welcher ein Subtyp aller anderer Typen ist), deshalb können sie NULL bei jedem Parameter jeder Funktion verwenden.

Es ist sehr wichtig zu verstehen, wie sich NULL und Funktionen mit NULL-Parametern in Datenbanken und instantOLAP verhalten. Wenn Sie z.B. Wert aus einer Datenbank (genauer einem Datenbank-basiertem Cube) lesen möchten, dann erhalten Sie NULL als Ergebnis, wenn der Wert nicht in der Datenbank existiert (z.B. wenn ein Umsatz für einen bestimmten Tag vorhanden ist). Die Funktion CUBE wird dann ebenfalls NULL für diesen Wert als Ergebnis haben und alle weiterverarbeitenden Funktionen bekommen diese NULL wiederum als Parameter.

Die meisten Funktionen liefern eine NULL als Ergebnis wenn einer ihrer Parameter NULL ist, da keine sinnvolle Berechnung eines Ergebnisses möglich ist. Es existieren jedoch einige spezielle Funktionen für den Umgang mit NULL, die wichtigsten davon sind ISNULL, EXISTS und ZERO.

Beispiele

```
Amount( Time:Dec/2004 )
```

Ergibt NULL, wenn kein Wert für Dezember 2004 existiert

```
DIV( NULL, 5 )
```

Ergibt immer NULL (unmöglich zu berechnen)

Siehe auch

EXISTS (auf Seite 370), ISNULL (auf Seite 385), ZERO (auf Seite 432)

String-Konstanten

Syntax

```
<String-Constant> := "" { <character> } "" | ''' { <character> } '''
```

Beschreibung

Zeichenketten sind Konstanten vom Typ String. In Ausdrücken müssen Sie die Zeichenketten in einfache oder doppelte Anführungszeichen setzen (wie in Programmiersprachen üblich). Sie können beide Arten von Anführungszeichen verwenden, die jeweils andere Art darf dann auch in der Zeichenkette selbst vorkommen, ohne zu einer Fehlermeldung zu führen.

Beispiele

```
'Hello World'
```

```
"Jim's car"
```

Siehe auch

String

Key-Konstanten

Syntax

```
<Key-Constant> = <dimension-name> ':' <id> | <dimension-name> ':' <id> ''
```

Beschreibung

Key-Konstanten verweisen auf einen bestimmten Schlüssel innerhalb einer Dimension. Wenn Sie Key-Konstanten in Abfragen verwenden, dann bleiben diese unbeeinflusst von aktuellen Selektionen und Filtern, da es sich um einen konstanten Verweis auf einen Schlüssel handelt.

Jede Key-Konstante wird als Dimensions-Name, gefolgt von einem Doppelpunkt ":" und der ID des Schlüssels, formuliert. Wenn die ID Leer- oder Sonderzeichen enthält, müssen Sie diese ausserdem in einfache oder doppelte Anführungszeichen setzen, um keine Fehlermeldung vom Parser zu erhalten.

Beispiele

```
Product:Coffee
```

```
Time:'01.01.2004'
```

Siehe auch

Key

Funktionen

ABC

Syntax

```
<abc-expression> := 'ABC(' <Key> ',' <Number> ',' <Number> ',' <Number> ')'
```

Seit

2.1

Ergebnis-Typ

Key

Beschreibung

Die Funktion ABC hilft Ihnen dabei, ABC-Analysen zu entwerfen. Mit dieser Funktion können Sie eine Menge von Schlüsseln nach einer bestimmten Kennzahl analysieren und dabei die Schlüssel ermitteln, die die obersten N% der Kennzahl (in Summe) bilden.

Der erste Parameter bestimmt die Menge der zu analysierenden Schlüssel. Der zweite Parameter ist der zu analysierende Ausdruck, in der Regel ein einfache Kennzahl. Der dritte und der vierte Parameter bestimmen den Bereich (durch einen Minimum und Maximum-Wert, der jeweils zwischen 0.0 und 1.0 liegen muss), den Sie als Ergebnis erhalten möchten.

Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
ABC( LEVEL( Products, 1 ), Amount(), 0.5, 1.0 )
```

Ermittle die Produkte, die (in Summe) die obersten 50% der Beträge ausmachen

ABS

Syntax

```
<abs-expression> := 'ABS(' <Number> ')'
```

Seit

2.0

Ergebnis-Typ

Number

Beschreibung

Diese Funktion gibt den absoluten (nicht vorzeichenbehafteten) Wert des Parameters zurück. Wenn der Parameter nicht negativ ist, dann wird er direkt zurückgegeben. Ist er negativ, dann wird die Negierung zurückgegeben. Ist er NULL (auf Seite 343), dann wird NULL zurückgegeben.

Beispiele

```
ABS( 1.0 ) --> 1.0
```

```
ABS( -1 ) --> 1
```

```
ABS( 0 ) --> 0
```

```
ABS( NULL ) --> NULL
```

Siehe auch

CEIL (auf Seite 354), FIRST (auf Seite 374)

ADD

Syntax

```
<add-expression> := 'ADD(' <Any> [ ',' { <Any> } ] )'
```

Seit

1.0

Ergebnis-Typ

Abhängig vom Typ der Parameter (siehe unten).

Beschreibung

Die Funktion ADD addiert oder verknüpft Werte jeden Typs. Die Wirkungsweise dieser Funktion hängt vom Typ der Parameter ab:

- Wenn alle Parameter Integer-Zahlen sind, ist das Ergebnis die Summe aller Zahlen und vom Typ Integer

- Wenn alle Parameter Zahlen sind, ist das Ergebnis die Summe aller Zahlen und vom Typ Number
- Wenn alle Parameter Schlüssel sind, ist das Ergebnis eine Liste mit allen Schlüssel und vom Typ Key
- Wenn alle Parameter Zeichenketten sind, ist das Ergebnis eine zusammengefügte Zeichenkette und vom Typ String
- Ansonsten ist das Ergebnis eine Zeichenkette in der alle Parameter als Strings zusammengefügt wurden

NULL (auf Seite 343)-Werte werden ignoriert, d.h. auch wenn ein Parameter eine NULL enthält, wird das Ergebnis nur aus den anderen Werten zusammengesetzt.

Diese Funktion ist nicht typsicher, da alles "addiert" und nicht auf seinen Typ hin überprüft wird. Verwenden Sie die SUM-Funktion, um eine rein mathematische Addierung von Zahlen durchzuführen.

Anstelle dieser Funktionen können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "+" verwenden.

Beispiele

```
ADD( 10, 20 ) --> 30
```

```
ADD( 'Hello', ' ', 'World' ) --> 'Hello World'
```

```
ADD( 'Hello', 10 ) --> 'Hello10'
```

```
ADD( Product:A, Product:B ) --> Product:A | Product:B
```

Siehe auch

DIV (auf Seite 365), MUL (auf Seite 402), SUB (auf Seite 419), SUM (auf Seite 420)

ALL

Syntax

```
<all-expression> := 'ALL(' <String> ')'
```

Seit

2.1

Ergebnis-Typ

Key

Beschreibung

Die Funktion ALL gibt alle Schlüssel einer Dimension oder einer Dimensions-Ebene zurück. Der Name der Dimension oder der Ebene muss als Zeichenkette an die Funktion übergeben werden.

Diese Funktion ist ähnlich der Funktion LEVEL, erlaubt aber die Übergabe der Dimensions-und Ebenennamen als Zeichenketten.

Beispiele

```
ALL( 'Time' )
```

Ergibt alle Schlüssel der Dimension "Time"

```
ALL( 'Week' )
```

Ergibt alle Wochen (eine Ebene der Dimension "Zeit")

Siehe auch

FAMILY (auf Seite 372), LEVEL (auf Seite 392)

ANCESTORS

Syntax

```
<ancestors-expression> := 'ANCESTORS(' <Key> ')'
```

Seit

1.2

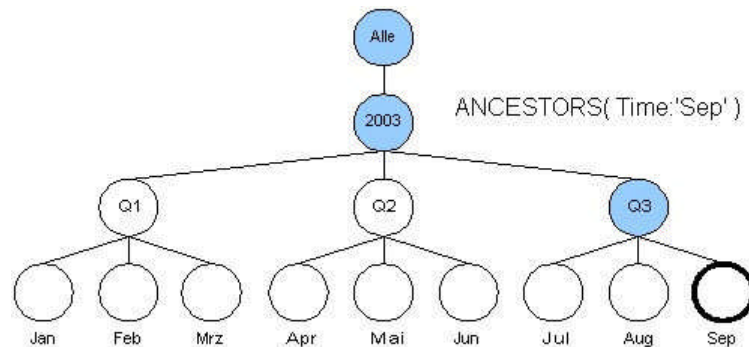
Ergebnis-Typ

Key

Beschreibung

Diese Funktion ermittelt alle Schlüssel oberhalb der Parameter-Schlüssel in deren Hierarchie (die Väter, deren Väter usw. bis hin zum Wurzel-Element der Dimensionen).

Wenn der Parameter NULL ist, dann gibt die Funktion NULL zurück.



Beispiele

`ANCESTORS(Time: '01.07.2003')`

Ergibt z.B. `Time: 'Jul/2003'` + `Time: '2003'` + `Time: 'Complete Period'`

Siehe auch

`CLUSTER` (auf Seite 356), `FILTER` (auf Seite 373), `PARENT` (auf Seite 409), `PEDIGREE` (auf Seite 410)

AND

Syntax

`<and-expression> := 'AND(' <Boolean> ',' <Boolean> ')'`

`<and-expression> := <Boolean> 'AND' <Boolean>`

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Der logische Funktion "AND" gibt die Boolean Konstanten TRUE, FALSE oder NULL zurück, abhängig von den Werten der Parameter:

- Wenn beide Parameter TRUE ergeben, gibt die Funktion auch TRUE zurück
- Wenn einer der Parameter FALSE ergibt, gibt die Funktion FALSE zurück
- Wenn einer Parameter NULL ergibt, gibt die Funktion NULL zurück

Anstelle diese Funktion können Sie auch den Operator "AND" verwenden.

Beispiele

```
AND( TRUE, TRUE ) --> TRUE
```

```
TRUE and TRUE --> TRUE
```

```
AND( FALSE, TRUE ) --> FALSE
```

```
AND( TRUE, FALSE ) --> FALSE
```

```
AND( TRUE, NULL ) --> NULL
```

```
AND( FALSE, NULL ) --> NULL
```

```
AND( NULL, NULL ) --> NULL
```

Siehe auch

NOT (auf Seite 407), OR (auf Seite 408)

ATTRIBUTENAMES

Syntax

```
<attributes-expression> := 'ATTRIBUTES(' <Key> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Diese Funktion gibt die Namen aller Attribute der Schlüssel zurück, die als Parameter übergeben werden.

Beispiele

```
ATTRIBUTENAMES( Time:'01.07.2003' )
```

Ergibt z.B. 'Weekday' + 'DayID' + ...

Siehe auch

ATTRIBUTES (auf Seite 352), DIMENSIONNAME (auf Seite 363)

ATTRIBUTENV

Syntax

```
<attributes-expression> := 'ATTRIBUTENV(' <Key> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Diese Funktion gibt die Namen und Werte aller Attribute der Schlüssel (aus dem Parameter) als Liste von Zeichenketten zurück. Die Werte eines Attribute werden in einer einzigen, durch Kommata separierten, Zeichenkette zurückgegeben. Wenn der Parameter NULL (auf Seite 343) ist gibt die Funktion NULL zurück.

Beispiele

```
ATTRIBUTENV( Product:Product1 )
```

Ergibt z.B. 'ProductID' + 'P1' + 'Customer' + 'C1, C2, C3' + ...

Siehe auch

ATTRIBUTENAMES (auf Seite 351), ATTRIBUTES (auf Seite 352)

ATTRIBUTES

Syntax

```
<attributes-expression> := 'ATTRIBUTES(' <Key> ')'
```

Seit

2.0

Ergebnis-Typ

Value

Beschreibung

Diese Funktion gibt alle Attribute der Schlüssel, die als Parameter übergeben werden, in einer Liste zurück. Wenn NULL (auf Seite 343) als Parameter übergeben wird, gibt die Funktion NULL zurück.

Beispiele

```
ATTRIBUTES( Time:'01.07.2003' )
```

Ergibt z.B. Weekday:'Tue' + Week:'26/2003' + ...

Siehe auch

ATTRIBUTENAMES (auf Seite 351)

AVG

Syntax

```
<avg-expression> := 'AVG(' <Number>, { ',' <Number> } ')'
```

Seit

1.0

Ergebnis-Typ

Double

Beschreibung

Die Funktion "AVG" ermittelt den Durchschnittswert aller Werte aller Parameter.

Wenn einer oder mehrere der Werte NULL (auf Seite 343) sind werden diese für die Bildung des Durchschnitts ignoriert. Wenn Sie NULL-Werte nicht ignorieren sondern als die Zahl 0 interpretieren möchten, dann können Sie dazu die Funktion ZERO verwenden.

Wenn nur NULL-Werte übergeben werden, dann gibt diese Funktion NULL zurück.

Beispiele

```
AVG( 2, 4 ) --> 3
```

```
AVG( Amount( LEVEL( Time, 3 ) ) ):
```

Ergibt den Durchschnittsbertrag für alle Tage (Ebene 3 der Dimension Time)

```
AVG( ZERO( Amount( LEVEL( Time, 3 ) ) ) ):
```

Wie oben, jedoch wird NULL als 0 gewertet

```
AVG( NULL ) --> NULL
```

Siehe auch

ADD (auf Seite 347), COUNT (auf Seite 359), MAX (auf Seite 398),
MIN (auf Seite 400), ZERO (auf Seite 432)

BEAUTIFY

Syntax

```
<beautify-expression> := 'BEAUTIFY(' <String> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion BEAUTIFY ändert die Gross/Kleinschreibung um Texte lesbarer zu gestalten. Nach dem Aufruf dieser Funktion wird der jeweils erste Buchstabe jeden Wortes gross geschrieben, der Rest klein. Wenn Sie eine NULL (auf Seite 343) übergeben, dann gibt diese Funktion auch NULL zurück.

Beispiele

```
BEAUTIFY( 'hello WORLD' ) --> 'Hello World'
```

Siehe auch

TOLOWER (auf Seite 422), TOUPPER (auf Seite 424)

CEIL

Syntax

```
<ceil-expression> := 'CEIL(' <Number> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion gibt den kleinsten ganzen Wert zurück, der nicht kleiner als der Parameter ist (Aufrundung). Spezielle Fälle sind:

- Wenn der Parameter bereits ein ganzer Wert ist, wird genau dieser Wert wieder zurückgegeben
- Wenn der Parameter NULL (auf Seite 343) ist, wird auch NULL zurückgegeben

Beispiele

```
CEIL( 1.5 ) --> 2
```

```
CEIL( -1.5 ) --> -1
```

```
CEIL( 5 ) --> 5
```

```
CEIL( NULL ) --> NULL
```

Siehe auch

FLOOR (auf Seite 375), ROUND (auf Seite 415)

CHILDREN

Syntax

```
<children-expression> := 'CHILDREN(' <Key> ')'
```

Seit

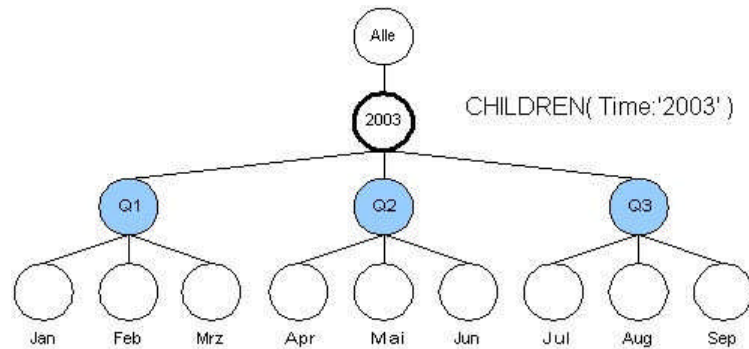
1.0

Ergebnis-Typ

Key

Beschreibung

Diese Funktion ermittelt alle Kind-Elemente der Schlüssel, die als Parameter übergeben werden. Kinder eines Schlüssels sind die Schlüssel, die in der Hierarchie direkt unter diesem positioniert sind.



Wenn NULL (auf Seite 343) als Parameter übergeben wird, dann gibt die Funktion NULL zurück.

Beispiele

```
CHILDREN( Time: '2003' )
```

Ergibt z.B. Time:'Jan/2003' + Time:'Feb/2030' + ... + Time:'Dez/2003'

```
CHILDREN( NULL ) --> NULL
```

Siehe auch

ANCESTORS (auf Seite 349), FAMILY (auf Seite 372), LEAFS (auf Seite 388), NEIGHBOURS (auf Seite 403), PARENT (auf Seite 409), PEDIGREE (auf Seite 410)

CLUSTER

Syntax

```
<cluster-expression> := 'CLUSTER(' <Key> ')'
```

Seit

2.1.2

Ergebnis-Typ

Key

Beschreibung

Die CLUSTER-Funktion ersetzt alle Schlüssel durch ihr Väter, wenn die ihre Geschwister ebenfalls in der Liste vorkommen. Das Ergebnis dieser Funktion wird solange erneut "geclustert" bis die Anzahl der Schlüssel sich nicht mehr verringern lässt.

Diese Funktion ist sehr nützlich, um die Anzahl der zu ladenden Werte zu vermindert, z.B. wenn Sie eine Kennzahl für einen grossen Zeitraum summieren möchten.

Wenn der Parameter NULL (auf Seite 343) ist gibt diese Funktion NULL zurück.

Beispiele

```
CLUSTER( Time:'Q1/2003' + Time:'Q2/2003' + Time:'Q3/2003' +  
Time:'Q4/2003' + Time:'Q1/2003' ) --> Time:'2003' +  
Time:'Q1/2004'
```

```
CLUSTER( NULL ) --> NULL
```

```
SUM( Amount( CLUSTER( Time ) ) ):
```

Ist eine effiziente Aggregation der Kennzahl "Amount" für einen langen Zeitraum

CONCAT

Syntax

```
<concat-expression> := 'CONCAT(' <String> ',' <String> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion "CONCAT" setzt eine neue Zeichenkette aus allen Zeichenketten zusammen, die als erster Parameter übergeben werden. Die Zeichekette des zweiten, optionalen Parameters wird dabei als Trennzeichen verwendet. Das Standard-Trennzeichen lautet ",".

Wenn Sie NULL (auf Seite 343) als ersten Parameter übergeben, dann gibt die Funktion NULL zurück.

Beispiele

```
CONCAT( LEVEL( Customer, 1 ).Name )
```

Ergibt z.B. 'Customer1,Customer2,Customer3'

```
CONCAT( LEVEL( Customer, 1 ).Name, ';' )
```

Ergibt z.B. 'Customer1;Customer2;Customer3'

Siehe auch

ADD (auf Seite 347)

CONTAINS

Syntax

```
<contains-expression> := 'CONTAINS(' <Any> ',' <Any> ')'
```

Seit

1.2

Ergebnis-Typ

Boolean

Beschreibung

Die Funktion CONTAINS überprüft, ob alle Elemente des zweiten Parameters im ersten vorhanden sind. Ist das der Fall, gibt sie TRUE zurück, ansonsten FALSE. NULL (auf Seite 343) wird wie ein normales Element behandelt.

Beispiele

```
CONTAINS( FAMILY( Time:'Complete Period' ), LEVEL( Time, 1 ) ) --> TRUE
```

```
CONTAINS( FAMILY( Time:'Complete Period' ), 'Hello World' ) --> FALSE
```

```
CONTAINS( FAMILY( Time:'Complete Period' ), NULL ) --> NULL
```

```
CONTAINS( NULL, LEVEL( Time, 1 ) ) --> FALSE
```

Siehe auch

EXISTS (auf Seite 370), IN (auf Seite 383)

COUNT

Syntax

```
<count-expression> := 'COUNT(' <Any> ')'
```

Seit

1.2

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion gibt die Grösse des Parameters zurück, also die Anzahl der Elemente in ihr. Wenn der Parameter NULL (auf Seite 343) ist, gibt diese Funktion NULL zurück.

Beispiele

```
COUNT( LEVEL( Fact, 0 ) ) --> 1 (Der Root-Key)
```

```
COUNT( NULL ) --> NULL
```

Siehe auch

EXISTS (auf Seite 370)

CUBE

Syntax

```
<cube-expression> := 'CUBE(' [ <Key> { ',' <Key> } ] ')'
```

```
<cube-expression> := '[' [ <Key> { ',' <Key> } ] ]'
```

```
<cube-expression> := <fact-name> '(' [ <Key> { ',' <Key> } ] ')'
```

Seit

1.0

Ergebnis-Typ

Entspricht dem Typ der zurückgegebenen Kennzal(en).

Beschreibung

Mit der Funktion CUBE können Sie die Daten aus den Cubes auslesen. Ohne weitere Parameter wird diese Funktion genau die Werte aus, die der aktuellen Selektion entsprechend.

Die aktuelle Selektion wird durch die Überschriften in Pivot-Tabellen, Abfragen- und Block-Filtern, Selektoren etc. bestimmt.

Sie könnten z.B. eine Abfrage mit der Kennzahl "Fact:Amount" in der X-Achse und dem Schlüssel "Product:Product1" auf der Y-Achse definieren. Ausserdem könnten Sie auch einen Selektor mit Jahren zum Bericht hinzufügen. Wenn der Benutzer dann das Jahr 2003 auswählt wäre die Selektion für die eine Zelle in der Pivot-Tabelle "Fact:Amount, Product:Product1, Time:2003". Wenn Sie dann die CUBE-Funktion für diese Zelle verwenden, wird genau dieser Wert aus den Cubes gelesen.

Wenn Sie einen Wert auslesen möchten, der nicht der aktuellen Selektion entspricht, können Sie weitere Key-Expressions als Parameter an diese Funktion übergeben. Die Ergebnisse dieser Expressios werden dann auf die Selektion angewendet bevor damit die Werte ausgelesen werden. Wenn Sie z.B. die Funktion "CUBE(NEXT(Time))" im obigen Beispiel verwenden würden, dann würden Sie den Wert "Fact:Amount, Product:Product1, Time:2004" auslesen. Oder Sie könnten z.B. "CUBE(Fact:Price)" aufrufen, um eine andere Kennzahl (in diesem Fall den Preis) auszulesen.

Beachten Sie, dass die CUBE-Funktion mehr als eine Kennzahl auslesen und zurückgeben kann. Das passiert wenn die aktuelle Selektion mehr als einen Schlüssel für mindestens eine Dimension besitzt. Wenn Sie z.B. im obigen Beispiel die Funktion "CUBE(CHILDREN(Time))" aufrufen, dann erhalten Sie 12 Werte, einen je Monat.

Mathematisch betrachtet entspricht die Anzahl der zurückgegebenen Werte $(N1) * (N2) * \dots * (Nd)$, wobei N_x der Anzahl der Selektierten Schlüssel für Dimension x und d der Anzahl der Dimensionen entspricht.

Der Rückgabetyt dieser Funktion hängt von den selektierten Kennzahlen ab. Wenn genau eine Kennzahl selektiert ist, wird deren Typ als Rückgabetyt verwendet. Wenn mehr als eine Kennzahl selektiert ist, entspricht der Rückgabetyt dem Supertyp aller Kennzahlen (was im ungünstigsten Fall Any wäre).

Es gibt zwei Kurzformen für diese Funktion: Eckige Klammer und automatisch generierte Kennzahl-Funktionen. Die Klammer-Form ist sehr ähnlich dem direkten Funktionsaufruf, anstatt CUBE auszuschreiben müssen Sie nur alle Parameter in eckige Klammern [] setzen. Die Funktion "CUBE(NEXT(Time))" entspricht dann "[NEXT(Time)]".

Die automatisch generierten Kennzahl-Funktionen können dann verwendet werden, wenn Sie genau wissen welche Kennzahl Sie auslesen möchten (was der Normalfall ist). In diesem Fall können Sie die Funktion verwenden, die automatisch beim Anlegen einer Kennzahl erstellt wurde und genauso heisst. Wenn Sie z.B. eine Kennzahl "Amount" definiert haben, dann existiert auch eine Funktion "Amount". Dann können Sie auf die Kennzahl auch mit "Amount()" anstatt mit "[Fact:Amount]" zugreifen. Sie können ausserdem weitere Parameter an diese Funktion übergeben, z.B. "Amount(NEXT(Time))".

Beachten Sie, dass nur für Kennzahlen mit wohlgeformten Namen automatisch Funktionen angelegt werden. Wenn der Name Ihrer Kennzahl Sonderzeichen oder Leerzeichen enthält, kann das System keine Kennzahl-Funktion für diese anlegen.

Beispiele

`CUBE()`

Ergibt die aktuell selektierten Werte

`[]`

Wie oben

`CUBE(NEXT(Time))`

Ergibt die aktuell selektierten Werte für das nächste Zeit-Element

`[NEXT(Time)]`

Wie oben

`[NEXT(Time), Fact:Amount]`

Ergibt den Wert von "Amount" für das nächste Zeit-Element

`CUBE(NEXT(Time), Fact:Amount)`

Wie oben

`Amount(NEXT(Time))`

Wie oben

Siehe auch

ZERO (auf Seite 432)

DEPTH

Syntax

```
<depth-expression> := 'DEPTH(' <Key> ')'
```

Seit

2.1

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion ermittelt die Tiefe einer Dimension (definiert durch den Parameter) und gibt diese zurück. Wenn der Parameter NULL (auf Seite 343) ist gibt die Funktion NULL zurück.

Beispiele

```
DEPTH( Time ) --> 4
```

```
DEPTH( NULL ) --> NULL
```

Siehe auch

LEVELNAMES (auf Seite 393)

DIMENSIONATTRIBUTENAMES

Syntax

```
<dimensionattributenames-expression> :=  
'DIMENSIONATTRIBUTENAMES(' <Key> ')'
```

Seit

2.1

Ergebnis-Typ

String

Beschreibung

Ermittelt die Namen aller Attribute der übergebenen Dimension (bzw. der Dimensionen der übergebenen Schlüssel). Wenn der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
DIMENSIONATTRIBUTENAMES( Product )
```

Ergibt z.B. 'ProductID' + 'Customer' + 'Text' + ...

Siehe auch

ATTRIBUTENAMES (auf Seite 351)

DIMENSIONNAME

Syntax

```
<dimensionname-expression> := 'DIMENSIONNAME(' <Key> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Gibt die Dimensions-Namen der Schlüssel wieder, die als Parameter übergeben werden. Wenn der Parameter mehr als einen Schlüssel enthält, dann werden entsprechend viele Namen zurückgegeben. Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion auch NULL zurück.

Beispiele

```
DIMENSIONNAME( Product ) --> 'Product'
```

```
DIMENSIONNAME( Product:Product1 ) --> 'Product'
```

Siehe auch

DIMENSIONNAMES (auf Seite 363)

DIMENSIONNAMES

Syntax

```
<dimensionnames-expression> := 'DIMENSIONNAMES()'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Gibt die Namen aller Dimensionen als Liste zurück.

Beispiele

```
DIMENSIONNAMES ( )
```

Ergibt z.B. 'Fact' + 'Product' + 'Time' + ...

Siehe auch

DIMENSIONNAME (auf Seite 363)

DISTINCT

Syntax

```
<distinct-expression> := 'DISTINCT(' <Any> ')'
```

Seit

2.0

Ergebnis-Typ

Entspricht dem Parameter-Typ

Beschreibung

Die Funktion "DISTINCT" eliminiert mehrfache Wert aus der als Parameter übergebenen List. D.H. jeder Wert der mehr als einmal vorkommt wird nur noch ein einziges Mal im Ergebnis vorkommen. Das gilt auch für den Wert NULL (auf Seite 343).

Beispiele

```
DISTINCT( Product:Product1 ) --> Product:Product1
```

```
DISTINCT( Product:Product1 | Product:Product2 |  
Product:Product1 ) --> Product:Product1 + Product:Product2
```

DIV

Syntax

```
<div-expression> := 'DIV(' <Number> ',' <Number> ')'
```

```
<div-expression> := <Number> '/' <Number>
```

Seit

1.0

Ergebnis-Typ

Double

Beschreibung

Die Funktion DIV teilt jeden Wert aus dem ersten Parameter durch den entsprechenden Wert aus dem zweiten Parameter und gibt die Ergebnisse als Liste von Zahlen zurück. Beide Parameter müssen die gleiche Anzahl von Werten enthalten.

Wenn einer der beiden Parameter NULL (auf Seite 343) ist wird die Funktion NULL zurückgeben. Wenn der Divisor (der zweite Parameter) 0 ist gibt die Funktion die Zeichenkette "ERR" zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "/" verwenden.

Beispiele

```
DIV( 10, 2 ) --> 5
```

```
10 / 2 --> 5
```

```
DIV( 10, NULL ) --> NULL
```

```
DIV( NULL, 5 ) --> NULL
```

```
DIV( 10, 0 ) --> 'ERR'
```

```
DIV( JOIN( 10, 5 ), JOIN( 2, 1 ) ) --> 5 | 5
```

Siehe auch

ADD (auf Seite 347), MUL (auf Seite 402)

DLOOKUP

Syntax

```
<dlookup-expression> := 'DLOOKUP(' <Key> { ',' <Key> } ')'
```

Seit

2.1.2

Ergebnis-Typ

Key

Beschreibung

Diese Funktion ermittelt, durch direkte Abfrage der Cubes, für welche Schlüssel des ersten Parameters eine bestimmte Kennzahl (definiert durch die folgenden Parameter) verfügbar ist (also nicht NULL ist). Bei der Abfrage der Cubes wird ebenfalls der aktuelle Filter beachtet. Die DLOOKUP-Funktion ist sehr wichtig zum Erstellen von Berichten wenn Sie Kennzahlen von sehr dünn befüllten Cubes darstellen möchten.

Die DLOOKUP-Funktion funktioniert sehr ähnlich wie die LOOKUP-Funktion, ist aber einfacher und schneller:

DLOOKUP kann nur ganze Dimensionen oder Dimensions-Ebenen ermitteln. Daher wird als erster Parameter nur ein Dimensions- oder Ebenen-Name erlaubt. Ausserdem können Sie mit einem Aufruf immer nur Schlüssel für eine einzige Dimension ermitteln.

DLOOKUP beachtet beim Suchen keine Formeln sondern ermittelt nur Schlüssel, die direkt aus einem Cube geladen werden können

Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
DLOOKUP( Article, Fact:Amount )
```

ergibt alle Artikel, für die unter dem aktuellen Filter die Kennzahl "Amount" in einem beliebigen Cube vorkommt

```
DLOOKUP( Article::Group, Fact:Amount )
```

Ergibt alle Artikelgruppen, für die unter dem aktuellen Filter die Kennzahl "Amount" in einem beliebigen Cube vorkommt

```
DLOOKUP( Article::Group + Article::Category, Fact:Amount )
```

Ergibt alle Artikelgruppen und -kategorien

```
DLOOKUP( Article::Group, Fact:Amount + PREV( Time ) )
```

Ergibt alle Artikelgruppen für den Vorzeitraum

```
DLOOKUP( NULL, NULL ) --> NULL
```

Siehe auch

LOOKUP (auf Seite 395), SORT (auf Seite 415)

DRILLLEVEL

Syntax

```
<drilllevel-expression> := 'DRILLLEVEL()'
```

Seit

2.1

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion ermittelt die aktuelle Drilldown-Ebene für die aktuelle Überschrift (Header). Ein Header ohne geöffneten Drilldown besitzt den Level 0, nach einem Drilldown 1 usw. Header ohne möglichen Drilldown geben immer 0 zurück.

Der Aufruf dieser Funktion ist nur in Pivot-Tabellen zulässig.

Beispiele

```
DRILLLEVEL( )
```

Ergibt 0, wenn kein Drilldown stattgefunden hat

Ergibt 1, wenn der Header wurde einmal per Drilldown geöffnet wurde

EMPTY

Syntax

```
<empty-expression> := 'EMPTY()'
```

Seit

2.1

Ergebnis-Typ

All

Beschreibung

Diese Funktion gibt eine leere Liste zurück. Der der Ergebnis-Typ dieser Funktion All ist, können Sie sie als Parameter für jede andere Funktion verwenden.

Examples

```
EMPTY( )
```

ENDSWITH

Syntax

```
<endswith-expression> := 'ENDSWITH(' <String>, <String> ')'
```

Seit

2.1

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion ermittelt, ob die Zeichenkette aus dem ersten Parameter mit der Zeichenkette aus dem zweiten Parameter endet. Wenn einer der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
ENDSWITH( 'Hello World', 'World' ) --> TRUE
```

```
ENDSWITH( 'Hello World', 'Hello' ) --> FALSE
```

```
ENDSWITH( 'Hello World', 'NULL' ) --> NULL
```

Siehe auch

STARTSWITH (auf Seite 417)

EQUAL

Syntax

```
<equal-expression> := 'EQUAL(' <Any> ',' <Any> ')'
```

```
<equal-expression> := <Any> '=' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion testet beide Parameter auf Gleichheit. Sie gelten als gleich, wenn beide Parameter die gleiche Anzahl Elemente besitzen und jedes Element aus Parameter 1 an der selben Position in Parameter 2 vorkommt.

Anstelle der Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "=" verwenden.

Beispiele

```
EQUAL( 10, 10 ) --> TRUE
```

```
EQUAL( 10, 20 ) --> FALSE
```

```
EQUAL( 10 | 10, 10 ) --> TRUE
```

```
EQUAL( 10, 'Hello World' ) --> FALSE
```

```
EQUAL( 10, NULL ) --> NULL
```

```
EQUAL( NULL, NULL ) --> NULL
```

Siehe auch

GREATER (auf Seite 376), GREATER_OR_EQUAL (auf Seite 377), LESS (auf Seite 390), LESS_OR_EQUAL (auf Seite 391), UNEQUAL (auf Seite 425)

EVAL

Syntax

```
<eval-expression> := 'EVAL(' <String> ')'
```

Seit

2.0

Ergebnis-Typ

Der Ergebnis-Typ entspricht dem Typ der Expression, die als Parameter übergeben wird.

Beschreibung

Die Funktion EVAL ermöglicht das Parsen und Auswerten von Ausdrücken zur Laufzeit. Der Parameter ist eine Zeichenkette, die einen gültigen Ausdruck enthalten muss.

Wenn der Parameter NULL (auf Seite 343) ist gibt diese Funktion NULL zurück.

Beispiele

```
EVAL( '10 * 2' ) --> 5
```

```
EVAL( $param )
```

Parsen und Berechnen der Parameters mit dem Namen 'param'

```
EVAL( NULL ) --> NULL
```

EXISTS

Syntax

```
<exists-expression> := 'EXISTS(' <Any> ')'
```

Seit

1.1

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob der Parameter mindestens einen anderen Wert als NULL (auf Seite 343) enthält.

Beispiele

```
EXISTS( NEXT( Time ) )
```

Ergibt TRUE, wenn das aktuelle Element aus der Dimension Time einen Nachfolger besitzt

```
EXISTS( 1 ) --> TRUE
```

```
EXISTS( NULL ) --> FALSE
```

Siehe auch

ISNULL (auf Seite 385)

EXP

Syntax

```
<exp-expression> := 'EXP(' <Number>, <Number> ')'
```

Seit

2.1

Ergebnis-Typ

Number

Beschreibung

Diese Funktion berechnet den Exponenten (definiert durch den zweiten Parameter) des ersten Parameters. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
EXP( 10, 2 ) --> 100
```

```
EXP( 3, 3 ) --> 27
```

```
EXP( 10, NULL ) --> NULL
```

```
EXP( NULL, 2 ) --> NULL
```

FACTROOT

Syntax

```
<factroot-expression> := 'FACTROOT()'
```

Seit

1.2

Ergebnis-Typ

Key

Beschreibung

Die Funktion "FACTROOT" gibt das Wurzel-Element der Kennzahl-Dimension als Liste zurück.

Beispiele

```
FACTROOT( )
```

Ergibt z.B. Fact:'All Facts'

Siehe auch

NONFACTROOTS (auf Seite 405)

FAMILY

Syntax

```
<family-expression> := 'FAMILY(' <Key> ')'
```

Seit

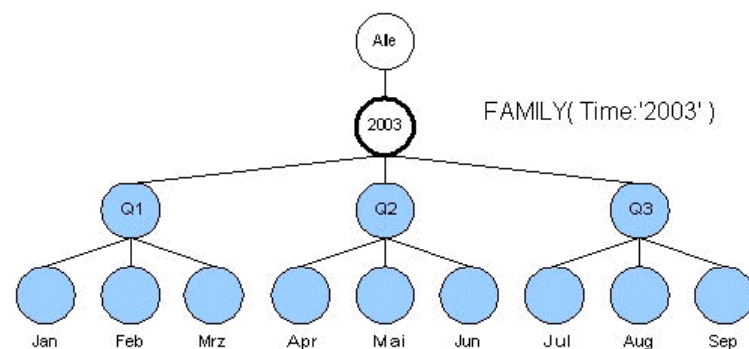
1.2

Ergebnis-Typ

Key

Beschreibung

Die FAMILY Funktion berechnet die "Familie" für jeden Schlüssel im Parameter und gibt diese zurück. Die Familie eines Schlüssels ist der Schlüssel selbst, seine Kinder, die Kinder seiner Kinder usw. bis runter zu den Blättern. Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.



Beispiele

```
FAMILY( Time:'2003' )
```

Ergibt z.B. Time:'Jan/2003' + Time:'01.01.2003' + ...
Time:'31.12.2003'

```
FAMILY( NULL ) --> NULL
```

Siehe auch

ALL (auf Seite 348), ANCESTORS (auf Seite 349), CHILDREN (auf Seite 355), LEAFS (auf Seite 388), PARENT (auf Seite 409)

FILTER

Syntax

```
<filter-expression> := 'FILTER()'
```

Since

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion "FILTER" gibt die aktuelle Selektion als Zeichenkette zurück. Die aktuelle Selektion ist die Zusammenstellung aller Selektionen z.B. in einer Zelle. Diese Funktion ist zur Analyse bzw. zum Debugging gedacht und Sie können damit die Selektion in einem Berichts-Element darstellen.

Beispiele

```
FILTER( )
```

Ergibt z.B. die Zeichenkette "[Time=2003;Product=A]"

FIND

Syntax

```
<find-expression> := 'FIND( <Key> ',' <String> )'
```

Seit

2.0

Ergebnis-Typ

Key

Beschreibung

Die Funktion FIND sucht in einer Dimension (die durch den ersten Parameter bestimmt wird) Schlüssel mittels eines Suchmusters, das durch den zweiten Parameter definiert wird. Im Suchmuster können Sie die Platzhalter "*" und "?" verwenden. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
FIND( Time, 'Jan/2003' )
```

Ergibt z.B. Time:'Jan/2003'

```
FIND( Time, 'J*/2003' )
```

Ergibt z.B. Time:'Jan/2003', Time:'Jun/2003', Time:'Jul/2003'

```
FIND( Time, NULL ) --> NULL
```

```
FIND( NULL, 'J*/2003' ) --> NULL
```

Siehe auch

NOW (auf Seite 408)

FIRST

Syntax

```
<first-expression> := 'FIRST(' <Any> [ ',' <Integer> ] )'
```

Seit

1.2

Ergebnis-Typ

Entspricht dem Typ des ersten Parameters

Beschreibung

Diese Funktion gibt die ersten N Elemente des ersten Parameters zurück. Wenn Sie keine Grösse N (mit dem zweiten Parameter) angeben, dann wird nur das erste Element zurückgegeben. Wenn einer der Parameter NULL (auf Seite 343) ist, dann wird auch NULL zurückgegeben.

Beispiele

```
FIRST( LEVEL( Time, 2 ) )
```

Ergibt z.B. Time:'Jan/2003'

```
FIRST( LEVEL( Time, 2 ), 3 )
```

Ergibt z.B. Time:'Jan/2003' + Time:'Feb/2003' + Time:'Mrz/2003'

```
FIRST( NULL ) --> NULL
```

Siehe auch

LAST (auf Seite 387), NEXT (auf Seite 404), PREV (auf Seite 412)

FLOOR

Syntax

```
<floor-expression> := 'FLOOR(' <Number> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion gibt den grössten Wert zurückgeben, der nicht grösser als der Parameter ist (anbrunden). Wenn der Parameter NULL (auf Seite 343) ist, dann gibt diese Funktion NULL zurück.

Beispiele

```
FLOOR( 1.5 ) --> 1
```

```
FLOOR( -1.5 ) --> -2
```

```
FLOOR( 5 ) --> 5
```

FLOOR(NULL) --> NULL

Siehe auch

ABS (auf Seite 346), CEIL (auf Seite 354), ROUND (auf Seite 415)

FOREACH

Syntax

<foreach-expression> := 'FOREACH(' <Key> ',' <Any> ')'

Seit

1.2

Ergebnis-Typ

Entspricht dem Typ des zweiten Parameters.

Beschreibung

Die Funktion FOREACH berechnet den zweiten Parameter für jeden Schlüssel aus dem ersten und gibt alle Ergebnisse in einer Liste zurück. Wenn einer der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Als erstes wird Parameter 1 ausgewertet dessen Ergebnis eine Liste von Dimensions-Schlüsseln ist. Die aktuelle Selektion (der Filter) wird dann für jeden dieser Schlüssel einmal angepasst und der zweite Parameter wird jeweils mit diesem Filter aufgerufen. Alle Ergebnisse werden in einer Liste zusammengefasst, dessen Typ dem vom Parameter 2 entspricht.

Beispiele

```
FOREACH( LEVEL( Article, 2 ), Article.Color )
```

Ergibt alle Farben der Artikel der Ebene 2

Siehe auch

MATCH (auf Seite 396), SORT (auf Seite 415)

GREATER

Syntax

<greater-expression> := 'GREATER(' <Any> ',' <Any> ')'

```
<greater-expression> := <Any> '>' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft für jeden Wert des ersten Parameters ob dieser grösser ist als der entsprechende Wert aus dem zweiten Parameter. Wenn die Parameter eine verschiedene Anzahl von Werten enthalten, gibt die Funktion FALSE zurück. Wenn einer der Werte NULL (auf Seite 343) ist, gibt diese Funktion NULL zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) ">" verwenden.

Beispiele

```
GREATER( 2, 1 ) --> TRUE
```

```
2 > 1 --> TRUE
```

```
GREATER( 2, 2 ) --> FALSE
```

```
2 > 2 --> FALSE
```

```
GREATER( NULL, 2 ) --> NULL
```

```
GREATER( JOIN( 2, 3 ), JOIN( 1, 2 ) ) --> TRUE
```

Siehe auch

EQUAL (auf Seite 369), GREATER_OR_EQUAL (auf Seite 377), LESS (auf Seite 390), LESS_OR_EQUAL (auf Seite 391)

GREATER_OR_EQUAL

Syntax

```
<greater-expression> := 'GREATER_OR_EQUAL(' <Any> ',' <Any> ')'
```

```
<greater-expression> := <Any> '>=' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft für jeden Wert des ersten Parameters ob dieser grösser oder gleich als der entsprechende Wert aus dem zweiten Parameter ist. Wenn die Parameter eine verschiedene Anzahl von Werten enthalten, gibt die Funktion FALSE zurück. Wenn einer der Werte NULL (auf Seite 343) ist, gibt diese Funktion NULL zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) ">=" verwenden.

Beispiele

```
GREATER_OR_EQUAL( 2, 1 ) --> TRUE
```

```
2 >= 1 --> TRUE
```

```
GREATER_OR_EQUAL( 2, 2 ) --> TRUE
```

```
2 >= 2 --> TRUE
```

```
GREATER_OR_EQUAL( NULL, 2 ) --> NULL
```

```
GREATER_OR_EQUAL( JOIN( 2, 2 ), JOIN( 1, 2 ) ) --> TRUE
```

Siehe auch

EQUAL (auf Seite 369), GREATER (auf Seite 376), LESS (auf Seite 390), LESS_OR_EQUAL (auf Seite 391)

HASKEYS

Syntax

```
<haskeys-expression> := 'HASKEYS(' <Key> { ',' <Key> } ')'
```

Seit

1.1

Ergebnis-Typ

Boolean

Beschreibung

Die Funktion HASKEYS überprüft die aktuelle Selektion (den Filter) daraufhin, ob sie mindestens einen aus jedem Parameter enthält. Diese Funktion ist sehr nützlich bei der Formulierung von Match-Expression, z.B. in Cubes oder Formeln. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt diese Funktion NULL zurück.

Beispiele

```
HASKEYS( Fact:Amount )
```

Enthält die aktuelle Selektion die Kennzahl "Amount"?

```
HASKEYS( Fact:Amount, Time:'Jan/2003' )
```

Enthält die aktuelle Selektion die Kennzahl "Amount" und den Schlüssel für Januar?

Siehe auch

HASLEVEL (auf Seite 379), HASPOSITION (auf Seite 380)

HASLEVEL

Syntax

```
<haslevel-expression> := 'HASLEVEL(' <Key> { ',' <Integer> } ')'
```

Seit

1.1

Ergebnis-Typ

Boolean

Beschreibung

Wie die Funktion HASKEYS überprüft diese Funktion die Schlüssel aus der aktuellen Selektion (dem Filter). Im Gegensatz zur Funktion HASKEYS überprüft diese Funktion, die aktuell selektierten Schlüssel einer Dimension in einer (oder mehreren) bestimmten Ebene der Dimensions-Hierarchie liegen. Der erste Parameter bestimmt die zu überprüfende Dimension, die folgenden sind Zahlen, die für die Ebenen stehen, auf die überprüft werden soll. Wenn alle selektierten Schlüssel zu einer dieser Ebenen gehören, gibt die Funktion TRUE zurück, ansonsten FALSE.

Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
HASLEVEL( Time, 1, 2 )
```

Gehört der aktuell selektierte Schlüssel der Dimension "Time" zur Ebene 1 oder 2?

Siehe auch

HASKEYS (auf Seite 378), HASPOSITION (auf Seite 380)

HASPOSITION

Syntax

```
<hasposition-expression> := 'HASPOSITION(' <Key> { ','  
<Integer> } ')'
```

Seit

2.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob die Schlüssel, die als Parameter 1 übergeben werden, eine Positionen aus dem zweiten oder folgenden Parameter besitzen. Die Position eines Schlüssels ist 0, wenn er das erste Kind seines Vaters ist. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt diese Funktion NULL zurück.

Beispiele

```
HASPOSITION( Time:'Jan/2004', 0 ) --> TRUE
```

```
HASPOSITION( Time:'Jan/2004', 1 ) --> FALSE
```

Siehe auch

POSITIONOF (auf Seite 411)

HASROLES

Syntax

```
<hasroles-expression> := 'HASROLES(' <String> { ','  
<String> } ')'
```

Seit

2.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob der aktuelle Benutzer alle Rollen besitzt, die als Parameter übergeben werden. Wenn einer der Parameter aus mehr als einem Wert besteht, dann reicht es aus wenn der Benutzer nur ein der Rollen dieses Parameters besitzt.

Beispiele

```
HASROLES( 'iolapAdmin', 'iolapUser' )
```

Ist der aktuelle Benutzer Administrator und Benutzer?

```
HASROLES( 'iolapAdmin' | 'iolapUser' )
```

Ist der aktuelle Benutzer Administrator oder Benutzer?

Siehe auch

HASUSER (auf Seite 381), USER (auf Seite 428)

HASUSER

Syntax

```
<hasuser-expression> := 'HASUSER(' <String> { ',' <String>  
} ')'
```

Seit

2.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob der Name des aktuellen Benutzer mit einem der Namen übereinstimmt, die als Parameter übergeben werden. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
HASUSER( 'guest', 'admin' )
```

Heisst der aktuelle Benutzer 'guest' oder 'admin'?

Siehe auch

HASROLES (auf Seite 380), USER (auf Seite 428)

IIF

Syntax

```
<iif-expression> := 'IIF(' <Boolean> ',' <Any> ',' <Any>
')'
```

Seit

1.2

Ergebnis-Typ

Entspricht dem Ergebnistyp des zweiten und dritten Parameters (dem Supertyp von beiden).

Beschreibung

Die IIF-Ausführung erlaubt die bedingte Ausführung von zwei Ausdrücken: Abhängig vom Ergebnis des ersten Parameters (der einen Ausdruck vom Typ Boolean enthalten muss) wird die Funktionen das Ergebnis des zweiten oder dritten Parameters oder NULL zurückgeben:

- Wenn der Parameter 1 TRUE (siehe "Boolean-Konstanten" auf Seite 342)TRUE als Ergebnis hat, wird das Ergebnis von Parameter 2 zurückgegeben
- Wenn der Parameter 1 FALSE (siehe "Boolean-Konstanten" auf Seite 342) als Ergebnis hat, wird das Ergebnis von Parameter 3 zurückgegeben
- Wenn der Parameter 1 NULL (auf Seite 343) als Ergebnis hat, wird NULL zurückgegeben

Beispiele

```
IIF( HASLEVEL( Time, 1 ), 'Year', IIF( HASLEVEL( Time, 2 ),
'Month', 'Day' ) )
```

Ergibt 'Year', 'Month' oder 'Day', abhängig von der Ebene der aktuell selektierten Zeit

Siehe auch

MATCH (auf Seite 396)

IN

Syntax

```
<in-expression> := 'IN(' <Any> ',' <Any> ')'
```

```
<in-expression> := <Any> 'IN' <Any>
```

Seit

1.2

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob alle Werte des ersten Parameters im zweiten vorhanden sind.

Anstelle dieser Funktion können Sie auch den Operator "IN" verwenden.

Beispiele

```
IN( Article.Color, 'Red' | 'Green' )
```

Ist der aktuelle Artikel rot oder grün?

```
Article.Color IN ( 'Red' | 'Green' )
```

Wie oben

Siehe auch

CONTAINS (auf Seite 358), EXISTS (auf Seite 370)

INTERSECT

Syntax

```
<intersect-expression> := 'INTERSECT(' <Any> { ',' <Any> }  
' )'
```

Seit

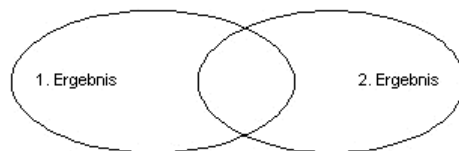
1.2

Ergebnis-Typ

Supertyp der Typen aller Parameter.

Beschreibung

Die Funktion INTERSECT berechnet die Schnittmenge aller Elemente aus allen Parameters. Das sind nur die die Elemente, die in allen Parametern vorkommen. Der Ergebnis-Typ dieser Funktion ist der Supertyp von allen Parameter-Typen.



Beispiele

```
INTERSECT( Article.Customer, Region.Customer )
```

Ergibt die Kunden sowohl des aktuellen Artikels als auch der aktuellen Region

```
INTERSECT( ( 10 | 20 | 30 ), ( 20 | 30 | 40 ) ) --> 20 | 30
```

Siehe auch

JOIN (auf Seite 386)

ISCHILDOF

Syntax

```
<ischildof-expression> := 'ISCHILDOF(' <Key> ',' <Key> ')'
```

Seit

1.2

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft ob alle Schlüssel des ersten Parameter gleich oder Kinder der Schlüssel aus dem zweiten Parameter sind. Ein Schlüssel A ist Kind eines Schlüssels B, wenn B der direkte Vater von A ist, oder der Vater von A Kind von B ist.

Wenn einer der Parameter NULL ist, gibt diese Funktion NULL zurück. Wenn der erste Parameter leer ist ergibt die Funktion TRUE.

Beispiele

```
ISCHILDOF( Time:'Jan/2003', Time:'2003' ) --> TRUE
```

```
ISCHILDOF( Time:'2003', Time:'Jan/2003' ) --> FALSE
```

```
ISCHILDOF( Time:'2003', NULL ) --> NULL
```

```
ISCHILDOF( NULL, Time:'Jan/2003' ) --> NULL
```

Siehe auch

ISPARENTOF (auf Seite 386)

ISNULL

Syntax

```
<isnull-expression> := 'ISNULL(' <Any> ')'
```

Seit

1.2

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob der Parameter mindestens eine NULL enthält.

Beispiele

```
ISNULL( NULL ) --> TRUE
```

```
ISNULL( 10 ) --> FALSE
```

```
ISNULL( ( 10 | NULL ) ) --> TRUE
```

Siehe auch

EXISTS (auf Seite 370)

ISPARENTOF

Syntax

```
<isparentof-expression> := 'ISPARENTOF(' <Key> ',' <Key>
')'
```

Seit

1.2

Ergebnis-Typ

Boolean

Beschreibung

Überprüft, ob alle Schlüssel aus dem ersten Parameter Vater von jedem Schlüssel aus dem zweiten Parameter sind. Ein Schlüssel A ist Vater eines Schlüssels B, wenn B direkter Nachfahre von A ist oder Ein Kind von A Vater von B ist.

Wenn einer der Parameter NULL ist ergibt die Funktion NULL. Wenn der erste Parameter eine leere Liste ist gibt die Funktion TRUE zurück.

Beispiele

```
ISPARENTOF( Time:'2003', Time:'Jan/2003' ) --> TRUE
```

```
ISPARENTOF( Time:'Jan/2003', Time:'2003' ) --> FALSE
```

```
ISPARENTOF( Time:'2003', NULL ) --> NULL
```

```
ISPARENTOF( NULL, Time:'2003' ) --> NULL
```

Siehe auch

ISCHILDOF (auf Seite 384)

JOIN

Syntax

```
<join-expression> := 'JOIN(' <Any> { ',' <Any> } ')'
```

```
<join-expression> := <Any> '|' <Any>
```

Seit

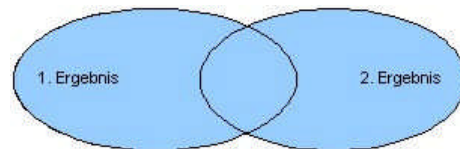
1.2

Ergebnis-Typ

Abhängig von den Typen der Parameter (der gemeinsame Supertyp).

Beschreibung

Die Funktion fügt alle Ergebnisse aller Parameter zu einer Liste zusammen und gibt diese zurück. Die Elemente des ersten Parameters werden als erstes zur Liste hinzugefügt, dann die des zweiten usw. Wenn einer der Parameter NULL ist wird diese NULL einfach zum Ergebnis hinzugefügt.



Anstelle der Funktion JOIN können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "|" verwenden

Beispiele

```
JOIN( LEVEL( Time, 1 ), LEVEL( Time, 2 ) )
```

entspricht LEVEL(Time, 1, 2)

```
JOIN( NULL ) --> NULL
```

Siehe auch

DISTINCT (auf Seite 364), INTERSECT (auf Seite 383)

LAST

Syntax

```
<last-expression> := 'LAST(' <Any> [ ',' <Integer> ] )'
```

Seit

1.2

Ergebnis-Typ

Der Typ des ersten Parameters.

Beschreibung

Diese Funktion gibt die letzten N Elemente des ersten Parameters zurück. Wenn Sie keine Grösse N (mit dem zweiten Parameter) angeben, dann wird nur das letzte Element zurückgegeben. Wenn einer der Parameter NULL ist, dann wird auch NULL zurückgegeben.

Beispiele

```
LAST( LEVEL( Time, 2 ) )
```

Ergibt z.B. Time:'Dez/2003'

```
LAST( LEVEL( Time, 2 ), 3 )
```

Ergibt z.B. Time:'Okt/2003' + Time:'Nov/2003' + Time:'Dez/2003'

```
LAST( NULL ) --> NULL
```

Siehe auch

FIRST (auf Seite 374)

LEAFS

Syntax

```
<leafs-expression> := 'LEAFS(' <Key> ')'
```

Seit

1.1

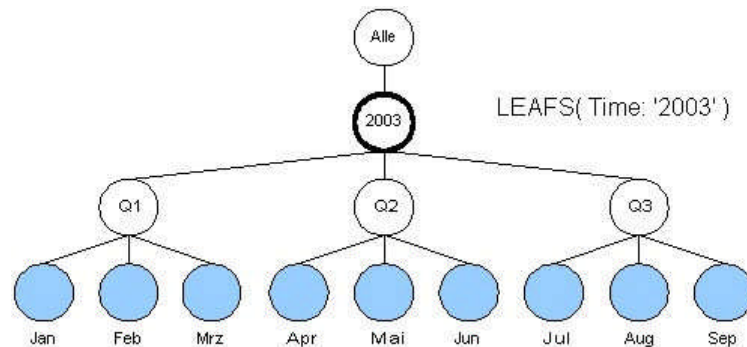
Ergebnis-Typ

Key

Beschreibung

Gibt alle Schlüssel unterhalb der Schlüssel aus dem Parameter zurück, die keine eigenen Kinder mehr besitzen (diese werden dann Leafs = Blätter genannt). Die Leafs müssen keine unmittelbaren Kinder der Ausgangsschlüssel sein und können sich auch weiter unten in der Hierarchie befinden.

Wenn der Parameter NULL ist gibt die Funktion NULL zurück. Wenn der Parameter bereits nur aus Leafs besteht wird eine leere Menge zurückgegeben.



Beispiele

`LEAFS(Time: '2003')`

Ergibt z.B. `Time:'Jan/2003' + ... + Time:'Dez/2003'`

`LEAFS(NULL) --> NULL`

Siehe auch

CHILDREN (auf Seite 355), FAMILY (auf Seite 372), NONLEAFS (auf Seite 406)

LEFT

Syntax

`<left-expression> := 'LEFT(' <String> ',' <Integer> ')'`

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion LEFT gibt die ersten N (definiert durch den zweiten Parameter) Zeichen aus der Zeichenkette zurück, die als erster Parameter übergeben wird. Wenn die Länge der Zeichenkette kleiner als N ist, wird die komplette Zeichenkette zurückgegeben.

Wenn einer der Parameter NULL ist, wird NULL zurückgegeben.

Beispiele

```
LEFT( 'Hello world', 5 ) --> 'Hello'
```

```
LEFT( NULL, 5 ) --> NULL
```

```
LEFT( 'Hello world', NULL ) --> NULL
```

Siehe auch

RIGHT (auf Seite 414), SUBSTR (auf Seite 419)

LESS

Syntax

```
<less-expression> := 'LESS(' <Any> ',' <Any> ')'
```

```
<less-expression> := <Any> '<' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft für jeden Wert des ersten Parameters ob dieser kleiner ist als der entsprechende Wert aus dem zweiten Parameter. Wenn die Parameter eine verschiedene Anzahl von Werten enthalten, gibt die Funktion FALSE zurück. Wenn einer der Werte NULL ist, gibt diese Funktion NULL zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "<" verwenden.

Beispiele

```
LESS( 1, 2 ) --> TRUE
```

```
1 < 2 --> TRUE
```

```
LESS( 2, 2 ) --> FALSE
```

```
2 < 2 --> FALSE
```

```
LESS( NULL, 2 ) --> NULL
```

```
LESS( JOIN( 2, 3 ), JOIN( 1, 2 ) ) --> FALSE
```

Siehe auch

EQUAL (auf Seite 369), GREATER (auf Seite 376), GREATER_OR_EQUAL (auf Seite 377), LESS_OR_EQUAL (auf Seite 391), UNEQUAL (auf Seite 425)

LESS_OR_EQUAL

Syntax

```
<lessorequal-expression> := 'LESSOREQUAL(' <Any> ',' <Any> ')'
```

```
<lessorequal-expression> := <Any> '<=' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft für jeden Wert des ersten Parameters ob dieser kleiner oder gleich als der entsprechende Wert aus dem zweiten Parameter ist. Wenn die Parameter eine verschiedene Anzahl von Werten enthalten, gibt die Funktion FALSE zurück. Wenn einer der Werte NULL ist, gibt diese Funktion NULL zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "<=" verwenden.

Beispiele

```
LESS_OR_EQUAL( 1, 2 ) --> TRUE
```

```
1 <= 2 --> TRUE
```

```
LESS_OR_EQUAL( 2, 2 ) --> TRUE
```

```
2 <= 2 --> TRUE
```

```
LESS_OR_EQUAL( NULL, 2 ) --> NULL
```

```
LESS_OR_EQUAL( JOIN( 2, 3 ), JOIN( 2, 2 ) ) --> TRUE
```

Siehe auch

EQUAL (auf Seite 369), GREATER (auf Seite 376),
GREATER_OR_EQUAL (auf Seite 377), LESS (auf Seite 390),
UNEQUAL (auf Seite 425)

LEVEL

Syntax

```
<level-expression> := 'LEVEL(' <Key> { ',' <Integer> } ')'
```

Seit

1.1

Ergebnis-Typ

Key

Beschreibung

Diese Funktion gibt komplette Ebenen einer Dimensions (definiert durch Parameter 1) zurück. Die gewünschten Ebenen werden durch die optionalen Parameter 2 - n definiert. Wenn der erste Parameter NULL (auf Seite 343) ist gibt diese Funktion NULL zurück. Wenn keine Ebene definiert wird gibt die Funktion eine leere Liste zurück.

Beachten Sie, dass diese Funktion nicht von der aktuellen Selektion (dem Filter) beeinflusst wird - sie gibt immer vollständige und ungefilterte Ebenen zurück.

Beispiele

```
LEVEL( Time, 0 )
```

Ergibt den Time:'Complete Period' (Root-Key der Dimension Time)

```
LEVEL( Time, 1, 2 )
```

Ergibt z.B. Time:'2003' + Time:'Jan/2003' + ... + Time:'Dez/2003'

```
LEVEL( Time ) --> Leeres Ergebnis
```

```
LEVEL( NULL ) --> NULL
```

```
LEVEL( Time, NULL ) --> NULL
```

Siehe auch

ALL (auf Seite 348), LEVELOF (auf Seite 393)

LEVELNAMES

Syntax

```
<levelnames-expression> := 'LEVELNAMES(' <Key> ')'
```

Seit

2.1

Ergebnis-Typ

String

Beschreibung

Diese Funktion ermittelt die Namen aller Hierarchie-Ebenen der Dimension, die als Parameter übergeben wird, und gibt sie in einer Liste zurück. Die Namen sind von oben nach unten sortiert. Wenn der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Beispiele

```
LEVELNAMES( Time )
```

Ergibt z.B. 'TimeRoot', 'Year', 'Month', 'Day'

Siehe auch

DEPTH (auf Seite 362)

LEVELOF

Syntax

```
<levelof-expression> := 'LEVELOF(' <Key> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Ermittelt, zu welchen Hierarchie-Ebenen die Schlüssel aus dem Parameter gehören. Die oberste Ebene einer Dimension (in der nur das Wurzel-Element liegt) hat die Ebene 0, alle darunterliegenden Ebenen haben entsprechend höhere Nummer. Wenn der Parameter NULL (auf Seite 343) gibt die Funktion NULL zurück.

Beispiele

```
LEVELOF( Time:'Complete Time' ) --> 0
```

```
LEVELOF( Time:'2000' ) --> 1
```

```
LEVELOF( Time:'Jan/2000' ) --> 2
```

```
LEVELOF( Time:'01.01.2000' ) --> 3
```

```
LEVELOF( NULL ) --> NULL
```

Siehe auch

HASLEVEL (auf Seite 379), POSITIONOF (auf Seite 411)

LIMIT

Syntax

```
<limit-expression> := 'LIMIT(' <String> ',' <Integer> ')'
```

Seit

1.2

Ergebnis-Typ

String

Beschreibung

Diese Funktion limitiert die Zeichenketten, die als erster Parameter übergeben werden, in ihrer Maximallänge. Wenn eine Zeichenkette länger ist als die durch Parameter 2 definierte Maximallänge, dann wird sie auf diese Länge (minus drei Zeichen) reduziert und drei Punkte "..." werden angehängt. Ansonsten werden die Zeichenketten unverändert zurückgegeben.

Wenn einer der Parameter NULL (auf Seite 343) ist gibt die Funktion NULL zurück.

Beispiele

```
LIMIT( 'Hello World', 8 ) --> 'Hello...'
```

Siehe auch

BEAUTIFY (auf Seite 354), SUBSTR (auf Seite 419), TOLOWER (auf Seite 422), TOUPPER (auf Seite 424)

LOOKUP

Syntax

```
<lookup-expression> := 'LOOKUP(' <Key> { ',' <Key> } ')'
```

Seit

2.0

Ergebnis-Typ

Key

Beschreibung

Diese Funktion ermittelt, durch direkte Abfrage der Cubes, für welche Schlüssel des ersten Parameters eine bestimmte Kennzahl (definiert durch die folgenden Parameter) verfügbar ist (also nicht NULL ist). Bei der Abfrage der Cubes wird ebenfalls der aktuelle Filter beachtet. Die LOOKUP-Funktion ist sehr wichtig zum Erstellen von Berichten wenn Sie Kennzahlen von sehr dünn befüllten Cubes darstellen möchten.

Wenn einer der Argumente NULL (auf Seite 343) ist gibt diese Funktion NULL zurück.

Beispiele

```
LOOKUP( LEVEL( Artikel, 1 ), Fact:Amount )
```

Ergibt alle Alle Artikel mit einem Wert für die Kennzahl "Amount" werden ermittelt

Siehe auch

DLOOKUP (auf Seite 366), SORT (auf Seite 415)

MATCH

Syntax

```
<match-expression> := 'MATCH(' <Key> ',' <Boolean> ')'
```

```
<match-expression> := <Key> '?' <Boolean>
```

Seit

1.2

Ergebnis-Typ

Key

Beschreibung

Diese Funktion hilft eine Menge von Schlüsseln mit einem Ausdruck nach einem bestimmten Kriterium zu filtern. Die Funktion gibt die Teilmenge der Schlüssel (aus dem ersten Parameter) zurück, für die der Ausdruck im zweiten Parameter TRUE ergibt.

Wie die Funktionen SORT und FOREACH erzeugt diese Funktionen eine Kopie der aktuellen Selektion (Filter) für jeden Schlüssel und wendet diesen Schlüssel darauf an. Mit diesem Filter wird dann die Match-Expression (der zweite Parameter) ausgeführt. Wenn das Ergebnis TRUE ist, wird der entsprechende Schlüssel zum Ergebnis hinzugefügt.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "?" verwenden.

Beispiele

```
MATCH( LEVEL( Product, 1 ), Product.ProductType = 'Type A' )
```

Ergibt alle Produkte vom Typ "A"

```
LEVEL( Product, 1 ) ? Product.ProductType = 'Type A'
```

Wie oben

Siehe auch

FIND (auf Seite 373), FOREACH (auf Seite 376), SORT (auf Seite 415)

MATRIX

Syntax

```
<matrix-expression> := 'MATRIX(' <Number> ',' <Number> [  
' ,' <Number> [ ',' <Number> ] ] )'
```

Seit

2.0

Ergebnis-Typ

Value

Beschreibung

Die Funktion MATRIX erlaubt den direkten Zugriff auf die Zellen in der aktuellen Pivot-Tabelle. Sie können diese Funktion z.B. dazu verwenden, Summenzeilen oder -spalten in Ihren Berichten darzustellen.

Die Funktion erwartet mindestens zwei Parameter: Die X- und die Y-Position der Zelle dessen Wert sie auslesen möchten. Wenn nur zwei Parameter übergeben werden, dann ermittelt diese Funktion genau eine Zelle. Wenn der dritte und vierte Parameter mit angegeben werden, wird ein gesamter Bereich der Pivot-Tabelle ausgelesen und es werden mehr Werte zurückgegeben. Der dritte Parameter bestimmt die letzte Spalte und der vierte die unterste Zeile des auszulesenden Bereichs. Wenn Sie z.B. 5 Spalten und 5 Zeilen auslesen, dann werden 25 Werte zurückgegeben. Leere Zellen geben eine NULL zurück.

In Verbindung mit dieser Funktion sind auch die Funktionen X, Y, MIN_X, MIN_Y, MAX_X and MAX_Y interessant. Diese Funktionen können Ihnen helfen, die Position der aktuellen Zelle oder die Grösse einer Pivot-Tabelle zu ermitteln.

Beispiele

```
ADD( TO_NUMBER( MATRIX( X(), MIN_Y(), X(), Y - 1() ) ) )
```

Erzeugt eine Summen-Zeile

Siehe auch

MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

MAX

Syntax

```
<max-expression> := 'MAX(' <Number> { ',' <Number> } ')'
```

Seit

1.0

Ergebnis-Typ

Double

Beschreibung

Die MAX-Funktion ermittelt den maximalen Wert aus allen Parametern. NULL-Werte werden bei der Ermittlung des maximalen Wertes nicht beachtet. Wenn alle Werte NULL (auf Seite 343) sind gibt diese Funktion NULL zurück.

Beispiele

```
MAX( Amount( LEVEL( Time, 3 ) )
```

Errechnet den grössten Betrag über alle Tage

Siehe auch

ADD (auf Seite 347), AVG (auf Seite 353), MIN (auf Seite 400), SUM (auf Seite 420)

MAX_X

Syntax

```
<max_x-expression> := 'MAX_X()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Wenn diese Funktion innerhalb von Überschriften oder Zellen einer Pivot-Tabelle verwendet wird, dann gibt sie die Spaltennummer der letzten Spalte zurück. Ausserhalb von Pivot-Tabellen ist diese Funktion nicht zulässig.

Beispiele

```
MATRIX( MAX_X(), Y() )
```

Ergibt den Wert der letzten Spalte in der aktuellen Zeile

Siehe auch

MATRIX (auf Seite 397), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

MAX_Y

Syntax

```
<max_y-expression> := 'MAX_Y()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Wenn diese Funktion innerhalb von Überschriften oder Zellen einer Pivot-Tabelle verwendet wird, dann gibt sie die Zeilennummer der letzten Zeile zurück. Ausserhalb von Pivot-Tabellen ist diese Funktion nicht zulässig.

Beispiele

```
MATRIX(X(), MAX_Y() )
```

Ergibt den Wert der letzten Zeile in der aktuellen Spalte

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

MIN

Syntax

```
<min-expression> := 'MIN(' <Number> { ',' <Number> } ')'
```

Seit

1.0

Ergebnis-Typ

Double

Beschreibung

Die MIN-Funktion ermittelt den kleinsten Wert aus allen Parametern. NULL-Werte werden bei der Ermittlung des maximalen Wertes nicht beachtet. Wenn alle Werte NULL (auf Seite 343) sind gibt diese Funktion NULL zurück.

Beispiele

```
MAX( Amount( LEVEL( Time, 3 ) )
```

Ergibt den kleinsten Betrag über alle Tage

Siehe auch

ADD (auf Seite 347), AVG (auf Seite 353), MAX (auf Seite 398), SUM (auf Seite 420)

MIN_X

Syntax

```
<min_x-expression> := 'MIN_X()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Wenn diese Funktion innerhalb von Überschriften oder Zellen einer Pivot-Tabelle verwendet wird, dann gibt sie die Spaltennummer der erste Spalte zurück. Ausserhalb von Pivot-Tabellen ist diese Funktion nicht zulässig.

Beispiele

```
MATRIX( MAX_X(), Y() )
```

Ergibt den Wert der ersten Spalte in der aktuellen Zeile

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_Y (auf Seite 401), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

MIN_Y

Syntax

```
<min_y-expression> := 'MIN_Y()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Wenn diese Funktion innerhalb von Überschriften oder Zellen einer Pivot-Tabelle verwendet wird, dann gibt sie die Zeilennummer der ersten Zeile zurück. Ausserhalb von Pivot-Tabellen ist diese Funktion nicht zulässig.

Beispiele

```
MATRIX( MAX_X(), Y() )
```

Ergibt den Wert der ersten Zeile in der aktuellen Spalte

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

MOD

Syntax

```
<mod-expression> := 'MOD(' <Number> ',' <Number> ')'
```

```
<mod-expression> := <Number> '%' <Number>
```

Seit

1.2

Ergebnis-Typ

Double

Beschreibung

Diese Funktion berechnet den Divisions-Rest, der beim Teilen des ersten Parameters durch den zweiten entsteht. Wenn einer der beiden Parameter NULL (auf Seite 343) ist dann gibt diese Funktion NULL zurück.

Anstatt dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "%" verwenden.

Beispiele

```
MOD( 5, 2 ) --> 1
```

```
94 % 10 --> 5
```

```
MOD( NULL, 1 ) --> NULL
```

```
10 % NULL --> NULL
```

Siehe auch

ADD (auf Seite 347), DIV (auf Seite 365), MUL (auf Seite 402), SUM (auf Seite 420)

MUL

Syntax

```
<mul-expression> := 'MUL(' <Number> ',' <Number> ')'
```

```
<mul-expression> := <Number> '*' <Number>
```

Seit

1.0

Ergebnis-Typ

Double

Beschreibung

Diese Funktion multipliziert alle Werte aller Parameter miteinander. Wenn einer der Werte NULL ist, dann gibt diese Funktion NULL zurück.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "*" verwenden.

Beispiele

```
MUL( 10, 42 ) --> 420
```

```
10 * 42 --> 420
```

```
10 * 42 * NULL --> NULL
```

Siehe auch

ADD (auf Seite 347), DIV (auf Seite 365), MOD (auf Seite 402), SUM (auf Seite 420)

NEIGHBOURS

Syntax

```
<neighbours-expression> := 'NEIGHBOURS(' <Key> ')'
```

Seit

1.2

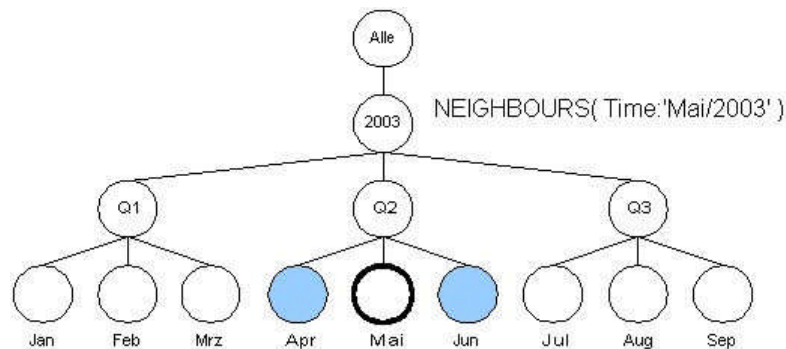
Ergebnis-Typ

Key

Beschreibung

Diese Funktion ermittelt alle "Nachbarn" der Schlüssel, die als Parameter übergeben werden. Die "Nachbarn" eines Schlüssel sind alle Schlüssel mit dem selben Vater ohne den Schlüssel selbst.

Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.



Beispiele

NEIGHBOURS(Time: 'Mai/2003')

Ergibt z.B. Time: 'Apr/2003' + Time: 'Jun/2003'

Siehe auch

CHILDREN (auf Seite 355), LEVEL (auf Seite 392), PARENT (auf Seite 409)

NEXT

Syntax

<next-expression> := 'NEXT(' <Key> [',' <Key>])'

Seit

1.0

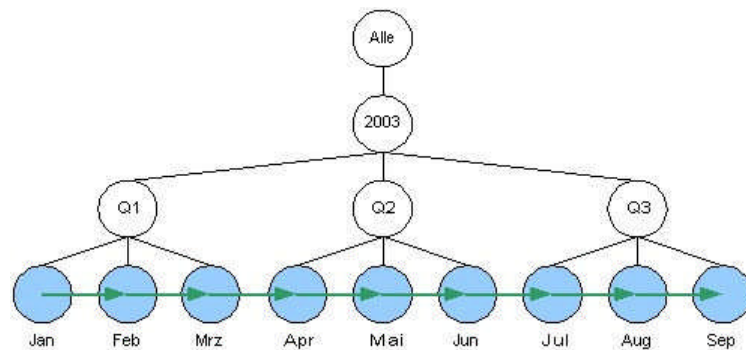
Ergebnis-Typ

Key

Beschreibung

Berechnet die nachfolgenden Schlüssel der Schlüssel aus Parameter 1. Der Nachfolger eines Schlüssels ist der nächste Schlüssel innerhalb der selben Ebene der Dimension. Der Nachfolger muss nicht unbedingt den selben Vater haben wie der Schlüssel selbst.

Wenn kein Abstand (durch Parameter 2) definiert wird, dann wird der direkt nächste Schlüssel zurückgegeben. Wenn eine Distanz N bestimmt wird, dann wird der Schlüssel mit dem Abstand N ermittelt. Z.B. entspricht der Ausdruck "NEXT(Time, 2)" dem Ausdruck "NEXT(NEXT(Time))"



Wenn der Parameter 1 NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück. Wenn der Schlüssel keinen Nachfolger besitzt, dann wird eine leere Liste zurückgegeben.

Beispiele

```
NEXT( Time: 'Jan/2003' ) --> Time: 'Feb/2003'
```

```
NEXT( Time: 'Jan/2003', 3 ) --> Time: 'Apr/2003'
```

```
NEXT( Time: 'Dez/2003' ) --> Leeres Ergebnis
```

```
NEXT( NULL ) --> NULL
```

Siehe auch

PREV (auf Seite 412), UPPERNEXT (auf Seite 426), UPPERPREV (auf Seite 427)

NONFACTROOTS

Syntax

```
<nonfactroots-expression> := 'NONFACTROOTS()'
```

Seit

1.2

Ergebnis-Typ

Key

Beschreibung

Gibt alle Wurzel aller Dimension (ohne der Kennzahl-Dimension) in einer Liste zurück.

Beispiele

```
NONFACTROOTS ( )
```

Ergibt z.B. Time:'Complete Period' + ... + Product:'All Products'

Siehe auch

FACTROOT (auf Seite 371)

NONLEAFS

Syntax

```
<nonleafs-expression> := 'NONLEAFS(' <Key> ')'
```

Seit

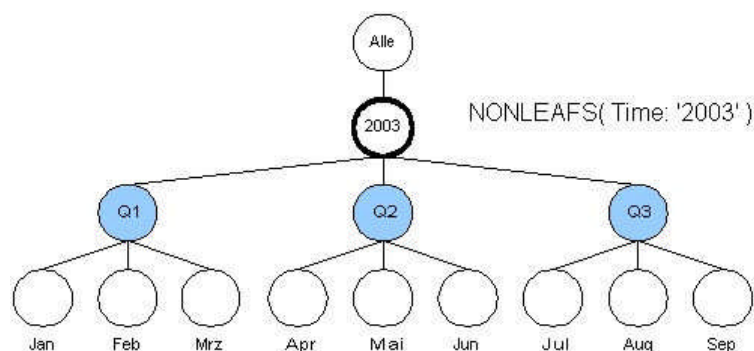
1.2

Ergebnis-Typ

Key

Beschreibung

Die Funktion NONLEAFS ermittelt alle Schlüssel unterhalb der Schlüssel aus dem Parameter, die keine eigenen Kinder besitzen. Die Blätter (Leafs) der Hierarchie werden nicht zum Ergebnis hinzugefügt. Wenn der Parameter NULL (auf Seite 343) ist, dann ist das Ergebnis der Funktion ebenfalls NULL.



Beispiele

```
NONLEAFS( Time: '2000' )
```

Ergibt z.B. Time:'Jan/2000', ..., Time:'Dez/2000'

Siehe auch

ANCESTORS (auf Seite 349), CHILDREN (auf Seite 355), FAMILY (auf Seite 372), LEAFS (auf Seite 388), PARENT (auf Seite 409)

NOT

Syntax

```
<not-expression> := 'NOT(' <Boolean> ')'
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese logische Funktion negiert den Parameter 1:

- Wenn der Parameter TRUE (siehe "Boolean-Konstanten" auf Seite 342) TRUE ist, dann gibt die Funktion FALSE zurück
- Wenn der Parameter FALSE (siehe "Boolean-Konstanten" auf Seite 342) ist, dann gibt die Funktion TRUE zurück
- Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück

Beispiele

```
NOT( TRUE ) --> FALSE
```

```
NOT( FALSE ) --> TRUE
```

```
NOT( NULL ) --> NULL
```

Siehe auch

AND (auf Seite 350), OR (auf Seite 408)

NOW

Syntax

```
<now-expression> := 'NOW(' <Key> ',' <String> ')'
```

Seit

1.2

Ergebnis-Typ

Key

Beschreibung

Die Funktion NOW findet den Schlüssel aus einer Zeit-Dimension, der dem aktuellen Tag, Monat, Jahr etc. entspricht. Solch eine Funktion ist notwendig, da in instantOLAP keine spezielle Zeit-Dimension existiert. Deshalb müssen Sie den aktuellen Schlüssel in Ihrer eigenen Zeit-Dimension mit dieser Funktion suchen.

Um den aktuellen Schlüsseln zu finden müssen Sie die Dimension selbst als ersten Parameter übergeben. Mit dem zweiten Parameter bestimmen die das Suchmuster als Zeichenkette in Form eines Datum-Formats, mit dessen Ergebnis der Schlüssel dann gesucht wird.

Beispiele

```
NOW( Time, 'yyyy' ) --> Time:'2002'
```

```
NOW( Time, 'MMM/yyyy' ) --> Time:'Jan/2002'
```

```
NOW( Time, 'MMM/yy' ) --> NULL
```

Siehe auch

FIND (auf Seite 373)

OR

Syntax

```
<or-expression> := 'OR(' <Boolean> ',' <Boolean> ')'
```

```
<or-expression> := <Boolean> 'OR' <Boolean>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Die logische Funktion OR ergibt TRUE, FALSE oder NULL abhängig von den Parametern:

- Wenn einer der Parameter TRUE (siehe "Boolean-Konstanten" auf Seite 342) ergibt, dann ergibt die Funktion ebenfalls TRUE
- Wenn beide Parameter FALSE (siehe "Boolean-Konstanten" auf Seite 342) ergeben, dann ergibt die Funktion FALSE,
- Wenn einer der Parameter NULL (auf Seite 343) ergibt, dann ergibt die Funktion ebenfalls NULL

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "or" verwenden.

Beispiele

```
OR( TRUE, TRUE ) --> TRUE
```

```
OR( FALSE, TRUE ) --> TRUE
```

```
OR( TRUE, FALSE ) --> TRUE
```

```
OR( TRUE, NULL ) --> TRUE
```

```
OR( FALSE, NULL ) --> NULL
```

```
OR( NULL, NULL ) --> NULL
```

Siehe auch

AND (auf Seite 350), NOT (auf Seite 407)

PARENT

Syntax

```
<parent-expression> := 'PARENT(' <Key> ')'
```

Seit

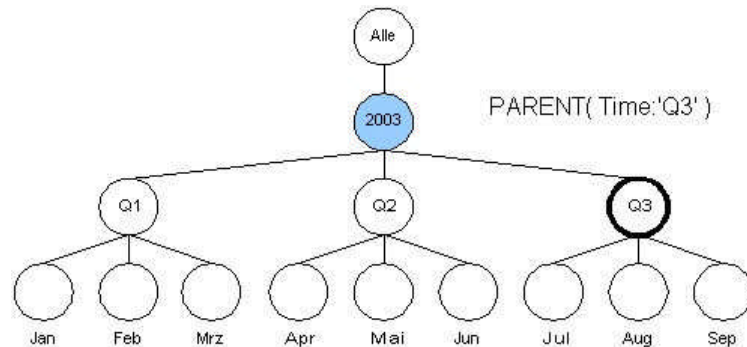
1.0

Ergebnis-Typ

Key

Beschreibung

Die Funktion PARENT ermittelt die Vater der Schlüssel aus dem Parameter. Der Vater ist der Schlüssel, der sich in der Hierarchie direkt über einem Schlüssel befindet. Alle Schlüssel bis auf das Wurzel-Element einer Dimension haben Väter. Wenn der Parameter NULL (auf Seite 343) ist, dann gibt diese Funktion NULL zurück.



Beispiele

```
PARENT( Time:'Jan/2003' )
```

Ergibt z.B. Time:'Q1/2003'

```
PARENT( Time:'Complete Period' )
```

Ergibt NULL (wenn 'Complete Period' der Root-Key ist)

```
PARENT( NULL ) --> NULL
```

Siehe auch

CHILDREN (auf Seite 355), FAMILY (auf Seite 372), ISPARENTOF (auf Seite 386), PEDIGREE (auf Seite 410)

PEDIGREE

Syntax

```
<pedigree-expression> := 'PEDIGREE(' <Key> ')'
```

Seit

2.1.2

Ergebnis-Typ

Key

Beschreibung

Diese Funktion gibt die Schlüssel aus dem Parameter und all ihre "Vorfahren" zurück. Das Ergebnis ist eine eindeutige und sortiert List - auch wenn verschiedene Schlüssel aus dem Parameter die selben Vorfahren besitzen, werden diese nur einmal im Ergebnis erscheinen.

Ist der Parameter NULL (auf Seite 343), dann gibt diese Funktion NULL zurück.

Beispiele

```
NEIGHBOURS( Time:'Jun/2004', Time:'May/2004' )
```

Ergibt z.B. Time:'Complete Period' + Time:'2004' + Time:'May/2004'
+ Time:'Jun/2004'

Siehe auch

ANCESTORS (auf Seite 349)

POSITIONOF

Syntax

```
<positionof-expression> := 'POSITIONOF(' <Key> ')
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Die Funktion POSITIONOF ermittelt die Position eines Schlüssels unterhalb ihres Vaters und gibt diese zurück. Das erste Kind eines Vaters hat die Position 0. Wenn der Parameter mehr als einen Schlüssel umfasst, dann werden entsprechend viele Positionen zurückgegeben. Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
POSITIONOF( Time:'01.01.2002' ) --> 0
```

```
POSITIONOF( Time:'03.01.2002' ) --> 2
```

Siehe auch

LEVELOF (auf Seite 393)

PREV**Syntax**

```
<prev-expression> := 'NEXT(' <Key> [ ',' <Integer> ] )'
```

Seit

1.0

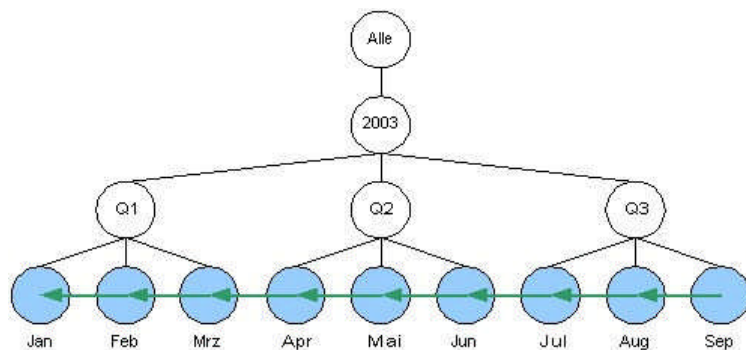
Ergebnis-Typ

Key

Beschreibung

Berechnet die Vorgänger-Schlüssel der Schlüssel aus Parameter 1. Der Vorgänger eines Schlüssels ist der vorherige Schlüssel innerhalb der selben Ebene der Dimension. Der Vorgänger muss nicht unbedingt den selben Vater haben wie der Schlüssel selbst.

Wenn kein Abstand (durch Parameter 2) definiert wird, dann wird der direkt vorhergehende Schlüssel zurückgegeben. Wenn eine Distanz N bestimmt wird, dann wird der Schlüssel mit dem Abstand N ermittelt. Z.B. entspricht der Ausdruck "PREV(Time, 2)" dem Ausdruck PREV(PREV(Time))".



Wenn der Parameter 1 NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück. Wenn der Schlüssel keinen Vorgänger besitzt, dann wird eine leere Liste zurückgegeben.

Beispiele

```
PREV( Time:'Dez/2003' ) --> Time:'Nov/2003'
```

```
PREV( Time:'Dez/2003', 3 ) --> Time:'Sep/2003'
```

```
PREV( Time:'Jan/2003 ) --> NULL
```

```
PREV( NULL ) --> NULL
```

Siehe auch

NEXT (auf Seite 404), UPPERNEXT (auf Seite 426), UPPERPREV (auf Seite 427)

REPLACE

Syntax

```
<replace-expression> := 'REPLACE(' <String> ',' <String> ',' <String> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion REPLACE ersetzt in der ersten Zeichenjette alle Vorkommnisse der Zeichenkette, die durch den zweiten Parameter definiert wird, durch den die Zeichenkette des dritten Parameters. Wenn einer der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
REPLACE( 'Hello World', 'World', 'Customer' ) --> 'Hello Customer'
```

Siehe auch

LEFT (auf Seite 389), RIGHT (auf Seite 414), STRLEN (auf Seite 418), SUBSTR (auf Seite 419)

REVERSE

Syntax

```
<reverse-expression> := 'REVERSE(' <Any> ')'
```

Seit

1.2

Ergebnis-Typ

Entspricht dem Typ des Parameters.

Beschreibung

Diese Funktion gibt alle Werte aus dem Parameter in umgekehrter Reihenfolge (rückwärts) zurück.

Beispiele

```
REVERSE( LEVEL( Zeit, 3 ) )
```

Ergibt z.B. Time:'Dez/2003' + ... + Time:'Jan/2003'

Siehe auch

INTERSECT (auf Seite 383), JOIN (auf Seite 386), WITHOUT (auf Seite 428)

RIGHT

Syntax

```
<right-expression> := 'RIGHT(' <String> ',' <Integer> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion RIGHT gibt die letzten N (definiert durch den zweiten Parameter) Zeichen aus der Zeichenkette zurück, die als erster Parameter übergeben wird. Wenn die Länge der Zeichenkette kleiner als N ist, wird die komplette Zeichenkette zurückgegeben.

Wenn einer der Parameter NULL (auf Seite 343) ist, wird NULL zurückgegeben.

Beispiele

```
RIGHT( 'Hello world', 5 ) --> 'world'
```

```
RIGHT( NULL, 5 ) --> NULL
```

```
RIGHT( 'Hello world', NULL ) --> NULL
```

Siehe auch

LEFT (auf Seite 389), SUBSTR (auf Seite 419)

ROUND

Syntax

```
<round-expression> := 'ROUND(' <Number> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Die Funktion ROUND gibt den nächsten Integer-Wert zum Parameter zurück. Das Ergebnis wird zur Integer-Zahl gerundet, indem 0.5 addiert und dann der Teil vor dem Komma genommen wird.

Wenn der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
ROUND( 0.5 ) --> 1
```

```
ROUND( 0.4 ) --> 0
```

```
ROUND( NULL ) --> NULL
```

Siehe auch

CEIL (auf Seite 354), FLOOR (auf Seite 375)

SORT

Syntax

```
<sort-expression> := 'SORT(' <Key> ',' <Value> [ ','  
<Integer> [ ',' <Boolean> ] ] )'
```

Seit

2.0

Ergebnis-Typ

Key

Beschreibung

Mit der Funktion SORT kann eine Menge von Schlüsseln an einem beliebigen Kriterium sortiert werden. Der erste Parameter bestimmt die Schlüssel, die sortiert werden sollen, der zweite das Sortier-Kriterium selbst. Wie bei den Funktionen MATCH und FOREACH wird für jeden Schlüssel aus dem ersten Parameter eine Kopie des aktuellen Filters erstellt, auf dem der Schlüssel angewendet wird. Mit diesem Filter wird dann die Kriterium-Funktion ausgeführt. Nach diesem Ergebnis werden dann die Schlüssel sortiert.

Mit dem (optionalen) dritten Parameter können Sie eine maximale Größe für das Ergebnis bestimmen. Das ermöglicht Ihnen die ersten oder letzten N Schlüssel aus der sortierten Liste zu ermitteln und so z.B. eine Top-10 Liste zu ermitteln. Der Standard-Wert für den dritten Parameter ist NULL, bei dem keine Limitierung vorgenommen wird.

Der (ebenfalls optionale) vierte Parameter bestimmt die Sortier-Reihenfolge des Ergebnisses: Wenn der Parameter TRUE ist, dann werden die Schlüssel absteigend sortiert. Ist er FALSE (die Standard-Einstellung), dann werden sie aufsteigend sortiert.

Wie die LOOKUP-Funktion kann auch diese Funktion ggf. direkte Abfragen auf die Cubes ausführen, z.B. durch das unmittelbare Senden eines SQL-Statements an eine Datenbank.

Beispiele

```
SORT( LEVEL( Article, 3 ), Artikel.Farbe )
```

Ergibt alle Artikel, sortiert nach ihrer Farbe

```
SORT( LEVEL( Article, 3 ), Amount(), 10, true )
```

Ergibt die Top-10 Artikel, sortiert nach dem Umsatz

Siehe auch

FOREACH (auf Seite 376), LOOKUP (auf Seite 395), MATCH (auf Seite 396)

SPLIT

Syntax

```
<split-expression> := 'SPLIT(' <String> ',' <String> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion SPLIT kann verwendet werden, um eine Zeichenkette in mehrere Teile aufzuteilen. Der erste Parameter ist die Zeichenkette, die aufgeteilt werden soll, der zweite der Trenner, nachdem die Zeichenkette aufgeteilt werden soll. Das Ergebnis ist eine Liste von Zeichenketten, jeweils ohne den Trenner.

Wenn ein Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
SPLIT( 'Split,this,string', ',' ) --> 'Split' | 'this' |  
'string'
```

```
SPLIT( NULL, ',' ) --> NULL
```

```
SPLIT( 'Split,this,string', NULL ) --> NULL
```

Siehe auch

LEFT (auf Seite 389), RIGHT (auf Seite 414), STRLEN (auf Seite 418),
SUBSTR (auf Seite 419)

STARTSWITH

Syntax

```
<startswith-expression> := 'STARTSWITH(' <String>, <String>  
)'
```

Seit

2.1

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion ermittelt, ob die Zeichenkette aus dem ersten Parameter mit der Zeichenkette aus dem zweiten Parameter beginnt. Wenn einer der Parameter NULL (auf Seite 343) ist, dann gibt die Funktion NULL zurück.

Beispiele

```
STARTSWITH( 'Hello World', 'World' ) --> FALSE
```

```
STARTSWITH( 'Hello World', 'Hello' ) --> TRUE
```

```
STARTSWITH( 'Hello World', 'NULL' ) --> NULL
```

Siehe auch

ENDSWITH (auf Seite 368)

STRLEN

Syntax

```
<strlen-expression> := 'STRLEN(' <String> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Die Funktion STRLEN ermittelt die Länge der Zeichenkette(n), die als Parameter übergeben werden. Wenn der Parameter NULL (auf Seite 343) ist, gibt die Funktion auch NULL zurück.

Beispiele

```
STRLEN( 'Hello world' ) --> 11
```

```
STRLEN( NULL ) --> NULL
```

Siehe auch

LEFT (auf Seite 389), RIGHT (auf Seite 414), SPLIT (auf Seite 417),
SUBSTR (auf Seite 419)

SUB

Syntax

```
<number-expression> := 'SUB(' <Number> ',' <Number> ')'
```

```
<number-expression> := <Number> '-' <Number>
```

Seit

1.0

Ergebnis-Typ

Number

Beschreibung

Diese Funktion ermittelt die Differenz zwischen Parameter 1 und Parameter 2. Wenn einer der Parameter NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Anstatt der Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "-" verwenden.

Beispiele

```
SUB( 10, 5 ) --> 5
```

```
-10 - 20 --> -30
```

```
SUB( 10, NULL ) -->NULL
```

Siehe auch

ADD (auf Seite 347), DIV (auf Seite 365), MUL (auf Seite 402)

SUBSTR

Syntax

```
<substr-expression> := 'SUBSTR(' <String> ',' <Integer> ','  
<Integer> ')'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Die Funktion SUBSTR ermittelt einen Ausschnitt der Zeichenkette, die als erster Parameter übergeben wird. Der Ausschnitt beginnt beim Index, der durch den zweiten Parameter bestimmt wird, und endet bei Index, der durch den dritten Parameter (-1) bestimmt wird. Die Länge des Ergebnisses beträgt daher Parameter 2 - Parameter 1.

Wenn einer der Parameter NULL (auf Seite 343) ist gibt die Funktion NULL zurück.

Beispiele

```
SUBSTR( 'Hello Welt', 0, 5 ) --> 'Hello'
```

```
SUBSTR( NULL, 1, 5 ) --> NULL
```

Siehe auch

LEFT (auf Seite 389), RIGHT (auf Seite 414), STRLEN (auf Seite 418)

SUM

Syntax

```
<sum-expression> := 'SUM(' <Number> [ ',' { <Number> } ] )'
```

Seit

2.0.3

Ergebnis-Typ

Double

Beschreibung

Diese Funktion berechnet die Summe aller Werte aller Parameter. Im Gegensatz zur ADD-Funktion akzeptiert die Funktion nur Zahlen und gibt auch nur Zahlen zurück.

Wenn einer der Werte NULL (auf Seite 343) ist, gibt die Funktion die Summe aller anderen Werte zurück. Wenn alle Werte NULL sind, gibt die Funktion NULL zurück.

Beispiele

```
SUM( 10, 20 ) --> 30
```

```
SUM( Amount( CHILDREN( Product ) ) )
```

Ergibt die Summe aller Beträge aller (Unter-) Produkte

Siehe auch

ADD (auf Seite 347)

TIMESTAMP

Syntax

```
<timestamp-expression> := 'TIMESTAMP(' <String> ')'
```

Seit

1.2

Ergebnis-Typ

String

Beschreibung

Diese Funktion gibt das aktuelle Datum als Zeichenkette zurück. Das Format das Datum wird durch den Parameter bestimmt, der ein Datums-Format enthalten muss. Wenn das Format NULL (auf Seite 343) ist gibt die Funktion NULL zurück.

Beispiele

```
TIMESTAMP( 'yyyy' ) --> '2002'
```

```
TIMESTAMP( 'yyyyMMdHHmm' ) --> '200208011321'
```

Siehe auch

NOW (auf Seite 408)

TOLOWER

Syntax

```
<tolower-expression> := 'TOLOWER(' <String> ')'
```

Seit

1.2

Ergebnis-Typ

String

Beschreibung

Wandelt alle Buchstaben der Zeichenkette in Kleinbuchstaben um. Wenn der Parameter NULL (auf Seite 343) ist wird NULL zurückgegeben.

Beispiele

```
TOLOWER( 'Hello World' ) --> 'hello world'
```

Siehe auch

BEAUTIFY (auf Seite 354), TOUPPER (auf Seite 424)

TONUMBER

Syntax

```
<tonumber-expression> := 'TONUMBER(' <Any> ')'
```

Seit

1.2

Ergebnis-Typ

Number

Beschreibung

Diese Funktion wandelt den Wert aus dem Parameter (wenn möglich) in eine Zahl um:

- Wenn der Wert eine Zeichenkette ist wird dieser geparsed. Abhängig vom Inhalt der Zeichenkette ist das Ergebnis eine ganze (Integer) Zahl, (Double) Zahl oder NULL (unkonvertierbar).

- Wenn der Wert eine Zahl ist, wird diese unverändert zurückgegeben.
- Für jeden anderen Typ wird NULL (auf Seite 343) zurückgegeben.
- Für NULL (auf Seite 343) wird NULL zurückgegeben.

Beispiele

```
TONUMBER( '10' ) --> 10
```

```
TONUMBER( '10.0' ) --> 10
```

```
TONUMBER( '10.2' ) --> 10.2
```

```
TONUMBER( 10 ) --> 10
```

```
TONUMBER( 'Hello World' ) --> NULL
```

```
TONUMBER( NULL ) --> NULL
```

Siehe auch

TOSTRING (auf Seite 423)

TOSTRING

Syntax

```
<tostring-expression> := 'TOSTRING(' <Any> [ ',' <String> ]  
' )'
```

Seit

1.1

Ergebnis-Typ

String

Beschreibung

Die Funktion TOSTRING konvertiert Parameter jeden Typ in eine Zeichenkette. Abhängig vom Typ des Parameters finden verschiedene Konvertierungen statt:

- Zeichenketten (String) werden ohne Veränderungen zurückgegeben
- Ganze Zahlen (Integer) werden in einen Text ohne Nachkommawert umgewandelt
- Reelle Zahlen (Double) werden in einen Text mit Nachkommawert umgewandelt
- Logische Wert (Boolean) werden in 'TRUE' oder 'FALSE' umgewandelt

- Datumsangaben (Date) werden mit dem lokalen Datumsformat umgewandelt
- Von Schlüssel (Key) wird die ID zurückgegeben
- Für NULL (auf Seite 343) gibt die Funktion NULL zurück

Mit dem (optionalen) zweiten Parameter können Sie die Konvertierung genauer definieren und ein Zahlen- oder Datumsformat (abhängig vom Typ des Wertes) übergeben.

Beispiele

```
TOSTRING( 10 ) --> '10'
```

```
TOSTRING( 10.2 ) --> '10.2'
```

```
TOSTRING( 10.2, '0.00' ) --> '10.20'
```

```
TOSTRING( Time:'Jan/2003' ) --> 'Jan/2003'
```

```
TOSTRING( TRUE ) --> 'TRUE'
```

```
TOSTRING( NULL ) --> NULL
```

Siehe auch

TONUMBER (auf Seite 422)

TOUPPER

Syntax

```
<toupper-expression> := 'TOUPPER(' <String> ')'
```

Seit

1.2

Ergebnis-Typ

String

Beschreibung

Wandelt alle Buchstaben der Zeichenkette in Grossbuchstaben um. Wenn der Parameter NULL (auf Seite 343) ist wird NULL zurückgegeben.

Beispiele

```
TOLOWER( 'Hello World' ) --> 'HELLO WORLD'
```

```
TOUPPER( NULL ) --> NULL
```

Siehe auch

BEAUTIFY (auf Seite 354), TOLOWER (auf Seite 422)

UNEQUAL

Syntax

```
<unequal-expression> := 'UNEQUAL(' <Any> ',' <Any> ')'
```

```
<unequal-expression> := <Any> '<>' <Any>
```

Seit

1.0

Ergebnis-Typ

Boolean

Beschreibung

Diese Funktion überprüft, ob die Werte aus beiden Parametern nicht gleich sind. Diese Funktion ergibt genau das negierte Ergebnis der EQUAL-Funktion. Wenn einer der Parameter NULL (auf Seite 343) ist wird NULL zurückgegeben.

Anstelle dieser Funktion können Sie auch den Operator (siehe "Operatoren" auf Seite 334) "<>" verwenden.

Beispiele

```
UNEQUAL( 10, 20 ) --> TRUE
```

```
10 <> 20 --> TRUE
```

```
UNEQUAL( 10, 10 ) --> FALSE
```

```
UNEQUAL( 10, 'Hello World' ) --> TRUE
```

```
UNEQUAL( 10, NULL ) --> NULL
```

Siehe auch

EQUAL (auf Seite 369), GREATER (auf Seite 376),
GREATER_OR_EQUAL (auf Seite 377), LESS (auf Seite 390),
LESS_OR_EQUAL (auf Seite 391)

UPPERNEXT

Syntax

```
<uppernext-expression> := 'UPPERNEXT(' <Key> ')'
```

Seit

1.1

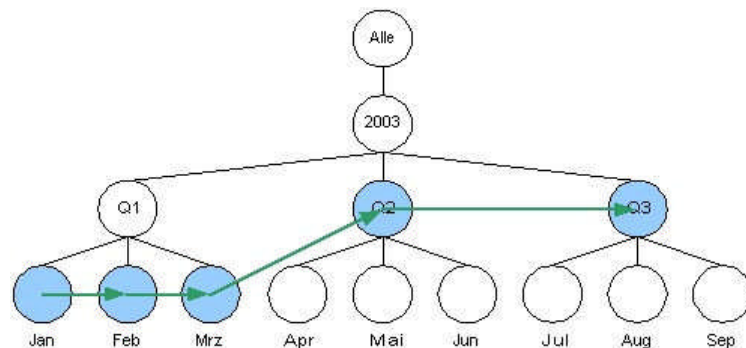
Ergebnis-Typ

Key

Beschreibung

Die Funktion UPPERNEXT ermittelt, wie die Funktion NEXT, den Nachfolger eines Schlüssels. Aber im Gegensatz zur Funktion NEXT wird für das letzte Kind eines Vaters nicht der Nachfolger, sondern der Nachfolger des Vaters selbst zurückgegeben.

Diese Funktion (und die Funktion UPPERPREV) sind sehr hilfreich bei der Berechnung von Aggregationen und Year-To-Date Berechnungen, da sie auf höher aggregierte Schlüssel zugreifen.



Wenn der Parameter NULL ist gibt diese Funktion NULL zurück.

Beispiele

```
UPPERNEXT( Time:'Nov/2003' ) --> Time:'Dec/2003'
```

```
UPPERNEXT( Time:'Dec/2003' ) --> Time:'2004'
```

```
UPPERNEXT( NULL ) --> NULL
```

Siehe auch

NEXT (auf Seite 404), PREV (auf Seite 412), UPPERPREV (auf Seite 427)

UPPERPREV

Syntax

```
<upperprev-expression> := 'UPPERPREV(' <Key> ')'
```

Seit

1.1

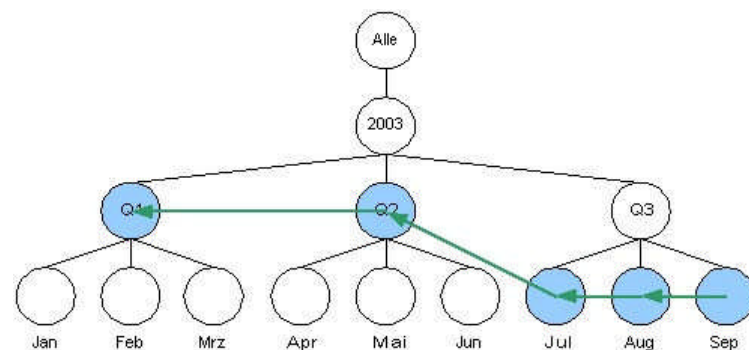
Ergebnis-Typ

Key

Beschreibung

Die Funktion UPPERPREV ermittelt, wie die Funktion PREV, den Vorgänger eines Schlüssels. Aber im Gegensatz zur Funktion PREV wird für das erste Kind eines Vaters nicht der Vorgänger, sondern der Vorgänger des Vaters selbst zurückgegeben.

Diese Funktion (und die Funktion UPPERNEXT) sind sehr hilfreich bei der Berechnung von Aggregationen und Year-To-Date Berechnungen, da sie auf höher aggregierte Schlüssel zugreifen.



Wenn der Parameter NULL (auf Seite 343) ist gibt diese Funktion NULL zurück.

Beispiele

```
UPPERPREV( Time:'Feb/2004' ) --> Time:'Jan/2004'
```

```
UPPERPREV( Time:'Jan/2004' ) --> Time:'2003'
```

```
UPPERPREV( NULL ) --> NULL
```

Siehe auch

NEXT (auf Seite 404), PREV (auf Seite 412), UPPERNEXT (auf Seite 426)

USER

Syntax

```
<User-Expression> := 'USER()'
```

Seit

2.0

Ergebnis-Typ

String

Beschreibung

Funktion USER gibt den Login-Namen des aktuellen Benutzers zurück. Das Ergebnis dieser Funktion ist äquivalent zur Variable \$USER.

Beispiele

```
USER()
```

Ergibt z.B. 'admin' (d.h. der aktuelle Benutzer ist 'admin')

Siehe auch

HASROLES (auf Seite 380), HASUSER (auf Seite 381)

WITHOUT

Syntax

```
<without-expression> := 'WITHOUT(' <Any> ',' <Any> ')'
```

Seit

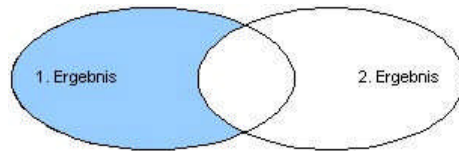
1.2

Ergebnis-Typ

Der gemeinsame Supertyp beider Parameter.

Beschreibung

Die Funktion WITHOUT gibt alle Werte aus dem ersten Parameter zurück, die nicht im zweiten Parameter vorkommen. Wenn einer der Parameter NULL (auf Seite 343) ist wird NULL zurückgegeben.



Beispiele

```
WITHOUT( CHILDREN( Jahr:'2004' ), Zeit:'Jan/2004' )
```

Ergibt alle Monate aus dem Jahr 2004 ohne Januar

Siehe auch

INTERSECT (auf Seite 383), JOIN (auf Seite 386)

X

Syntax

```
<x-expression> := 'X()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion ermittelt die aktuelle X-Position (Spaltennummer) des Headers oder der Zelle einer Pivot-Tabelle, in der sie gerade ausgeführt wird. Ausserhalb von Pivot-Tabellen ist der Aufruf dieser Funktion nicht zulässig.

Beispiele

```
background="IIF( X() % 2 = 0, 'grey', 'white' )"
```

In einem Query-Header kann die Funktion X() z.B. zum Einfärben von Spalten verwendet werden

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), XHEADER (auf Seite 430), Y (auf Seite 431), YHEADER (auf Seite 431)

XHEADER

Syntax

```
<xheader-expression> := 'XHEADER(' <String> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Mit Hilfe der XHEADER Funktion kann die Position einer oder mehrerer Überschriften aus der X-Achse der aktuellen Pivot-Tabelle ermittelt werden. Der Parameter ist ein Suchmuster, mit dem die Überschriften gesucht werden. Das Ergebnis ist eine Liste von Spaltennummern oder NULL, wenn keine Überschrift gefunden wurde.

Im Suchmuster können sie die Platzhalter "*" und "?" verwenden. Wenn das Pattern NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Der Aufruf dieser Funktion ausserhalb von Pivot-Tabellen ist nicht zulässig.

Beispiele

```
XHEADER( 'Sum' )
```

Ergibt die Position der Überschrift 'Sum'

```
XHEADER( 'Article*' )
```

Ergibt die Positionen aller Überschriften, die mit 'Article' beginnen

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), X (auf Seite 429), Y (auf Seite 431), YHEADER (auf Seite 431)

Y

Syntax

```
<y-expression> := 'Y()'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Diese Funktion ermittelt die aktuelle Y-Position (Zeilennummer) des Headers oder der Zelle einer Pivot-Tabelle, in der sie gerade ausgeführt wird. Ausserhalb von Pivot-Tabellen ist der Aufruf dieser Funktion nicht zulässig.

Beispiele

```
background="IIF( Y() % 2 = 0, 'grey', 'white' )"
```

In einem Query-Header kann die Funktion Y() z.B. zum Einfärben von Zeilen verwendet werden

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 400), X (auf Seite 429), XHEADER (auf Seite 430), YHEADER (auf Seite 431)

YHEADER

Syntax

```
<yheader-expression> := 'YHEADER(' <String> ')'
```

Seit

2.0

Ergebnis-Typ

Integer

Beschreibung

Mit Hilfe der YHEADER Funktion kann die Position einer oder mehrerer Überschriften aus der Y-Achse der aktuellen Pivot-Tabelle ermittelt werden. Der Parameter ist ein Suchmuster, mit dem die Überschriften gesucht werden. Das Ergebnis ist eine Liste von Spaltennummern oder NULL, wenn keine Überschrift gefunden wurde.

Im Suchmuster können sie die Platzhalter "*" und "?" verwenden. Wenn das Pattern NULL (auf Seite 343) ist, gibt die Funktion NULL zurück.

Der Aufruf dieser Funktion ausserhalb von Pivot-Tabellen ist nicht zulässig.

Beispiele

```
YHEADER( 'Sum' )
```

Ergibt Position der Überschrift 'Sum'

```
YHEADER( 'Article*' )
```

Ergibt die Positionen aller Überschriften, die mit 'Article' beginnen

Siehe auch

MATRIX (auf Seite 397), MAX_X (auf Seite 398), MAX_Y (auf Seite 399), MIN_X (auf Seite 400), MIN_Y (auf Seite 401), X (auf Seite 429), XHEADER (auf Seite 430), Y (auf Seite 431)

ZERO

Syntax

```
<zero-expression> := 'ZERO(' <Number> ')'
```

Seit

1.1

Ergebnis-Type

Number

Beschreibung

Die Funktion ZERO wandelt alle NULL (auf Seite 343)-Werte aus dem Parameter in die ganze Zahl "0" um. Alle anderen Werte werden ohne Veränderung zurückgegeben. Da die meisten Funktionen nicht mit NULL rechnen können, dient diese Funktion ggf. zur Vorkonvertierung von NULL in eine echte Zahl.

Beispiele

```
ZERO( Amount( ) )
```

```
AVG( Zero( Amount( LEAFS( Zeit ) ) ) ) )
```

Siehe auch

CUBE (auf Seite 359), EXISTS (auf Seite 370), ISNULL (auf Seite 385)

KAPITEL 6

Formate

In diesem Kapitel

Cron-Patterns	436
Datums-Formate	437
Farben	439
Zahlen-Formate	440

Cron-Patterns

Syntax

```
cron-pattern := [ [ [ [ <month-pattern> ' ' ] <day-pattern>
' ' ] <day-of-week-pattern> ' ' ] <hour-pattern> ' ' ]
<minute-pattern>
```

```
month-pattern := '*' | { (1..12) { , (1..12) }
```

```
day-pattern := '*' | { (1..31) { , (1..31) }
```

```
day-of-week-pattern := '*' | { (1..7) { , (1..7) }
```

```
hour-pattern := '*' | { (0..23) { , (0..23) }
```

```
minute-pattern := '*' | { (0..59) { , (0..59) }
```

Beschreibung

Cron-Patterns werden in verschiedenen Elementen von instantOLAP verwendet, u.A. zum Anstossen der Dimensions-Synchronisieren oder zum Verschicken von E-Mails. Cron-Patterns enthalten jeweils einen Teil für jeden Teil der Zeit: Einen für den Monat, einen für den Tag, einen für den Wochentag, einen für die Stunde und einen für die Minute. Jeder Teil kann auf "*" gesetzt werden (das steht dann für jeden Monat, jeden Tag etc.) oder eine, durch Kommata separierte, Liste von Monaten, Tagen, Wochentagen, Stunden oder Minuten (jeweils als Zahl). enthalten. Die einzelnen Teile werden durch ein Leerzeichen getrennt.

Ein Cron-Pattern trifft dann zu, wenn für jeden Teil der Zeit entweder eine entsprechende Nummer im Cron-Pattern vorkommt oder der entsprechende Teil im Cron-Pattern auf "*" gesetzt wurde.

Beispiele

```
*
```

Jede Minute

```
* 0
```

Jede Stunde (00:00, 01:00, ..., 23:00)

```
* 0 0
```

Jeden Tag um Mitternacht

```
* 1 0
```

Jeden Tag um 01:00

Datums-Formate

Syntax

Datums-Formate werden über ein Pattern definiert, in dem die Elemente aus der folgenden Tabelle kombiniert werden können, um ein Datum in das gewünschte Zielformat zu konvertieren. Jedes Element wird bei der Konvertierung durch den entsprechenden Teil des Datums ersetzt.

Symbol	Bedeutung	Darstellung	Beispiel
G	Ära-Bezeichner	(Text)	AD
y	Jahr	(Zahl)	1996
M	Monat im Jahr	(Zahl & Text)	July & 07
q	Quartal im Jahr	(Zahl)	2
d	Tag im Monat	(Zahl)	10
h	Stunde in am/pm (1~12)	(Zahl)	12
H	Stunde im Tag (0~23)	(Zahl)	0
m	Minute in der Stunde	(Zahl)	30
s	Sekunde in der Minute	(Zahl)	55
S	Millisekunden	(Zahl)	989
E	Tag in der Woche	(Text)	Tuesday
D	Tag im Jahr	(Zahl)	190
F	Wochentag im Monat	(Zahl)	2 (2nd Wed in July)
w	Woche im Jahr	(Zahl)	27
W	Woche im Monat	(Zahl)	2
a	am/pm Darstellung	(Text)	PM
k	Stunde im Tag (1~24)	(Zahl)	24
K	Stunde in am/pm (0~11)	(Zahl)	0
z	Zeitzone	(Text)	Pacific Time Standard
'	Vor Sonderzeichen	(Delimiter)	

" Einfache (Literal) '
Anführungszeichen

Beispiele

'dd.MM.yyyy'

Ergibt z.B. '01.10.2004'

'dd. MMM yyyy'

Ergibt z.B. '01. Oktober 2004'

'EEE'

Ergibt z.B. 'Montag'

Farben

Syntax

```
<color> := <predefined-color> | '#' <hex> <hex> <hex>
```

Beschreibung

In instantOLAP wird jede Farbe durch eine Zeichenkette repräsentiert, die entweder einen vordefinierten Farb-Namen enthält (siehe unten) oder die Farbe über einen RGB-Wert beschreibt.

Vordefinierte Namen sind einfache, englische, Namen wie "red", "green" oder "blue".

Für einen RGB-Wert muss die Zeichenkette als erstes Zeichen ein "#" enthalten, gefolgt von einem sechs-stelligen Hexadezimal-Wert mit jeweils zwei Ziffern für den Rot-, Grün- und Blau-Anteil (bekannt aus HTML).

Vordefinierte Farben

white, black, red, green, blue, light_grey, grey, dark_grey, orange, yellow, cyan

Beispiele

```
'red' (Rot)
'green' (Grün)
'blue' (Blau)
'ffffff' (Weiss)
'000000' (Schwarz)
'ff0000' (Rot)
'00ff00' (Grün)
'0000ff' (Blau)
```

Zahlen-Formate

Syntax

Zahlen-Formate werden über ein Pattern definiert, in dem die Elemente aus der folgenden Tabelle kombiniert werden können, um eine Zahl in das gewünschte Zielformat zu konvertieren. Jedes Element wird bei der Konvertierung durch den entsprechenden Teil der Zahl ersetzt.

Symbol	Bedeutung
0	eine Ziffer
#	eine Ziffer, Null für fehlende Ziffern
,	Platzhalter für Dezimal-Trenner
E	trennt die Mantisse und den Exponenten im Exponential-Format
;	unterteiltl Formate
-	Prefix für negative Werte
%	Mit 100 multiplizieren und als Prozentzahl darstellen
?	Mit 1.000 multiplizieren und als Promille darstellen
×	Währungszeichen
.	Monitärerer Dezimal-Separator
X	jedes anderes Zeichen kann im Prefix und Suffix darstellen
'	Wird verwendet um Sonderzeichen im Prefix oder Suffix anzuführen

Beispiele

'00#'

Ergibt z.B. '005'

'#.###.##0.00'

Ergibt z.B. '7.532.238,00'

'000%'

Ergibt z.B. '037%'

SQL-Ausdrücke

SQL-Ausdrücke in instantOLAP

SQL-Keyloader und SQL-Cubes verwenden für ihre Definition SQL-Ausdrücke. Diese Ausdrücke haben eine sehr ähnliche Syntax wie SQL, bestehen aber nur aus Fragmenten eines kompletten Statements. Die kompletten Statements werden dann durch den SQL-Generator von instantOLAP aus diesen Fragmenten zusammengesetzt, wenn Sie zum Laden von Dimensionen oder Kennzahlen benötigt werden.

Typen

SQL-Ausdrücke verwenden, ebenso wie die instantOLAP-Formelsprache, ein Typsystem, das aber weniger restriktiv ist. Hauptsächlich wird zwischen Zeichenketten (String), Zahlen (Number) und logischen Werten (Boolean) unterschieden. Logische Ausdrücke werden z.B. für die Generierung einer WHERE-Clause (eines Filters) verwendet, alle anderen Parameter der Loader und Cubes akzeptieren jeden Datentyp.

Der SQL-Generator

Jedesmal, wenn ein Loader oder ein Cube eine SQL-Abfrage ausführt, wird vorher das Statement generiert. Bei der Generierung eines Statements werden immer folgende Schritte durchgeführt:

- 1 Alle benötigten und verwendeten SQL-Fragmente werden zusammengestellt und in den SELECT-Teil des Statements übertragen
- 2 Alle verwendeten Filter werden zusammengestellt und in die WHERE-Clause des Statements übertragen
- 3 Alle Tabellen, die von den Fragmenten und Filtern verwendet werden, werden ermittelt
- 4 Alle Links zwischen den Tabellen (und die dazu ggf. zusätzlich benötigten Tabellen) werden ermittelt, um die JOINS der Statements zu generieren
- 5 Alle nicht aggregierenden Ausdrücke werden in den GROUP BY Teil des Statements übertragen

Wenn mehr als eine Tabelle verwendet wird, dann sammelt der Generator alle möglichen Links zwischen diesen Tabellen zusammen und fügt diese (bzw. deren Tabellen) zum Statement hinzu. Wenn eine Tabelle nicht mit den anderen verbunden werden kann, dann bricht der Generator ab und wirft eine entsprechende Fehlermeldung. Die gleichzeitige Abfrage von Werten aus verschiedenen Tabellen, die nicht miteinander verknüpft sind, ist nicht möglich.

Hinweise für den Generator in den Datenbank-Einstellungen

Da instantOLAP die SQL-Stamments im ANSI SQL/92 Format generiert, werden einige SQL Möglichkeiten bestimmter Datenbank nicht standardmässig unterstützt und müssen entsprechend eingestellt werden:

- Der Concat-Operator: Die meisten Datenbanken verwenden den ANSI-Standard-Operator ||, um Zeichenketten aneinander zu hängen. Einige Datenbank unterstützen diesen jedoch nicht (z.B. MS Access, MS SQL Server oder MySQL) und verwenden stattdessen z.B. das Plus (+).
- Die maximale Anzahl von Werten in IN-Listen innerhalb der WHERE-Clause: instantOLAP verwendet solche Listen mit dem IN-Operator, um grosse Mengen von Filtern zu definieren (z.B. um eine Liste von IDs zu filtern). Diese Listen können sehr lang werden und die so maximale Länge des gesamten Statements überschreiten. In diesem Fall teilt instantOLAP das Statements in mehrere auf und führt diese einzeln aus. Die maximale Länge der Listen ist ebenfalls Datenbank-abhängig.
- Die Verwendung von Aliases (Namen) in WHERE, GROUP BY und ORDER BY: Einige Datenbanken verbieten die Verwendung von komplexen Ausdrücken im GROUP BY, WHERE oder ORDER BY Teil und erwartet an dieser Stelle stattdessen den Aliase (eine Art temporärer Spaltenname), der weiter vorne im Statement zum entsprechenden Ausdruck vergeben wurde. Andere Datenbanken kennen keine Aliase und ermöglichen stattdessen komplexe Ausdrücke an diesen Stellen.
- Datenbankabhängige Funktionen und Aggregationen: Die meisten Datenbanken besitzen eigene, proprietäre Funktionen die nicht Teil der ANSI SQL/92 Syntax sind aber sehr nützlich sein können.

Um diese Unstimmigkeiten abzudecken, können Sie in der Definition der Datenbanken die meisten Datenbank-abhängigen Parameter (der Concat-Operator, die maximale Länge für IN-Listen, die Verwendung von Aliasen etc.) einstellen. Für die gängigen Datenbanken existieren ausserdem bereits Voreinstellungen, die automatisch bei der Anbindung der entsprechenden Datenbanken übernommen werden.

Nur die proprietären Funktionen und Aggregationen müssen Sie innerhalb der SQL-Ausdrücke als solche markieren (durch die Verwendung des SQL Passthroughs).

In diesem Kapitel

Tabellen und Spalten.....	443
Konstanten	444
Operatoren	445
Funktionen.....	447
Aggregations-Funktionen	448
SQL Passthrough	450
Klammern	452

Tabellen und Spalten

Syntax

```
<column-expression> := [ <schema-name> '.' ] <table-name>
                        '.' <column-name>
```

```
<schema-name> := <string> | "'" <string> "'"
```

```
<table-name> := <string> | "'" <string> "'"
```

```
<column-name> := <string> | "'" <string> "'"
```

Beschreibung

Dies ist der einfachste Weg, um auf eine Spalte aus einer Tabelle zuzugreifen. Wie in reinem SQL müssen Sie den Tabellennamen, gefolgt von einem '.' und dem Spaltennamen, dazu angeben. Im Unterschied zu echtem SQL müssen Sie jedoch immer einen Tabellennamen vorranstellen, ein einzelner Spaltenname ist nicht erlaubt.

Optional können Sie auch Tabellen aus einem anderem Datenbankschema als dem Standard-Schema der Datenbank verwenden (wenn die Datenbank Schematas unterstützt). Wie in SQL müssen die dazu vor dem Tabellennamen den Schema-Namen, gefolgt von einem Punkt, angeben.

Wenn die Schema-, Tabellen- oder Spaltennamen Sonderzeichen oder Leerzeichen enthalten, dann können Sie die Namen in einfachen oder doppelten Anführungszeichen setzen. Der SQL-Generator setzt beim Generieren der SQL-Statements die Namen ebenfalls in den (jeweils für die Datenbank gültigen) Delimiter. Wenn Sie mittels "Drag&Drop" in der Workbench SQL-Expressions definieren, werden alle Namen immer in Anführungszeichen gesetzt.

Beispiele

```
SALES.AMOUNT
```

Greift auf die Spalte AMOUNT aus der Tabelle SALES zu

```
ERM.SALES.AMOUNT
```

Greift auf die Spalte AMOUNT der Tabelle SALES aus dem Schema ERM zu

```
'SALES'. 'AVG AMOUNT'
```

Greift auf die Spalte SALES der Tabelle AVG AMOUNT zu (mit einem Leerzeichen im Tabellen-Namen)

Konstanten

Syntax

```
<sql-constant> := <string-constant> | <number-constant> |  
<boolean-constant> | <null-constant>  
  
<string-constant> := '"' <string> '"'  
  
<number-constant> := <integer-constant> | <double-constant>  
  
<integer-constant> := ['-'] { <digit> }  
  
<double-constant> := ['-'] { <digit> } '.' { <digit> }  
  
<boolean-constant> := 'TRUE' | 'FALSE'  
  
<null-constant> := 'NULL'
```

Beschreibung

Wie im "echten" SQL können Sie Konstanten in Ihrem SQL-Ausdrücken verwenden. Es gibt verschiedene Typen von Konstanten (Integer, Double, Boolean oder String-Konstanten sowie NULL), die Sie verwenden können.

Beispiele

```
'Hello World'  
  
10  
  
-10  
  
10.5  
  
-10.5  
  
TRUE  
  
FALSE  
  
NULL
```

Operatoren

Syntax

<operator-expression> := <binary-operator-expression>

<binary-operator-expression> := <sql-expression> <operator>
<sql-expression>

Beschreibung

instantOLAP unterstützt nahezu alle aus SQL bekannten Operatoren inklusive "LIKE" und "IS NULL". In der folgenden Tabelle sind alle unterstützten Operatoren beschrieben:

Op.	Bedeutung	Bemerkung	Beispiel
	Verbindung von Zeichenkette n	Wird durch den jeweiligen Operator der DB ersetzt	CUSTOMER.FIRSTNAME CUSTOMER.LASTNAME
+	Addition	Für Nummern und Zeichenketten	
-	Subtraktion		
*	Multiplikation		SALES.QUANTITY * SALES.PRICE
/	Division		SALES.AMOUNT / SALES.PRICE
%	Modulo		
<	Kleiner als		SALES.PRICE < 2.00
<=	Kleiner oder gleich		SALES.PRICE <= 2.00
>	Grösser als		SALES.PRICE > 2.00
>=	Grösser oder gleich		SALES.PRICE >= 2.00
=	Gleich		
LIKE	Mustervergleich	? und % können als Platzhalter verwendet werden	CUSTOMER.NAME LIKE 'A%'
IN	Kommt der Wert in der Liste vor?		PRODUCT.STATE IN (1, 2, 5)

AND	Logisches Und	
OR	Logisches Oder	
NOT	Logisches Nicht	
IS NULL	Test auf NULL	CUSTOMER.NAME IS NULL
IS NOT NULL	Test auf nicht NULL	CUSTOMER.NAME IS NOT NULL

Obwohl der Concat-Operator Datenbank-abhängig ist (z.B. akzeptieren MS SQL Server und MS Access nur "+" als Operator), sollten Sie in instantOLAP immer "||" als Concat-Operator verwenden. Der wirkliche Operator kann in der Datenbank-Definition von instantOLAP eingestellt werden und der SQL-Generator ersetzt alle Vorkommnisse von "||" durch den wirklichen Operator. Ausserdem ist der Operator für die gängigen Datenbanken bereits voreingestellt.

Funktionen

Syntax

```
<function-expression> := <aggregation-expression> |  
<passthrough-expression>
```

Beschreibung

Es gibt zwei verschiedene Wege, Funktionen innerhalb von instantOLAP-SQL zu verwenden: Den Aufruf von Aggregations-Funktionen (die einzigen Funktionen, die ANSI-SQL kennt) und der Aufruf von datenbankabhängigen Funktionen mit Hilfe der "SQL Passthrough" Syntax. Der erste Fall ist der häufigste und wird hauptsächlich für die Anbindung von Kennzahlen an Fakten-Tabellen verwendet. Der zweite ist sehr flexibel und kann sowohl für die Anbindung von Kennzahlen als auch für andere Berechnungen und Transformationen in SQL verwendet werden.

Beispiele

```
SUM( SALES.AMOUNT )
```

Verwendet die Standard-SQL Function "SUM" zum Summieren aller Beträge

```
@TOCHAR( SALES.DATE, 'yyyy' )
```

Konvertiert das Datum in ihre Jahreszahl mit der Datenbankspezifischen Funktion "TOCHAR"

Aggregations-Funktionen

Syntax

```
<aggregation-expression> := <sum-expression> | <min-expression> | <max-expression> | <avg-expression> | <count-expression> | <count-distinct-expression> || <custom-aggregation-expression>
```

```
<sum-expression> := 'SUM(' <sql-expression> ')'
```

```
<min-expression> := 'MIN(' <sql-expression> ')'
```

```
<max-expression> := 'MAX(' <sql-expression> ')'
```

```
<avg-expression> := 'AVG(' <sql-expression> ')'
```

```
<count-expression> := 'COUNT(' <sql-expression> ')'
```

```
<count-distinct-expression> := 'COUNT DISTINCT(' <sql-expression> ')'
```

```
<custom-aggregation-expression> := '@@' <function-name> '(' <sql-expression> ')'
```

Beschreibung

Aggregations-Funktionen werden verwendet, um innerhalb von SQL-Cubes Kennzahlen an eine Fakten-Tabelle aus einer Datenbank zu binden. Meistens werden Kennzahlen mit einer aggregierenden Funktionen angebunden, damit das SQL-Statement als Ergebnis die Verdichtung (z.B. die Summe, den Durchschnitt, die Anzahl etc.) der Kennzahl für die jeweilig gewünschten Ebenen der verschiedenen Dimensionen ergibt.

Durch die Verwendung einer aggregierenden Funktion erweitert der SQL-Generator von instantOLAP automatisch den GROUP BY Teil des generierten Statements.

Es gibt verschiedene Aggregations-Funktionen in ANSI SQL: SUM, MIN, MAX, AVG und COUNT bzw. COUNT DISTINCT. Die meisten Datenbank kennen mehr als diese Funktionen, unterscheiden sich dabei aber in der Syntax. Wenn Sie eine dieser proprietären Funktionen verwenden möchten, dann müssen Sie eine SQL-Passthrough Syntax verwenden (die einen SQL-Ausdruck 1:1 und ohne vorherigen Syntax-Check an die Datenbank weiterleitet).

Beispiele

```
SUM( SALES.AMOUNT )
```

Berechnet die Summe alle Beträge

```
MIN( SALES.AMOUNT )
```

Berechnet den kleinsten Betrag

```
MAX( SALES.AMOUNT )
```

Berechnet den grössten Betrag

```
AVG( SALES.AMOUNT )
```

Berechnet den durchschnittlichen Betrag

```
COUNT( SALES.CUSTOMER_ID )
```

Berechnet die Anzahl Kunden

```
COUNT DISTINCT( SALES.CUSTOMER_ID )
```

Berechnet die Anzahl (eindeutiger) Kunden

```
@@LEAST( SALES.AMOUNT )
```

Verwendet die proprietäre Datenbankfunktion "LEAST"

SQL Passthrough

Syntax

```
<passthrough-expression> := <function-passthrough-expression> | <full-passthrough-expression>
```

```
<function-passthrough-expression> := [ '@' | '@@' ]  
<string> '(' { <sql-expression> } ')'
```

```
<full-passthrough-expression> := [ '@' | '@@' ] "  
<string> ""
```

Beschreibung

Der SQL-Parser von instantOLAP kennt und akzeptiert nur Funktionen aus dem ANSI-SQL 92 Standard (wie SUM, MIN, MAX usw) und wirft eine Fehlermeldung, wenn andere als diese Funktionen in SQL-Ausdrücken verwendet wird. Dennoch ist es möglich, mit Hilfe der "SQL Passthrough" Syntax von instantOLAP, eigene oder datenbank-spezifische Funktionen innerhalb von SQL-Ausdrücken zu verwenden.

Es gibt zwei verschiedene Formen eines solchen Passthroughs: Sie können nur den unbekanntem Funktionsnamen für den SQL-Parser kennzeichnen oder Sie können einen kompletten SQL-Ausdruck durchreichen. Der erste Weg lässt den Parser nur den unbekanntem Funktionsnamen akzeptieren, alle Parameter der Funktion werden aber trotzdem auf ihre Korrektheit hin überprüft. Der zweite übergibt einen kompletten Teilausdruck inkl. der Funktionsparameter, ohne diesen zu überprüfen. Der Vorteil der ersten Form ist, dass immer noch eine Überprüfung der Parameter stattfindet und Sie so Fehler vermeiden können. Der zweite Weg ist dagegen flexibler aber auch unsicherer, denn Sie können alles an die Datenbank übergeben und bekommen ggf. erst zur Laufzeit Fehler von der Datenbank gemeldet.

Um nur einen Funktionsnamen zu markieren, müssen Sie den Passthrough-Operator "@" vor den Funktionsnamen stellen (z.B. @SUBSTR). Um einen ganzen Ausdruck zu übergeben, muss unmittelbar nach dem Operator eine, durch Anführungszeichen umschlossene, Zeichenkette folgen. Der Inhalt dieser Zeichenkette wird dann unverändert in das SQL-Statement übernommen.

Ausserdem gibt es auch zwei verschiedene Operatoren, mit denen Sie einen solchen Passthrough einleiten können: Der einfache Operator "@" und der aggregierende Operator "@@". Da SQL zwischen einfachen und aggregierenden Funktionen unterscheidet, müssen Sie dem SQL-Parser kenntlich machen, ob Ihre Funktion eine solche Aggregation durchführt. Wenn das der Fall ist, wird der SQL-Generator die entsprechende GROUP BY Clause für die nicht-aggregierenden Spalten im Statement hinzufügen.

Beispiele

```
@SUBSTR( SALES.DATE, 1, 4 )
```

Die Funktion "SUBSTR" wird durchgereicht, aber alle Parameter werden weiterhin überprüft

```
@ "CONVERT( CHAR(10) SALES.DATE )" 
```

Vollständiger "Passthrough" des Ausdrucks

```
@ 'COUNT( DISTINCT SALES.CUSTOMER_ID )'
```

Übergibt eine (aggregierenden) SQL-Expression

Klammern

Syntax

```
<bracket-expression> := '(' <sql-expression> ')'
```

Beschreibung

Klammern innerhalb von SQL-Ausdrücken erlauben, die Reihenfolge ihrer Ausführung zu kontrollieren. Durch das Umschliessen von Teilen eines Ausdrucks durch Klammern zwingen Sie die Datenbank dazu, den Teil innerhalb der Klammern als Erstes, vor dem Rest des Ausdrucks, zu berechnen. Klammern können beliebig kombiniert und ineinander verschachtelt werden - die Klammern werden dann von innen nach aussen berechnet und aufgelöst.

Beispiele

```
( 10 + 20 ) * 30 --> 900
```

```
10 + 20 * 30 --> 610
```

```
( CUSTOMER.FIRSTNAME IS NULL ) OR ( CUSTOMER.LASTNAME IS  
NULL )
```

Index

3

3DDepth • 108, 147, 148
3DModeOn • 108, 147, 187

A

ABC • 346
Abfragen • 52
ABS • 346, 376
ADD • 335, 347, 354, 358, 365, 398, 400, 402, 403, 419, 421
Aggregations-Funktionen • 448
Alias • 233
Align • 63, 67, 99
All • 331, 368
ALL • 348, 373, 392
ANCESTORS • 349, 356, 373, 407, 411
AND • 336, 342, 350, 407, 409
Angle • 187
Any • 331, 347, 358, 359, 364, 369, 370, 374, 376, 377, 382, 383, 385, 386, 387, 390, 391, 413, 422, 423, 425, 428
Assertion • 63
Attribute • 304, 305, 308, 315, 316
ATTRIBUTENAMES • 337, 351, 352, 353, 363
ATTRIBUTENV • 352
ATTRIBUTES • 337, 351, 352

Ä

Äußere Blöcke • 35

A

Author • 18, 19, 20, 102, 103
AutoLabelSpacingOn • 109, 148
AVG • 353, 398, 400

B

Background • 38, 39, 64, 67, 84, 109, 148, 188
Balkendiagramm-Eigenschaften • 147
BarAlignment • 149
BarLabelAngle • 149
BarLabelColors • 150
BarLabelFont • 150
BarLabelsOn • 150
BarLabelStyle • 151

BarOutlineColor • 151
BarOutlineOff • 152
BarType • 152
BarWidth • 153
BEAUTIFY • 354, 395, 422, 425
Bericht • 18
Berichts-Eigenschaften • 17
Boolean • 30, 60, 61, 78, 79, 81, 87, 94, 95, 99, 108, 111, 112, 113, 115, 120, 121, 124, 129, 136, 137, 139, 143, 145, 146, 147, 148, 150, 152, 155, 156, 158, 163, 164, 165, 167, 171, 177, 178, 179, 180, 184, 185, 186, 196, 197, 198, 200, 203, 205, 206, 208, 298, 302, 311, 313, 319, 321, 325, 331, 342, 350, 358, 368, 369, 370, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 390, 391, 396, 407, 408, 409, 415, 418, 425
Boolean-Konstanten • 342, 382, 407, 409
Border • 38, 39
Bottom-Color • 65, 68, 89, 94, 98
Bottom-Padding • 66, 69, 90, 94, 98

C

Catalog • 212
CEIL • 347, 354, 376, 415
Cell-Align • 63, 66, 78
Cell-Background • 65, 67, 71
Cell-Bottom-Color • 66, 68, 76, 77
Cell-Bottom-Padding • 66, 68, 76, 78
Cell-Font • 69, 70, 82
Cell-Font-Size • 69, 70, 83
Cell-Font-Weight • 69, 70, 84
Cell-Foreground • 67, 70, 84
Cell-Left-Color • 68, 71, 76, 77, 89
Cell-Left-Padding • 69, 71, 76, 78, 90
Cell-Link • 72, 73, 74, 75, 90
Cell-Link-Icon • 73, 74, 75, 91
Cell-Link-Keys • 73, 75, 92
Cell-Link-Name • 73, 74, 75, 92
Cell-Link-Target • 73, 74, 75, 93
Cell-Right-Color • 68, 75, 77, 94
Cell-Right-Padding • 69, 76, 78, 94
Cell-Top-Color • 68, 76, 77, 98
Cell-Top-Padding • 69, 76, 77, 98
Cell-Vertical-Align • 67, 78, 99

Charset • 212
ChartBackground • 110, 149, 153
ChartForeground • 110, 149, 154
ChartTitle • 111, 154, 188
CHILDREN • 355, 373, 389, 404, 407, 410
CLUSTER • 350, 356
Color • 38, 39
Column • 243, 272, 277, 313, 315, 317
Column-end bracket • 213
Column-start bracket • 213
Complete • 295, 300, 309, 312
CONCAT • 357
Concat-Operator • 214
ConnectedLinesOn • 111
CONTAINS • 358, 383
Copy-To-Result • 102, 104
Corner-Align • 52
Corner-Background • 52, 55
Corner-Bottom-Color • 53, 56, 57, 58
Corner-Font • 53, 54, 55
Corner-Font-Size • 54
Corner-Font-Weight • 54, 55
Corner-Foreground • 53, 55
Corner-Left-Color • 53, 55, 57, 58
Corner-Right-Color • 53, 56, 58
Corner-Text • 57
Corner-Top-Color • 53, 56, 57, 58
COUNT • 354, 359
Cron Pattern • 245
Cron-Pattern • 321, 323, 324, 326
Cron-Patterns • 245, 321, 324, 436
CSS • 18, 26
CSV-Attribute • 277
CSV-Column • 243
CSV-Cube • 309
CSV-Dimension • 315
CSV-Fact • 313
CSV-Keyloader • 272
CSV-Source • 239, 272, 310
CUBE • 335, 359, 433

D

Das Typsystem • 330
Database • 254, 296
Datasource • 214
Date • 18, 19, 20, 102, 103
Datenbank • 212
Datums-Formate • 281, 282, 284, 285, 286, 287, 290, 437
Default • 28, 30
Default Text-Attribute • 245, 249
DefaultGridLinesColor • 111, 154
DefaultGridLinesOn • 112, 155
Delimiter • 239, 241

Depth • 189
DEPTH • 362, 393
Description • 20
DetachedDistance • 189
DetachedSlices • 189
Diagramm-Eigenschaften • 107
Die Formelsprache • 329
Dimension • 245, 251, 265, 268, 277, 289, 291, 305, 308, 315, 316
DIMENSIONATTRIBUTENAMES • 362
Dimensionen und Selektionen • 333, 334
DIMENSIONNAME • 351, 363, 364
DIMENSIONNAMES • 363
Dimensions-Ebenen • 333
Direction • 235
Distinct • 254
DISTINCT • 364, 387
DIV • 335, 348, 365, 402, 403, 419
DLOOKUP • 366, 395
Double • 63, 153, 187, 189, 331, 343, 353, 365, 398, 400, 402, 403, 420
Double-Konstanten • 342, 343
Drilldown • 78, 80, 81
Drilldown-Encapsulate • 79, 81
Drilldown-Iteration • 79, 80, 81
Drilldown-Prefetch • 79, 80, 81
DRILLLEVEL • 367
Drop IN with NULL • 215

E

Ebenen-Funktionen • 340
EMPTY • 367
Enable Load • 296, 297
Enable Store • 297
End • 281, 282, 284, 287, 288
Endshift • 281, 282, 288
ENDSWITH • 368, 418
EQUAL • 335, 369, 377, 378, 391, 392, 425
Escape Character • 215
EVAL • 339, 369
Exclude Pattern • 282, 283, 284
Exclude Values • 283
EXISTS • 344, 358, 359, 370, 383, 386, 433
EXP • 371
Export • 103
Expression • 318

F

Fact • 302, 303, 313, 318, 320
FACTROOT • 371, 406
FAMILY • 349, 356, 372, 389, 407, 410
Farben • 439
File-Cache • 324

Filename • 324
 Filter • 20, 35, 40, 58, 60, 81, 88
 FILTER • 350, 373
 FIND • 373, 396, 408
 FIRST • 347, 374, 388
 First Line As Names • 239
 FloatingLabelFont • 112, 155, 190
 FloatingOnLegendOff • 113, 156
 FLOOR • 355, 375, 415
 Font • 40, 41, 42, 69, 82, 83, 84, 113, 156, 190
 Font Size • 41, 42
 Font Weight • 41, 42
 Font-Size • 70, 82, 83, 84
 Font-Weight • 70, 82, 83
 FOREACH • 376, 396, 416
 Foreground • 65, 71, 84, 114, 149, 157, 188, 191
 Format • 42, 45, 51, 84, 86, 248, 255, 265, 273, 278, 306, 317
 Formate • 273, 276, 278, 306, 317, 435
 Formats • 21
 Formel • 318
 Formula • 85, 104, 105
 Funktionen • 346, 447
 Funktionsaufrufe • 336, 340

G

GraphInsets • 114, 157, 191
 GREATER • 335, 369, 376, 378, 391, 392, 425
 GREATER_OR_EQUAL • 335, 369, 377, 391, 392, 425
 GridAdjustmentOn • 115, 158
 GridImage • 115, 158
 GridLineColors • 115, 158
 GridLines • 116, 159
 GridLinesColor • 116, 159
 Group By Index • 216

H

HASKEYS • 378, 380
 HASLEVEL • 379, 394
 HASPOSITION • 379, 380
 HASROLES • 380, 382, 428
 HASUSER • 381, 428
 Height • 28, 34, 45, 51, 86, 100

I

Icon • 21, 23
 ID • 248
 Ignore Missing Targets • 265, 266
 IIF • 382
 IN • 335, 358, 383
 Include • 327
 Innere Blöcke • 38

Input • 87
 InsideLabelColor • 192
 InsideLabelFont • 192
 Integer • 28, 34, 42, 45, 49, 51, 54, 66, 68, 69, 71, 76, 77, 83, 86, 89, 94, 98, 100, 108, 119, 123, 124, 125, 126, 127, 129, 132, 134, 142, 147, 149, 162, 165, 166, 167, 168, 169, 172, 173, 175, 182, 187, 195, 198, 202, 331, 343, 347, 355, 359, 362, 367, 374, 375, 379, 380, 387, 389, 392, 393, 394, 398, 399, 400, 401, 411, 412, 414, 415, 418, 419, 429, 430, 431
 Integer-Konstanten • 342
 INTERSECT • 383, 387, 414, 429
 ISCHILDOF • 384, 386
 ISNULL • 344, 371, 385, 433
 ISPARENTOF • 385, 386, 410
 Iteration • 35, 45, 82, 87, 97

J

JDBC-Driver • 217
 JOIN • 335, 384, 386, 414, 429

K

Kennzahl-Funktionen • 341
 Key • 20, 35, 40, 45, 48, 58, 59, 73, 80, 81, 87, 91, 248, 332, 333, 345, 346, 348, 349, 351, 352, 355, 356, 359, 362, 363, 366, 372, 373, 374, 376, 378, 379, 380, 384, 386, 388, 392, 393, 395, 396, 403, 404, 405, 406, 408, 409, 410, 411, 412, 415, 416, 426, 427
 Key-Attribute • 251
 Key-Konstanten • 342, 345
 Klammern • 336, 452
 Kommentare • 102
 Konfigurations-Eigenschaften • 211
 Konstanten • 342, 444
 Konventionen • 15

L

Label_0 • 117, 118, 160, 161, 192, 193, 194
 LabelUrl_0 • 117, 118, 160, 161, 193, 194
 LabelUrlTarget_0 • 117, 118, 160, 161, 193, 194
 LAST • 375, 387
 LEAFS • 356, 373, 388, 407
 LEFT • 389, 413, 415, 417, 419, 420
 Left-Color • 66, 88, 94, 98
 Left-Padding • 66, 89, 94, 98
 LegendColors • 118, 161, 194

LegendColumns • 119, 162, 195
LegendFont • 119, 162, 195
LegendImage • 120, 163, 195
LegendOn • 120, 163, 196
LegendPosition • 121, 164, 196
LegendReverseOn • 121, 164, 197
LESS • 335, 369, 377, 378, 390, 392, 425
LESS_OR_EQUAL • 335, 369, 377, 378, 391, 425
LEVEL • 334, 349, 392, 404
Levelname • 273
LEVELNAMES • 362, 393
LEVELOF • 392, 393, 412
LIMIT • 394
Limit Syntax • 217
Line-Dimension • 298, 310
LineStroke • 122
LineWidth • 122
Liniendiagramm-Eigenschaften • 108
Link • 46, 47, 48, 49, 73, 90, 91, 92, 93, 235
Link-Expression • 237
Link-Icon • 47, 48, 49, 73, 90, 91, 92, 93
Link-Keys • 47, 48, 49, 74, 90, 91, 92, 93
Link-Name • 47, 48, 49, 75, 90, 91, 92, 93
Link-Target • 47, 48, 49, 75, 90, 91, 92, 93
Load Links • 218
Load Table-Sizes • 219
Locale Format • 281, 284, 288
Logo • 22, 25
LOOKUP • 367, 395, 416
LowerRange • 123, 165

M

Match • 298, 302, 311, 313, 319, 321, 325
MATCH • 335, 376, 383, 396, 416
MATRIX • 397, 399, 401, 430, 431, 432
Max • 293, 294
MAX • 354, 398, 400
Max Age • 322, 323, 326
Max connection age • 219
Max connection count • 220
Max IN-Count • 220
Max Size • 323
MAX_X • 397, 398, 399, 401, 430, 431, 432
MAX_Y • 397, 399, 401, 430, 431, 432
MaxValueLineCount • 123, 165
Memory-Cache • 321

Min • 293
MIN • 354, 398, 400
MIN_X • 397, 399, 400, 401, 430, 431, 432
MIN_Y • 397, 399, 401, 430, 432
MOD • 335, 402, 403
Mode • 255, 274, 284
Model • 23, 327
MUL • 335, 348, 365, 402, 419
MultiColorOn • 165

N

Name • 24, 27, 29, 30, 32, 221, 230, 231, 233, 236, 240, 243, 246, 251, 267, 279, 289, 299
NEIGHBOURS • 356, 403
NEXT • 375, 404, 413, 426, 427
NONFACTROOTS • 372, 405
NONLEAFS • 389, 406
NOT • 342, 351, 407, 409
NOW • 374, 408, 421
NULL • 342, 343, 346, 347, 348, 352, 353, 354, 355, 356, 357, 358, 359, 362, 363, 364, 365, 366, 368, 370, 371, 372, 374, 375, 376, 377, 378, 379, 380, 381, 382, 392, 393, 394, 395, 398, 400, 402, 404, 405, 406, 407, 409, 410, 411, 412, 413, 414, 415, 417, 418, 419, 420, 421, 422, 423, 424, 425, 427, 429, 430, 432
Null-ID • 306
Null-Value • 256, 267, 274, 279, 317
Number • 123, 124, 130, 131, 165, 166, 172, 174, 332, 346, 347, 348, 353, 354, 365, 371, 375, 397, 398, 400, 402, 415, 419, 420, 422, 432
Number-Keyloader • 293

O

Omit-Percentage • 307
Operator • 237
Operatoren • 334, 336, 348, 365, 369, 377, 378, 387, 390, 391, 396, 402, 403, 409, 419, 425, 445
Options • 28, 29, 30
OR • 336, 342, 351, 407, 408
Orientation • 36
OutsideLabelFont • 198
OutsiteLabelColor • 197

P

Padding • 49
PARENT • 350, 356, 373, 404, 407, 409
Parent Format • 257
Parent Pattern • 285

Parent Search-Attribute • 257
 Parent SQL-Expression • 258, 259
 Parent Trim • 259
 Parent-Column • 275
 Parent-Format • 276
 Password • 222
 Pattern • 286, 289, 290
 PEDIGREE • 350, 356, 410
 PercentDecimalCount • 198
 PercentLabelsOn • 198
 PercentLabelStyle • 199
 PieLabelFont • 199
 PieLabelsOn • 200
 PieRotationOn • 200
 PointingLabelColor • 201
 PointingLabelFont • 201
 POSITIONOF • 380, 394, 411
 PREV • 375, 405, 412, 426, 427
 Print Height • 24, 25
 Print Logo • 24, 25
 Print Width • 24, 25

Q

Quote • 239, 241

R

Range • 124, 166
 RangeAdjusted • 124, 166
 RangeAdjusterOn • 124, 167
 RangeAdjusterPosition • 125, 167
 RangeAxisLabel • 125, 168
 RangeAxisLabelAngle • 126, 168
 RangeAxisLabelFont • 126, 168
 RangeColor • 127, 169
 RangeDecimalCount • 127, 169
 RangeLabelFont • 127, 170
 RangeLabelPostfix • 128, 170
 RangeLabelPrefix • 128, 170
 RangeLabelsOff • 129, 171
 RangeOn • 129, 171
 RangePosition • 129, 172
 RangeStep • 130, 172
 Recursive • 260
 Relink-Attribute • 265, 267, 289, 290
 REPLACE • 413
 REVERSE • 413
 RIGHT • 390, 413, 414, 417, 419, 420
 Right-Color • 66, 76, 89, 93, 98
 Right-Padding • 66, 76, 90, 94, 98
 Rotate • 94
 ROUND • 355, 376, 415

S

SampleAxisLabel • 130, 173
 SampleAxisLabelAngle • 173
 SampleAxisLabelFont • 131, 173

SampleAxisRange • 131, 174
 SampleColors • 132, 174, 202
 SampleDecimalCount • 132, 175, 202
 SampleHighlightImage • 132
 SampleHighlightOn • 133
 SampleHighlightStyle • 134
 SampleHightlightSize • 133
 SampleLabelAngle • 134, 175
 SampleLabelColors • 135, 175, 202
 SampleLabelFont • 135, 176
 SampleLabelSelectionColor • 176
 SampleLabelsOn • 135, 177, 203
 SampleLabelStyle • 136, 177, 203
 SampleScrollerOn • 136, 178
 Schema • 221
 Select • 28
 SelectionStyle • 204
 SeriesLabelColors • 137, 178, 204
 SeriesLabelFont • 179
 SeriesLabelsOn • 137, 179, 205
 SeriesLabelStyle • 138, 179
 SeriesLineOff • 138
 Single IN Operator • 223
 SingleClickURLOn • 139, 180, 205
 SliceSeperatorColor • 206
 SliceSeperatorOn • 206
 Sort • 95, 96
 SORT • 367, 376, 395, 396, 415
 Sort-Descending • 95
 Source Expression • 237
 Spalte • 231
 Span All Dimensions • 295, 299, 309, 311
 SPLIT • 417, 419
 SQL Passthrough • 450
 SQL-Attribute • 265
 SQL-Ausdrücke • 233, 237, 238, 259, 261, 262, 263, 268, 300, 301, 303, 308, 441
 SQL-Check • 261
 SQL-Cube • 295
 SQL-Dimension • 304
 SQL-Expression • 262, 268, 303, 308
 SQL-Fact • 302
 SQL-Keyloader • 254
 SQL-Order • 263, 300, 301
 SQL-Where • 233, 263, 300, 301
 StackedOn • 139
 Start • 281, 284, 287, 288
 Startline • 240, 241
 Startshift • 282, 288
 STARTSWITH • 368, 417
 Storage • 247

String • 18, 22, 24, 25, 26, 29, 31, 32, 38, 39, 40, 41, 46, 47, 48, 49, 50, 52, 53, 54, 55, 56, 57, 58, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 78, 82, 83, 84, 88, 90, 91, 92, 93, 96, 97, 98, 104, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 125, 126, 127, 128, 130, 131, 132, 133, 134, 135, 136, 137, 138, 140, 141, 142, 143, 144, 145, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 168, 169, 170, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 201, 202, 203, 204, 206, 207, 208, 209, 332, 345, 348, 351, 352, 354, 357, 362, 363, 364, 368, 369, 373, 380, 381, 389, 393, 394, 408, 413, 414, 417, 418, 419, 420, 421, 422, 423, 424, 428, 430, 431

String-Konstanten • 342, 344

STRLEN • 413, 417, 418, 420

Stylesheet • 19, 26

SUB • 335, 348, 419

Subcube • 59

Submit • 30

SUBSTR • 390, 395, 413, 415, 417, 419

SUM • 348, 398, 400, 402, 403, 420

Supress-Cols • 60, 62

Supress-Rows • 61

Syntax • 333

T

Tabelle • 230

Tabellen und Spalten • 443

Table • 234

Table Names • 224, 226

Table Types • 225

Table-end bracket • 223

Table-start bracket • 224

Target Attribute • 269

Target Expression • 238

TargetLabelsPosition • 140

TargetValueLine • 140, 180

Text • 31, 32, 96, 97, 104

ThousandsDelimiter • 141, 181, 206

Time-Attribute • 289

Time-Keyloader • 281

Timeout • 226

TIMESTAMP • 421

Title • 26, 31, 32, 50, 97

TitleFont • 141, 181, 207

TOLOWER • 354, 395, 422, 425

TONUMBER • 422, 424

Top-Color • 66, 77, 89, 94, 97

Top-Padding • 66, 78, 90, 94, 98

Tortendiagramm-Eigenschaften • 187

TOSTRING • 335, 423

TOUPPER • 354, 395, 422, 424

Trim • 242, 264, 270

Type • 32, 231, 244, 249

U

Überschriften • 63

UNEQUAL • 335, 369, 391, 392, 425

Unique • 252, 270, 280, 291

Unit • 250

UPPERNEXT • 405, 413, 426, 427

UPPERPREV • 405, 413, 426, 427

URL • 226, 242

UriTarget • 141, 182, 207

Use Aliases • 227

Use Filter • 33

User • 223, 228

USER • 381, 382, 428

V

Value • 28, 30, 85, 95, 138, 253, 318, 332, 352, 397, 415

ValueLabelAngle • 142, 182

ValueLabelFont • 142, 183

ValueLabelPostfix • 143, 183

ValueLabelPrefix • 143, 183, 208

ValueLabelsOn • 143, 184, 208

ValueLabelStyle • 144, 184, 209

ValueLinesColor • 144, 185

ValueLinesOn • 145, 185, 186

Vertical-Align • 78, 98

Visible • 27, 99

VisibleSamples • 145

W

Width • 29, 34, 45, 51, 87, 100

WITHOUT • 414, 428

X

X • 397, 399, 401, 429, 430, 431, 432

XHEADER • 397, 399, 401, 430, 431, 432

Y

Y • 397, 399, 401, 430, 431, 432

YHEADER • 397, 399, 401, 430, 431

Z

Zahlen-Formate • 248, 440

ZERO • 344, 354, 361, 432
ZoomOn • 146, 186
Z-Order • 100
Zugriff auf Attribute • 337
Zugriff auf Variablen • 337