

Multidimensional Expressions (MDX) Reference

SQL Server 2012 Books Online

Reference



Microsoft[®]

Multidimensional Expressions (MDX) Reference

SQL Server 2012 Books Online

Summary: Multidimensional Expressions (MDX) is the query language that you use to work with and retrieve multidimensional data in Microsoft Analysis Services. MDX is based on the XML for Analysis (XMLA) specification, with specific extensions for SQL Server Analysis Services. MDX utilizes expressions composed of identifiers, values, statements, functions, and operators that Analysis Services can evaluate to retrieve an object (for example a set or a member), or a scalar value (for example, a string or a number).

Category: Reference

Applies to: SQL Server 2012

Source: SQL Server Books Online ([link to source content](#))

E-book publication date: June 2012

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft and the trademarks listed at

<http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Contents

Multidimensional Expressions (MDX) Reference	8
MDX Syntax Elements	8
Identifiers	10
Using Regular Identifiers.....	10
Using Delimited Identifiers.....	11
Expressions.....	12
Using Cube and Subcube Expressions.....	14
Using Dimension Expressions	16
Using Member Expressions.....	17
Using Tuple Expressions.....	18
Using Set Expressions.....	19
Using Scalar Expressions	20
Working with Empty Values.....	22
Operators (MDX Syntax)	25
Arithmetic Operators.....	28
Bitwise Operators.....	29
Comparison Operators	29
Concatenation Operators	31
Set Operators	31
Unary Operators.....	32
Assignment Operators	33
Functions (MDX Syntax)	33
Using String Functions.....	35
Using Mathematical Functions	37
Using Logical Functions.....	37
Using Member Functions.....	38
Using Tuple Functions.....	39
Using Set Functions	39
Using Dimension, Hierarchy, and Level Functions	43
Using Stored Procedures	44
Comments (MDX Syntax).....	44
Reserved Keywords (MDX Syntax)	46
MDX Language Reference.....	46
MDX Syntax Conventions	47
MDX Statement Reference.....	48
MDX Scripting Statements	48
MDX Data Definition Statements.....	54
MDX Data Manipulation Statements.....	94
MDX Operator Reference	113
-- (Comment).....	115

- (Except)	117
- (Negative)	118
- (Subtract).....	119
* (Crossjoin).....	120
* (Multiply).....	121
/ (Divide).....	122
^ (Power).....	124
/*...*/ (Comment).....	125
// (Comment).....	126
: (Range)	127
+ (Add).....	128
+ (Positive).....	129
+ (String Concatenation)	130
+ (Union).....	130
< (Less Than)	132
<= (Less Than or Equal To).....	133
<> (Not Equal To).....	134
= (Equal To).....	135
> (Greater Than).....	136
>= (Greater Than or Equal To).....	137
AND.....	138
IS140	
NOT	141
OR.....	142
XOR	144
MDX Function Reference.....	145
AddCalculatedMembers.....	157
Aggregate.....	158
AllMembers.....	162
Ancestor	164
Ancestors	166
Ascendants	168
Avg	169
Axis.....	172
BottomCount.....	174
BottomPercent.....	175
BottomSum	176
CalculationCurrentPass.....	177
CalculationPassValue.....	178
CASE Statement.....	179
Children	182
ClosingPeriod	183
CoalesceEmpty.....	185
Correlation.....	187
Count (Dimension)	188
Count (Hierarchy Levels)	188

Count (Set).....	189
Count (Tuple).....	192
Cousin	192
Covariance	194
CovarianceN.....	195
Crossjoin.....	196
Current.....	198
CurrentMember.....	199
CurrentOrdinal	202
CustomData	203
DataMember.....	204
DefaultMember	205
Descendants	206
Dimension.....	211
Dimensions.....	212
Distinct.....	214
DistinctCount.....	215
DrilldownLevel.....	216
DrilldownLevelBottom.....	218
DrilldownLevelTop.....	220
DrilldownMember.....	221
DrilldownMemberBottom.....	223
DrilldownMemberTop.....	225
DrillupLevel.....	226
DrillupMember.....	227
Error	230
Except	230
Exists	231
Extract.....	233
Filter	235
FirstChild	236
FirstSibling.....	237
Generate.....	238
Head.....	240
Hierarchize.....	242
Hierarchy	244
IIf244	
Instr	247
Intersect.....	250
IsAncestor	252
IsEmpty	253
IsGeneration.....	254
IsLeaf.....	255
IsSibling	255
Item (Member).....	256
Item (Tuple).....	257

KPIGoal.....	259
KPIStatus	260
KPITrend.....	261
KPIWeight.....	262
KPICurrentTimeMember.....	262
KPIValue.....	263
Lag.....	264
LastChild.....	265
LastPeriods	265
LastSibling.....	267
Lead.....	267
Leaves.....	268
Level.....	269
Levels.....	270
LinkMember.....	271
LinRegIntercept	272
LinRegPoint.....	273
LinRegR2	275
LinRegSlope.....	276
LinRegVariance	277
LookupCube.....	278
Max.....	279
MeasureGroupMeasures.....	281
Median.....	281
Members (Set).....	283
Members (String).....	284
MemberToStr	285
MemberValue.....	286
Min	287
Mtd.....	288
Name.....	289
NameToSet.....	290
NextMember.....	291
NonEmpty.....	292
NonEmptyCrossjoin	294
OpeningPeriod.....	295
Order.....	296
Ordinal	301
ParallelPeriod.....	302
Parent.....	303
PeriodsToDate.....	304
Predict	306
PrevMember	307
Properties.....	309
Qtd.....	311
Rank.....	312

RollupChildren	314
Root.....	316
SetToArray.....	317
SetToStr.....	318
Siblings.....	319
Stddev	320
StddevP.....	320
Stdev.....	320
StdevP	321
StripCalculatedMembers.....	322
StrToMember	324
StrToSet	326
StrToTuple	327
StrToValue	329
Subset.....	330
Sum	331
Tail	333
This	334
ToggleDrillState.....	335
TopCount.....	337
TopPercent	338
TopSum	341
TupleToStr	343
Union.....	344
UniqueName.....	346
UnknownMember.....	347
Unorder.....	349
UserName.....	350
ValidMeasure.....	350
Value	352
Var	353
Variance.....	354
VarianceP	354
VarP.....	354
VisualTotals	355
Wtd.....	356
Ytd	357
MDX Reserved Words.....	358

Multidimensional Expressions (MDX) Reference

In this Section

Topic	Description
Defining Assignments and Other Script Commands	Describes the various syntax elements available in the MDX language for MDX expressions, statements, and scripts.
MDX Language Reference (MDX)	Describes the statements, operators, and functions that define the MDX language.

See Also

[Querying Multidimensional Data \(Analysis Services - Multidimensional Data\)](#)

[Analysis Services Scripting Language Reference](#)

[Retrieving Data from an Analytical Data Source](#)

[Creating and Editing MDX Scripts](#)

MDX Syntax Elements

Multidimensional Expressions (MDX) has several elements that are used by, or influence, most statements:

Term	Definition
Identifiers	Identifiers are the names of objects such as cubes, dimensions, members, and measures.
Data Types	Define the types of data that are contained by cells, member properties, and cell properties. MDX supports only the OLE VARIANT data type. For more information about the coercion, conversion, and manipulation of the VARIANT data type,

Term	Definition
	see "VARIANT and VARIANTARG" in the Platform SDK documentation.
Multidimensional Expressions (MDX) Reference	Expressions are units of syntax that Microsoft SQL Server Analysis Services can resolve to single (scalar) values or objects. Expressions include functions that return a single value, a set expression, and so on.
Operators	Operators are syntax elements that work with one or more simple MDX expressions to make more complex MDX expressions.
Functions	Functions are syntax elements that take zero, one, or more input values, and return a scalar value or an object. Examples include the Sum function for adding several values, the Members function for returning a set of members from a dimension or level, and so on.
Comments	Comments are pieces of text that are inserted into MDX statements or scripts to explain the purpose of the statement. Analysis Services does not execute comments .
Reserved Keywords	Reserved keywords are words that are reserved for the use of MDX and should not be used for object names used in MDX statements.
Members, Tuples, and Sets	Members, tuples and sets are core concepts of multidimensional data that you must understand before you create an MDX query.

See Also

[Multidimensional Expressions \(MDX\) Reference](#)

Identifiers

An identifier is the name of an Microsoft SQL Server Analysis Services object. Every Analysis Services object can and must have an identifier. This includes cubes, dimensions, hierarchies, levels, members, and so on. You use the identifier of an object to reference the object in Multidimensional Expressions (MDX) statements.

Depending on how you name the object, the identifier of the object identifier will be either a regular or delimited identifier.



Note

Both regular and delimited identifiers must contain from 1 through 100 characters.

Using Regular Identifiers

A regular identifier is an object name that complies with the following formatting rules for regular identifiers. A regular identifier can be used with or without delimiters.

Formatting Rules for Regular Identifiers

1. The first character must be one of the following:
 - A letter as defined by the Unicode Standard 2.0. Besides letter characters from other languages, the Unicode definition of letters includes Latin characters from a through z and from A through Z.
 - The underscore (_).
2. Subsequent characters can be:
 - Letters as defined in the Unicode Standard 2.0.
 - Decimal numbers from either Basic Latin or other national scripts.
 - The underscore (_).
3. The identifier must not be an MDX reserved keyword. Reserved keywords are case-insensitive in MDX. For more information, see [MDX Syntax Elements \(MDX\)](#).
4. Embedded spaces or special characters are not allowed.

Examples of Regular Identifiers

In the following MDX statement, the identifiers, *Measures*, *Product*, and *Style*, comply with the formatting rules for regular identifiers. These regular identifiers do not need delimiters.

```
SELECT Measures.MEMBERS ON COLUMNS,  
Product.Style.CHILDREN ON ROWS  
FROM [Adventure Works]
```

Although not required, you could also use delimiters with regular identifiers. In the following MDX statement, the `Measures`, `Product`, and `Style` regular identifiers have been correctly delimited by using brackets.

```
SELECT [Measures].MEMBERS ON COLUMNS,  
[Product].[Style].CHILDREN ON ROWS  
FROM [Adventure Works]
```

Using Delimited Identifiers

An identifier that does not comply with the formatting rules for regular identifiers must always be delimited by using brackets ([]).

Note

Delimiters are for identifiers only. Delimiters cannot be used for keywords, whether or not the keywords are marked as reserved in Analysis Services.

You use a delimited identifier in the following situations:

- When the name of an object or part of the name uses reserved words.
We recommend that reserved keywords not be used as object names. Databases upgraded from earlier versions of Analysis Services may contain identifiers that include words not reserved in the earlier version, but are reserved words for SQL Server Analysis Services. Until you can change the identifier for the object, you can reference the object using a delimited identifier.
- When the name of an object uses characters not listed as qualified identifiers.
Analysis Services allows a delimited identifier to use any character in the current code page. However, indiscriminate use of special characters in an object name may make MDX statements and scripts difficult to read and maintain.

Formatting Rules for Delimited Identifiers

The body of a delimited identifier can contain any combination of characters in the current code page, including the delimiting characters themselves. If the body of the delimited identifier contains delimiting characters, special handling is required:

- If the body of the identifier contains only a left bracket ([), no additional handling is required.
- If the body of the identifier contains a right bracket (]), you must specify two right brackets (]]).

Examples of Delimited Identifiers

In the following hypothetical MDX statement, `Sales Volume`, `Sales Cube`, and `select` are delimited identifiers:

-- The [Sales Volume] and [Sales Cube] identifiers contain a space.

```
SELECT Measures.[Sales Volume]
```

```
FROM [Sales Cube]
```

```
WHERE Product.[select]
```

-- The [select] identifier is a reserved keyword.

In this next example, the name of an object is `Total Profit [Domestic]`. To reference this object, you must use the following delimited identifier:

```
[Total Profit [Domestic]]]
```

Notice that the left bracket before `Domestic` did not have to be changed to create the delimited identifier. However, the right bracket following `Domestic` had to be replaced with two right brackets.

Delimiting Identifiers with Multiple Parts

When you use qualified object names you may have to delimit more than one of the identifiers that make up the object name. For example, the `Front Brakes` identifier in the following code needs delimiting.

```
SELECT [Measures].MEMBERS ON COLUMNS,
```

```
[Product].[Product].[Front Brakes] ON ROWS
```

```
FROM [Adventure Works]
```

In addition, the `Measures` identifier in the previous example was delimited to demonstrate delimiting more than one identifier.

See Also

[MDX Language Reference](#)

[MDX Query Fundamentals \(MDX\)](#)

[MDX Syntax Elements \(MDX\)](#)

Expressions

An expression is a combination of identifiers, values, and operators that Microsoft SQL Server Analysis Services can evaluate to get a result. The data can be used in several different places when accessing or changing data. For example, you can use an expression as part of the data to be retrieved by a query or as a search condition to look for data that meets a set of criteria.

Simple and Complex Expressions

An expression can be simple or complex in MDX:

A simple expression can be one of the following expressions:

Constant

A constant is a symbol that represents a single, specific value in MDX. String, numeric, and date values can be rendered as constants. Unlike numeric constants, string and date constants must be delimited with single quote (') characters.

Scalar function

A scalar function returns a single value within the context of evaluation in MDX. This distinction is important to understanding how MDX resolves scalar functions, because most MDX expressions, statements, and scripts are evaluated not over a single data element, but iteratively over a group of data elements such as cells or members. At the time the scalar function is evaluated, however, the function is typically reviewing a single data element.

Object identifier

MDX is object-oriented because of the nature of multidimensional data. Object identifiers are considered simple expressions in MDX. For more information on identifiers, see [MDX Query Fundamentals \(MDX\)](#).

A complex expression can be built from combinations of these entities joined by operators.

Expression Results

For a simple expression built of a single constant, variable, scalar function, or column name, the data type, collation, precision, scale, and value of the expression is the data type, collation, precision, scale, and value of the referenced element. Because MDX directly supports only the OLE VARIANT data type, coercion should not occur when working with simple expressions.

For a complex expression, coercion can occur when using two or more simple expressions with different data types.

Expression Examples

The following query shows examples of calculated measures whose definitions are simple expressions:

```
WITH
MEMBER MEASURES.CONSTANTVALUE AS 1
MEMBER MEASURES.SCALARFUNCTION AS [Date].[Calendar Year].CURRENTMEMBER.NAME
MEMBER MEASURES.OBJECTIDENTIFIER AS [Measures].[Internet Sales Amount]
SELECT
{MEASURES.CONSTANTVALUE,MEASURES.SCALARFUNCTION,MEASURES.OBJECTIDENTIFIER }
ON 0,
[Date].[Calendar Year].MEMBERS ON 1
FROM [Adventure Works]
```

An expression can also be a calculation, such as `[Measures].[Discount Amount] * 1.5`. The following example demonstrates the use of a calculation to define a member in an MDX SELECT statement:

```
WITH
    MEMBER [Measures].[Special Discount] AS
        [Measures].[Discount Amount] * 1.5
SELECT
    [Measures].[Special Discount] on COLUMNS,
    NON EMPTY [Product].[Product].MEMBERS ON Rows
FROM [Adventure Works]
WHERE [Product].[Category].[Bikes]
```

In This Section

Topic	Description
Using Cube and Subcube Expressions	Defines cube and subcube expressions.
Using Dimension Expressions	Defines dimension expressions.
Using Member Expressions	Defines member expressions.
Using Tuple Expressions	Defines tuple expressions.
Using Set Expressions	Defines set expressions.
Using Scalar Expressions	Defines scalar expressions.
Working with Empty Values	Describes what an empty value is and how such values are handled.

See Also

[MDX Language Reference](#)

[MDX Query Fundamentals \(MDX\)](#)

Using Cube and Subcube Expressions

You use cube and subcube expressions in Multidimensional Expressions (MDX) statements to define, manipulate, or retrieve data from a cube or subcube.

Cube Expressions

A cube expression contains either a cube identifier or the CURRENTCUBE keyword, and therefore can only be simple expressions. Many MDX statements use the CURRENTCUBE keyword to identify the current cube context instead of requiring a cube identifier.

A cube identifier appears as Cube_Name in BNF notation descriptions of MDX statements.

Cube expressions may appear in several places. In an MDX SELECT statement they specify the cube from which data is to be retrieved. In the following example query, the expression [Adventure Works] refers to the cube of that name:

```
SELECT [Measures].[Internet Sales Amount] ON COLUMNS
FROM [Adventure Works]
```

In the CREATE MEMBER statement, the cube expression specifies which cube the calculated member you are creating is to appear on. In the following example, the statement creates a calculated measure on the Measures dimension of the Adventure Works cube:

```
CREATE MEMBER [Adventure Works].[Measures].[Test] AS 1
```

When you use the CREATE MEMBER statement inside an MDX Script, the name of the cube can be replaced with the CURRENTCUBE keyword, since the cube where the calculated member is to be created must be the same cube that the MDX Script belongs to, as shown in the following example:

```
CREATE MEMBER CURRENTCUBE.[Measures].[Test] AS 1;
```

Doing this makes it easier to copy and paste calculated member definitions from one cube to another since the name of the cube is no longer hard-coded.

SubCube Expressions

A subcube expression can contain a subcube identifier or an MDX statement that returns a subcube. If the subcube expression contains a subcube identifier, it will be a simple expression. If it contains an MDX statement that returns a subcube, it is a complex statement. The MDX SELECT statement, for example, returns a subcube and can be used where subcube expressions are allowed, as shown in the following example:

```
SELECT [Measures].MEMBERS ON COLUMNS,
[Date].[Calendar Year].MEMBERS ON ROWS
FROM
(SELECT [Measures].[Internet Sales Amount] ON COLUMNS,
[Date].[Calendar Year].[2004] ON ROWS
FROM [Adventure Works])
```

This use of a SELECT statement in the FROM clause is also referred to as a subselect.

Another common scenario where subcube expressions are encountered is when making scoped assignments in an MDX Script. In the following example, the SCOPE statement is used to limit an assignment to a subcube consisting of [Measures].[Internet Sales Amount]:


```
SCOPE ([Measures].[Internet Sales Amount]);  
    This=1;  
END SCOPE;
```

A subcube identifier appears as `Subcube_Name`. in BNF notation descriptions of MDX statements.

See Also

[The Basic MDX Query \(MDX\)](#)

[Building Subcubes in MDX \(MDX\)](#)

[CREATE SUBCUBE Statement \(MDX\)](#)

[Expressions \(MDX\)](#)

[SCOPE Statement \(MDX\)](#)

Using Dimension Expressions

You typically use dimension and hierarchy expressions when passing parameters to functions in Multidimensional Expressions (MDX) to return members, sets, or tuples from a hierarchy.

Dimension expressions can only be simple expressions because they are object identifiers. See [Expressions \(MDX\)](#) for an explanation of simple and complex expressions.

Dimension Expressions

A dimension expression either contains a dimension identifier or a dimension function.

Dimension expressions are rarely used on their own. Instead, you will usually want to specify a hierarchy on a dimension. The only exception is when you are working with the Measures dimension, which has no hierarchies.

The following example shows a calculated member that uses the expression `[Measures]` along with the `.Members` and `Count()` functions to return the number of members on the Measures dimension:

```
WITH MEMBER [Measures].[MeasureCount] AS  
COUNT ([Measures].MEMBERS)  
SELECT [Measures].[MeasureCount] ON 0  
FROM [Adventure Works]
```

A dimension identifier appears as `Dimension_Name` in the BNF notation used to describe MDX statements.

Hierarchy Expressions

Similarly, a hierarchy expression contains either a hierarchy identifier or a hierarchy function. The following example shows the use of the hierarchy expression `[Date].[Calendar]`, along with the

.Levels and .Count functions, to return the number of levels in the Calendar hierarchy of the Date dimension:

```
WITH MEMBER [Measures].[CalendarLevelCount] AS  
[Date].[Calendar].Levels.Count  
SELECT [Measures].[CalendarLevelCount] ON 0  
FROM [Adventure Works]
```

The most common scenario where hierarchy expressions are used is in conjunction with the .Members function, to return all the members on a hierarchy. The following example returns all the members of [Date].[Calendar] on the rows axis:

```
SELECT [Measures].[Internet Sales Amount] ON 0,  
[Date].[Calendar].MEMBERS ON 1  
FROM [Adventure Works]
```

A hierarchy identifier appears as Dimension_Name.Hierarchy_Name in the BNF notation used to describe MDX statements.

See Also

[Expressions \(MDX\)](#)

Using Member Expressions

A member expression contains a member identifier, a member function, or an expression that can be converted to a member.

Member identifiers can come in many different formats. The simplest form of a member identifier consists of the member's name. For example:

```
SELECT Amount ON 0  
FROM [Adventure Works]
```

However, if there are several members with the same name on different hierarchies, there is no method to determine which member the query will return. For example, the following query requests data for a member with the name [CY 2004]. The query executes successfully, but there are at least six members with that name in the Adventure Works cube:

```
SELECT [CY 2004] ON 0  
FROM [Adventure Works]
```

Therefore, the most reliable form of member identifier is the member's unique name, which guarantees to identify a specific member in a cube. Analysis Services can generate unique names in several ways, but a unique name is always composed of at least two identifiers: the

dimension name, and the member name or member key. A unique name appears in the following format:

```
Dimension_Name.[Hierarchy_Name.] [[{Member_Name | &Member_Key}.]... ]  
{Member_Name | &Member_Key}
```

Here are some examples of member unique names from the Adventure Works cube:

```
[Measures].[Amount]  
[Date].[Calendar Year].&[2004]  
[Date].[Calendar].[Calendar Quarter].&[2004]&[1]  
[Employee].[Employees].&[112]  
[Product].[Product Categories].[All Products]
```

Many MDX functions exist that return members. For a full list, see [MDX Function Reference \(MDX\)](#)

Note

For more information about member names and member keys, see [Working with Members, Tuples, and Sets \(MDX\)](#).

See Also

[Expressions \(MDX\)](#)

Using Tuple Expressions

A tuple is made up of one member from every dimension that is contained within a cube. Therefore, a tuple uniquely identifies a single cell within the cube.

Note

A tuple that references one or more members that are not valid is known as an empty tuple.

The complete expression of a tuple identifier is made up of one or more explicitly specified members, framed in parentheses:

```
(Member_expression [ ,Member_expression ... ] )
```

A tuple can be fully qualified, can contain implicit members, or can contain a single member.

Tuples and Implicit Members

A tuple that explicitly specifies a single member from every dimension that is contained within a cube is known as a fully qualified tuple. However, a tuple does not have to be fully qualified.

Any dimension not explicitly referenced within a tuple is implicitly referenced. The member for the implicitly referenced dimension depends on the structure of the dimension and the attribute relationships defined within it. If there is an explicit reference to a hierarchy on the same dimension as the implicitly referenced hierarchy, and there is a direct or indirect relationship defined between the explicitly referenced hierarchy and the implicitly referenced hierarchy, then the tuple behaves as if it contains the member on the implicitly referenced hierarchy that exists with the member on the explicitly referenced hierarchy. For example, if a cube contains a Customer dimension with City and Country attributes, and there is a relationship defined between these two attributes so that a City has one Country and a Country can contain many Cities, then explicitly including the City 'London' in your tuple implicitly references the Country 'United Kingdom'. However, if no attribute relationships are defined, the relationship is in the opposite direction (for example, although City might have a relationship with Country, you cannot determine the City someone lives in just from knowing the Country they live in) or there are no direct relationships between the two attributes defined (there could be a relationship defined from Customer to City and Customer to Country, but no relationship defined between City and Country) then the following rules apply:

- If the implicitly referenced hierarchy has a default member, the default member is added to the tuple.
- If the implicitly referenced hierarchy has no default member, the **(All)** member of the default hierarchy is used.
- If the implicitly referenced hierarchy has no default member the first member of the topmost level of the hierarchy is used.

One-Member Tuples

If the tuple expression has a single member, MDX converts the member into a one-member tuple for the purposes of evaluating the expression. In other words, providing the member expression `[Measures].[TestMeasure]` instead of a tuple expression is functionally equivalent to the tuple expression `([Measures].[TestMeasure])`.

See Also

[Expressions \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Using Set Expressions

A set consists of an ordered list of zero or more tuples. A set that does not contain any tuples is known as an empty set.

The complete expression of a set consists of zero or more explicitly specified tuples, framed in curly braces:

```
{ { Tuple_expression | Member_expression } [, { Tuple_expression | Member_expression } ] ... }
```

The member expressions specified in a set expression are converted to one-member tuple expressions.

Example

The following example shows two set expressions used on the Columns and Rows axes of a query:

```
SELECT
{[Measures].[Internet Sales Amount], [Measures].[Internet Tax Amount]} ON
COLUMNS,
{([Product].[Product Categories].[Category].&[4], [Date].[Calendar].[Calendar
Year].&[2004]),
([Product].[Product Categories].[Category].&[1], [Date].[Calendar].[Calendar
Year].&[2003]),
([Product].[Product Categories].[Category].&[3], [Date].[Calendar].[Calendar
Year].&[2004])}
ON ROWS
FROM [Adventure Works]
```

On the Columns axis, the set

```
{[Measures].[Internet Sales Amount], [Measures].[Internet Tax Amount]}
```

consists of two members from the Measures dimension. On the Rows axis, the set

```
{([Product].[Product Categories].[Category].&[4], [Date].[Calendar].[Calendar Year].&[2004]),
([Product].[Product Categories].[Category].&[1], [Date].[Calendar].[Calendar Year].&[2003]),
([Product].[Product Categories].[Category].&[3], [Date].[Calendar].[Calendar Year].&[2004])}
```

consists of three tuples, each of which contains two explicit references to members on the Product Categories hierarchy of the Product dimension and the Calendar hierarchy of the Date dimension.

For examples of functions that return sets, see [Working with Members, Tuples, and Sets \(MDX\)](#).

See Also

[Expressions \(MDX\)](#)

Using Scalar Expressions

In Multidimensional Expressions (MDX), a scalar expression is an element of MDX syntax that, when evaluated, returns a single value within the context of evaluation.

Scalar expressions include string, numeric, and date expressions in MDX.

Scalar expressions are typically used in calculated member definitions, as calculated members must return a scalar value. The following query shows examples of calculated members on the Measures dimension that use different types of scalar expression:

```
WITH
MEMBER MEASURES.NumericValue AS 10
MEMBER MEASURES.NumericExpression AS 10 + 10
MEMBER MEASURES.NumericExpressionBasedOnMeasure AS [Measures].[Internet Sales Amount] + 10
MEMBER MEASURES.StringValue AS "10"
MEMBER MEASURES.ConcatenatedString AS "10" + "10"
MEMBER MEASURES.StringFunction AS MEASURES.CURRENTMEMBER.NAME
MEMBER MEASURES.TodaysDate AS NOW()
SELECT
{MEASURES.NumericValue,MEASURES.NumericExpression,MEASURES.NumericExpressionBasedOnMeasure,
MEASURES.StringValue, MEASURES.ConcatenatedString, MEASURES.StringFunction,
MEASURES.TodaysDate}
ON COLUMNS
FROM [Adventure Works]
```

The only data type that a measure, calculated or otherwise, can return is the OLE Variant type. Therefore, sometimes you might need to cast a measure value to a particular type to receive the behavior you expect. The following query shows an example of this:

```
WITH
//Two calculated measures that return strings
MEMBER MEASURES.NumericString1 AS "10"
MEMBER MEASURES.NumericString2 AS "10"
//In this case, the + operator acts to concatenate the strings
MEMBER MEASURES.Concatenation AS MEASURES.NumericString1 +
MEASURES.NumericString2
//Casting one value to an integer with the CINT function causes the second
measure
//to be treated as an integer too, so that the + operator now acts to add the
values
MEMBER MEASURES.Addition AS CINT(MEASURES.NumericString1) +
MEASURES.NumericString2
SELECT
```

```
{MEASURES.NumericString1,MEASURES.NumericString2,MEASURES.Concatenation,MEASURES.Addition }
ON COLUMNS
FROM [Adventure Works]
```

See Also

[Expressions \(MDX\)](#)

Working with Empty Values

An empty value indicates that a specific member, tuple, or cell is empty. An empty cell value indicates either that the data for the specified cell cannot be found in the underlying fact table, or that the tuple for the specified cell represents a combination of members that is not applicable for the cube.

Note

Although an empty value is different from a value of zero, an empty value is typically treated as zero most of the time.

The following query illustrates the behavior of empty and zero values:

```
WITH
//A calculated Product Category that always returns 0
MEMBER [Product].[Category].[All Products].ReturnZero AS 0
//Will return true for any null value
MEMBER MEASURES.ISEMPYDemo AS ISEMPY([Measures].[Internet Tax Amount])
//Will true for any null or zero value
//To be clear: the expression 0=null always returns true in MDX
MEMBER MEASURES.IsZero AS [Measures].[Internet Tax Amount]=0
SELECT
{[Measures].[Internet Tax Amount],MEASURES.ISEMPYDemo,MEASURES.IsZero}
ON COLUMNS,
[Product].[Category].[Category].ALLMEMBERS
ON ROWS
FROM [Adventure Works]
WHERE ([Date].[Calendar].[Calendar Year].&[2001])
```

The following information applies to empty values:

- The IsEmpty function returns **TRUE** if and only if the cell identified by the tuple specified in the function is empty. Otherwise, the function returns **FALSE**.



Note

The **IsEmpty** function cannot determine whether a member expression returns a null value. To determine whether a null member is returned from an expression, use the IS operator.

- When the empty cell value is an operand for any one of the numeric operators (+, -, *, /), the empty cell value is treated as zero if the other operand is a nonempty value. If both operands are empty, the numeric operator returns the empty cell value.
- When the empty cell value is an operand for the string concatenation operator (+), the empty cell value is treated as an empty string if the other operand is a nonempty value. If both operands are empty, the string concatenation operator returns the empty cell value.
- When the empty cell value is an operand for any one of the comparison operators (=, <>, >=, <=, >, <), the empty cell value is treated as zero or an empty string, depending on whether the data type of the other operand is numeric or string, respectively. If both operands are empty, both operands are treated as zero.
- When collating numeric values, the empty cell value collates in the same place as zero. Between the empty cell value and zero, empty collates before zero.
- When collating string values, the empty cell value collates in the same place as the empty string. Between the empty cell value and the empty string, empty collates before an empty string.

Dealing with Empty Values in MDX Statements and Cubes

In Multidimensional Expressions (MDX) statements, you can look for empty values and then perform certain calculations on cells with valid (that is, not empty) data. Eliminating empty values when performing calculations can be important because certain calculations, such as an average, can be inaccurate if empty cell values are included.

If empty values are stored in your underlying fact table data, by default they will be converted to zeroes when the cube is processed. You can use the **Null Processing** option on a measure to control whether null facts are converted into 0, converted to an empty value, or even throws an error during processing. If you do not want empty cell values appearing in your query results, you should create queries, calculated members, or MDX Script statements that eliminate the empty values or replace them with some other value.

To remove empty rows or columns from a query, you can use the NON EMPTY statement before the axis set definition. For example, the following query only returns the Product Category Bikes because that is the only Category that was sold in the Calendar Year 2001:

```
SELECT
{[Measures].[Internet Tax Amount]}
ON COLUMNS,
//Comment out the following line to display all the empty rows for other
Categories
```



```

NON EMPTY
[Product].[Category].[Category].MEMBERS
ON ROWS
FROM [Adventure Works]
WHERE ([Date].[Calendar].[Calendar Year].&[2001])

```

More generally, to remove empty tuples from a set you can use the NonEmpty function. The following query shows two calculated measures, one of which counts the number of Product Categories and the second shows the number of Product Categories which have values for the measure [Internet Tax Amount] and the Calendar Year 2001:

```

WITH
MEMBER MEASURES.CategoryCount AS
COUNT ([Product].[Category].[Category].MEMBERS)
MEMBER MEASURES.NonEmptyCategoryCountFor2001 AS
COUNT (
NONEMPTY (
[Product].[Category].[Category].MEMBERS
, ([Date].[Calendar].[Calendar Year].&[2001], [Measures].[Internet Tax
Amount]))
))
SELECT
{MEASURES.CategoryCount,MEASURES.NonEmptyCategoryCountFor2001 }
ON COLUMNS
FROM [Adventure Works]

```

For more information, see [NonEmpty \(MDX\)](#).

Empty Values and Comparison Operators

When empty values are present in data, logical and comparison operators can potentially return a third result of EMPTY instead of just TRUE or FALSE. This need for three-valued logic is a source of many application errors. These tables outline the effect of introducing empty value comparisons.

This table shows the results of applying an AND operator to two Boolean operands.

AND	TRUE	EMPTY	FALSE
TRUE	TRUE	FALSE	FALSE
EMPTY	FALSE	EMPTY	FALSE

AND	TRUE	EMPTY	FALSE
FALSE	FALSE	FALSE	FALSE

This table shows the results of applying an OR operator to two Boolean operands.

OR	TRUE	FALSE
TRUE	TRUE	TRUE
EMPTY	TRUE	TRUE
FALSE	TRUE	FALSE

This table shows how the NOT operator negates, or reverses, the result of a Boolean operator.

Boolean expression to which the NOT operator is applied	Evaluates to
TRUE	FALSE
EMPTY	EMPTY
FALSE	TRUE

See Also

[Expressions \(MDX\)](#)

[MDX Operator Reference \(MDX\)](#)

[Expressions \(MDX\)](#)

Operators (MDX Syntax)

In Multidimensional Expressions (MDX), operators let you perform the following actions:

- Change data, either permanently or temporarily.
- Search for values or objects that meet a specified condition.
- Implement a decision between values or expressions.
- Test for specific conditions before beginning or committing a transaction, or before executing specific statements.

MDX supports the operators listed in the following table:

To perform this type of operation	Use
Assigns a value to a variable, or associates a result set column with an alias.	MDX Syntax Elements (MDX)
Addition, subtraction, multiplication, division.	Arithmetic Operators
Test for the truth of a condition, such as AND, OR, NOT, and XOR.	Bitwise Operators
Compare a value against another value or an expression.	Comparison Operators
Either permanently or temporarily combine two strings into one string.	Concatenation Operators
Either permanently or temporarily combine two set expressions into a single set.	Set Operators
Performs an operation on one operand.	Unary Operators

Note

In queries, anyone who can see the data in the cube to be used with some type of operator can perform operations. However, you need the appropriate permissions before you can successfully change the data.

When using multiple operators, the order in which MDX evaluates the operators is important. Similarly, the user of operators may require that one data type be converted into another data type before the operators can be evaluated.

Evaluating Complex Expressions

You can build an expression by using operators to combine several smaller expressions. In these complex expressions, MDX evaluates the operators in order based on the definition of operator precedence used by Microsoft SQL Server Analysis Services. MDX performs operators with higher precedence before performing operators with lower precedence.

Understanding Operator Precedence

The following list shows operator precedence, from highest to lowest. Operators in the same line are equal in precedence, and are evaluated from left to right unless otherwise forced by parenthesis:

- IS
- AS

- DISTINCT
- :
- ^
- /, *
- +, -
- EXISTING
- <>, >=, =, <=, >, <
- NOT
- AND
- XOR
- OR

For more information about operators in MDX, see [MDX Operator Reference \(MDX\)](#).

Determining Results

When you combine simple expressions to form a complex expression, the rules for the operators combined with the rules for data type precedence determine the data type of the resulting value.

If the result is a character or Unicode value, the rules for the operators combined with the rules for collation precedence determines the collation of the result. For more information about collations, see [Working with Languages and Collations \(SSAS\)](#).

There are also rules that determine the precision, scale, and length of the result based on the precision, scale, and length of the simple expressions.

Converting Data Types

MDX implicitly converts an object to a different type when that object is used in an expression that requires a different type. The following table defines the conversion rules for each object.

Original Type	Type Needed	Conversion
Level	Set	<level>.members
Hierarchy	Member	<hierarchy>.defaultmember
Member	Tuple	(<Member>)
Tuple	Member	<tuple>.item(0)
Tuple	Scalar	<tuple>.value

See Also

[MDX Operator Reference](#)

[MDX Syntax Elements \(MDX\)](#)

Arithmetic Operators

You can use arithmetic operators in Multidimensional Expressions (MDX) for any arithmetic computations, including addition, subtraction, multiplication, and division.

MDX supports the arithmetic operators listed in the following table.

Operator	Description
+ (Add)	Adds two numbers.
/ (Divide)	Divides one number by another number.
* (Multiply)	Multiplies two numbers.
- (Subtract)	Subtracts two numbers.
^ (Power)	Raises one number by another number.

Note

MDX does not include a function to obtain the square root of a number. To obtain the square root of a number, raise it to the power of 0.5 using the ^ operator.

Order of Precedence

The following rules determine the order of precedence for arithmetic operators in an MDX expression:

- When there is more than one arithmetic operator in an expression, MDX performs multiplication and division first, followed by subtraction and addition.
- When all arithmetic operators in an expression have the same level of precedence, the order of execution is left to right.
- Expressions within parentheses take precedence over all other operations.

See Also

[Operators \(MDX Syntax\)](#)

[Operators \(MDX Syntax\)](#)

Bitwise Operators

Logical operators evaluate values and return a Boolean value. In Multidimensional Expressions (MDX), logical operators do not perform bitwise operations.

MDX supports the logical operators listed in the following table.

Operator	Description
AND	Performs a logical conjunction on two numeric expressions.
IS	Performs a logical comparison on two object expressions.
NOT	Performs a logical negation on a numeric expression.
OR	Performs a logical disjunction on two numeric expressions.
XOR	Performs a logical exclusion on two numeric expressions.

See Also

[Operators \(MDX Syntax\)](#)

[Operators \(MDX Syntax\)](#)

Comparison Operators

You use comparison operators with scalar data. You can use comparison operators in any Multidimensional Expressions (MDX) expression.

To check for a condition, you can also use comparison operators in MDX statements and functions, such as the MDX Iif function. However, if you use comparison operators to check for a condition, make sure that you have appropriate permissions before trying to change data based upon that condition. Anyone that has access to the actual data and can query that data can use comparison operators in additional queries. But this access does not mean that these individuals have or should have the appropriate permissions to change data. Also, to maintain data integrity, limit the number of people that can query and change data.

Comparison operators evaluate to a Boolean data type, returning TRUE or FALSE based on the outcome of the tested condition.

MDX supports the comparison operators listed in the following table.

Operator	Description
= (Equal To)	<p>For non-null arguments, returns TRUE if the left argument is equal to the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value, unless the comparison <code>0=null</code> is made, in which case the Boolean contains TRUE.</p>
<> (Not Equal To)	<p>For non-null arguments, returns TRUE if the left argument is not equal to the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value.</p>
> (Greater Than)	<p>For non-null arguments, returns TRUE if the left argument has a value that is greater than the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value.</p>
>= (Greater Than or Equal To)	<p>For non-null arguments, returns TRUE if the left argument has a value that is higher than or equal to the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value.</p>
< (Less Than)	<p>For non-null arguments, returns TRUE if the left argument has a value that is less than than the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value.</p>
<= (Less Than or Equal To)	<p>For non-null arguments, returns TRUE if the left argument has a value that is lower than or equal to the right argument; otherwise, FALSE.</p> <p>If either or both arguments evaluate to a null value, the operator returns a null value.</p>

See Also

[Operators \(MDX Syntax\)](#)

[Operators \(MDX Syntax\)](#)

Concatenation Operators

The concatenation operator is the plus sign (+). You can combine, or concatenate, two or more character strings into a single character string. You can also concatenate binary strings.

The following code is an example of concatenation operator that combines the product name with the product's unique name:

```
WITH MEMBER Measures.ProductName AS
    Product.Product.CurrentMember.Name + " (" +
    Product.Product.CurrentMember.UniqueName + ")"
SELECT
    {Measures.ProductName} ON COLUMNS,
    Product.Product.Members ON ROWS
FROM [Adventure Works]
```

Language Considerations

When the strings used in a concatenation both have the same collation, the resulting concatenated string has the same collation as the inputs. When the strings used in a concatenation have different collations, the rules of collation precedence determine the collation of the resulting concatenated string. For more information, see [Operators \(MDX Syntax\)](#).

See Also

[MDX Operator Reference](#)

[Operators \(MDX Syntax\)](#)

Set Operators

In Multidimensional Expressions (MDX), set operators perform operations on members or sets, and return a set. You often use set operators as an alternate to several set functions in MDX expressions.

MDX supports the set operators listed in the following table.

Operator	Description
- (Except)	Returns the difference between two sets,

Operator	Description
	removing duplicate members. This operator is functionally equivalent to the Except function.
* (Crossjoin)	Returns the cross product of two sets. This operator is functionally equivalent to the Crossjoin function.
: (Range)	Returns a naturally ordered set, with the two specified members as endpoints and all members between the two specified members included as members of the set.
+ (Union)	Returns a union of two sets, excluding duplicate members. This operator is functionally equivalent to the Operators (MDX Syntax) function.

See Also

[MDX Function Reference](#)

[MDX Operator Reference](#)

[Operators \(MDX Syntax\)](#)

Unary Operators

In Multidimensional Expressions (MDX), unary operators perform an operation on a single operand, such as returning the negative or positive value of a numeric expression.

MDX supports the unary operators listed in the following table.

Operator	Description
- (Negative)	Returns the negative value of a numeric expression.
+ (Positive)	Returns the positive value of a numeric expression.

The following example demonstrates the use of a unary operator to return the negative value of a measure:

```

WITH
    MEMBER [Measures].[NegDiscountAmount] AS
        -[Measures].[Discount Amount]
SELECT
    {[Measures].[Discount Amount],[Measures].[NegDiscountAmount]} on COLUMNS,
    NON EMPTY [Product].[Product].MEMBERS ON Rows
FROM [Adventure Works]
WHERE [Product].[Category].[Bikes]

```

In addition, MDX uses special unary operators to determine the aggregation operation performed by the RollupChildren function. For more information on these special unary operators, see [Adding a Custom Aggregation to a Dimension](#).

See Also

[Operators \(MDX Syntax\)](#)

Assignment Operators

In Multidimensional Expressions (MDX), the assignment operator is the equal sign (=). The assignment operator is used to assign values to subcubes in MDX Script. For more information, see [MDX Scripting Fundamentals \(MDX\)](#).

See Also

[Operators \(MDX Syntax\)](#)

Functions (MDX Syntax)

Multidimensional Expressions (MDX) has several categories of intrinsic functions to perform certain operations. The following table lists the function categories that are available in MDX.

Note

For more information about individual functions, see [MDX Syntax Elements \(MDX\)](#).

Function Category	Description
Array functions	Provide arrays for use in stored procedures. For more information, see Using Stored Procedures .
Dimension functions	Return a reference to a dimension from a

Function Category	Description
	<p>hierarchy, level, or member.</p> <p>For more information, see Using Dimension, Hierarchy, and Level Functions.</p>
Hierarchy functions	<p>Return a reference to a hierarchy from a level or member.</p> <p>For more information, see Using Dimension, Hierarchy, and Level Functions.</p>
Level functions	<p>Return a reference to a level from a member, dimension, hierarchy, or from a string expression.</p> <p>For more information, see Using Dimension, Hierarchy, and Level Functions.</p>
Logical functions	<p>Perform logical operations and comparisons on objects and expressions.</p> <p>For more information, see Using Logical Functions.</p>
Member functions	<p>Return a reference to a member from other objects or from a string expression.</p> <p>For more information, see Using Member Functions.</p>
Numeric functions	<p>Perform mathematical and statistical functions on objects and expressions.</p> <p>For more information, see Using Mathematical Functions.</p>
Set functions	<p>Return a reference to a set from other objects or from a string expression.</p> <p>For more information, see Using Set Functions.</p>
String functions	<p>Return string values from other objects or from the server.</p> <p>For more information, see Using String Functions.</p>
Tuple functions	<p>Return a reference to a tuple from a set or from a string expression.</p> <p>For more information, see Using Tuple</p>

Function Category	Description
	Functions.

Uses of Functions

Functions can be used or included in any MDX expression. Functions can also be nested (one function used inside another function).

See Also

[MDX Syntax Elements \(MDX\)](#)

Using String Functions

You can use string functions on nearly every object in Multidimensional Expressions (MDX). In stored procedures, you use string functions primarily to convert the object to a string representation. You also use string functions to evaluate a string expression over an object in order to return a value.

The most widely used string functions are **Name** and **Uniquename**. Respectively, these functions return the name and unique name of an object. Mostly, they are used when debugging calculations to discover what member a function is returning.

Examples

The following example queries show how to use these functions:

```
WITH
    //Returns the name of the current Product on rows
    MEMBER [Measures].[ProductName] AS [Product].[Product].CurrentMember.Name
    //Returns the uniquename of the current Product on rows
    MEMBER [Measures].[ProductUniqueName] AS
[Product].[Product].CurrentMember.Uniquename
    //Returns the name of the Product dimension
    MEMBER [Measures].[ProductDimensionName] AS [Product].Name
SELECT
{[Measures].[ProductName],[Measures].[ProductUniqueName],[Measures].[ProductD
imensionName]}
    ON COLUMNS,
    [Product].[Product].MEMBERS ON ROWS
FROM [Adventure Works]
```

The **Generate** function can be used to execute a string function on every member of a set and concatenate the results. This also can be useful when debugging calculations as it allows you to visualize the contents of a set. The following example shows how to use it in this way:

```
WITH
    //Returns the names of the current Product and its ancestors up to the All
Member
    MEMBER [Measures].[AncestorNames] AS
    GENERATE (
ASCENDANTS([Product].[Product Categories].CurrentMember)
, [Product].[Product Categories].CurrentMember.Name, ", ")
SELECT
    {[Measures].[AncestorNames]}
    ON COLUMNS,
    [Product].[Product Categories].MEMBERS ON ROWS
FROM [Adventure Works]
```

Another group of widely used string functions are those that enable you to cast a string containing the uniqueness of an object or an expression which resolves to the object into the object itself. The following example query demonstrates how the **StrToMember** and **StrToSet** functions do this:

```
SELECT
    {StrToMember("[Measures].[Inter" + "net Sales Amount]")}
    ON COLUMNS,
    StrToSet("{
[Product].[Product Categories].[Category].&[3],
[Product].[Product Categories].[Product].&[477],
[Product].[Product Categories].[Product].&[788],
[Product].[Product Categories].[Product].&[708],
[Product].[Product Categories].[Product].&[711]
}")
    ON ROWS
FROM [Adventure Works]
```



Note

The **StrToMember** and **StrToSet** functions should be used with caution. They can lead to poor query performance if they are used within calculation definitions.

See Also

[Generate \(MDX\)](#)

[Name \(MDX\)](#)

[UniqueName \(MDX\)](#)

[Using Stored Procedures \(MDX\)](#)

[Stored Procedures \(MDX\)](#)

[StrToMember \(MDX\)](#)

[StrToSet \(MDX\)](#)

Using Mathematical Functions

A mathematical function performs a mathematical operation on numeric expressions and returns the result of the operation.

By default, a number passed to a mathematical function will be interpreted as a double precision floating point number.

See Also

[Functions \(MDX Syntax\)](#)

Using Logical Functions

A logical function performs a logical operation or comparison on objects and expressions and returns a Boolean value. Logical functions are essential in Multidimensional Expressions (MDX) to determine the position of a member.

The most commonly used logical function is the **IsEmpty** function. For more information on how to use the **IsEmpty** function, see [Working with Empty Values](#).

The following query shows how to use the **IsLeaf** and **IsAncestor** functions:

```
WITH
//Returns true if the CurrentMember on Calendar is a leaf member, ie it has
no children
MEMBER MEASURES.[IsLeafDemo] AS IsLeaf([Date].[Calendar].CurrentMember)
//Returns true if the CurrentMember on Calendar is an Ancestor of July 1st
2001
MEMBER MEASURES.[IsAncestorDemo] AS
IsAncestor([Date].[Calendar].CurrentMember, [Date].[Calendar].[Date].&[1])
SELECT{MEASURES.[IsLeafDemo],MEASURES.[IsAncestorDemo] } ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[Functions \(MDX Syntax\)](#)

Using Member Functions

A member function is an Multidimensional Expressions (MDX) function that returns a member. Member functions, like tuple functions and set functions, are essential to negotiating the multidimensional structures found in Analysis Services.

Of the many member functions in MDX, the most important is the **CurrentMember** function, which is used to determine the current member on a hierarchy. The following query illustrates how to use it, along with the **Parent**, **Ancestor**, and **Prevmember** functions:

```
WITH
//Returns the name of the currentmember on the Calendar hierarchy
MEMBER MEASURES.[CurrentMemberDemo] AS [Date].[Calendar].CurrentMember.Name
//Returns the name of the parent of the currentmember on the Calendar
hierarchy
MEMBER MEASURES.[ParentDemo] AS [Date].[Calendar].CurrentMember.Parent.Name
//Returns the name of the ancestor of the currentmember on the Calendar
hierarchy at the Year level
MEMBER MEASURES.[AncestorDemo] AS ANCESTOR([Date].[Calendar].CurrentMember,
[Date].[Calendar].[Calendar Year]).Name
//Returns the name of the member before the currentmember on the Calendar
hierarchy
MEMBER MEASURES.[PrevMemberDemo] AS
[Date].[Calendar].CurrentMember.Prevmember.Name

SELECT{MEASURES.[CurrentMemberDemo],MEASURES.[ParentDemo],MEASURES.[AncestorD
emo],MEASURES.[PrevMemberDemo] } ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[Using Set Functions](#)

[Using Tuple Functions](#)

[Using Set Functions](#)

Using Tuple Functions

A tuple function retrieves a tuple from a set or retrieves a tuple by resolving the string representation of a tuple.

Tuple functions, like member functions and set functions, are essential to negotiating the multidimensional structures found in Analysis Services.

There are three tuple functions in MDX, [Current \(MDX\)](#), [Item \(Tuple\) \(MDX\)](#) and [StrToTuple \(MDX\)](#). The following example query shows how to use of each of them:

```
WITH
//Creates a set of tuples consisting of Years and Countries
SET MyTuples AS [Date].[Calendar Year].[Calendar Year].MEMBERS *
[Customer].[Country].[Country].MEMBERS
//Returns a string representation of that set using the Current and Generate
functions
MEMBER MEASURES.CURRENTDEMO AS GENERATE(MyTuples,
TUPLETOSTR(MyTuples.CURRENT), ", ")
//Retrieves the fourth tuple from that set and displays it as a string
MEMBER MEASURES.ITEMDEMO AS TUPLETOSTR(MyTuples.ITEM(3))
//Creates a tuple consisting of the measure Internet Sales Amount and the
country Australia from a string
MEMBER MEASURES.STRTOTUPLEDEMO AS STRTOTUPLE("([Measures].[Internet Sales
Amount]" + ", [Customer].[Country].&[Australia])")
SELECT{MEASURES.CURRENTDEMO,MEASURES.ITEMDEMO,MEASURES.STRTOTUPLEDEMO} ON
COLUMNS
FROM [Adventure Works]
```

See Also

[Using Set Functions](#)

[Using Member Functions](#)

[Using Set Functions](#)

Using Set Functions

A set function retrieves a set from a dimension, hierarchy, level, or by traversing the absolute and relative locations of members within these objects, constructing sets in a variety of ways.

Set functions, like member functions and tuple functions, are essential to negotiating the multidimensional structures found in Analysis Services. Set functions are also essential to obtaining results from Multidimensional Expressions (MDX) queries because set expressions define the axes of an MDX query.

One of the most common set functions is the [Members \(Set\) \(MDX\)](#) function, which retrieves a set containing all of the members from a dimension, hierarchy, or level. The following is an example of its use within a query:

```
SELECT
//Returns all of the members on the Measures dimension
[Measures].MEMBERS
ON Columns,
//Returns all of the members on the Calendar Year level of the Calendar Year
Hierarchy
//on the Date dimension
[Date].[Calendar Year].[Calendar Year].MEMBERS
ON Rows
FROM [Adventure Works]
```

Another commonly used function is the [Crossjoin \(MDX\)](#) function. It returns a set of tuples representing the cartesian product of the sets passed into it as parameters. In practical terms, this function enables you to create 'nested' or 'crosstabbed' axes in queries:

```
SELECT
//Returns all of the members on the Measures dimension
[Measures].MEMBERS
ON Columns,
//Returns a set containing every combination of all of the members
//on the Calendar Year level of the Calendar Year Hierarchy
//on the Date dimension and all of the members on the Category level
//of the Category hierarchy on the Product dimension
Crossjoin(
[Date].[Calendar Year].[Calendar Year].MEMBERS,
[Product].[Category].[Category].MEMBERS)
ON Rows
FROM [Adventure Works]
```

The [Descendants \(MDX\)](#) function is similar the **Children** function, but is more powerful. It returns the descendants of any member at one or more levels in a hierarchy:

```
SELECT
[Measures].[Internet Sales Amount]
ON Columns,
//Returns a set containing all of the Dates beneath Calendar Year
//2004 in the Calendar hierarchy of the Date dimension
```

```
DESCENDANTS(  
[Date].[Calendar].[Calendar Year].&[2004]  
, [Date].[Calendar].[Date])
```

ON Rows

```
FROM [Adventure Works]
```

The [Order \(MDX\)](#) function enables you to order the contents of a set in ascending or descending order according to a particular numeric expression. The following query returns the same members on rows as the previous query, but now orders them by the Internet Sales Amount measure:

```
SELECT  
[Measures].[Internet Sales Amount]  
ON Columns,  
//Returns a set containing all of the Dates beneath Calendar Year  
//2004 in the Calendar hierarchy of the Date dimension  
//ordered by Internet Sales Amount  
ORDER(  
DESCENDANTS(  
[Date].[Calendar].[Calendar Year].&[2004]  
, [Date].[Calendar].[Date])  
, [Measures].[Internet Sales Amount], BDESC)  
ON Rows  
FROM [Adventure Works]
```

This query also illustrates how the set returned from one set function, Descendants, can be passed as a parameter to another set function, Order.

Filtering a set according to certain criteria is very useful when writing queries, and for this purpose you can use the [Filter \(MDX\)](#) function, as shown in the following example:

```
SELECT  
[Measures].[Internet Sales Amount]  
ON Columns,  
//Returns a set containing all of the Dates beneath Calendar Year  
//2004 in the Calendar hierarchy of the Date dimension  
//where Internet Sales Amount is greater than $70000  
FILTER(  
DESCENDANTS(  
[Date].[Calendar].[Calendar Year].&[2004]  
, [Date].[Calendar].[Date])
```

```
, [Measures].[Internet Sales Amount]>70000)
```

```
ON Rows
```

```
FROM [Adventure Works]
```

Other, more sophisticated functions exist that allow you to filter a set in other ways. For example, the following query shows the [TopCount \(MDX\)](#) function returns the top n items in a set:

```
SELECT
```

```
[Measures].[Internet Sales Amount]
```

```
ON Columns,
```

```
//Returns a set containing the top 10 Dates beneath Calendar Year
```

```
//2004 in the Calendar hierarchy of the Date dimension by Internet Sales  
Amount
```

```
TOPCOUNT (
```

```
DESCENDANTS (
```

```
[Date].[Calendar].[Calendar Year].&[2004]
```

```
, [Date].[Calendar].[Date])
```

```
,10, [Measures].[Internet Sales Amount])
```

```
ON Rows
```

```
FROM [Adventure Works]
```

Finally it is possible to perform a number of logical set operations using functions such as [Intersect \(MDX\)](#), [Union \(MDX\)](#) and [Except \(MDX\)](#) functions. The following query shows examples of the latter two functions:

```
SELECT
```

```
//Returns a set containing the Measures Internet Sales Amount, Internet Tax  
Amount and
```

```
//Internet Total Product Cost
```

```
UNION (
```

```
{[Measures].[Internet Sales Amount], [Measures].[Internet Tax Amount]}
```

```
, {[Measures].[Internet Total Product Cost]}
```

```
)
```

```
ON Columns,
```

```
//Returns a set containing all of the Dates beneath Calendar Year
```

```
//2004 in the Calendar hierarchy of the Date dimension
```

```
//except the January 1st 2004
```

```
EXCEPT (
```

```
DESCENDANTS (
```

```
[Date].[Calendar].[Calendar Year].&[2004]
, [Date].[Calendar].[Date])
,{[Date].[Calendar].[Date].&[915]})
ON Rows
FROM [Adventure Works]
```

See Also

[Using Tuple Functions](#)

[Using Member Functions](#)

[Using Tuple Functions](#)

Using Dimension, Hierarchy, and Level Functions

Dimension, hierarchy, and level functions are useful for traversing the multidimensional structures found in Analysis Services. Typically, you use such functions in conjunction with other functions to obtain information about the members of a dimension, hierarchy, or level.

The following example shows how to use the **.Dimension**, **.Hierarchy**, and **.Level** functions:

```
WITH
MEMBER MEASURES.DIMENSIONNAME AS
[Date].[Calendar].CURRENTMEMBER.DIMENSION.NAME
MEMBER MEASURES.HIERARCHYNAME AS
[Date].[Calendar].CURRENTMEMBER.HIERARCHY.NAME
MEMBER MEASURES.LEVELNAME AS [Date].[Calendar].LEVEL.NAME
SELECT
{MEASURES.DIMENSIONNAME, MEASURES.HIERARCHYNAME, MEASURES.LEVELNAME}
ON Columns,
[Date].[Calendar].MEMBERS
ON Rows
FROM [Adventure Works]
```

See Also

[Dimension \(MDX\)](#)

[Functions \(MDX Syntax\)](#)

[Hierarchy \(MDX\)](#)

[Level \(MDX\)](#)

Using Stored Procedures

You can extend the functionality of Analysis Services and Multidimensional Expressions (MDX) by writing .NET stored procedures or user-defined functions. For more information, see [ADOMD.NET Server Programming](#)

When you reference or call a stored procedure, you specify the function name followed by parentheses. Within the parentheses, you can specify expressions called arguments that provide the data to be passed into the parameters. When you call a function, you must supply argument values for all of the parameters, and you must specify the argument values in the same sequence in which the parameters are defined in the user-defined function.

The following example query assumes that you have an assembly named SampleAssembly registered on your Analysis Services Server:

```
SELECT SampleAssembly.RandomSample([Geography].[State-Province].Members, 5)
on ROWS,
[Date].[Calendar].[Calendar Year] on COLUMNS
FROM [Adventure Works]
WHERE [Measures].[Reseller Freight Cost]
```

Note

Stored procedure is the terminology used in Microsoft SQL Server Analysis Services for these types of functions. Earlier versions of Analysis Services called these types of functions as *user-defined functions*.

Types of stored procedures

Analysis Services supports both COM and CLR assemblies. CLR assemblies are recommended because of the enhanced security available to CLR assemblies. If Microsoft Office Excel is installed on the server, Excel functions are also available.

Note

Microsoft Visual Basic for Applications (VBA) COM Assemblies are registered automatically.

See Also

[Functions \(MDX Syntax\)](#)

Comments (MDX Syntax)

Comments are non-executing text strings in program code. (Comments are also known as remarks). You can use comments to document code, or temporarily disable parts of Multidimensional Expressions (MDX) statements and scripts being diagnosed. By using comments to document code, you can make future program code maintenance easier. You

frequently use comments to record the program name, the author name, and the dates of major code changes. You can also use comments to describe complex calculations or explain a programming method.

Comments in MDX follow these guidelines:

- All alphanumeric characters or symbols can be used within the comment. Microsoft SQL Server Analysis Services ignores all characters within a comment.
- There is no maximum length for a comment within a statement or script. A comment can be made up of one or more lines.

MDX supports three types of commenting characters:

// (double forward slashes)

These comment characters can be used on the same line as code to be run or on a line by themselves. Everything from the double forward slashes to the end of the line is part of the comment. For a multiple-line comment, the double forward slashes must appear at the starting of each comment line. For more information, see [MDX Syntax Elements \(MDX\)](#).

-- (double hyphens)

These comment characters can be used on the same line as code to be run or on a line by themselves. Everything from the double hyphens to the end of the line is part of the comment. For a multiple-line comment, the double hyphens must appear at the starting of each comment line. For more information, see [-- \(Comment\) \(MDX\)](#).

/* ... */ (forward slash-asterisk character pairs)

These comment characters can be used on the same line as code to be run, on lines by themselves, or even within executable code. Everything from the open comment pair (/*) to the close comment pair (*/) is considered part of the comment. For a multiple-line comment, the open-comment character pair (/*) must start the comment, and the close-comment character pair (*/) must end the comment. No other comment characters can appear on any lines of the comment. For more information, see [/* ... */ \(Comment\)](#).

Example

The following query shows examples of all three types of comment:

```
//An example of a comment using the double-forward slash
--An example of a comment using the double-hyphen
/*An example of a
multi-line
comment*/
SELECT
{[Measures].[Internet Sales Amount]}
ON Columns,
```

```
[Date].[Calendar].MEMBERS
ON Rows
FROM [Adventure Works]
```

See Also

[MDX Syntax Elements \(MDX\)](#)

Reserved Keywords (MDX Syntax)

Microsoft SQL Server Analysis Services reserves certain keywords for its exclusive use. For a list of reserved keywords, see [MDX Syntax Elements \(MDX\)](#).

Reserved keywords follow these guidelines:

- You cannot include reserved keywords in a Multidimensional Expressions (MDX) statement in any location except that defined by Analysis Services.
- No objects in the database should be specific a name that matches a reserved keyword. If such a name exists, the object must always be referred to using delimited identifiers. Although this method does allow for object names to be reserved words, using keywords to name objects should be avoided.
- Use a naming convention that avoids using reserved keywords. Consonants or vowels can be removed if an object name must look like a reserved keyword.

See Also

[MDX Syntax Elements \(MDX\)](#)

MDX Language Reference

The reference documentation for Multidimensional Expressions (MDX) is grouped into sections, as described in the following table.

In This Section

Topic	Description
Multidimensional Expressions (MDX) Reference	Briefly describes the syntax conventions used in the MDX Language Reference.
MDX Statement Reference	Describes the scripting, data definition, and data manipulation statements available in

Topic	Description
	the MDX language.
MDX Operator Reference	Lists the operators available in the MDX language.
MDX Function Reference	Describes the functions available in the MDX language.
MDX Reserved Words	Provides a list of words reserved for use by the MDX language.

See Also

[Multidimensional Expressions \(MDX\) Reference](#)

MDX Syntax Conventions

The diagrams for Multidimensional Expressions (MDX) syntax in the MDX Language Reference use these conventions.

Convention	Usage
<i>italic</i>	Indicates user-supplied arguments of MDX syntax.
(vertical bar)	Separates syntax items within brackets or braces. You can select only one of the items.
[] (brackets)	Indicates optional syntax items. Do not type the brackets.
[,] ...n	Indicates that the preceding item can be repeated any number of times. The items are sometimes separated by commas.
<label> ::=	Indicates the name for a block of syntax. This convention is used to group and label portions of lengthy syntax or a unit of syntax that can be used in more than one place within a statement. Each location in which the block of syntax can be used is indicated with the label enclosed in angle

Convention	Usage
	brackets: <label>.

See Also

[MDX Language Reference \(MDX\)](#)

MDX Statement Reference

Multidimensional Expressions (MDX) statements are separated into three groups, as described in the following table.

In this Section

Topic	Description
MDX Language Reference (MDX)	Contains information about MDX scripting statements that manage query context, scope, and the control of flow within MDX scripts.
MDX Data Definition Statements	Contains information about MDX data definition statements that create, drop, and manipulate multidimensional objects.
MDX Data Manipulation Statements	Contains information about MDX data manipulation statements that retrieve and manipulate data from multidimensional objects.

See Also

[MDX Language Reference](#)

MDX Scripting Statements

In Multidimensional Expressions (MDX), the following statements manage context, scope, and control of flow within MDX scripts.

In this Section

Topic	Description
MDX Scripting Fundamentals (MDX)	Calculates a subcube, optionally determining the solve order of dimensions included within the subcube.
CASE Statement (MDX)	Lets you conditionally return specific values from multiple comparisons.
Existing Statement (MDX)	Forces a specified set to be evaluated within the current context.
FREEZE Statement	Locks the cell values of a specified subcube to their current values.
IF Statement (MDX)	Executes a statement if the condition is true.
SCOPE Statement	Limits the scope of specified MDX statements to a specified subcube.

See Also

- [MDX Statement Reference \(MDX\)](#)
- [MDX Data Definition Statements](#)
- [MDX Data Manipulation Statements](#)
- [MDX Scripting Fundamentals \(MDX\)](#)

CALCULATE Statement

Populates each cell in a cube with an aggregate value.

Syntax

CALCULATE

Arguments

None

Remarks

The CALCULATE statement is automatically included as the first statement in a cube's MDX script when you create a cube by using SQL Server Data Tools (SSDT). The CALCULATE statement tells each cell in the cube to aggregate from lower granularity cells. After a cell is aggregated, if you subsequently populate lower granularity cells by using expressions, it impacts the aggregated value of higher granularity cells. You almost always want this aggregation to happen, but you can remove it or cause other statements to execute before this statement.

The CALCULATE statement cannot be included in a nested subcube within the MDX script. A nested subcube is defined by using the SCOPE statement. For more information about the SCOPE statement, see [Defining Assignments and Other Script Commands](#).



Note

Calculated members are not aggregated.

See Also

[MDX Scripting Statements \(MDX\)](#)

[MDX Scripting Fundamentals \(MDX\)](#)

[Creating and Editing MDX Scripts](#)

FREEZE Statement

Locks the cell values of a specified subcube to their current values. When the cell values are locked, changes to other cells have no effect on the cells that are locked.

Syntax

```
FREEZE Subcube_Expression
```

Arguments

Subcube_Expression

A valid Multidimensional Expressions (MDX) expression that returns a subcube.

Remarks

The **FREEZE** statement locks the values of cells in a specified subcube, preventing subsequent statements in an MDX script from changing their values in subsequent calculation passes.

In the following example, A and B represent subcubes in an MDX calculation script:

```
B = 2;
```

```
A = B;
```

```
B = 3
```

At this point, both A and B are equal to 3.

We now insert the **Freeze** function to lock the cells in the A subcube:

```
B = 2;
```

```
A = B;
```

```
FREEZE (A) ;
```

```
B = 3
```

A is now equal to 2, and B is equal to 3.

See Also

[MDX Scripting Statements \(MDX\)](#)

IF Statement

Executes a statement if the condition is true.

Syntax

```
IF expression THEN assignment END IF
```

Arguments

expression

A Multidimensional Expressions (MDX) expression that evaluates to a Boolean that returns true or false.

assignment

An MDX expression that assigns a value to either a subcube or a calculated property.

Remarks

Use the IF statement for control flow, which is unlike the [MDX Function Reference \(MDX\)](#) function and the [CASE Statement \(MDX\)](#) that can only be used to return values or objects.

Examples

In the following example, the scope is restricted to the Country level of the Customers Geography hierarchy in the Customers dimension. If the current measure is Internet Sales Amount, then the Internet Sales Amount is set to 10:

```
SCOPE ([Customer].[Customer Geography].[Country].MEMBERS) ;
```

```
    IF Measures.CurrentMember IS [Measures].[Internet Sales Amount] THEN this =  
10 END IF;
```

```
END SCOPE;
```

See Also

[MDX Function Reference \(MDX\)](#)

SCOPE Statement

Limits the scope of specified Multidimensional Expressions (MDX) statements to a specified subcube.

Syntax

```
SCOPE(Subcube_Expression)
  [ MDX_Statement ]
END SCOPE
```

```
Subcube_Expression ::= (Auxiliary_Subcube [, Auxiliary_Subcube,...n])
```

```
Auxiliary_Subcube ::=
  Limited_Set
  | Root([dimension_name])
  | Leaves([dimension_name])
```

```
Limited_Set ::=
  single_tuple
  | member
  | Common_Grain_Members
  | hierarchy.members
  | level.members
  | {}
  | Descendants
    (
      Member
    , [level
      [
        , SELF
        | AFTER
          | BEFORE
          | SELF_AND_AFTER
```

```

        | SELF_AND_BEFORE
        | SELF_BEFORE_AFTER
        | LEAVES
    ]
)
[* <limited set>]

```

Arguments

Subcube_Expression

A valid MDX subcube expression.

MDX_Statement

A valid MDX statement.

Common_Grain_Members

A valid MDX statement that evaluates to members that have the same grain.

single_tuple

A single tuple.

Remarks

The SCOPE statement determines the subcube that will be affected by the running of one or more MDX statements. Unless an MDX statement is framed within a SCOPE statement, the implicit scope of an MDX statement is the entire cube.

Note

Hidden members are exposed in SCOPE statements.

SCOPE statements will create subcubes that expose "holes" regardless of the **MDX Compatibility** setting. For example, the statement, `Scope(Customer.State.members)`, can include the states in countries or regions that do not contain states, but for which otherwise invisible placeholder members were inserted.

Calculated members and named sets created within a SCOPE statement are unaffected by the SCOPE statement.

Example

The following example, from the MDX calculation script in the Adventure Works sample solution, defines the current scope as fiscal quarter in fiscal year 2005 and the sales amount quota measure, and then assigns a value to the cells in the current scope by using the **ParallelPeriod** function. The example then modifies the scope using another SCOPE statement, and then performs another assignment using the `This (MDX)` function.

```

Scope
(

```

```

[Date].[Fiscal Year].&[2005],
[Date].[Fiscal].[Fiscal Quarter].Members,
[Measures].[Sales Amount Quota]
) ;

```

```

This = ParallelPeriod
    (
        [Date].[Fiscal].[Fiscal Year], 1,
        [Date].[Fiscal].CurrentMember
    ) * 1.35 ;

```

```

/*-- Allocate equally to months in FY 2002 -----*/

```

```

Scope
(
    [Date].[Fiscal Year].&[2002],
    [Date].[Fiscal].[Month].Members
) ;

```

```

This = [Date].[Fiscal].CurrentMember.Parent / 3 ;

```

```

End Scope ;

```

```

End Scope ;

```

See Also

[MDX Scripting Statements \(MDX\)](#)

MDX Data Definition Statements

In Multidimensional Expressions (MDX), data definition statements create, drop, and manipulate multidimensional objects. The following table lists the available data definition statements.

In this Section

Topic	Description
MDX Scripting Statements (MDX)	Alters the structure of a specified cube.
CREATE ACTION Statement	Creates an action that can be associated with a cube, dimension, hierarchy, or subordinate object.
CREATE CELL CALCULATION Statement	Creates a calculation that evaluates an MDX expression over a specified set of tuples within a cube.
CREATE GLOBAL CUBE Statement (MDX)	Creates and populates a locally persisted cube, based on a subcube from a cube on the server. A connection to the server is not required to connect to the locally persisted cube.
CREATE MEMBER Statement	Creates a calculated member.
CREATE SESSION CUBE Statement (MDX)	Creates and populates a cube available to all queries in the same the session, based on cubes on the server.
CREATE SET Statement	Creates a named set for a specified cube.
CREATE SUBCUBE Statement	Redefines the cube space of a specified cube or subcube to a specified subcube.
DROP ACTION Statement	Deletes a specified action from a specified cube.
DROP CELL CALCULATION Statement	Removes the specified cell calculation.
DROP MEMBER Statement	Removes a calculated member.
DROP SET Statement	Removes a named set.
DROP SUBCUBE Statement	Drops a specified subcube, reverting to the previously defined cube or subcube definition with the specified name.
REFRESH CUBE Statement	Refreshes the client cache for a cube.

See Also

[MDX Statement Reference \(MDX\)](#)

[MDX Data Manipulation Statements](#)

[MDX Scripting Statements](#)

ALTER CUBE Statement

Alters the structure of a specified cube.

Syntax

ALTER CUBE

```
  Cube_Name | CURRENTCUBE  
  <alter clause>  
  [ < alter clause> ...n]
```

< alter clause> ::=

```
  <create dimension member clause>  
  | <remove dimension member clause>  
  | <move dimension member clause>  
  | <update clause>  
  | <create cell calculation clause>
```

<create dimension member clause> ::=

```
CREATE DIMENSION MEMBER [ParentName.]MemberName  
  , [[KEY = Key_Value]  
  | [Property_Name = Property_Value[, ...n]]
```

<dropping clause>::=

DROP

```
  DIMENSION MEMBER Member_Name  
    Member_Name ...n ]  
  [WITH DESCENDANTS]  
  | [ SESSION ] [ CALCULATED ] MEMBER Member_Name  
    [ ,Member_Name,...n ]  
  | SET Set_Name  
    [ ,Set_Name,...n ]  
  | [ SESSION ] CELL CALCULATION CellCalc_Name  
    [ ,CellCalc_Name,...n ]  
  | ACTION Action_Name
```

```

<move dimension member clause> ::=
MOVE DIMENSION MEMBER MemberName
    [, SKIPPED_LEVELS = Unsigned_Integer]
    [WITH DESCENDANTS]
    UNDER ParentName

```

```

<update clause> ::=
UPDATE
    CUSTOM ROLLUP FOR MEMBER MemberName
        [, MemberName, ...n] AS MDX_Expression
| DIMENSION Dimension_Name | Hierarchy_Name
    , DEFAULT_MEMBER = MDX_Expression
| DIMENSION MEMBER MemberName AS
[MDX_Expression]
    [Property_Name = Property_Value [, ...n]]

```

```

<create cell calculation clause> ::=
CELL CALCULATION Calculation_Name
    FOR Set_Expression AS MDX_Expression
        [ [ CONDITION = 'Logical_Expression' ]
| [ DISABLED = { TRUE | FALSE } ]
| [ DESCRIPTION = String ]
| [ CALCULATION_PASS_NUMBER = Integer ]
| [ CALCULATION_PASS_DEPTH = Integer ]
| [ SOLVE_ORDER = Integer ]
| [ Calculation_Name = Scalar_Expression ], ...n]

```

Creating a Dimension Member

A new row is added to the underlying dimension table.

Arguments

ParentName

A valid string expression that provides the name of the parent of the new dimension member, unless the dimension member is being created at the root.

MemberName

A valid string expression that provides a member name.

Key_Value

A valid scalar expression that defines the new dimension member's key value.

Property_Name

A valid Multidimensional Expressions (MDX) identifier that represents a member property.

Property_Value

A valid Multidimensional Expressions (MDX) scalar expression that defines the calculated member property's value.

Dropping a Dimension Member

Dropping a dimension member from a write-enabled dimension deletes the member and its corresponding row from the underlying dimension table.

Arguments**Cube_Name**

A valid string expression providing a cube name.

Member_Name

A valid string expression providing a member name or member key.

Remarks

If the WITH DESCENDANTS clause is not used, children of a dropped member become children of the dropped member's parent. If the WITH DESCENDANTS clause is used, all descendants and their rows in the dimension table are also dropped.

**Note**

For information about dropping calculated members, named sets, actions, and cell calculations, see [DROP MEMBER Statement \(MDX\)](#), [DROP SET Statement \(MDX\)](#), [DROP ACTION Statement \(MDX\)](#), and [DROP CELL CALCULATION Statement \(MDX\)](#).

Updating the Default Dimension Member

This clause updates the default member of a cube and is used in the MDX calculation script to define a default member. The default member can be specified for the database dimension, a cube dimension, or for a user's login. The default member can also be changed during a session.

Arguments**Dimension_Name**

A valid string that provides the name of a dimension.

MDX_Expression

A valid MDX expression that returns a single member.

Remarks

The specified MDX expression can be static or dynamic.

Moving a Dimension Member

A row is modified in the underlying dimension table.

Arguments**ParentName**

A valid string expression that provides the name of the new parent for the dimension member being moved.

MemberName

A valid string expression that provides a member name.

Unsigned_Integer

A valid number specifying the number of levels to skip.

If the WITH DESCENDANTS clause is specified, the entire tree is moved. If the WITH DESCENDANTS clause is not specified, the children of a moved parent become the children of the moved member's parent. The effect of a move is simply to update the values for the parent key column in the underlying dimension table.

Updating a Dimension Member

The UPDATE DIMENSION MEMBER clause allows you to modify properties of a member as well as the custom member formula associated with a member.

Arguments**MemberName**

A valid string expression that provides a member name.

MDX_Expression

A valid MDX expression that returns a single member.

Property_Value

A valid MDX scalar expression that defines the calculated member property's value.

Creating a Cell Calculation

For more information about creating a cell calculation using the ALTER CUBE statement, see [CREATE CELL CALCULATION Statement \(MDX\)](#).

See Also

[MDX Data Definition Statements \(MDX\)](#)

CREATE ACTION Statement

Creates an action that can be associated with a cube, dimension, hierarchy, or subordinate object.

Syntax

```
CREATE ACTION CURRENTCUBE | Cube_Name
    .Action_Name <action body>
<action body> ::=
FOR
    CUBE
    | Hierarchy_Name [MEMBERS]
    | Level_Name [MEMBERS]
    | CELLS
    | SET }
AS 'MDX_Expression'
    [, TYPE = '
        { URL
        | HTML
        | STATEMENT
        | DATASET
        | ROWSET
        | COMMANDLINE
        | PROPRIETARY }
    '
    [, INVOCATION = 'INTERACTIVE | ON_OPEN | BATCH ' ]
    [, APPLICATION = String_Expression ]
    [, DESCRIPTION = String_Expression ]
    [, CAPTION = 'MDX_Expression' ]
```

Arguments

Cube_Name

A valid string that provides a cube name.

Action_Name

A valid string that provides the name of the action being created.

Hierarchy_Name

A valid string that provides a hierarchy name.

Level_Name

A valid string that provides a level name.

Member_Name

A valid string that provides a member name or member key.

MDX_Expression

A valid MDX expression.

String_Expression

A valid string expression.

Remarks


It is possible for client applications to create and run actions that are unsafe; it is also possible for client applications to use unsafe functions. To avoid these situations, use the **Safety Options** property. For more information, see Safety Options Property.

 **Note**

This statement is included for backwards compatibility. Actions new to SQL Server Analysis Services, such as Drillthrough or Report actions, are not supported.

Action Types

The following table describes the different types of actions available in Microsoft SQL Server Analysis Services.

Action type	Description
URL	<p>The returned action string is a URL that should be opened using an Internet browser.</p> <p> Note If this action does not start with <code>http://</code> or <code>https://</code>, the action will be unavailable to the browser unless SafetyOptions is set to DBPROPVAL_MSMD_SAFETY_OPTIONS_ALLOW_ALL.</p>
HTML	<p>The returned action string is an HTML script. The string should be saved to a file and the file should be rendered using an Internet browser. In this case, a whole script may be run as part</p>

Action type	Description
	of the generated HTML.
STATEMENT	The returned action string is a statement that needs to be executed by setting the ICommand::SetText method of a command object to the string and calling the ICommand::Execute method. If the command does not succeed, an error is returned.
DATASET	The returned action string is an MDX statement that needs to be run by setting the ICommand::SetText method of a command object to the string and calling the ICommand::Execute method. The requested interface ID (IID) should be IDataset . The command succeeds if a data set has been created. The client application should allow the user to browse the returned data set.
ROWSET	Similar to DATASET , but instead of requesting an IID of IDataset , the client application should ask for an IID of IRowset . The command succeeds if a rowset has been created. The client application should allow the user to browse the returned rowset.
COMMANDLINE	The client application should execute the action string. The string is a command line.
PROPRIETARY	A client application should not display, nor execute the action unless the application has a custom, nongeneric knowledge of the specific action. Proprietary actions are not returned to the client application unless the client application explicitly asks for these by setting the appropriate restriction on the APPLICATION_NAME .

Invocation Types

The following table describes the different types of invocations available in Analysis Services. The invocation type is used only by the client application to help determine when to invoke the action. The invocation type does not actually determine the invocation behavior of the action.

Invocation type	Description
INTERACTIVE	The action should be invoked by the client application through user interaction.

Invocation type	Description
ON_OPEN	The action should be invoked by the client application when the target object is opened. This invocation type is not currently implemented.
BATCH	The action should be invoked by the client application when the target object is involved in a batch operation, as determined by the client application. This invocation type is not currently implemented.

Scope

Each action is defined for a specific cube and has a unique name in that cube. An action can have one of the scopes listed in the following table.

Cube scope

For actions independent of specific dimensions, members, or cells; for example: "Launch terminal emulation for AS/400 production system".

Dimension scope

The action applies to a specific dimension. These actions are not dependent on specific selection of levels or members.

Level scope

The action applies to a specific dimension level. These actions are not dependent on specific selection of a member in that dimension.

Member scope

The action applies to specific level members.

Cell scope

The action applies to specific cells only.

Set scope

The action applies to a set only. The name, **ActionParameterSet**, is reserved for use by the application inside the expression of the action.

See Also

[MDX Data Definition Statements \(MDX\)](#)

CREATE CELL CALCULATION Statement

Creates a calculation that evaluates a Multidimensional Expressions (MDX) expression over a specified set of tuples within a cube.

Syntax

```
[WITH <CELL CALCULATION clause> Calculation_Name  
[,WITH <CELL CALCULATION clause> Calculation_Name...n]  
CREATE CELL CALCULATION CURRENTCUBE | Cube_Name.Calculation_Name
```

```
<CELL CALCULATION clause> ::=  
FOR Set_Expression AS 'MDX_Expression'  
  [ [ CONDITION = 'Logical_Expression' ]  
  | [ DISABLED = { TRUE | FALSE } ]  
  | [ DESCRIPTION =String ]  
  | [ CALCULATION_PASS_NUMBER = Integer]  
  | [ CALCULATION_PASS_DEPTH = Integer]  
  | [ SOLVE_ORDER = Integer]  
  | [ Calculation_Name= Scalar_Expression ], ...n]
```

Arguments

Cube_Name

A valid string that provides a cube name.

Calculation_Name

A valid string that provides a cell calculation name.

Set_Expression

A valid MDX expression that returns a set.

String

A valid string value.

MDX_Expression

A valid MDX expression.

Logical_Expression

A valid MDX logical expression.

Integer

A valid integer value.

Calculation_Name

A valid string that provides the name of a cell calculation property.

Scalar_Expression

A valid MDX scalar expression.

Remarks

By using calculated cells, the client application can specify a rollup value for a particular set of cells, instead of for an entire set of cells as in the case of a custom rollup formula or a calculated member. For example, it is possible to specify that any cell in the set defined by `{ [Canada], [Time].[2000] }` can contain a value that is defined by a formula. Any other cells that are not contained within this set are computed normally.

Note

The Backus-Naur Form (BNF) of `{*(<comment> | <whitespace> | <newline>)}` will be parsed as `{*}` for backwards compatibility.

See Also

[MDX Data Definition Statements \(MDX\)](#)

[Creating Query-Scoped Cell Calculations \(MDX\)](#)

[Building Cell Calculations in MDX \(MDX\)](#)

[Using Cell Properties \(MDX\)](#)

[FORMAT_STRING Contents \(MDX\)](#)

[FORE_COLOR and BACK_COLOR Contents \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

CREATE GLOBAL CUBE Statement

Creates and populates a locally persisted cube, based on a subcube from a cube on the server. A connection to the server is not required to connect to the locally persisted cube. For more information about local cubes, see [Local Cubes](#).

Syntax

```
CREATE GLOBAL CUBE local_cube_name STORAGE 'Cube_Location'  
FROM source_cube_name (<param list>)
```

<param list> ::= <param> ,<param list> | <param>

<param> ::= <dims list> | <measures list>

<measures list>::= <measure>[, <measures list>]

<dims list>::= <dim def> [, <dims list>]

<measure>::= MEASURE *source_cube_name.measure_name* [<visibility qualifier>] [AS *measure_name*]

<dim def>::= <source dim def> | <derived dim def>

<source dim def>::= DIMENSION *source_cube_name.dimension_name* [<dim flags>]
[<visibility qualifier>] [AS *dimension_name*] [FROM <dim from clause>] [<dim content def>]

<dim flags>::= NOT_RELATED_TO_FACTS

<dim from clause>::= < dim DM from clause> | <reg dim from clause>

<dim DM from clause>::= *dm_model_name*> COLUMN *column_name*

<dim reg from clause>::= *dimension_name*

<dim content def>::= (<level list> [, <grouping list>] [, <member slice list>] [, <default member>])

<level list>::= <level def> [, <level list>]

<level def>::= LEVEL *level_name* [<level type>] [AS *level_name*] [<level content def>]

<level content def>::= (<property list>) | NO_PROPERTIES

<level type>::= GROUPING

<property list>::= <property def> [, <property list>]

<property def>::= PROPERTY *property_name*

<grouping list>::= <grouping entity> [,<grouping list>]

<grouping entity>::= GROUP **group_level_name.group_name** (<mixed list>)

<grp mixed list>::= <grp mixed element> [,<grp mixed list>]

<grp mixed element>::= <grouping entity> | <member def>

<member slice list>::= <member list>

<member list>::= <member def> [, <member list>]

<member def>::= MEMBER **member_name**

<default member>::= DEFAULT_MEMBER AS **MDX_expression**

<visibility qualifier>::= HIDDEN

Syntax Elements

local_cube_name

The name of the local cube.

'Cube_Location'

The name and path for the locally persisted cube.

source_cube_name

The name of the cube on which the local cube is based.

source_cube_name.measure_name

The fully qualified name of the source measure being included in the local cube. Calculated members of the Measures dimension are not permitted.

measure_name

The name of the measure in the local cube.

source_cube_name.dimension_name

The fully qualified name of the source dimension being included in the local cube.

dimension_name

The name of the dimension in the local cube.

FROM <dim from clause>

Valid specification for derived dimension definition only.

NOT_RELATED_TO_FACTS

Valid specification for derived dimension definition only.

<level type>

Valid specification for derived dimension definition only.

Remarks

A local cube is defined in terms of the measures and definitions that define it. There are two types of dimensions.

- Source dimensions - These are dimensions that were part of one of more source cubes
- Derived dimensions - These are dimensions that provide new analysis capabilities. A derived dimension can be a regular dimension defined based on a source dimension that is either sliced vertically or horizontally, or contains custom grouping of dimension members. A derived dimension can also be a data mining dimension based on a data mining model.



Note

The Dimension keyword can refer to either dimensions or hierarchies.

In a local cube, you can perform the following tasks:

- Eliminate dimensions that exist in the source cube
- Add or eliminate hierarchies from a dimension
- Eliminate measure groups or specific measures

The CREATE GLOBAL CUBE statement follows these rules:

- The CREATE GLOBAL CUBE statement automatically copies all commands, such as calculated measures or actions, to the local cube. If a command contains a Multidimensional Expressions (MDX) expression that references the parent cube explicitly, the local cube cannot run that command. To prevent this problem, use the **CURRENTCUBE** keyword when defining MDX expressions for commands. The **CURRENTCUBE** keyword uses the current cube context when referencing a cube within an MDX expression.
- A global cube that is created from an existing global cube in a local cube file cannot be saved in the same local cube file. For example, you create a global cube named SalesLocal1 and save this cube to the C:\SalesLocal.cub file. You then connect to the C:\SalesLocal.cub file and create a second global cube named SalesLocal2. If you now try to save the SalesLocal2 global cube to the C:\SalesLocal.cub file, you will receive an error. However, you can save the SalesLocal2 global cube to a different local cube file.
- Global cubes do not support distinct count measures. Because cubes that include distinct count measures are nonadditive, the CREATE GLOBAL CUBE statement cannot support the creation or use of distinct count measures.

- When adding a measure to a local cube, you must also include at least one dimension that is related to the measure being added.
- When adding a parent-child hierarchy to a local cube, levels and filters on a parent-child hierarchy are ignored and the entire parent-child hierarchy is included.
- Member properties are not supported in local cubes.
- You cannot create a local cube from a perspective.
- When you include a semi-additive measure to a local cube, the following rules apply:
 - You must include the Account dimension if the AggregateFunction property for the measure being added is ByAccount.
 - You must include the entire Time dimension if the AggregateFunction property measure being added is FirstChild, LastChild, FirstNonEmpty, LastNonEmpty, or AverageOfChildren.
- Data mining dimensions cannot be added to a local cube.
- Reference dimensions are materialized and added as regular dimensions.
- When you include a many-to-many dimension, the following rules apply:
 - You must add the entire many-to-many dimension.
 - You must add the intermediary measure group.
 - You must add the entirety of all dimensions common to the two measure groups involved in the many-to-many relationship.

The following example demonstrates creating a local, persisted version of the Adventure Works cube that contains only the Reseller Sales Amount measure, the Reseller dimension, and the Date dimension.

```
CREATE GLOBAL CUBE [LocalReseller]
  Storage 'C:\LocalAWReseller1.cub'
  FROM [Adventure Works]
  (
    MEASURE [Adventure Works].[Reseller Sales Amount],
    DIMENSION [Adventure Works].[Reseller],
    DIMENSION [Adventure Works].[Date]
  )
```

The following example demonstrates slicing when you create a local cube. The global cube that is created is based on the Adventure Works cube sliced vertically by the 2005 member of the Fiscal Year level, and horizontally by the Fiscal Year and Month levels.

```
CREATE GLOBAL CUBE [LocalReseller]
  Storage 'C:\LocalAWReseller2.cub'
  FROM [Adventure Works]
```

```

(
    MEASURE [Adventure Works].[Reseller Sales Amount],
    DIMENSION [Adventure Works].[Reseller],
    DIMENSION [Adventure Works].[Date]
    (
        LEVEL [Fiscal Year],
        LEVEL [Month],
        MEMBER [Date].[Fiscal].[Fiscal Year].&[2005]
    )
)

```

See Also

[CREATE SESSION CUBE Statement \(MDX\)](#)

[CREATE SESSION CUBE Statement \(MDX\)](#)

CREATE KPI Statement

Creates a key performance indicator (KPI). A KPI is a collection of calculations that are associated with a measure group in a cube and are used to evaluate business or scenario success.

Syntax

```

CREATE KPI CURRENTCUBE | <Cube Name>.KPI_Name AS KPI_Value
    [,Property_Name = Property_Value, ...n]

```

Arguments

KPI_Name

A valid string that provides a KPI name.

KPI_Value

A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Property_Name

A valid string that provides the name of a KPI property.

Property_Value

A valid scalar expression that defines the KPI property's value.

Remarks

Specifying a cube other than the cube that is currently connected causes an error. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

KPI Properties

The following table lists all KPI properties. None of these properties have a default value. Therefore, until a specific value has been assigned to a KPI property, queries against that properties will return a null value.

Property identifier	Meaning
GOAL	A valid MDX expression that returns a numeric value.
STATUS	A valid MDX expression that returns a numeric value.
STATUS_GRAPHIC	A string that defines a set of graphic images that will be used by the client application.
TREND	A valid MDX expression that returns a numeric value.
TREND_GRAPHIC	A string that defines a set of graphic images that will be used by the client application.
WEIGHT	A valid MDX expression that returns a numeric value.
CURRENT_TIME_MEMBER	A valid MDX expression that returns a member in the time dimension. CURRENT_TIME_MEMBER sets the reference point for all relative time functions
PARENT_KPI	A string that specifies the name of the parent KPI.
CAPTION	A string that the client application uses as the caption for the KPI.
DISPLAY_FOLDER	A string that specifies the path of the display folder where the KPI is to be shown by the client application. The folder level separator is defined by the client

Property identifier	Meaning
	application. For the tools and clients supplied by Analysis Services, the backslash (\) is the level separator. To provide multiple display folders for a defined member, use a semicolon (;) to separate the folders
ASSOCIATED_MEASURE_GROUP	A string that specifies the name of the measure group to which all MDX calculations should be referred.

The values for the GOAL, STATUS, and TREND properties are MDX expressions that should evaluate between -1 to 1. However, it is the client application that defines the actual range of values for these properties. When you use the tools and clients supplied by Analysis Services to browse KPIs, values less than -1 are treated as -1, and values larger than 1 are treated as 1.

Both STATUS_GRAPHIC and TREND_GRAPHIC are string values that the client application uses to identify the correct set of images to display. These strings also define the behavior of the display function. This behavior includes the number of states to display (typically, this is an odd number) and which images to use for each of those states.

KPI Graphics in SQL Server Data Tools

In SQL Server Data Tools (SSDT), KPI graphics can have either three or five states. The following table defines the values for each of those states.

Number of states for KPI graphic	Value of those states
3	Bad = -1 to -0.5 OK = -0.4999 to 0.4999 Good = 0.50 to 1
5	Bad = -1 to -0.75 Risk = -0.7499 to -0.25 OK = -0.2499 to 0.2499 Rising = 0.25 to 0.7499 Good = 0.75 to 1

Note

For some graphics, such as the reversed gauge or reversed status arrow, the range is inverted. That is, -1 is good, and 1 is bad.

In SQL Server Data Tools (SSDT), the name of the KPI graphic determines whether the graphic has three or five states. The following table lists the usage, name, and number of states that SQL Server Data Tools (SSDT) associates with its KPI graphics.

Use of graphic	Name of KPI graphic	Number of states
Status	Shapes	3
Status	Traffic Light	3
Status	Road Signs	3
Status	Gauge	3
Status	Reversed Gauge	5
Status	Thermometer	3
Status	Cylinder	3
Status	Faces	3
Status	Variance arrow	3
Trend	Standard Arrow	3
Trend	Status Arrow	3
Trend	Reversed status arrow	5
Trend	Faces	3

See Also

[DROP KPI Statement \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

CREATE MEASURE statement

Creates a measure in a Tabular Model.

Syntax

```
CREATE MEASURE Table_Name[Measure_Name] = DAX_Expression
[; CREATE MEASURE ...n]
```

Arguments

Table_Name

A valid string literal that provides the name of the table where the measure will be created.

Measure_Name

A valid string literal that provides a measure name.

DAX_Expression

A valid DAX expression that returns a single scalar value.

Remarks

The Measure_Name must be enclosed in square parenthesis.

The CREATE MEASURE statement can only be used inside of a MDX script definition; see [MdxScript Element \(ASSL\)](#).

You can also define a calculated member for use by a single query. To define a calculated member that is limited to a single query, you use the WITH clause in the SELECT statement. For more information, see [Building Measures in MDX](#).

See Also

[MDX Data Definition Statements \(MDX\)](#)

CREATE MEMBER Statement

Creates a calculated member.

Syntax

```
CREATE [ SESSION ] [HIDDEN] [ CALCULATED ] MEMBER CURRENTCUBE |  
Cube_Name.Member_Name  
    AS MDX_Expression  
    [,Property_Name = Property_Value, ...n]  
.....[,SCOPE_ISOLATION = CUBE]
```

Arguments

Cube_Name

A valid string expression that provides the name of the cube where the member will be created.

Member_Name

A valid string expression that provides a member name. Specify a fully qualified name to create a member within a dimension other than the Measures dimension. If you do not

provide a fully qualified member name, the member will be created in the Measures dimension.

MDX_Expression

A valid Multidimensional Expressions (MDX) expression.

Property_Name

A valid string that provides the name of a calculated member property.

Property_Value

A valid scalar expression that defines the calculated member property's value.

Remarks

The CREATE MEMBER statement defines calculated members that are available throughout the session, and therefore, can be used in multiple queries during the session. For more information, see [Creating Session-Scoped Calculated Members \(MDX\)](#).

You can also define a calculated member for use by a single query. To define a calculated member that is limited to a single query, you use the WITH clause in the SELECT statement. For more information, see [Using WITH to Create Calculated Members \(MDX\)](#).

Property_Name can refer to either standard or optional calculated member properties. Standard member properties are listed later in this topic. Calculated members created with CREATE MEMBER without a **SESSION** value have session scope. Additionally, strings inside calculated member definitions are delimited with double quotation marks. This is different from the method defined by OLE DB, which specifies that strings should be delimited by single quotation marks.

Specifying a cube other than the cube that is currently connected causes an error. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

For more information about member properties that are defined by OLE DB, see the OLE DB documentation.

Scope

A calculated member can occur within one of the scopes listed in the following table.

Query scope

The visibility and lifetime of the calculated member is limited to the query. The calculated member is defined in an individual query. Query scope overrides session scope. For more information, see [Using WITH to Create Calculated Members \(MDX\)](#).

Session scope

The visibility and lifetime of the calculated member is limited to the session in which it is created. (The lifetime is less than the session duration if a DROP MEMBER statement is issued on the calculated member.) The CREATE MEMBER statement creates a calculated member with session scope.

Scope Isolation

When a cube Multidimensional Expressions (MDX) script contains calculated members, by default the calculated members are resolved before any session-scoped calculations are resolved and before any query-defined calculations are resolved.

Note

In certain scenarios, the **Aggregate** (MDX) function and the **VisualTotals** (MDX) function do not exhibit this behavior.

The behavior allows generic client applications to work with cubes that contain complex calculations, without having to take into account the specific implementation of the calculations. However, in certain scenarios, you might want to execute session or query-scoped calculated members before certain calculations in the cube, and neither the **Aggregate** function nor the **VisualTotals** function are applicable. To accomplish this, use the **SCOPE_ISOLATION** calculation property.

Example

The following script is an example of a scenario where the **SCOPE_ISOLATION** calculation property is required to produce the correct result.

Cube's MDX Script:

```
CREATE MEMBER CURRENTCUBE.Measures.ProfitRatio AS 'Measures.[Store Sales]/Measures.[Store Cost]', SOLVE_ORDER = 10
```

MDX Query:

```
WITH MEMBER [Customer].[Customers].[USA]. USAWithoutWA AS  
[Customer].[Customers].[Country].&[USA] - [Customer].[Customers].[State  
Province.&[WA], SOLVE_ORDER=5  
SELECT {USAWithoutWA} ON 0 FROM SALES  
WHERE ProfitRatio
```

The desired result of the previous query is the ratio of sales for USA without WA, to store cost for USA without WA. The previous query does not return the desired result; it returns the ratio of USA minus the ratio of WA, which is a meaningless result. To achieve the desired result, you can use the **SCOPE_ISOLATION** calculation property.

MDX Query using the **SCOPE_ISOLATION** calculation property:

```
WITH MEMBER [Customer].[Customers].[USA]. USAWithoutWA AS  
[Customer].[Customers].[Country].&[USA] - [Customer].[Customers].[State  
Province.&[WA], SOLVE_ORDER=5  
,SCOPE_ISOLATION=CUBE  
SELECT {USAWithoutWA} ON 0 FROM SALES  
WHERE ProfitRatio
```

Standard Properties

Each calculated member has a set of default properties. When a client application is connected to Microsoft Analysis Services, the default properties are either supported, or available to be supported, as the administrator chooses.

Additional member properties may be available, depending upon the cube definition. The following properties represent information relevant to the dimension level in the cube.

Property identifier	Meaning
SOLVE_ORDER	The order in which the calculated member will be solved in cases where a calculated member references one other calculated member (that is, where calculated members intersect each other).
FORMAT_STRING	A Microsoft Office style format string that the client application can use when displaying cell values.
VISIBLE	<p>A value that indicates whether the calculated member is visible in a schema rowset. Visible calculated members can be added to a set with the AddCalculatedMembers function. A nonzero value indicates that the calculated member is visible. The default value for this property is Visible.</p> <p>Calculated members that are not visible (where this value is set to zero) are generally used as intermediate steps in more complex calculated members. These calculated members can also be referred to by other types of members, such as measures.</p>
NON_EMPTY_BEHAVIOR	The measure or set that is used to determine the behavior of calculated members when resolving empty cells.
CAPTION	A string that the client application uses as the caption for the member.
DISPLAY_FOLDER	A string that identifies the path of the display folder that the client application

Property identifier	Meaning
	uses to show the member. The folder level separator is defined by the client application. For the tools and clients supplied by Analysis Services, the backslash (\) is the level separator. To provide multiple display folders for a defined member, use a semicolon (;) to separate the folders.
ASSOCIATED_MEASURE_GROUP	The name of the measure group to which this member is associated.

See Also

[DROP MEMBER Statement \(MDX\)](#)

[UPDATE MEMBER Statement \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

CREATE SESSION CUBE Statement

Creates and populates a session cube from an existing server cube. The session cube is only visible within the current session; it cannot be browsed or queried from any other session. The session cube is implicitly deleted when the session is closed.

Syntax

```
CREATE SESSION CUBE session_cube_name FROM <cube list> (<param list>)
```

```
<cube list> ::= source_cube_name [, <cube list>]
```

```
<param list> ::= <param> [, <param list> | <param>]
```

```
<param> ::= <dims list> | <measures list>
```

```
<measures list> ::= <measure> [, <measures list>]
```

```
<dims list> ::= <dim def> [, <dims list>]
```

<measure>::= MEASURE *source_cube_name.measure_name* [<visibility qualifier>] [AS *measure_name*]

<dim def>::= <source dim def> | <derived dim def>

<source dim def>::= DIMENSION *source_cube_name.dimension_name* [<dim flags>] [<visibility qualifier>] [AS *dimension_name*] [FROM <dim from clause>] [<dim content def>]

<dim flags>::= NOT_RELATED_TO_FACTS

<dim from clause>::= <reg dim from clause>

<dim reg from clause>::= *dimension_name*

<dim content def>::= (<level list> [, <grouping list>] [, <member slice list>] [, <default member>])

<level list>::= <level def> [, <level list>]

<level def>::= LEVEL *level_name* [<level type>] [AS *level_name*] [<level content def>]

<level content def>::= (<property list>) | NO_PROPERTIES

<level type>::= GROUPING

<property list>::= <property def> [, <property list>]

<property def>::= PROPERTY *property_name*

<grouping list>::= <grouping entity> [, <grouping list>]

<grouping entity>::= GROUP *group_level_name.group_name* (<mixed list>)

<grp mixed list>::= <grp mixed element> [, <grp mixed list>]

<grp mixed element>::= <grouping entity> | <member def>

<member slice list>::= <member list>

<member list>::= <member def> [, <member list>]

<member def>::= MEMBER *member_name*

<default member>::= DEFAULT_MEMBER AS *MDX_expression*

<visibility qualifier>::= HIDDEN

Syntax Elements

session_cube_name

The name of the session cube.

source_cube_name

The name of the cube on which the session cube is based.

source_cube_name.measure_name

The fully qualified name of the source measure being included in the session cube.
Calculated members of the Measures dimension are not permitted.

measure_name

The name of the measure in the session cube.

source_cube_name.dimension_name

The fully qualified name of the source dimension being included in the session cube.

dimension_name

The name of the dimension in the session cube.

FROM <dim from clause>

Valid specification for derived dimension definition only.

NOT_RELATED_TO_FACTS

Valid specification for derived dimension definition only.

<level type>

Valid specification for derived dimension definition only.

Remarks

Unlike server and local cubes, a session cube is not persisted beyond the session that created the session cube. A session cube is defined in terms of the measures and definitions that define it. There are two types of dimensions.

- Source dimensions - These are dimensions that were part of one of more source cubes.
- Derived dimensions - These are dimensions that provide new analysis capabilities. A derived dimension can be a regular dimension defined based on a source dimension that is either sliced vertically or horizontally, or contains custom grouping of dimension members. A derived dimension can also be a data mining dimension based on a data mining model.



Note

The Dimension keyword can refer to either dimensions or hierarchies.

Session cubes are used primarily for dynamic grouping of attribute members into custom member groups by client applications, such as Microsoft Excel. In a session cube, you can perform the following tasks:

- Eliminate dimensions that exist in the source cube.
- Add or eliminate hierarchies from a dimension.
- Eliminate measure groups or specific measures.
- Add a new attribute, based on attribute binding, for purposes of creating groups against an existing attribute..



Important

Security on session cube objects is inherited from the underlying source objects. Other objects, such as actions and calculation scripts, are also inherited by the session cube.

The CREATE SESSION CUBE statement follows these rules:

- You cannot perform grouping on parent-child hierarchies.
- You cannot perform grouping on ROLAP dimensions.
- You cannot perform grouping on linked dimensions.
- You cannot perform grouping on levels with custom rollups.
- You cannot perform grouping on discretized attribute hierarchies.
- You cannot perform grouping on unnatural hierarchies, which are hierarchies with many-to-many relationships between levels (such as age and gender).
- Explicit references to a cube name in MDX script are broken by grouping because the session cube has a different name. Use the CURRENTCUBE keyword instead.
- You cannot perform grouping on dimensions with explicit default members.
- When performing grouping, session-scoped calculated members on the original server cube are dropped.
- When performing grouping on a cube dimension in a server cube, the grouping affects all cube dimensions based on the same dimension.

Example

The following example demonstrates creating a session-scoped version of the Adventure Works cube that contains the Reseller Sales Amount measure, the Reseller dimension, the Product dimension, the Geography dimension, and the Date dimension. Within this session cube, two groups are created; one group contains countries in Europe and one group contains groups in North America. This sample is a simplified version of a CREATE SESSION CUBE statement issued by Microsoft Excel when a user creates a custom grouping of members.

```
CREATE SESSION CUBE [Adventure Works_XL_GROUPING1]
FROM [Adventure Works]
( MEASURE [Adventure Works].[Internet Sales Amount]
,MEASURE [Adventure Works].[Reseller Sales Amount]
, DIMENSION [Adventure Works].[Date].[Calendar]
, DIMENSION [Adventure Works].[Date].[Calendar Year]
, DIMENSION [Adventure Works].[Date].[Calendar Semester]
, DIMENSION [Adventure Works].[Date].[Calendar Quarter]
, DIMENSION [Adventure Works].[Date].[Month Name]
, DIMENSION [Adventure Works].[Date].[Date]
, DIMENSION [Adventure Works].[Geography].[Country]
    HIDDEN AS _XL_GROUPING81
, DIMENSION [Adventure Works].[Geography].[State-Province]
, DIMENSION [Adventure Works].[Geography].[City]
, DIMENSION [Adventure Works].[Geography].[Postal Code]
, DIMENSION [Adventure Works].[Geography].[Geography]
, DIMENSION [Adventure Works].[Product].[Product Categories]
, DIMENSION [Adventure Works].[Product].[Category]
, DIMENSION [Adventure Works].[Product].[Subcategory]
, DIMENSION [Adventure Works].[Product].[Product]
, DIMENSION [Adventure Works].[Product].[Product Key]
, DIMENSION [Adventure Works].[Reseller].[Reseller]
, DIMENSION [Adventure Works].[Reseller].[Geography Key]
, DIMENSION [Geography].[Country]
    NOT_RELATED_TO_FACTS FROM _XL_GROUPING81
    ( LEVEL [(All)]
    ,LEVEL [Country1] GROUPING
    ,LEVEL [Country]
```

```

    ,GROUP [Country1].[CountryX1_Grp_1]
        ( MEMBER [Geography].[Country].&[Canada]
          ,MEMBER [Geography].[Country].&[United States] )
    ,GROUP [Country1].[CountryX1_Grp_2]
        ( MEMBER [Geography].[Country].&[France]
          ,MEMBER [Geography].[Country].&[Germany]
          ,MEMBER [Geography].[Country].&[United Kingdom] )
    )
)

```

See Also

[CREATE GLOBAL CUBE Statement \(MDX\)](#)

[CREATE GLOBAL CUBE Statement \(MDX\)](#)

CREATE SET Statement

Creates a named set with session scope for the current cube.

Syntax

```

CREATE [SESSION] [ STATIC | DYNAMIC ] [HIDDEN] SET
  CURRENTCUBE | Cube_Name
    .Set_Name AS 'Set_Expression'
    [,Property_Name = Property_Value, ...n]

```

Arguments

Cube_Name

A valid string expression that provides the name of the cube.

Set_Name

A valid string expression that provides the name for the named set being created.

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Property_Name

A valid string that provides the name of a set property.

Property_Value

A valid scalar expression that defines the set property's value.

Remarks

A named set is a set of dimension members (or an expression that defines a set) that you create to use again. For example, a named set makes it possible to define a set of dimension members that consists of the set of the top ten stores by sales. This set can be defined statically, or by means of a function like TopCount. This named set can then be used wherever the set of the top 10 stores is needed.

The CREATE SET statement creates a named set that remains available throughout the session, and therefore, can be used in multiple queries in a session. For more information, see [MDX Data Definition Statements \(MDX\)](#).

You can also define a named set for use by a single query. To define such a set, you use the WITH clause in the SELECT statement. For more information about the WITH clause, see [Using WITH to Create Named Sets](#).

The Set_Expression clause can contain any function that supports MDX syntax. Sets created with the CREATE SET statement that do not specify the SESSION clause have session scope. Use the WITH clause to create a set with query scope.

Specifying a cube other than the cube that is currently connected causes an error. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

Scope

A user-defined set can occur within one of the scopes listed in the following table.

Query scope

The visibility and lifetime of the set is limited to the query. The set is defined in an individual query. Query scope overrides session scope. For more information, see [Using WITH to Create Named Sets](#).

Session scope

The visibility and lifetime of the set is limited to the session in which it is created. (The lifetime is less than the session duration if a DROP SET statement is issued on the set.) The CREATE SET statement creates a set with session scope. Use the WITH clause to create a set with query scope.

Example

The following example creates a set called Core Products. The SELECT query then demonstrates calling the newly created set. The CREATE SET statement must be executed before the SELECT query can be executed - they cannot be executed in the same batch.

```
CREATE SET [Adventure Works].[Core Products] AS  
'{[Product].[Category].[Bikes]}'
```

```
SELECT [Core Products] ON 0  
FROM [Adventure Works]
```

Set Evaluation

Set evaluation can be defined to occur differently; it can be defined to occur only once at set creation or can be defined to occur every time the set is used.

STATIC

Indicates that the set is evaluated only once at the time the CREATE SET statement is evaluated.

DYNAMIC

Indicates that the set is to be evaluated every time it is used in a query.

Set Visibility

The set can be either visible or not to other users who query the cube.

HIDDEN

Specifies that the set is not visible to users who query the cube.

Standard Properties

Each set has a set of default properties. When a client application is connected to Microsoft Analysis Services, the default properties are either supported, or available to be supported, as the administrator chooses.

Property identifier	Meaning
CAPTION	A string that the client application uses as the caption for the set.
DISPLAY_FOLDER	A string that identifies the path of the display folder that the client application uses to show the set. The folder level separator is defined by the client application. For the tools and clients supplied by Analysis Services, the backslash (\) is the level separator. To provide multiple display folders for a defined set, use a semicolon (;) to separate the folders.

See Also

[DROP SET Statement \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

CREATE SUBCUBE Statement

Redefines the cube space of a specified cube or subcube to a specified subcube. This statement changes the apparent cube space for subsequent operations.

Syntax

```
CREATE SUBCUBE Cube_Name AS Select_Statement  
| NON VISUAL ( Select_Statement )
```

Arguments

Cube_Name

The valid string expression that provides the name of the cube or perspective that is being restricted, which becomes the name of the subcube.

Select_Statement

A valid Multidimensional Expressions (MDX) SELECT expression that does not contain WITH, NON EMPTY, or HAVING clauses, and does not request dimension or cell properties.

See [SELECT Statement \(MDX\)](#) for a detailed syntax explanation on Select statements and the **NON VISUAL** clause.

Remarks

When default members are excluded in the definition of a subcube, coordinates will correspondingly change. For attributes that can be aggregated, the default member is moved to the [All] member. For attributes that cannot be aggregated, the default member is moved to a member that exists in the subcube. The following table contains example subcube and default member combinations.

Original default member	Can be aggregated	Subselect	Revised default member
Time.Year.All	Yes	{Time.Year.2003}	No change
Time.Year.[1997]	Yes	{Time.Year.2003}	Time.Year.All
Time.Year.[1997]	No	{Time.Year.2003}	Time.Year.[2003]
Time.Year.[1997]	Yes	{Time.Year.2003, Time.Year.2004}	Time.Year.All
Time.Year.[1997]	No	{Time.Year.2003, Time.Year.2004}	Either Time.Year.[2003] or Time.Year.[2004]

[All] members will always exist in a subcube.

Session objects created in the context of a subcube are dropped when the subcube is dropped.

For more information about subcubes, see [DROP SUBCUBE Statement \(MDX\)](#).

Example

The following example creates a subcube that restricts the apparent cube space to members that exist with the country of Canada. It then uses the **MEMBERS** function to return all members of the Country level of the Geography user-defined hierarchy - returning only the country of Canada.

```
CREATE SUBCUBE [Adventure Works] AS
    SELECT [Geography].[Country].&[Canada] ON 0
    FROM [Adventure Works]

SELECT [Geography].[Country].[Country].MEMBERS ON 0
    FROM [Adventure Works]
```

The following example creates a subcube that restricts the apparent cube space to {Accessories, Clothing} members in Products.Category and {[Value Added Reseller], [Warehouse]} in Resellers.[Business Type].

```
CREATE SUBCUBE [Adventure Works] AS
    Select {[Category].Accessories, [Category].Clothing} on 0,
           {[Business Type].[Value Added Reseller], [Business
Type].[Warehouse]} on 1
    from [Adventure Works]
```

Querying the subcube for all members in Products.Category and Resellers.[Business Type] with the following MDX:

```
select [Category].members on 0,
       [Business Type].members on 1
    from [Adventure Works]
    where [Measures].[Reseller Sales Amount]
```

Yields the following results:

	All Products	Accessories	Clothing
All Resellers	\$2,031,079.39	\$506,172.45	\$1,524,906.93
Value Added Reseller	\$767,388.52	\$175,002.81	\$592,385.71

Warehouse	\$1,263,690.86	\$331,169.64	\$932,521.23
-----------	----------------	--------------	--------------

Dropping and recreating the subcube using the NON VISUAL clause will create a subcube that keeps the true totals for all members in Products.Category and Resellers.[Business Type], whether they are visible or not in the subcube.

```
CREATE SUBCUBE [Adventure Works] AS
    NON VISUAL (Select {[Category].Accessories, [Category].Clothing} on 0,
                {[Business Type].[Value Added Reseller], [Business
Type].[Warehouse]} on 1
                from [Adventure Works])
```

Issuing the same MDX query from above:

```
select [Category].members on 0,
       [Business Type].members on 1
from [Adventure Works]
where [Measures].[Reseller Sales Amount]
```

Yields the following different results:

	All Products	Accessories	Clothing
All Resellers	\$80,450,596.98	\$571,297.93	\$1,777,840.84
Value Added Reseller	\$34,967,517.33	\$175,002.81	\$592,385.71
Warehouse	\$38,726,913.48	\$331,169.64	\$932,521.23

The [All Products] and [All Resellers], column and row respectively, contains totals for all members not only those visible ones.

See Also

[Key Concepts in MDX \(MDX\)](#)

[MDX Scripting Statements \(MDX\)](#)

[DROP SUBCUBE Statement](#)

[SELECT Statement \(MDX\)](#)

DROP ACTION Statement

Deletes a specified action from a specified cube.

Syntax

```
DROP ACTION CURRENTCUBE | Cube_Name  
    Action_Name
```

Arguments

Cube_Name

A valid string expression that provides the cube name.

Action_Name

A valid string expression that provides the name of the action being dropped.

See Also

[MDX Data Definition Statements \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

DROP CELL CALCULATION Statement

Removes the specified cell calculation.

Syntax

```
DROP [ SESSION ] CELL CALCULATION CURRENTCUBE | Cube_Name.CellCalc_Name
```

Arguments

Cube_Name

A valid string expression that provides the name of a cube expression.

CellCalc_Name

A valid string expression that provides the name of the cell calculation to be dropped.

See Also

[MDX Data Definition Statements \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

DROP KPI Statement

Drops the specified key performance indicator (KPI) from the specified cube.

Syntax

DROP KPI CURRENTCUBE | `Cube_Name.KPI_Name`

Arguments

Cube_Name

A valid string that specifies the cube name.

KPI_Name

A valid string that specifies the name of the KPI that is to be dropped.

See Also

[CREATE KPI Statement \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

DROP MEMBER Statement

Removes a calculated member.

Syntax

```
DROP MEMBER  
CURRENTCUBE | Cube_Name  
        .Member_Name  
        [,CURRENTCUBE | Cube_Name.Member_Name ...n]
```

Arguments

Cube_Name

A valid string expression that provides a cube name.

Member_Identifier

A valid string expression that provides a member name or member key.

See Also

[MDX Data Definition Statements \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

DROP SET Statement

Removes a named set.

Syntax

DROP [SESSION] SET

<Cube_Reference>.Set_Name
[, <Cube_Reference>.Set_Name ...n]

<Cube_Reference> ::= {CURRENTCUBE | Cube_Name}

Arguments

Cube_Name

A valid string expression that provides the name of the cube.

Set_Name

A valid string expression that provides that name of the named set to be dropped.

Remarks

For more information about named sets, see [MDX Data Definition Statements \(MDX\)](#).

See Also

[MDX Data Definition Statements \(MDX\)](#)

DROP SUBCUBE Statement

Drops a specified subcube, reverting to the previously defined cube or subcube definition with the specified name.

Syntax

DROP SUBCUBE Subcube_Name

Arguments

Subcube_Name

A valid string expression that provides a subcube name.

See Also

[CREATE SUBCUBE Statement \(MDX\)](#)

[CREATE SUBCUBE Statement](#)

REFRESH CUBE Statement

Refreshes the client cache for a cube.

Syntax

```
REFRESH CUBE Cube_Name
```

Arguments

Cube_Name

A valid string expression that provides a cube name.

Remarks

For client applications connected to an instance of Microsoft SQL Server Analysis Services, this statement causes the memory cached on the client application to be synchronized with the server. While this will ordinarily be detected and updated automatically, the length of time before this happens depends on the client connection string settings. The REFRESH CUBE statement refreshes data immediately.

For client applications connected to a local cube, the REFRESH CUBE statement causes the local cube file to be rebuilt.

Important

Named sets that are stored on the server are not refreshed.

See Also

[MDX Data Definition Statements \(MDX\)](#)

UPDATE MEMBER Statement

Updates an existing calculated member.

Syntax

```
UPDATE MEMBER Cube_Name.Member_Name  
    AS MDX_Expression  
    [,Property_Name = Property_Value, ...n]  
.....[,SCOPE_ISOLATION = CUBE]
```

Arguments

Cube_Name

A valid string that specifies the name of the cube that contains the member.

Member_Name

A valid string that specifies the name of an existing member.

MDX_Expression

A valid Multidimensional Expressions (MDX) expression to which the member is to be updated.

Property_Name

A valid string that specifies the name of a calculated member property.

Property_Value

A valid scalar expression that specifies the property value for the calculated member.

Remarks

The UPDATE MEMBER statement updates an existing calculated member while preserving the relative precedence of this member with respect to other calculations. Therefore, you cannot use the UPDATE MEMBER statement to change SOLVEORDER.

An UPDATE MEMBER statement cannot be specified in the MDX script for a cube.

Specifying a cube other than the cube that is currently connected causes an error. Therefore, you should use CURRENTCUBE in place of a cube name to denote the current cube.

For more information about member properties that are defined by OLE DB, see the OLE DB documentation.

Standard Properties

Each member has a set of default properties. The following table lists those default properties.

Property identifier	Meaning
FORMAT_STRING	A Microsoft Office style format string that the client application can use to display cell values.
VISIBLE	<p>A value that indicates whether the calculated member is visible in a schema rowset. Visible calculated members can be added to a set with the AddCalculatedMembers function. A nonzero value indicates that the calculated member is visible. The default value for this property is Visible.</p> <p>Calculated members that are not visible are generally used as intermediate steps in more complex calculated members. These calculated members can also be referred to by other types of members, such as</p>

Property identifier	Meaning
	measures.
NON_EMPTY_BEHAVIOR	The measure or set that MDX uses to determine the behavior of calculated members when resolving empty cells.
CAPTION	A string value that specifies the caption that the client application uses to display the member.
DISPLAY_FOLDER	A string value that specifies the path of the display folder where the member is to be shown by the client application. The folder level separator is defined by the client application. For the tools and clients supplied by Analysis Services, the backslash (\) as the level separator. To provide multiple display folders for a defined member, use a semicolon (;) to separate the folders.
ASSOCIATED_MEASURE_GROUP	The name of the measure group to which this member is associated.

See Also

[DROP MEMBER Statement \(MDX\)](#)

[CREATE MEMBER Statement \(MDX\)](#)

[MDX Data Definition Statements \(MDX\)](#)

MDX Data Manipulation Statements

In Multidimensional Expressions (MDX), data manipulation statements retrieve and manipulate data from multidimensional objects. The following table lists the data manipulation statements in MDX.

In this Section

Topic	Description
MDX Scripting Statements (MDX)	Runs a stored procedure that returns a void either in the current scope or optionally on

Topic	Description
	a specified cube.
CLEAR CALCULATIONS Statement	Removes all calculations from the cube and returns the cube to calculation pass 0.
DRILLTHROUGH Statement	Retrieves the rowsets that were used to create a specified cell in a cube.
SELECT Statement (MDX)	Retrieves data from a specified cube.
UPDATE CUBE Statement	Updates the value of a specified leaf or nonleaf cell in a cube, optionally allocating the value for a specified non-leaf cell across dependent leaf cells.

See Also

[MDX Statement Reference](#)

[MDX Data Definition Statements](#)

[MDX Scripting Statements](#)

CALL Statement

Runs a stored procedure that returns a void either in the current scope or optionally on a specified cube.

Syntax

```
CALL SP_Name
    [ (SP_Argument
        [, SP_Argument ,...n]
    ) ]
[ON Cube_Expression]
```

Arguments

SP_Name

A valid string expression that provides the name of a stored procedure.

SP_Argument

A valid string expression that provides an argument to the called stored procedure.

Cube_Expression

A valid string cube expression providing the name of the cube.

Remarks

The **CALL** statement runs a specified registered stored procedure, optionally including one or more arguments for the specified stored procedure. The **CALL** statement is for use only with stored procedures that return voids. This statement cannot be combined with other functions or operators within an MDX expression. Registered stored procedures that return values can be called directly within MDX expressions and combined with other MDX functions and operators.

If a cube is not specified, the statement runs the stored procedure on the current cube.

Note

If the stored procedure is not registered on the client, the **CALL** statement attempts to call the stored procedure from an instance of Microsoft SQL Server Analysis Services.

See Also

[Using Stored Procedures \(MDX\)](#)

[Stored Procedures \(MDX\)](#)

CLEAR CALCULATIONS Statement

Removes all calculations from the cube and returns the cube to calculation pass 0.

Syntax

```
CLEAR CALCULATIONS [FROM Cube_Expression]
```

Arguments

Cube_Expression

A valid Multidimensional Expressions (MDX) cube expression.

Remarks

The **FROM** clause can be omitted when the context of the cube is known, such as in an MDX script.

Note

This statement can only be executed by a server or database administrator, or a member of a role that has access to the source data in the cube (that is, ReadSourceData=true)

See Also

[MDX Data Manipulation Statements \(MDX\)](#)

DRILLTHROUGH Statement

Retrieves the underlying table rows that were used to create a specified cell in a cube.

Syntax

```
DRILLTHROUGH [MAXROWS Unsigned_Integer]  
    <MDX SELECT statement>  
    [RETURN Set_of_Attributes_and_Measures  
        [,Set_of_Attributes_and_Measures ...]  
    ]
```

Arguments

Unsigned_Integer

A positive integer value.

MDX SELECT statement

Any valid Multidimensional Expressions (MDX) expressions SELECT statement.

Set_of_Attributes_and_Measures

A comma-separated list of dimension attributes and measures.

Remarks

Drillthrough is an operation in which an end user selects a single cell from a cube and retrieves a result set from the source data for that cell in order to get more detailed information. By default, a drillthrough result set is derived from the table rows that were evaluated to calculate the value of the selected cube cell. For end users to drill through, their client applications must support this capability. In Microsoft SQL Server Analysis Services, the results are retrieved directly from MOLAP storage, unless ROLAP partitions or dimensions are queried.

Important

Drillthrough security is based on the general security options defined on the cube. If a user cannot get some data by using MDX, drillthrough will also restrict the user in the exactly the same manner.

An MDX statement specifies the subject cell. The value specified by the **MAXROWS** argument indicates the maximum number of rows that should be returned by the resulting rowset.

By default, the maximum number of rows that are returned is 10,000 rows. This means that if you leave **MAXROWS** unspecified, you will get 10,000 rows or less. If this value is too low for your scenario, you can set **MAXROWS** to a higher number, such as `MAXROWS 20000`. If it is too low overall, you can increase the default by changing the

OLAP\Query\DefaultDrillthroughMaxRows server property. For more information about changing this property, see [Configure Server Properties in Analysis Services](#).

Unless otherwise specified, the columns returned include all granularity attributes for all dimensions related to the measure group of the specified measure, other than many-to-many dimensions. Cube dimensions are preceded by \$ to distinguish between dimensions and measure groups. The **RETURN** clause is used to specify the columns returned by the drillthrough query. The following functions can be applied to a single attribute or measure by the **RETURN** clause.

Name(attribute_name)

Returns the name of the specified attribute member.

UniqueName(attribute_name)

Returns the unique name of the specified attribute member.

Key(attribute_name[, N])

Returns the key of the specified attribute member, where N specifies column in the composite key (if any). The default value for N is 1.

Caption(attribute_name)

Returns the caption of the specified attribute member.

MemberValue(attribute_name)

Returns the member value of the specified attribute member.

Translation(attribute_name[, N])

Returns the translated value of the specified attribute member, where N is the language.

CustomRollup(attribute_name)

Returns the custom rollup expression of the specified attribute member.

CustomRollupProperties(attribute_name)

Returns the custom rollup properties of the specified attribute member.

UnaryOperator(attribute_name)

Returns the unary operator of the specified attribute member.

Example

The following example specifies cell for the month of July, 2007 for the reseller sales amount measure (the default measure) for the country of Australia. The RETURN clause specifies that the date of each sale, the product model name, the employee name, the sales amount, the tax amount and the product cost values that underlie this cell be returned.

```
DRILLTHROUGH
SELECT
    ([Date].[Calendar].[Month].[July 2007])
ON 0
FROM [Adventure Works]
```

```

WHERE [Geography].[Country].[Australia]
RETURN
    [$Date].[Date]
    ,KEY([$Product].[Model Name])
    ,NAME([$Employee].[Employee])
    ,[Reseller Sales].[Reseller Sales Amount]
    ,[Reseller Sales].[Reseller Tax Amount]
    ,[Reseller Sales].[Reseller Standard Product Cost]

```

See Also

[MDX Data Manipulation Statements \(MDX\)](#)

SELECT Statement

Retrieves data from a specified cube.

Syntax

```

[ WITH <SELECT WITH clause>
  [ , <SELECT WITH clause>...n ]
]
SELECT
  [ *
  | ( <SELECT query axis clause>
      [ , <SELECT query axis clause>,...n ]
    )
  ]
FROM
  <SELECT subcube clause>
  [ <SELECT slicer axis clause> ]
  [ <SELECT cell property list clause> ]

<SELECT WITH clause> ::=
  ( CELL CALCULATION <CREATE CELL CALCULATION body clause> )
| ( [ CALCULATED ] MEMBER <CREATE MEMBER body clause> )
| ( SET <CREATE SET body clause> )
| ( MEASURE = <measure body clause> )

```

<SELECT query axis clause> ::=
[NON EMPTY] **Set_Expression**
[<SELECT dimension property list clause>]
ON
 Integer_Expression
| AXIS(**Integer**)
| COLUMNS
| ROWS
| PAGES
| SECTIONS
| CHAPTERS

<SELECT subcube clause> ::=
 Cube_Name
| [NON VISUAL] (SELECT
 [*
| (<SELECT query axis clause> [,
 <SELECT query axis clause>,...n])
|
 FROM
 <SELECT subcube clause>
 <SELECT slicer axis clause>)

<SELECT slicer axis clause> ::=
 WHERE **Tuple_Expression**

<SELECT cell property list clause> ::=
[CELL] PROPERTIES **CellProperty_Name**
[, **CellProperty_Name**,...n]

<SELECT dimension property list clause> ::=
[DIMENSION] PROPERTIES
 (**DimensionProperty_Name**
 [, **DimensionProperty_Name**,...n])

```
| (LevelProperty_Name  
  [, LevelProperty_Name,...n ] )  
| (MemberProperty_Name  
  [, MemberProperty_Name,...n ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Integer

An integer between 0 and 127.

Cube_Name

A valid string that provides a cube name.

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

CellProperty_Name

A valid string that represents a cell property.

DimensionProperty_Name

A valid string that represents a dimension property.

LevelProperty_Name

A valid string that represents a level property.

MemberProperty_Name

A valid string that represents a member property.

Remarks

The `<SELECT slicer axis clause>` expression must contain members in dimensions and hierarchies other than those referenced in the specified `<SELECT query axis clause>` expressions.

If an attribute in the cube is omitted from the specified `<SELECT query axis clause>` expressions and the `<SELECT slicer axis clause>` value, the attribute's default member is implicitly added to the slicer axis.

The `NON VISUAL` option in the subselect statement enables you to filter out members while keeping the true totals instead of filtered totals. This enables you to query for the top ten sales (persons/products/regions) and obtain the true total of sales for all queried members, instead of the total value of sales for the top ten returned. See the examples below for more information.

Calculated members can be included in `<SELECT query axis clause>` whenever the connection was opened using the connection string parameter `subqueries=1`; see [Supported XMLA](#)

[Properties \(XMLA\)](#) and

P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString for parameter usage. An example is provided on calculated members in subselects.

Autoexists

When two or more attributes of the dimension are used in a SELECT statement, Analysis Services evaluates the attributes' expressions to make sure that the members of those attributes are properly confined to meet the criteria of all other attributes. For example, suppose you are working with attributes from the Geography dimension. If you have one expression that returns all members from the City attribute, and another expression that confines members from the Country attribute to all countries in Europe, then this will result in the City members being confined to only those cities that belong to countries in Europe. This characteristic of Analysis Services is called Autoexists and applies only to attributes in the same dimension. Autoexists only applies to attributes from the same dimension because it tries to prevent the dimension records excluded in one attribute expression from being included by the other attribute expressions. Autoexists can also be understood as the resulting intersection of the different attributes expressions over the dimension records. See the following examples below:

```
//Obtain the Top 10 best reseller selling products by Name
```

```
with member [Measures].[PCT Discount] AS '[Measures].[Discount Amount]/[Measures].[Reseller Sales Amount]', FORMAT_STRING = 'Percent'
    set Top10SellingProducts as 'topcount([Product].[Model Name].children, 10, [Measures].[Reseller Sales Amount])'
    set Preferred10Products as '
        {[Product].[Model Name].&[Mountain-200],
        [Product].[Model Name].&[Road-250],
        [Product].[Model Name].&[Mountain-100],
        [Product].[Model Name].&[Road-650],
        [Product].[Model Name].&[Touring-1000],
        [Product].[Model Name].&[Road-550-W],
        [Product].[Model Name].&[Road-350-W],
        [Product].[Model Name].&[HL Mountain Frame],
        [Product].[Model Name].&[Road-150],
        [Product].[Model Name].&[Touring-3000]
        }'

select {[Measures].[Reseller Sales Amount], [Measures].[Discount Amount], [Measures].[PCT Discount]} on 0,
    Top10SellingProducts on 1
```

from [Adventure Works]

The obtained result set is:

	Reseller Sales Amount	Discount Amount	PCT Discount
Mountain-200	\$14,356,699.36	\$19,012.71	0.13%
Road-250	\$9,377,457.68	\$4,032.47	0.04%
Mountain-100	\$8,568,958.27	\$139,393.27	1.63%
Road-650	\$7,442,141.81	\$39,698.30	0.53%
Touring-1000	\$6,723,794.29	\$166,144.17	2.47%
Road-550-W	\$3,668,383.88	\$1,901.97	0.05%
Road-350-W	\$3,665,932.31	\$20,946.50	0.57%
HL Mountain Frame	\$3,365,069.27	\$174.11	0.01%
Road-150	\$2,363,805.16	\$0.00	0.00%
Touring-3000	\$2,046,508.26	\$79,582.15	3.89%

The obtained set of products seems to be the same as Preferred10Products; so, verifying the Preferred10Products set:

```
with member [Measures].[PCT Discount] AS '[Measures].[Discount Amount]/[Measures].[Reseller Sales Amount]', FORMAT_STRING = 'Percent'
    set Top10SellingProducts as 'topcount([Product].[Model Name].children, 10, [Measures].[Reseller Sales Amount])'
    set Preferred10Products as '
        {[Product].[Model Name].&[Mountain-200],
        [Product].[Model Name].&[Road-250],
        [Product].[Model Name].&[Mountain-100],
        [Product].[Model Name].&[Road-650],
        [Product].[Model Name].&[Touring-1000],
        [Product].[Model Name].&[Road-550-W],
        [Product].[Model Name].&[Road-350-W],
        [Product].[Model Name].&[HL Mountain Frame],
        [Product].[Model Name].&[Road-150],
        [Product].[Model Name].&[Touring-3000]
        }'
```



```

select {[Measures].[Reseller Sales Amount], [Measures].[Discount Amount],
[Measures].[PCT Discount]} on 0,
    Preferred10Products on 1
from [Adventure Works]

```

As per the following results, both sets (Top10SellingProducts, Preferred10Products) are the same

	Reseller Sales Amount	Discount Amount	PCT Discount
Mountain-200	\$14,356,699.36	\$19,012.71	0.13%
Road-250	\$9,377,457.68	\$4,032.47	0.04%
Mountain-100	\$8,568,958.27	\$139,393.27	1.63%
Road-650	\$7,442,141.81	\$39,698.30	0.53%
Touring-1000	\$6,723,794.29	\$166,144.17	2.47%
Road-550-W	\$3,668,383.88	\$1,901.97	0.05%
Road-350-W	\$3,665,932.31	\$20,946.50	0.57%
HL Mountain Frame	\$3,365,069.27	\$174.11	0.01%
Road-150	\$2,363,805.16	\$0.00	0.00%
Touring-3000	\$2,046,508.26	\$79,582.15	3.89%

In the previous examples we created two sets: one as a calculated expression and the other as a constant expression. These examples illustrate the different flavors of Autoexists.

Autoexists can be applied deep or shallow to the expressions. The default setting is deep. The following example will illustrate the concept of deep Autoexists. In the example we are filtering Top10SellingProducts by [Product].[Product Line] attribute for those in [Mountain] group. Note that both attributes (slicer and axis) belong to the same dimension, [Product].

```

with member [Measures].[PCT Discount] AS '[Measures].[Discount
Amount]/[Measures].[Reseller Sales Amount]', FORMAT_STRING = 'Percent'
    set Top10SellingProducts as 'topcount([Product].[Model Name].children,
10, [Measures].[Reseller Sales Amount])'
// Preferred10Products set removed for clarity
select {[Measures].[Reseller Sales Amount], [Measures].[Discount Amount],
[Measures].[PCT Discount]} on 0,
    Top10SellingProducts on 1
from [Adventure Works]

```

where [Product].[Product Line].[Mountain]

Produces the following result set:

	Reseller Sales Amount	Discount Amount	PCT Discount
Mountain-200	\$14,356,699.36	\$19,012.71	0.13%
Mountain-100	\$8,568,958.27	\$139,393.27	1.63%
HL Mountain Frame	\$3,365,069.27	\$174.11	0.01%
Mountain-300	\$1,907,249.38	\$876.95	0.05%
Mountain-500	\$1,067,327.31	\$17,266.09	1.62%
Mountain-400-W	\$592,450.05	\$303.49	0.05%
LL Mountain Frame	\$521,864.42	\$252.41	0.05%
ML Mountain Frame-W	\$482,953.16	\$206.95	0.04%
ML Mountain Frame	\$343,785.29	\$161.82	0.05%
Women's Mountain Shorts	\$260,304.09	\$6,675.56	2.56%

In the previous result set we have seven newcomers to the list of Top10SellingProducts and Mountain-200, Mountain-100 and HL Mountain Frame have moved to the top of the list. In the previous result set those three values were interspersed

This is called Deep Autoexists, because the Top10SellingProducts set is evaluated to meet the slicing conditions of the query. Deep Autoexists means that all expressions will be evaluated to meet the deepest possible space after applying the slicer expressions, the sub select expressions in the axis, and so on.

However, one might want to be able to do the analysis over the Top10SellingProducts as equivalent to Preferred10Products, as in the following example:

```
with member [Measures].[PCT Discount] AS '[Measures].[Discount Amount]/[Measures].[Reseller Sales Amount]', FORMAT_STRING = 'Percent'
    set Top10SellingProducts as 'topcount([Product].[Model Name].children, 10, [Measures].[Reseller Sales Amount])'
    set Preferred10Products as '
        {[Product].[Model Name].&[Mountain-200],
        [Product].[Model Name].&[Road-250],
        [Product].[Model Name].&[Mountain-100],
```

```

[Product].[Model Name].&[Road-650],
[Product].[Model Name].&[Touring-1000],
[Product].[Model Name].&[Road-550-W],
[Product].[Model Name].&[Road-350-W],
[Product].[Model Name].&[HL Mountain Frame],
[Product].[Model Name].&[Road-150],
[Product].[Model Name].&[Touring-3000]
}'

```

```

select {[Measures].[Reseller Sales Amount], [Measures].[Discount Amount],
[Measures].[PCT Discount]} on 0,
    Preferred10Products on 1
from [Adventure Works]
where [Product].[Product Line].[Mountain]

```

Produces the following result set:

	Reseller Sales Amount	Discount Amount	PCT Discount
Mountain-200	\$14,356,699.36	\$19,012.71	0.13%
Mountain-100	\$8,568,958.27	\$139,393.27	1.63%
HL Mountain Frame	\$3,365,069.27	\$174.11	0.01%

In the above results, the slicing gives a result that contains only those products from Preferred10Products that are part of the [Mountain] group in [Product].[Product Line]; as expected, because Preferred10Products is a constant expression.

This result set is also understood as shallow Autoexists. This is because the expression is evaluated before the slicing clause. In the previous example, the expression was a constant expression for illustration purposes in order to introduce the concept.

Autoexists behavior can be modified at the session level using the **Autoexists** connection string property. The following example begins by opening a new session and adding the Autoexists=3 property to the connection string. You must open a new connection in order to do the example. Once the connection is established with the Autoexist setting it will remain in effect until that connection is finished.

```

with member [Measures].[PCT Discount] AS '[Measures].[Discount
Amount]/[Measures].[Reseller Sales Amount]', FORMAT_STRING = 'Percent'
    set Top10SellingProducts as 'topcount([Product].[Model Name].children,
10, [Measures].[Reseller Sales Amount])'

```

```
//Preferred10Products set removed for clarity
```

```
select {[Measures].[Reseller Sales Amount], [Measures].[Discount Amount],  
[Measures].[PCT Discount]} on 0,  
    Top10SellingProducts on 1
```

```
from [Adventure Works]
```

```
where [Product].[Product Line].[Mountain]
```

The following result set now shows the shallow behavior of Autoexists.

	Reseller Sales Amount	Discount Amount	PCT Discount
Mountain-200	\$14,356,699.36	\$19,012.71	0.13%
Mountain-100	\$8,568,958.27	\$139,393.27	1.63%
HL Mountain Frame	\$3,365,069.27	\$174.11	0.01%

Autoexists behavior can be modified by using the `AUTOEXISTS=[1|2|3]` parameter in the connection string; see [Supported XMLA Properties \(XMLA\)](#) and **P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString** for parameter usage.

Examples

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first eight months of calendar year 2003 that are contained in the `Date` dimension, from the **Adventure Works** cube.

```
WITH MEMBER [Date].[Calendar].[First8Months2003] AS  
    Aggregate (  
        PeriodsToDate (  
            [Date].[Calendar].[Calendar Year],  
            [Date].[Calendar].[Month].[August 2003]  
        )  
    )  
SELECT  
    [Date].[Calendar].[First8Months2003] ON COLUMNS,  
    [Product].[Category].Children ON ROWS  
FROM
```

[Adventure Works]

WHERE

[Measures].[Order Quantity]

To understand **NON VISUAL**, the following example is a query of [Adventure Works] to obtain [Reseller Sales Amount] figures in a table where the product categories are the columns and the reseller business types are the rows. Note that totals are given for both products and resellers.

The following SELECT statement:

```
select [Category].members on 0,  
       [Business Type].members on 1  
from [Adventure Works]  
where [Measures].[Reseller Sales Amount]
```

Produces the following results:

	All Products	Accessories	Bikes	Clothing	Components
All Resellers	\$80,450,596.98	\$571,297.93	\$66,302,381.56	\$1,777,840.84	\$11,799,076.66
Specialty Bike Shop	\$6,756,166.18	\$65,125.48	\$6,080,117.73	\$252,933.91	\$357,989.07
Value Added Reseller	\$34,967,517.33	\$175,002.81	\$30,892,354.33	\$592,385.71	\$3,307,774.48
Warehouse	\$38,726,913.48	\$331,169.64	\$29,329,909.50	\$932,521.23	\$8,133,313.11

To produce a table with data only for the Accessories and Clothing products, the Value Added Reseller and Warehouse resellers, yet keeping the overall totals could be written as follows using NON VISUAL:

```
select [Category].members on 0,  
       [Business Type].members on 1  
from NON VISUAL (Select {[Category].Accessories, [Category].Clothing} on  
0,  
                  {[Business Type].[Value Added Reseller], [Business  
Type].[Warehouse]} on 1
```

```

from [Adventure Works])
where [Measures].[Reseller Sales Amount]

```

Produces the following results:

	All Products	Accessories	Clothing
All Resellers	\$80,450,596.98	\$571,297.93	\$1,777,840.84
Value Added Reseller	\$34,967,517.33	\$175,002.81	\$592,385.71
Warehouse	\$38,726,913.48	\$331,169.64	\$932,521.23

To produce a table that visually totals the columns but for row totals brings the true total of all [Category], the following query should be issued:

```

select [Category].members on 0,
       [Business Type].members on 1
from NON VISUAL (Select {[Category].Accessories, [Category].Clothing} on 0
                 from ( Select {[Business Type].[Value Added Reseller],
[Business Type].[Warehouse]} on 0
                       from [Adventure Works])
                 )
where [Measures].[Reseller Sales Amount]

```

Note how NON VISUAL is only applied to [Category].

The above query produces the following results:

	All Products	Accessories	Clothing
All Resellers	\$73,694,430.80	\$506,172.45	\$1,524,906.93
Value Added Reseller	\$34,967,517.33	\$175,002.81	\$592,385.71
Warehouse	\$38,726,913.48	\$331,169.64	\$932,521.23

When compared with the previous results, you can observe that the [All Resellers] row now adds up to the displayed values for [Value Added Reseller] and [Warehouse] but that the [All Products] column shows the total value for all products, including those not displayed.

The following example demonstrates how to use calculated members in subselects to filter on them. To be able to reproduce this sample, the connection must be established using the connection string parameter subqueries=1.

```
select Measures.allmembers on 0
from (
    Select { [Measures].[Reseller Sales Amount]
            , [Measures].[Reseller Total Product Cost]
            , [Measures].[Reseller Gross Profit]
            , [Measures].[Reseller Gross Profit Margin]
            } on 0
    from [Adventure Works]
)
```

The above query produces the following results:

Reseller Sales Amount	Reseller Total Product Cost	Reseller Gross Profit	Reseller Gross Profit Margin
\$80,450,596.98	\$79,980,114.38	\$470,482.60	0.58%

See Also

[Restricting the Query with Query and Slicer Axes \(MDX\)](#)

[MDX Data Manipulation Statements \(MDX\)](#)

[Restricting the Query with Query and Slicer Axes \(MDX\)](#)

UPDATE CUBE Statement

Updates the value of a specified leaf or nonleaf cell in a cube, optionally allocating the value for a specified non-leaf cell across dependent leaf cells.

Syntax

```
UPDATE [ CUBE ] Cube_Name
SET
    <update clause>
    [, <update clause> ...n ]
```

<update clause> ::=

```

Tuple_Expression[.VALUE]= New_Value
[
  NO_ALLOCATION
| USE_EQUAL_ALLOCATION
  | USE_EQUAL_INCREMENT
  | USE_WEIGHTED_ALLOCATION [ BY Weight_Expression]
  | USE_WEIGHTED_INCREMENT [ BY Weight_Expression]
]

```

Arguments

Cube_Name

A valid string that provides the name of a cube.

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

New_Value

A valid numeric expression.

Weight_Expression

A valid Multidimensional Expressions (MDX) numeric expression that returns a decimal value between 0 and 1.

Remarks

The cell specified by the tuple expression can be any valid cell in the multidimensional space (that is, the cell does not have to be a leaf cell). However, the cell must be aggregated with the Sum aggregate function and must not include a calculated member in the tuple that is used to identify the cell.

It may be helpful to think of the **UPDATE CUBE** statement as a subroutine that will automatically generate a series of individual cell writeback operations to leaf and non-leaf cells that will roll up into a specified sum.

The following table describes the methods of allocation.

Allocation method	Description
USE_EQUAL_ALLOCATION	Every leaf cell that contributes to the updated cell will be assigned an equal value based on the following expression: $\langle \text{leaf cell value} \rangle = \frac{\langle \text{New Value} \rangle}{\text{Count}(\text{leaf cells that$

Allocation method	Description
	are contained in <tuple>)
USE_EQUAL_INCREMENT	<p>Every leaf cell that contributes to the updated cell will be changed according to the following expression:</p> $\langle \text{leaf cell value} \rangle = \langle \text{leaf cell value} \rangle + (\langle \text{New Value} \rangle - \langle \text{existing value} \rangle) / \mathbf{Count}(\text{leaf cells contained in } \langle \text{tuple} \rangle)$
USE_WEIGHTED_ALLOCATION	<p>Every leaf cell that contributes to the updated cell will be assigned an equal value that is based on the following expression:</p> $\langle \text{leaf cell value} \rangle = \langle \text{New Value} \rangle * \text{Weight_Expression}$
USE_WEIGHTED_INCREMENT	<p>Every leaf cell that contributes to the updated cell will be changed according to the following expression:</p> $\langle \text{leaf cell value} \rangle = \langle \text{leaf cell value} \rangle + (\langle \text{New Value} \rangle - \langle \text{existing value} \rangle) * \text{Weight_Expression}$

If a weight expression is not specified, the **UPDATE CUBE** statement implicitly uses the following expression:

$$\text{Weight_Expression} = \langle \text{leaf cell value} \rangle / \langle \text{existing value} \rangle$$

A weight expression should be expressed as a decimal value between zero (0) and 1. This value specifies the ratio of the allocated value that you want to assign to the leaf cells that are affected by the allocation. The client application programmer's has the responsibility of creating expressions whose rollup aggregate values will equal the allocated value of the expression.

Caution

The client application must consider the allocation of all dimensions concurrently to avoid possible unexpected results, including incorrect rollup values or inconsistent data.

Each **UPDATE CUBE** allocation should be considered to be atomic for transactional purposes. This means, that if any one of the allocation operations fails for any reason, such as an error in a formula or a security violation, the whole UPDATE CUBE operation will fail. Before the

calculations of the individual allocation operations are processed, a snapshot of the data is taken to ensure that the resulting calculations are correct.

 **Caution**

When used on a measure that contains integers, the `USE_WEIGHTED_ALLOCATION` method can return imprecise results caused by incremental rounding changes.

 **Important**

When updated cells do not overlap, the **Update Isolation Level** connection string property can be used to enhance performance for UPDATE CUBE.

See Also

[MDX Data Manipulation Statements \(MDX\)](#)

P:Microsoft.AnalysisServices.AdomdClient.AdomdConnection.ConnectionString

MDX Operator Reference

The Multidimensional Expressions (MDX) language supports arithmetic, logical, comparison, set, string, and unary operators. The following table lists the supported operators and their descriptions.

In This Section

Topic	Description
-- (Comment)	Indicates comment text that is provided by the user.
- (Except)	Performs a set operation that returns the difference between two sets, removing duplicate members.
- (Negative) (MDX)	Performs a unary operation that returns the negative value of a numeric expression.
- (Subtract) (MDX)	Performs an arithmetic operation that subtracts one number from another number.
* (Crossjoin)	Performs a set operation that returns the cross product of two sets.
* (Multiply) (MDX)	Performs an arithmetic operation that multiplies two numbers.

Topic	Description
/(Divide) (MDX)	Performs an arithmetic operation that divides one number by another number.
^ (Power)	Performs an arithmetic operation that raises one number by another number.
/*...*/ (Comment)	Indicates comment text that is provided by the user.
// (Comment)	Indicates user-provided text.
:(Range)	Performs a set operation that returns a naturally ordered set, with the two specified members as endpoints, and all members between the two specified members included as members of the set.
+ (Add) (MDX)	Performs an arithmetic operation that adds two numbers.
+ (Positive) (MDX)	Performs a unary operation that returns the positive value of a numeric expression.
+ (String Concatenation) (MDX)	Performs a string operation that concatenates two or more character strings, tuples, or a combination of strings and tuples.
+ (Union)	Performs a set operation that returns a union of two sets, removing duplicates.
< (Less Than) (MDX)	Performs a comparison operation that determines whether the value of one MDX expression is less than the value of another MDX expression.
<= (Less Than or Equal To) (MDX)	Performs a comparison operation that determines whether the value of one MDX expression is less than or equal to the value of another MDX expression.
<> (Not Equal To) (MDX)	Performs a comparison operation that determines whether the value of one MDX expression is not equal to the value of another MDX expression.
= (Equal To)	Performs a comparison operation that

Topic	Description
	determines whether the value of one MDX expression is equal to the value of another MDX expression.
> (Greater Than) (MDX)	Performs a comparison operation that determines whether the value of one MDX expression is greater than the value of another MDX expression.
>= (Greater Than or Equal To) (MDX)	Performs a comparison operation that determines whether the value of one MDX expression is greater than or equal to the value of another MDX expression.
AND (MDX)	Performs a logical conjunction on two numeric expressions.
IS	Performs a logical comparison on two object expressions.
NOT (MDX)	Performs a logical negation on a numeric expression.
OR (MDX)	Performs a logical disjunction on two numeric expressions.
XOR	Performs a logical exclusion on two numeric expressions.

See Also

[MDX Language Reference](#)

-- (Comment)

Indicates comment text that is provided by the user.

Syntax

```
-- Comment_Text
```

Parameters

Parameter	Description
Comment_Text	The string that contains the text of the comment.

Remarks

Comments can be inserted on a separate line, nested at the end of a Multidimensional Expressions (MDX) script line, or nested within an MDX statement. The server does not evaluate the comment.

Use this operator for single-line or nested comments. Comments inserted with -- are delimited by the newline character.

There is no maximum length for comments.

Examples

The following example demonstrates the use of this operator.

```
-- This member returns the gross profit margin for product types  
-- and reseller types crossjoined by year.
```

```
SELECT  
    [Date].[Calendar].[Calendar Year].Members *  
    [Reseller].[Reseller Type].Children ON 0,  
    [Product].[Category].[Category].Members ON 1  
FROM -- Select from the Adventure Works cube.  
    [Adventure Works]  
WHERE  
    [Measures].[Gross Profit Margin]
```

See Also

[/*...*/ \(Comment\) \(MDX\)](#)

[// \(Comment\) \(MDX\)](#)

[MDX Operator Reference \(MDX\)](#)

- (Except)

Performs a set operation that returns the difference between two sets, removing duplicate members.

Syntax

```
Set_Expression - Set_Expression
```

Parameters

Parameter	Description
Set_Expression	A valid Multidimensional Expressions (MDX) expression that returns a set.

Return Value

A set that contains members that are not shared by both specified parameters.

Remarks

The - (**Except**) operator is functionally equivalent to the Except function.

Examples

The following example demonstrates the use of this operator:

```
// This query shows the quantity of orders for all product categories
// with the exception of Components.
SELECT
    [Measures].[Order Quantity] ON COLUMNS,
    [Product].[Product Categories].[All].Children
    - [Product].[Product Categories].[Components] ON ROWS
FROM
    [Adventure Works]
```

See Also

[MDX Operator Reference \(MDX\)](#)

- (Negative)

Performs a unary operation that returns the negative value of a numeric expression.

Syntax

- **Numeric_Expression**

Parameters

Parameter	Description
Numeric_Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A negative value that has the data type of the specified parameter.

Examples

The following example demonstrates the use of this operator.

```
-- This member creates a negative version of the
-- Reseller Freight Cost.
WITH MEMBER
    Measures.[Resell Cost as Negative]
    AS -Measures.[Reseller Freight Cost]
SELECT
    [Date].[Calendar Month of Year].Children ON COLUMNS,
    [Product].[Product Categories].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    {[Measures].[Resell Cost as Negative]}
```

See Also

[MDX Operator Reference \(MDX\)](#)

- (Subtract)

Performs an arithmetic operation that subtracts one number from another number.

Syntax

Numeric_Expression - Numeric_Expression

Parameters

Parameter	Description
Numeric_Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If one expression evaluates to a null value, the operator returns the result of the non-null expression.

Examples

The following example demonstrates the use of this operator.

```
-- This member returns the increase or decrease  
-- in gross profit margin over a month.
```

```
WITH MEMBER [Measures].[GPM Delta] AS  
(  
(Measures.[Gross Profit Margin]) -  
([Date].[Calendar].CurrentMember.PrevMember,  
Measures.[Gross Profit Margin])
```



```

), FORMAT_STRING = 'Percent'

SELECT
DESCENDANTS (
[Date].[Calendar].[Calendar Year].&[2002],
[Date].[Calendar].[Month]) ON 0,
[Product].[Category].[Category].Members ON 1
FROM
[Adventure Works]
WHERE
([Measures].[GPM Delta])

```

See Also

[MDX Operator Reference \(MDX\)](#)

* (Crossjoin)

Performs a set operation that returns the cross product of two sets.

Syntax

```
Set_Expression * Set_Expression
```

Parameter

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Return Value

A set that contains the cross product of both specified parameters.

Remarks

The * (**Crossjoin**) operator is functionally equivalent to the Crossjoin function.

Examples

The following example demonstrates the use of this operator.

```

-- This query returns the gross profit margin for product types
-- and reseller types crossjoined by year.
SELECT
    [Date].[Calendar].[Calendar Year].Members *
    [Reseller].[Reseller Type].Children ON 0,
    [Product].[Category].[Category].Members ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[Gross Profit Margin])

```

See Also

[MDX Operator Reference \(MDX\)](#)

* (Multiply)

Performs an arithmetic operation that multiplies two numbers.

Syntax

Numeric_Expression * Numeric_Expression

Parameters

Parameter	Description
Numeric_Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If one expression evaluates to a null value, the operator returns a null value.

See Also

[MDX Operator Reference \(MDX\)](#)

/ (Divide)

Performs an arithmetic operation that divides one number by another number.

Syntax

`Dividend / Divisor`

Parameters

Parameter	Description
Dividend	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.
Divisor	A valid MDX expression that returns a numeric value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

The actual value returned by the **/ (Divide)** operator represents the quotient of the first expression divided by the second expression.

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If Divisor evaluates to a null value, the operator raises an error. If both Divisor and Dividend evaluate to a null value, the operator returns a null value.

Examples

The following example demonstrates the use of this operator.

```
-- This query returns the freight cost per user,  
-- for products, averaged by month.
```

```
With Member [Measures].[Freight Per Customer] as  
    [Measures].[Internet Freight Cost]  
    /  
    [Measures].[Customer Count]
```

```
SELECT  
    [Ship Date].[Calendar].[Calendar Year] Members ON 0,  
    [Product].[Category].[Category].Members ON 1  
FROM  
    [Adventure Works]  
WHERE  
    ([Measures].[Freight Per Customer])
```

Dividing a non-zero or non-null value by zero or null will return the value Infinity, which is displayed in query results as the value "1.#INF". In most cases, you should check for division by zero to avoid this situation. The following example shows you how:

```
//Returns 1.#INF when Internet Sales Amount is zero or null  
Member [Measures].[Reseller to Internet Ratio] AS  
    [Measures].[Reseller Sales Amount]  
    /  
    [Measures].[Internet Sales Amount]  
//Traps the division by zero scenario and returns null instead of 1.#INF  
Member [Measures].[Reseller to Internet Ratio With Error Handling] AS  
    IIF([Measures].[Internet Sales Amount]=0, NULL,  
    [Measures].[Reseller Sales Amount]  
    /  
    [Measures].[Internet Sales Amount])  
  
SELECT  
    {[Measures].[Reseller to Internet Ratio],[Measures].[Reseller to Internet  
Ratio With Error Handling]} ON 0,
```

```
[Product].[Category].[Category].Members ON 1
FROM
[Adventure Works]
WHERE ([Date].[Calendar].[Calendar Year].&[2001])
```

See Also

[If \(MDX\)](#)

[MDX Operator Reference \(MDX\)](#)

^ (Power)

Performs an arithmetic operation that raises one number by another number.

Syntax

Numeric_Expression ^ Numeric_Expression

Parameters

Parameter	Description
Numeric_Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If one expression evaluates to a null value, the operator returns a null value.

See Also

[MDX Operator Reference \(MDX\)](#)

`/*...*/` (Comment)

Indicates comment text that is provided by the user.

Syntax

```
/* Comment_Text */
```

Parameters

Parameter	Description
Comment_Text	The string that contains the text of the comment.

Remarks

The server does not evaluate the text between the comment characters, `/*` and `*/`. Comments can be inserted on a separate line or within a Multidimensional Expressions (MDX) statement. Multiple-line comments must be indicated by `/*` and `*/`.

There is no maximum length for comments. Comments can be nested; for example, `/* Test /*Comment*/ Text*/` is an example of a nested comment.

Examples

The following example demonstrates the use of this operator.

```
/* This member returns the gross profit margin for product types
   and reseller types crossjoined by year. */
SELECT
    [Date].[Calendar].[Calendar Year].Members *
    [Reseller].[Reseller Type].Children ON 0,
    [Product].[Category].[Category].Members ON 1
FROM /* Select from the Adventure Works cube. */
    [Adventure Works]
WHERE
    [Measures].[Gross Profit Margin]
```

See Also

[// \(Comment\) \(MDX\)](#)

[-- \(Comment\) \(MDX\)](#)

[MDX Operator Reference \(MDX\)](#)

// (Comment)

Indicates user-provided text.

Syntax

```
// Comment_Text
```

Parameters

Parameter	Description
Comment_Text	The string that contains the text of the comment.

Remarks

Comments can be inserted on a separate line, nested at the end of a Multidimensional Expressions (MDX) script line, or nested within an MDX statement. The server does not evaluate the comment.

Use // for single-line comments only. Comments inserted with // are delimited by the newline character.

There is no maximum length for comments.

Examples

The following example demonstrates the use of this operator.

```
// This member returns the gross profit margin for product types  
// and reseller types crossjoined by year.
```

```
SELECT  
    [Date].[Calendar].[Calendar Year].Members *  
    [Reseller].[Reseller Type].Children ON 0,  
    [Product].[Category].[Category].Members ON 1
```

```
FROM // Select from the Adventure Works cube.  
    [Adventure Works]  
WHERE  
    [Measures].[Gross Profit Margin]
```

See Also

[/*...*/ \(Comment\) \(MDX\)](#)

[-- \(Comment\) \(MDX\)](#)

[MDX Operator Reference \(MDX\)](#)

: (Range)

Performs a set operation that returns a naturally ordered set, with the two specified members as endpoints, and all members between the two specified members included as members of the set.

Syntax

```
Member_Expression : Member_Expression
```

Parameters

Parameter	Description
Member_Expression	A valid Multidimensional Expressions (MDX) expression that returns a member.

Return Value

A set that contains the specified members and all members between the specified members.

Remarks

Both parameters must specify members within the same level and hierarchy of a given dimension. If both parameters specify the same member, the **: (Range)** operator returns a set that contains just the specified member. If the first parameter is Null, then the set contains all members from the beginning of the level of the member specified in the second parameter, up to and including that member. If the second parameter is Null, then the set contains all

members from the member specified in the first parameter, up to and including the last member on the same level.

This set operator has no functional equivalent in MDX.

Examples

The following example demonstrates the use of this operator.

```
-- This query returns the freight cost per user  
-- for products, averaged by month, for the first quarter.
```

With Member [Measures].[Freight Per Customer] as

```
(  
    [Measures].[Internet Freight Cost]  
    /  
    [Measures].[Customer Count]  
)  
  
SELECT  
    {[Ship Date].[Calendar].[Month].&[2004]&[1] : [Ship  
Date].[Calendar].[Month].&[2004]&[3]} ON 0,  
    [Product].[Category].[Category].Members ON 1  
FROM  
    [Adventure Works]  
WHERE  
    ([Measures].[Freight Per Customer])
```

See Also

[MDX Operator Reference \(MDX\)](#)

+ (Add)

Performs an arithmetic operation that adds two numbers.

Syntax

Numeric_Expression + Numeric_Expression

Parameters

Parameter	Description
Numeric Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If one expression evaluates to a null value, the operator returns the result of the other expression.

See Also

[MDX Operator Reference \(MDX\)](#)

+ (Positive)

Performs a unary operation that returns the positive value of a numeric expression.

Syntax

```
+ Numeric_Expression
```

Parameters

Parameter	Description
Numeric_Expression	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A positive value that has the data type of the specified parameter.

See Also

[MDX Operator Reference \(MDX\)](#)

+ (String Concatenation)

Performs a string operation that concatenates two or more character strings, tuples, or a combination of strings and tuples.

Syntax

```
String_Expression + String_Expression
```

Parameters

Parameter	Description
String_Expression	A valid Multidimensional Expressions (MDX) expression that returns a string value.

Return Value

A value with the data type of the parameter that has the higher precedence.

Remarks

Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression. If one expression evaluates to a null value, the operator returns the result of the other expression.

See Also

[MDX Operator Reference \(MDX\)](#)

+ (Union)

Performs a set operation that returns a union of two sets, removing duplicate members.

Syntax

Set_Expression + Set_Expression

Parameters

Parameter	Description
Set_Expression	A valid Multidimensional Expressions (MDX) expression that returns a set.

Return Value

A set that contains the members of both specified sets.

Remarks

The + (**Union**) operator is functionally equivalent to the [Union \(MDX\)](#) function.

Examples

The following example demonstrates the use of this operator.

```
-- This member returns the gross profit margin for each year for North American countries.
```

```
SELECT
    [Date].[Calendar].[Calendar Year].Members ON 0,
    {[Sales Territory].[Sales Territory].[Country].[United States]} +
    {[Sales Territory].[Sales Territory].[Country].[Canada]} ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[Gross Profit Margin])
```

See Also

[MDX Operator Reference \(MDX\)](#)

< (Less Than)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is less than the value of another MDX expression.

Syntax

```
MDX_Expression < MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if both parameters are non-null, and the first parameter has a value that is lower than the value of the second parameter.
- **false** if both parameters are non-null, and the first parameter has a value that is either equal to or greater than the value of the second parameter.
- null if either or both parameters evaluate to a null value.

Examples

The following example demonstrates the use of this operator.

```
-- This query returns the gross profit margin (GPM)  
-- for clothing sales where the GPM is less than 30%.
```

```
With Member [Measures].[LowGPM] as
```

```
    IIF(  
        [Measures].[Gross Profit Margin] < .3,  
        [Measures].[Gross Profit Margin],  
        null)
```

```
SELECT NON EMPTY
```

```
    [Sales Territory].[Sales Territory Country].Members ON 0,  
    [Product].[Category].[Clothing] ON 1
```

```
FROM
    [Adventure Works]
WHERE
    ([Measures].[LowGPM])
```

See Also

[MDX Operator Reference \(MDX\)](#)

<= (Less Than or Equal To)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is less than or equal to the value of another MDX expression.

Syntax

```
MDX_Expression <= MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if both parameters are non-null, and the first parameter has a value that is either less than or equal to the value of the second parameter.
- **false** if both parameters are non-null, and the first parameter has a value that greater than the value of the second parameter.
- null if either or both parameters evaluate to a null value.

Examples

The following example demonstrates the use of this operator.

```
-- This query returns the gross profit margin (GPM)
-- for Australia where the GPM is less than or equal to 30%.
```

With Member [Measures].[LowGPM] as

```

IIF(
    [Measures].[Gross Profit Margin] <= .5,
    [Measures].[Gross Profit Margin],
    null)
SELECT
NON EMPTY [Sales Territory].[Sales Territory Country].[Australia] ON 0,
    NON EMPTY [Product].[Category].Members ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[LowGPM])

```

See Also

[MDX Operator Reference \(MDX\)](#)

<> (Not Equal To)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is not equal to the value of another MDX expression.

Syntax

```
MDX_Expression <> MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if both parameters are non-null, and the first parameter is not equal to the second parameter.
- **false** if both parameters are non-null, and the first parameter is equal to the second parameter.

- null if either or both parameters evaluate to a null value.

See Also

[MDX Operator Reference \(MDX\)](#)

= (Equal To)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is equal to the value of another MDX expression.

Note

To compare objects, use the [IS \(MDX\)](#) operator. For example, use the IS operator when you are checking if the current member on a query axis is a specific member.

Syntax

```
MDX_Expression = MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if the value of the first parameter is equal to the value of the second parameter.
- **false** if the value of the first parameter is not equal to the value of the second parameter.
- **true** if both parameters are null, or one parameter is null and the other parameter is 0.

Examples

The following query shows examples of these conditions:

```
With
--Returns true
Member [Measures].bool1 as 1=1
--Returns false
Member [Measures].bool2 as 1=0
```



```

--Returns true
Member [Measures].bool3 as null=null
--Returns true
Member [Measures].bool4 as 0=null
--Returns false
Member [Measures].bool5 as 1=null
Select
{[Measures].bool1,[Measures].bool2,[Measures].bool3,[Measures].bool4,[Measures].bool5}
On 0
From [Adventure Works]

```

See Also

[MDX Operator Reference \(MDX\)](#)

> (Greater Than)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is greater than the value of another MDX expression.

Syntax

```
MDX_Expression > MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if both parameters are non-null, and the first parameter has a value that is greater than the value of the second parameter.
- **false** if both parameters are non-null, and the first parameter has a value that is either equal to or lower than the value of the second parameter.
- null if either or both parameters evaluate to a null value.

Examples

The following example query demonstrates the use of this operator.

```
-- This query returns the gross profit margin (GPM)
-- for Australia where the GPM is more than 50%.
With Member [Measures].[HighGPM] as
    IIF(
        [Measures].[Gross Profit Margin] > .5,
        [Measures].[Gross Profit Margin],
        null)
SELECT
NON EMPTY [Sales Territory].[Sales Territory Country].[Australia] ON 0,
    NON EMPTY [Product].[Category].Members ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[HighGPM])
```

See Also

[MDX Operator Reference \(MDX\)](#)

>= (Greater Than or Equal To)

Performs a comparison operation that determines whether the value of one Multidimensional Expressions (MDX) expression is greater than or equal to the value of another MDX expression.

Syntax

```
MDX_Expression >= MDX_Expression
```

Parameters

Parameter	Description
MDX_Expression	A valid MDX expression.

Return Value

A Boolean value based on the following conditions:

- **true** if the first parameter has a value that is either greater than or equal to the value of the second parameter.
- **false** if the first parameter has a value that is lower than the value of the second parameter.
- **true** if both parameters are null or if one parameter is null and the other parameter is 0.

Examples

The following example demonstrates the use of this operator.

```
-- This query returns the gross profit margin (GPM)
-- for Australia where the GPM is greater than or equal to 50%.
```

With Member [Measures].[HighGPM] as

```
IIF(
    [Measures].[Gross Profit Margin] >= .5,
    [Measures].[Gross Profit Margin],
    null)
SELECT
    NON EMPTY [Sales Territory].[Sales Territory Country].[Australia] ON 0,
    NON EMPTY [Product].[Category].Members ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[HighGPM])
```

See Also

[MDX Operator Reference \(MDX\)](#)

AND

Performs a logical conjunction on two numeric expressions.

Syntax

Expression1 **AND** Expression2

Parameters

Parameter	Description
Expression1	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.
Expression2	A valid MDX expression that returns a numeric value.

Return Value

A Boolean value that returns true if both parameters evaluate to **true**; otherwise, **false**.

Remarks

The **AND** operator treats both expressions as Boolean values (zero, 0, as **false**; otherwise, **true**) before the operator performs the logical conjunction. The following table illustrates how the **AND** operator performs the logical conjunction.

Expression1	Expression2	Return Value
true	true	true
true	false	false
false	true	false
false	false	false

Example

Code

```
-- This query returns the gross profit margin (GPM)
-- for clothing sales where the GPM is between 20% and 30%.
With Member [Measures].[LowGPM] as
    IIF(
        [Measures].[Gross Profit Margin] <= .3 AND
        [Measures].[Gross Profit Margin] >= .2,
```

```

    [Measures].[Gross Profit Margin],
    null)
SELECT NON EMPTY
    [Sales Territory].[Sales Territory Country].Members ON 0,
    [Product].[Category].[Clothing] ON 1
FROM
    [Adventure Works]
WHERE
    ([Measures].[LowGPM])

```

See Also

[MDX Operator Reference \(MDX\)](#)

IS

Performs a logical comparison on two object expressions.

Syntax

Expression1 IS (Expression2 | NULL)

Parameters

Parameter	Description
Expression1	A valid Multidimensional Expressions (MDX) expression that returns an MDX object reference.
Expression2	A valid MDX expression that returns an MDX object reference.

Return Value

A Boolean value that returns **true** if both arguments refer to the same object; otherwise, **false**. If the **NULL** keyword is specified, the operator returns **true** if Expression1 is **null**; otherwise, **false**.

Remarks

The **IS** operator is often used to determine whether tuples and members are idempotent, meaning that they are exactly equivalent.

Examples

The following example shows how to use the **IS** operator to check if the current member on an axis is a specific member:

With

```
//Returns TRUE if the currentmember is Bikes
Member [Measures].[IsBikes?] AS
[Product].[Category].CurrentMember IS [Product].[Category].&[1]
SELECT
{[Measures].[IsBikes?]} ON 0,
    [Product].[Category].[Category].Members ON 1
FROM
    [Adventure Works]
```

See Also

[MDX Operator Reference \(MDX\)](#)

NOT

Performs a logical negation on a numeric expression.

Syntax

NOT Expression1

Parameters

Parameter	Description
Expression1	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

Return Value

A Boolean value that returns **false** if the argument evaluates to **true**; otherwise, **true**.

Remarks

The **NOT** operator treats the expression as a Boolean value (zero, 0, as **false**; otherwise, **true**) before the operator performs the logical negation. The following table illustrates how the **NOT** operator performs the logical negation.

Expression1	Return Value
true	false
false	true

See Also

[MDX Operator Reference \(MDX\)](#)

OR

Performs a logical disjunction on two numeric expressions.

Syntax

Expression1 **OR** Expression2

Parameters

Parameter

Description

A valid Multidimensional Expressions (MDX) expression that returns a numeric value.

A valid MDX expression that returns a numeric value.

Return Value

A Boolean value that returns **true** if either or both arguments evaluate to **true**; otherwise, **false**.

Remarks

The **OR** operator treats both arguments as Boolean values (zero, 0, as **false**; otherwise, **true**) before the operator performs the logical disjunction. The following table illustrates how the **OR** operator performs the logical disjunction.

Expression1	Expression2	Return Value
true	true	true
true	false	true
false	true	true
false	false	false

Example

The following query contains a calculated measure that returns the string "MARRIED OR MALE" if the current member on the Gender hierarchy of the Customer dimension is Male or the current member on the Marital Status hierarchy of the Customer dimension is Married; otherwise it returns the string "UNMARRIED OR FEMALE".

```
WITH
MEMBER MEASURES.ORDEMO AS
IIF(
([Customer].[Gender].CURRENTMEMBER IS [Customer].[Gender].&[M])
OR
([Customer].[Marital Status].CURRENTMEMBER IS [Customer].[Marital
Status].&[M]),
"MARRIED OR MALE",
"UNMARRIED OR FEMALE")
SELECT [Customer].[Gender].[Gender].MEMBERS ON 0,
[Customer].[Marital Status].[Marital Status].MEMBERS ON 1
FROM [Adventure Works]
WHERE (MEASURES.ORDEMO)
```


See Also

[MDX Operator Reference \(MDX\)](#)

XOR

Performs a logical exclusion on two numeric expressions.

Syntax

Expression1 XOR Expression2

Parameters

Parameter	Description
Expression1	A valid Multidimensional Expressions (MDX) expression that returns a numeric value.
Expression2	A valid MDX expression that returns a numeric value.

Return Value

A Boolean value that returns **true** if one and only one argument evaluates to **true**; otherwise, **false**.

Remarks

The **XOR** operator treats both parameters as Boolean values (zero, 0, as **false**; otherwise, **true**) before the operator performs the logical exclusion. The following table illustrates how the **XOR** operator performs the logical exclusion.

Expression1	Expression2	Return Value
true	true	false
true	false	true
false	true	true

Expression1	Expression2	Return Value
false	false	false

See Also

[MDX Operator Reference \(MDX\)](#)

MDX Function Reference

Microsoft SQL Server Analysis Services provides for the use of functions in Multidimensional Expressions (MDX) syntax. Functions can be used in any valid MDX statement, and are frequently used in queries, custom rollup definitions, and other calculations. This section provides information about the MDX functions included with Analysis Services.

You can use the following tables to find functions by their category of return value, or you can select a function by name from the alphabetical list in the table of contents.

Array Functions

Function	Description
MDX Language Reference (MDX)	Converts one or more sets to an array for use in a user-defined function.

Hierarchy Functions

Function	Description
Hierarchy	Returns the hierarchy that contains a specified member or level.
Dimension	Returns the dimension that contains a specified member, level, or hierarchy.
Dimensions	Returns a hierarchy specified by a numeric or string expression.

Level Functions

Function	Description
Level	Returns the level of a member.
Levels	Returns the level whose position in a dimension or hierarchy is specified by a numeric expression or whose name is specified by a string expression.

Logical Functions

Function	Description
IsAncestor	Returns whether a specified member is an ancestor of another specified member.
IsEmpty	Returns whether the evaluated expression is the empty cell value.
IsGeneration	Returns whether a specified member is in a specified generation.
IsLeaf	Returns whether a specified member is a leaf member.
IsSibling	Returns whether a specified member is a sibling of another specified member.

Member Functions

Function	Description
Ancestor	Returns the ancestor of a member at a specified level or distance.
ClosingPeriod	Returns the last sibling among the descendants of a member at a specified level.

Function	Description
Cousin	Returns the child member with the same relative position under a parent member as the specified child member.
CurrentMember	Returns the current member along a specified dimension or hierarchy during iteration.
DataMember	Returns the system-generated data member that is associated with a nonleaf member of a dimension.
DefaultMember	Returns the default member of a dimension or hierarchy.
FirstChild	Returns the first child of a member.
FirstSibling	Returns the first child of the parent of a member.
Item (Member)	Returns a member from a specified tuple.
Lag	Returns the member that is a specified number of positions before a specified member along the member's dimension.
LastChild	Returns the last child of a specified member.
LastSibling	Returns the last child of the parent of a specified member.
Lead	Returns the member that is a specified number of positions following a specified member along the member's dimension.
LinkMember	Returns the member equivalent to a specified member in a specified hierarchy.
Members (Member)	Returns a member specified by a string expression.
NextMember	Returns the next member in the level that contains a specified member.
OpeningPeriod	Returns the first sibling among the descendants of a specified level, optionally at a specified member.

Function	Description
ParallelPeriod	Returns a member from a prior period in the same relative position as a specified member.
Parent	Returns the parent of a member.
PrevMember	Returns the previous member in the level that contains a specified member.
StrToMember	Returns the member specified by an MDX-formatted string.
UnknownMember (MDX)	Returns the unknown member associated with a level or member.
ValidMeasure	Returns a valid measure in a virtual cube by forcing inapplicable dimensions to their top level.

Numeric Functions

Function	Description
Aggregate	Returns a scalar value calculated by aggregating either measures or an optionally specified numeric expression over the tuples of a specified set.
Avg (MDX)	Returns the average value of measures or the average value of an optional numeric expression, evaluated over a specified set.
CalculationCurrentPass	Returns the current calculation pass of a cube for the specified query context.
CalculationPassValue	Returns the value of a MDX expression evaluated over the specified calculation pass of a cube.
CoalesceEmpty	Coalesces an empty cell value to a number or string and returns the coalesced value.
Correlation	Returns the correlation coefficient of two series evaluated over a set.

Function	Description
Count (Dimension)	Returns the number of dimensions in a cube.
Count (Level)	Returns the number of levels in a dimension or hierarchy.
Count (Set)	Returns the number of cells in a set.
Count (Tuple)	Returns the number of dimensions in a tuple.
Covariance	Returns the population covariance of two series evaluated over a set, using the biased population formula.
CovarianceN	Returns the sample covariance of two series evaluated over a set, using the unbiased population formula.
DistinctCount	Returns the number of distinct, nonempty tuples in a set.
IIf	Returns one of two values determined by a logical test.
LinRegIntercept	Calculates the linear regression of a set and returns the value of the intercept in the regression line, .
LinRegPoint	Calculates the linear regression of a set and returns the value of y in the regression line, .
LinRegR2	Calculates the linear regression of a set and returns the coefficient of determination, R2.
LinRegSlope	Calculates the linear regression of a set, and returns the value of the slope in the regression line, .
LinRegVariance	Calculates the linear regression of a set, and returns the variance associated with the regression line, .
LookupCube	Returns the value of an MDX expression evaluated over another specified cube in the same database.

Function	Description
Max (MDX)	Returns the maximum value of a numeric expression that is evaluated over a set.
Median	Returns the median value of a numeric expression that is evaluated over a set.
Min (MDX)	Returns the minimum value of a numeric expression that is evaluated over a set.
Ordinal	Returns the zero-based ordinal value associated with a level.
Predict	Returns a value of a numeric expression evaluated over a data mining model.
Rank	Returns the one-based rank of a specified tuple in a specified set.
RollupChildren	Returns a value generated by rolling up the values of the children of a specified member using the specified unary operator.
Stddev	Alias for Stdev .
StddevP	Alias for StdevP .
Stdev	Returns the sample standard deviation of a numeric expression evaluated over a set, using the unbiased population formula.
StdevP	Returns the population standard deviation of a numeric expression evaluated over a set, using the biased population formula.
StrToValue	Returns the value specified by an MDX-formatted string.
Sum (MDX)	Returns the sum of a numeric expression evaluated over a set.
Value	Returns the value of a measure.
Var	Returns the sample variance of a numeric expression evaluated over a set, using the unbiased population formula.
Variance	Alias for Var .
VarianceP	Alias for VarP .

Function	Description
VarP	Returns the population variance of a numeric expression evaluated over a set, using the biased population formula.

Set Functions

Function	Description
AddCalculatedMembers	Returns a set generated by adding calculated members to a specified set.
AllMembers	Returns a set that contains all members, including calculated members, of the specified dimension, hierarchy, or level.
Ancestors	Returns a set of all ancestors of a member at a specified level or distance.
Ascendants	Returns the set of the ascendants of a specified member, including the member itself.
Axis	Returns a set defined in an axis.
BottomCount	Sorts a set in ascending order, and returns the specified number of tuples with the lowest values.
BottomPercent	Sorts a set in ascending order, and returns a set of tuples with the lowest values whose cumulative total is equal to or less than a specified percentage.
BottomSum	Sorts a set in ascending order, and returns a set of tuples with the lowest values whose total is equal to or less than a specified value.
Children	Returns the children of a specified member.
Crossjoin	Returns the cross product of one or more sets.
CurrentOrdinal (MDX)	Returns the current iteration number within

Function	Description
	a set during iteration.
Descendants	Returns the set of descendants of a member at a specified level or distance, optionally including or excluding descendants in other levels.
Distinct	Returns a set, removing duplicate tuples from a specified set.
DrilldownLevel	Drills down the members of a set to one level below the lowest level represented in the set, or to one level below an optionally specified level of a member represented in the set.
DrilldownLevelBottom	Drills down the bottommost members of a set, at a specified level, to one level below.
DrilldownLevelTop	Drills down the topmost members of a set, at a specified level, to one level below.
DrilldownMember	Drills down the members in a specified set that are present in a second specified set. Alternatively, the function drills down on a set of tuples.
DrilldownMemberBottom	Drills down the members in a specified set that are present in a second specified set, limiting the result set to a specified number of members. Alternatively, this function also drills down on a set of tuples.
DrilldownMemberTop	Drills down the members in a specified set that are present in a second specified set, limiting the result set to a specified number of members. Alternatively, this function drills down on a set of tuples.
DrillupLevel	Drills up the members of a set that are below a specified level.
DrillupMember	Drills up the members in a specified set that are present in a second specified set.
Except	Finds the difference between two sets, optionally retaining duplicates.

Function	Description
Exists	Returns the set of members of one set that exist with one or more tuples of one or more other sets.
Extract	Returns a set of tuples from extracted dimension elements.
Filter	Returns the set that results from filtering a specified set based on a search condition.
Generate	Applies a set to each member of another set, and then joins the resulting sets by union. Alternatively, this function returns a concatenated string created by evaluating a string expression over a set.
Head	Returns the first specified number of elements in a set, while retaining duplicates.
Hierarchize	Orders the members of a set in a hierarchy.
Intersect	Returns the intersection of two input sets, optionally retaining duplicates.
LastPeriods	Returns a set of members up to and including a specified member.
Members (Set)	Returns the set of members in a dimension, level, or hierarchy.
Mtd	Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level in the Time dimension.
NameToSet	Returns a set that contains the member specified by an MDX-formatted string.
NonEmptyCrossjoin	Returns the cross product of one or more sets as a set, excluding empty tuples and tuples without associated fact table data.
Order	Arranges members of a specified set, optionally preserving or breaking the hierarchy.

Function	Description
PeriodsToDate	Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by a specified level in the Time dimension.
Qtd	Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Quarter level in the Time dimension.
Siblings	Returns the siblings of a specified member, including the member itself.
StripCalculatedMembers	Returns a set generated by removing calculated members from a specified set.
StrToSet	Returns the set specified by an MDX-formatted string.
Subset	Returns a subset of tuples from a specified set.
Tail	Returns a subset from the end of a set.
ToggleDrillState	Toggles the drill state of members.
TopCount	Sorts a set in descending order and returns the specified number of elements with the highest values.
TopPercent	Sorts a set in descending order, and returns a set of tuples with the highest values whose cumulative total is equal to or less than a specified percentage.
TopSum	Sorts a set and returns the topmost elements whose cumulative total is at least a specified value.
Union (MDX)	Returns the union of two sets, optionally retaining duplicates.
Unorder (MDX)	Removes any enforced ordering from a specified set.

Function	Description
VisualTotals	Returns a set generated by dynamically totaling child members in a specified set, optionally using a pattern for the name of the parent member in the resulting cellset.
Wtd	Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Week level in the Time dimension.
Ytd	Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level in the Time dimension.

String Functions

Function	Description
CalculationPassValue	Returns the value of an MDX expression evaluated over the specified calculation pass of a cube.
CoalesceEmpty	Coalesces an empty cell value to a number or string and returns the coalesced value.
Generate	Applies a set to each member of another set, and then joins the resulting sets by union. Alternatively, this function returns a concatenated string created by evaluating a string expression over a set.
Iif	Returns one of two values determined by a logical test.
LookupCube	Returns the value of an MDX expression evaluated over another specified cube in the same database.
MemberToStr	Returns an MDX-formatted string that

Function	Description
	corresponds to a specified member.
Name	Returns the name of a dimension, hierarchy, level, or member.
Properties	Returns a string, or a strongly-typed value, that contains a member property value.
SetToStr	Returns an MDX-formatted string of that corresponds to a specified set.
TupleToStr	Returns an MDX-formatted string that corresponds to specified tuple.
UniqueName	Returns the unique name of a specified dimension, hierarchy, level, or member.
UserName	Returns the domain name and user name of the current connection.

Subcube Functions

Function	Description
This	Returns the current subcube.
Leaves	Returns the set of leaf members in the specified dimension, member, or tuple.

Tuple Functions

Function	Description
Current	Returns the current tuple from a set during iteration.
Item (Tuple)	Returns a tuple from a set.
Root	Returns a tuple that consists of the All members from each attribute hierarchy in a cube, dimension, or tuple.

Function	Description
StrToTuple	Returns the tuple specified by an MDX-formatted string.

Other Functions

Function	Description
Error (MDX)	Raises an error, optionally providing a specified error message.

See Also

[MDX Language Reference \(MDX\)](#)

AddCalculatedMembers

Returns a set generated by adding calculated members to a specified set.

Syntax

AddCalculatedMembers(**Set_Expression**)

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

By default, MDX excludes calculated members when it resolves set functions. The **AddCalculatedMembers** function examines the set expression specified in Set_Expression, and includes calculated members that are siblings of the members contained within the scope of that set expression.

Note

This function can be used only with one-dimensional set expressions.

Examples

The following example demonstrates the use of this function.

```
-- This query returns the calculated members for the cube
-- by retrieving all members, and then excluding non-calculated members.
```

```
SELECT
    AddCalculatedMembers(
        { [Measures].Members }
        )-[Measures].Members ON COLUMNS
FROM [Adventure Works]
```

The following example returns the `Measures.[Unit Price]` member, in addition to all the calculated members in the **Measures** dimension, from the **Adventure Works** cube.

```
SELECT
    AddCalculatedMembers({Measures.[Unit Price]}) ON COLUMNS
FROM
    [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Aggregate

Returns a number that is calculated by aggregating over the cells returned by the set expression. If a numeric expression is not provided, this function aggregates each measure within the current query context by using the default aggregation operator that is specified for each measure. If a numeric expression is provided, this function first evaluates, and then sums, the numeric expression for each cell in the specified set.

Syntax

```
Aggregate(Set_Expression [ ,Numeric_Expression ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression

of cell coordinates that return a number.

Remarks

If a set of empty tuples or an empty set is specified, this function returns an empty value.

The following table describes how the **Aggregate** function behaves with different aggregation functions.

Aggregation Operator	Result
Sum	Returns the sum of values over the set.
Count	Returns the count of values over the set.
Max	Returns the maximum value over the set.
Min	Returns the minimum value over the set.
Semi-additive aggregation functions	Returns the calculation of semi-additive behavior over the set after projecting the shape to the time axis.
Distinct Count	<p>Aggregates across the fact data contributing to the subcube when the slicer axis includes a set.</p> <p>Returns the distinct count for each member of the set. The result depends on the security on the cells being aggregated, and not on the security on the cells that are required for the computation. Cell security on the set generates an error; cell security below the granularity of the specified set is ignored. Calculations on the set generate an error. Calculations below granularity of the set are ignored. Distinct count over a set that includes a member and one or more of its children returns the distinct count across facts contributing to the child member.</p>
Attributes that cannot be aggregated	Returns the sum of the values.
Mixed aggregation functions	Not supported, and raises an error.
Unary operators	Not respected; values are aggregated by summing.

Aggregation Operator	Result
Calculated measures	Solve order set to ensure calculated measure applies.
Calculated members	Normal rules apply, that is, the last solve order takes precedence.
Assignments	Assignments aggregate according to the measure aggregation function. If the measure aggregation function is distinct count, the assignment is summed.

Examples

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first eight months of calendar year 2003 that are contained in the `Date` dimension, from the **Adventure Works** cube.

```
WITH MEMBER [Date].[Calendar].[First8Months2003] AS
    Aggregate (
        PeriodsToDate (
            [Date].[Calendar].[Calendar Year],
            [Date].[Calendar].[Month].[August 2003]
        )
    )
SELECT
    [Date].[Calendar].[First8Months2003] ON COLUMNS,
    [Product].[Category].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    [Measures].[Order Quantity]
```

The following example aggregates over the first two months of the second semester of calendar year 2003.

```
WITH MEMBER [Date].[Calendar].[First2MonthsSecondSemester2003] AS
    Aggregate (
        PeriodsToDate (
            [Date].[Calendar].[Calendar Semester],
```

```

        [Date].[Calendar].[Month].[August 2003]
    )
)
SELECT
    [Date].[Calendar].[First2MonthsSecondSemester2003] ON COLUMNS,
    [Product].[Category].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    [Measures].[Order Quantity]

```

The following example returns the count of the resellers whose sales have declined over the previous time period, based on user-selected State-Province member values evaluated using the Aggregate function. The **Hierarchize** and **DrillDownLevel** functions are used to return values for declining sales for product categories in the Product dimension.

```

WITH MEMBER Measures.[Declining Reseller Sales] AS
    Count (
        Filter (
            Existing (Reseller.Reseller.Reseller),
            [Measures].[Reseller Sales Amount] < ([Measures].[Reseller Sales
Amount]),
            [Date].Calendar.PrevMember)
        )
    )
MEMBER [Geography].[State-Province].x AS
    Aggregate (
        {[Geography].[State-Province].&[WA]&[US]},
        [Geography].[State-Province].&[OR]&[US] }
    )
SELECT NON EMPTY Hierarchize (
    AddCalculatedMembers (
        {DrillDownLevel ({[Product].[All Products]})})
    )
)
    DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS

```

```
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
      [Date].[Calendar].[Calendar Quarter].&[2003]&[4],
      [Measures].[Declining Reseller Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

[Children \(MDX\)](#)

[Hierarchize \(MDX\)](#)

[Count \(Set\) \(MDX\)](#)

[Filter \(MDX\)](#)

[AddCalculatedMembers \(MDX\)](#)

[DrilldownLevel \(MDX\)](#)

[Properties \(MDX\)](#)

[PrevMember \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

AllMembers

Evaluates either a hierarchy or a level expression and returns a set that contains all members of the specified hierarchy or level, which includes all calculated members in the hierarchy or level.

Syntax

Hierarchy syntax

```
Hierarchy_Expression.AllMembers
```

Level syntax

```
Level_Expression.AllMembers
```

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Remarks

The **AllMembers** function returns a set that contains all members, which includes calculated members, in the specified hierarchy or level. The **AllMembers** function returns the calculated members even if the specified hierarchy or level contains no visible members.

Important

When a dimension contains only a single visible hierarchy, the hierarchy can be either referred to by the dimension name or by the hierarchy name, because the dimension name in this case is resolved to its only visible hierarchy. For example, `Measures.AllMembers` is a valid MDX expression because it resolves to the only hierarchy in the Measures dimension.

Note

The **AllMembers** function is semantically similar to the `AddCalculatedMembers (MDX)` function.

Examples

The following example returns all members in the `[Date].[Calendar Year]` attribute hierarchy on the column axis, this includes calculated members, and the set of all children of the `[Product].[Model Name]` attribute hierarchy on the row axis from the **Adventure Works** cube.

```
SELECT
    [Date].[Calendar Year].AllMembers ON COLUMNS,
    [Product].[Model Name].Children ON ROWS
FROM
    [Adventure Works]
```

The following example returns all members in the **Measures** dimension on the column axis, this includes all calculated members, and the set of all children of the `[Product].[Model Name]` attribute hierarchy on the row axis from the **Adventure Works** cube.

```
SELECT
    Measures.AllMembers ON COLUMNS,
    [Product].[Model Name].Children ON ROWS
FROM
    [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Children \(MDX\)](#)

Ancestor

A function that returns the ancestor of a specified member at a specified level or at a specified distance from the member.

Syntax

Level syntax

Ancestor(**Member_Expression**, **Level_Expression**)

Numeric syntax

Ancestor(**Member_Expression**, **Distance**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Distance

A valid numeric expression that specifies the distance from the specified member.

Remarks

With the **Ancestor** function, you provide the function with an MDX member expression and then provide either an MDX expression of a level that is an ancestor of the member or a numeric expression that represents the number of levels above that member. With this information, the **Ancestors** function returns the ancestor member at that level.

Note

To return a set containing the ancestor member, instead of just the ancestor member, use the [MDX Function Reference \(MDX\)](#) function.

If a level expression is specified, the **Ancestor** function returns the ancestor of specified member at the specified level. If the specified member is not within the same hierarchy as specified level, the function returns an error.

If a distance is specified, the **Ancestor** function returns the ancestor of the specified member that is the number of steps specified up in the hierarchy specified by the member expression. A member may be specified as a member of an attribute hierarchy, a user-defined hierarchy, or in

some cases, a parent-child hierarchy. A number of 1 returns a member's parent and a number of 2 returns a member's grandparent (if one exists). A number of 0 returns the member itself.

Note

Use this form of the **Ancestor** function for cases in which the level of the parent is unknown or cannot be named.

Examples

The following example uses a level expression and returns the Internet Sales Amount for each State-Province in Australia and its percent of the total Internet Sales Amount for Australia.

```
WITH MEMBER Measures.x AS [Measures].[Internet Sales Amount] /
(
  [Measures].[Internet Sales Amount],
  Ancestor
    (
      [Customer].[Customer Geography].CurrentMember,
      [Customer].[Customer Geography].[Country]
    )
), FORMAT_STRING = '0%'
SELECT {[Measures].[Internet Sales Amount], Measures.x} ON 0,
{
  Descendants
    (
      [Customer].[Customer Geography].[Country].&[Australia],
      [Customer].[Customer Geography].[State-Province], SELF
    )
} ON 1
FROM [Adventure Works]
```

The following example uses a numeric expression and returns the Internet Sales Amount for each State-Province in Australia and its percent of the total Internet Sales Amount for all countries.

```
WITH MEMBER Measures.x AS [Measures].[Internet Sales Amount] /
(
  [Measures].[Internet Sales Amount],
  Ancestor
    ([Customer].[Customer Geography].CurrentMember, 2)
)
```

```

    ), FORMAT_STRING = '0%'
SELECT {[Measures].[Internet Sales Amount], Measures.x} ON 0,
{
    Descendants
        (
            [Customer].[Customer Geography].[Country].&[Australia],
            [Customer].[Customer Geography].[State-Province], SELF
        )
} ON 1
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

Ancestors

A function that returns the set of all ancestors of a specified member at a specified level or at a specified distance from the member. With Microsoft SQL Server Analysis Services, the set returned will always consist of a single member - Analysis Services does not support multiple parents for a single member.

Syntax

Level syntax

Ancestors(**Member_Expression**, **Level_Expression**)

Numeric syntax

Ancestors(**Member_Expression**, **Distance**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Distance

A valid numeric expression that specifies the distance from the specified member.

Remarks

With the **Ancestors** function, you provide the function with an MDX member expression and then provide either an MDX expression of a level that is an ancestor of that member or a numeric expression that represents the number of levels above that member. With this information, the **Ancestors** function returns the set of members (which will be a set consisting of one member) at that level.

Note

To return an ancestor member, rather than an ancestor set, use the Ancestor function.

If a level expression is specified, the **Ancestors** function returns the set of all ancestors of the specified member at the specified level. If the specified member is not within the same hierarchy as the specified level, the function returns an error.

If a distance is specified, the **Ancestors** function returns the set of all members that are the number of steps specified up in the hierarchy specified by the member expression. A member may be specified as a member of an attribute hierarchy, a user-defined hierarchy, or, in some cases, a parent-child hierarchy. A number of 1 returns the set of members at the parent level and a number of 2 returns the set of members at the grandparent level (if one exists). A number of 0 returns the set including only the member itself.

Note

Use this form of the **Ancestors** function for cases in which the level of the parent is unknown or cannot be named.

Examples

The following example uses the **Ancestors** function to return the Internet Sales Amount measure for a member, its parent, and its grandparent. This example uses level expressions to specify the levels to be returned. The levels are in the same hierarchy as the member specified in the member expression.

```
SELECT {
    Ancestors([Product].[Product Categories].[Product].[Mountain-100 Silver,
38],[Product].[Product Categories].[Category]),
    Ancestors([Product].[Product Categories].[Product].[Mountain-100 Silver,
38],[Product].[Product Categories].[Subcategory]),
    Ancestors([Product].[Product Categories].[Product].[Mountain-100 Silver,
38],[Product].[Product Categories].[Product])
} ON 0,
[Measures].[Internet Sales Amount] ON 1
FROM [Adventure Works]
```

The following example uses the **Ancestors** function to return the Internet Sales Amount measure for a member, its parent, and its grandparent. This example uses numeric expressions

to specify the levels being returned. The levels are in the same hierarchy as the member specified in the member expression.

```
SELECT {
    Ancestors (
        [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],2
    ),
    Ancestors (
        [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],1
    ),
    Ancestors (
        [Product].[Product Categories].[Product].[Mountain-100 Silver, 38],0
    )
} ON 0,
[Measures].[Internet Sales Amount] ON 1
FROM [Adventure Works]
```

The following example uses the **Ancestors** function to return the Internet Sales Amount measure for the parent of a member of an attribute hierarchy. This example uses a numeric expression to specify the level being returned. Since the member in the member expression is a member of an attribute hierarchy, its parent is the [All] level.

```
SELECT {
    Ancestors (
        [Product].[Product].[Mountain-100 Silver, 38],1
    )
} ON 0,
[Measures].[Internet Sales Amount] ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Ascendants

Returns the set of the ascendants of a specified member, including the member itself.

Syntax

Ascendants(**Member_Expression**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **Ascendants** function returns all of the ancestors of a member from the member itself up to the top of the member's hierarchy; more specifically, it performs a post-order traversal of the hierarchy for the specified member, and then returns all ascendant members related to the member, including itself, in a set. This is in contrast to the Ancestor function, which returns a specific ascendant member, or ancestor, at a specific level.

Examples

The following example returns the count of reseller orders for the [Sales Territory].[Northwest] member and all the ascendants of that member from the **Adventure Works** cube. The **Ascendants** function constructs the set that includes the [Sales Territory].[Northwest] member and its ascendants for the ROWS axis.

```
SELECT
    Measures.[Reseller Order Count] ON COLUMNS,
    Order (
        Ascendants (
            [Sales Territory].[Northwest]
        ),
        DESC
    ) ON ROWS
FROM
    [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Avg

Evaluates a set and returns the average of the non empty values of the cells in the set, averaged over the measures in the set or over a specified measure.

Syntax

```
Avg( Set_Expression [ , Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a set of empty tuples or an empty set is specified, the **Avg** function returns an empty value. The **Avg** function calculates the average of the nonempty values of cells in the specified set by first calculating the sum of values across cells in the specified set, and then dividing the calculated sum by the count of nonempty cells in the specified set.

Note

Analysis Services ignores nulls when calculating the average value in a set of numbers.

If a specific numeric expression (typically a measure) is not specified, the **Avg** function averages each measure within the current query context. If a specific measure is provided, the **Avg** function first evaluates the measure over the set, and then the function calculates the average based on the specified measure.

Note

When using the **CurrentMember** function in a calculated member statement, you must specify a numeric expression because no default measure exists for the current coordinate in such a query context.

To force the inclusion of empty cells, the application must use the `CoalesceEmpty` function or specify a valid `Numeric_Expression` that supplies a value of zero (0) for empty values. For more information about empty cells, see the OLE DB documentation.

Examples

The following example returns the average for a measure over a specified set. Notice that the specified measure can be either the default measure for the members of the specified set or a specified measure.

```
WITH SET [NW Region] AS  
    { [Geography].[State-Province].[Washington]  
      , [Geography].[State-Province].[Oregon]
```

```

    , [Geography].[State-Province].[Idaho]}
MEMBER [Geography].[Geography].[NW Region Avg] AS
    AVG ([NW Region]
        --Uncomment the line below to get an average by Reseller Gross Profit
        Margin
        --otherwise the average will be by whatever the default measure is in the
        cube,
        --or whatever measure is specified in the query
        --, [Measures].[Reseller Gross Profit Margin]
    )
SELECT [Date].[Calendar Year].[Calendar Year].Members ON 0
FROM [Adventure Works]
WHERE ([Geography].[Geography].[NW Region Avg])

```

The following example returns the daily average of the Measures.[Gross Profit Margin] measure, calculated across the days of each month in the 2003 fiscal year, from the **Adventure Works** cube. The **Avg** function calculates the average from the set of days that are contained in each month of the [Ship Date].[Fiscal Time] hierarchy. The first version of the calculation shows the default behavior of Avg in excluding days that did not record any sales from the average, the second version shows how to include days with no sales in the average.

```

WITH MEMBER Measures.[Avg Gross Profit Margin] AS
    Avg(
        Descendants(
            [Ship Date].[Fiscal].CurrentMember,
            [Ship Date].[Fiscal].[Date]
        ),
        Measures.[Gross Profit Margin]
    ), format_String='percent'
MEMBER Measures.[Avg Gross Profit Margin Including Empty Days] AS
    Avg(
        Descendants(
            [Ship Date].[Fiscal].CurrentMember,
            [Ship Date].[Fiscal].[Date]
        ),
        CoalesceEmpty(Measures.[Gross Profit Margin],0)
    ), Format_String='percent'
SELECT

```

```
{Measures.[Avg Gross Profit Margin],Measures.[Avg Gross Profit Margin  
Including Empty Days]} ON COLUMNS,
```

```
[Ship Date].[Fiscal].[Fiscal Year].Members ON ROWS
```

```
FROM
```

```
[Adventure Works]
```

```
WHERE ([Product].[Product Categories].[Product].&[344])
```

The following example returns the daily average of the Measures.[Gross Profit Margin] measure, calculated across the days of each semester in the 2003 fiscal year, from the **Adventure Works** cube.

```
WITH MEMBER Measures.[Avg Gross Profit Margin] AS
```

```
    Avg (
```

```
        Descendants (
```

```
            [Ship Date].[Fiscal].CurrentMember,
```

```
            [Ship Date].[Fiscal].[Date]
```

```
        ),
```

```
        Measures.[Gross Profit Margin]
```

```
    )
```

```
SELECT
```

```
    Measures.[Avg Gross Profit Margin] ON COLUMNS,
```

```
    [Ship Date].[Fiscal].[Fiscal Year].[FY 2003].Children ON ROWS
```

```
FROM
```

```
[Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Axis

Returns the set of tuples on a specified axis.

Syntax

```
Axis(Axis_Number)
```

Arguments

Axis_Number

A valid numeric expression that specifies the axis number.

Remarks

The **Axis** function uses the zero-based position of an axis to return the set of tuples on an axis. For example, `Axis(0)` returns the COLUMNS axis, `Axis(1)` returns the ROWS axis, and so on. The **Axis** function cannot be used on the filter axis. This function can be used to make calculated members aware of the context of the query that is being run. For example, you might need a calculated member that provides the sum of only those members selected on the Rows axis. It can also be used to make the definition of one axis dependent on the definition of another. For example, by ordering the contents of the Rows axis according to the value of the first item on the Columns axis.



Note

An axis can reference only a prior axis. For example, `Axis(0)` must occur after the COLUMNS axis has been evaluated, such as on a ROW or PAGE axis.

Examples

The following example query shows how to use the Axis function:

```
WITH MEMBER MEASURES.AXISDEMO AS
SETTOSTR(AXIS(1))
SELECT MEASURES.AXISDEMO ON 0,
[Date].[Calendar Year].MEMBERS ON 1
FROM [Adventure Works]
```

The following example shows the use of the Axis function inside a calculated member:

```
WITH MEMBER MEASURES.AXISDEMO AS
SUM(AXIS(1), [Measures].[Internet Sales Amount])
SELECT {[Measures].[Internet Sales Amount],MEASURES.AXISDEMO} ON 0,
{[Date].[Calendar Year].&[2003], [Date].[Calendar Year].&[2004]} ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

BottomCount

Sorts a set in ascending order, and returns the specified number of tuples in the specified set with the lowest values.

Syntax

```
BottomCount(Set_Expression, Count [,Numeric_Expression])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, this function sorts the tuples in the specified set according to the value of the specified numeric expression as evaluated over the set, in ascending order. The **BottomCount** function then returns the specified number of tuples with the lowest value.

Important

The **BottomCount** function, like the TopCount function, always breaks the hierarchy.

If a numeric expression is not specified, the function returns the set of members in natural order, without any sorting, behaving like the Tail (MDX) function.

Example

The following example returns the Reseller Order Quantity measure by for each calendar year for the bottom five Product SubCategory sales, ordered based on the Reseller Sales Amount measure.

```
SELECT BottomCount([Product].[Product Categories].[Subcategory].Members
, 10
, [Measures].[Reseller Sales Amount]) ON 0,
[Date].[Calendar].[Calendar Year].Members ON 1

FROM
```

```
[Adventure Works]
WHERE
    [Measures].[Reseller Order Quantity]
```

See Also

[MDX Function Reference \(MDX\)](#)

BottomPercent

Sorts a set in ascending order, and returns a set of tuples with the lowest values whose cumulative total is equal to or greater than a specified percentage.

Syntax

```
BottomPercent(Set_Expression, Percentage, Numeric_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Percentage

A valid numeric expression that specifies the percentage of tuples to be returned.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **BottomPercent** function calculates the sum of the specified numeric expression evaluated over a specified set, sorting the set in ascending order. The function then returns the elements with the lowest values whose cumulative percentage of the total summed value is at least the specified percentage. This function returns the smallest subset of a set whose cumulative total is at least the specified percentage. The returned elements are ordered largest to smallest.

Important

The **BottomPercent** function, like the TopPercent function, always breaks the hierarchy. For more information, see Order function.

Example

The following example returns, for the Bike category, the smallest set of members of the City level in the Geography hierarchy in the Geography dimension for fiscal year 2003 whose cumulative total using the Reseller Sales Amount measure is at least 15% of the cumulative total (beginning with the members of this set with the smallest number of sales).

```
SELECT
[Product].[Product Categories].Bikes ON 0,
BottomPercent
    ( {[Geography].[Geography].[City].Members}
    , 15
    , ([Measures].[Reseller Sales Amount],[Product].[Product
Categories].Bikes)
    ) ON 1
FROM [Adventure Works]
WHERE ([Measures].[Reseller Sales Amount],[Date].[Fiscal].[Fiscal Year].[FY
2003])
```

See Also

[MDX Function Reference \(MDX\)](#)

BottomSum

Sorts a specified set in ascending order, and returns a set of tuples with the lowest values whose sum is equal to or less than a specified value.

Syntax

```
BottomSum(Set_Expression, Value, Numeric_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Value

A valid numeric expression that specifies the value against which each tuple is compared.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression

of cell coordinates that return a number.

Remarks

The **BottomSum** function calculates the sum of a specified measure evaluated over a specified set, sorting the set in ascending order. The function then returns the elements with the lowest values whose total of the specified numeric expression is at least the specified value (sum). This function returns the smallest subset of a set whose cumulative total is at least the specified value. The returned elements are ordered smallest to largest.

Important

The **BottomSum** function, like the TopSum function, always breaks the hierarchy.

Examples

The following example returns, for the Bike category, the smallest set of members of the City level in the Geography hierarchy in the Geography dimension for fiscal year 2003, and whose cumulative total, using the Reseller Sales Amount measure, is at least the sum of 50,000 (beginning with the members of this set with the smallest number of sales):

```
SELECT
[Product].[Product Categories].Bikes ON 0,
BottomSum
    ( {[Geography].[Geography].[City].Members}
    , 50000
    , ([Measures].[Reseller Sales Amount],[Product].[Product
Categories].Bikes)
    ) ON 1
FROM [Adventure Works]
WHERE ([Measures].[Reseller Sales Amount],[Date].[Fiscal].[Fiscal Year].[FY
2003])
```

See Also

[MDX Function Reference \(MDX\)](#)

CalculationCurrentPass

Returns the current calculation pass of a cube for the specified query context.

Syntax

CalculationCurrentPass()

Remarks

The **CalculationCurrentPass** function returns the zero-based index of the calculation pass for the current query context. With automatic recursion resolution in SQL Server Analysis Services, this function has little practical use.

See Also

[MDX Function Reference \(MDX\)](#)

[If \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

CalculationPassValue

Returns either the numeric or the string value of a Multidimensional Expressions (MDX) expression evaluated over the specified calculation pass of a cube.

Syntax

Numeric syntax

CalculationPassValue(**Numeric_Expression**,**Pass_Value** [, ABSOLUTE | RELATIVE [,ALL]])

String syntax

CalculationPassValue(**String_Expression**,**Pass_Value** [, ABSOLUTE | RELATIVE [,ALL]])

Arguments

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

String_Expression

A valid string expression that is typically a valid Multidimensional Expressions (MDX) expression of cell coordinates that return a number expressed as a string.

Pass_Value

A valid numeric expression that specifies the calculation pass number.

ABSOLUTE

An access flag value that specifies that the Pass_Value parameter contains the zero-based index of the calculation pass. ABSOLUTE is the default access flag value if no access flag value is specified.

RELATIVE

An access flag value that specifies that the `Pass_Value` parameter contains a relative offset from the calculation pass of the triggering calculation. If the offset resolves into a calculation pass index less than 0, calculation pass 0 is used and no error occurs.

ALL

When this flag is set, all values are null except for those loaded by the storage engine. When not set, the values are aggregated without any calculations applied.

Remarks

If a numeric expression is provided, the function returns a numeric value by evaluating the specified MDX numeric expression in the specified calculation pass, and optionally modified by an access flag and an access flag modifier.

If a string expression is provided, the function returns a string value by evaluating the specified MDX string expression in the specified calculation pass, and optionally modified by an access flag and an access flag modifier.

With automatic recursion resolution in SQL Server Analysis Services, this function has little practical use.

Note

Only administrators can use the **CalculationPassValue** function within an MDX script. An error occurs if an MDX script that contains this function is run in the context of a role that does not have administrator privileges.

See Also

[MDX Function Reference \(MDX\)](#)

[If \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

CASE Statement

Lets you conditionally return specific values from multiple comparisons. There are two types of case statements:

- A simple case statement that compares an expression to a set of simple expressions to return specific values.
- A searched case statement that evaluates a set of Boolean expressions to return specific values.

Syntax

Simple Case Statement

```
CASE [input_expression]
WHEN when_expression THEN when_true_result_expression
[...n]
[ELSE else_result_expression]
END
```

Search Case Statement

```
CASE
WHEN Boolean_expression THEN when_true_result_expression
[...n]
[ELSE else_result_expression]
END
```

Arguments

input_expression

A Multidimensional Expressions (MDX) expression that resolves to a scalar value.

when_expression

A specified scalar value against which the input_expression is evaluated, which when evaluated to true, returns the scalar value of the else_result_expression.

when_true_result_expression

The scalar value returned when the WHEN clause evaluates to true.

else_result_expression

The scalar value returned when none of the WHEN clauses evaluate to true.

Boolean_expression

An MDX expression that evaluates to a scalar value.

Remarks

If there is no ELSE clause, and all WHEN clauses evaluate to false, the result is an empty cell.

Simple Case Expression

MDX evaluates a simple case expression by resolving the input_expression to a scalar value. This scalar value is then compared to the scalar value of the when_expression. If the two scalar values match, the CASE statement returns the value of the when_true_expression. If the two scalar values do not match, the next WHEN clause is evaluated. If all of the WHEN clauses evaluate to false, the value of else_result_expression from the ELSE clause, if any, is returned.

In the following example, the Reseller Order Count measure is evaluated against several WHEN clauses and returns a result based on the value of the Reseller Order Count measure for each year. For Reseller Order Count values that do not match a scalar value specified in a when_expression in a WHEN clause, the scalar value of the else_result_expression is returned.

```
WITH MEMBER [Measures].x AS
CASE [Measures].[Reseller Order Count]
    WHEN 0 THEN 'NONE'
    WHEN 1 THEN 'SMALL'
    WHEN 2 THEN 'SMALL'
    WHEN 3 THEN 'MEDIUM'
    WHEN 4 THEN 'MEDIUM'
    WHEN 5 THEN 'LARGE'
    WHEN 6 THEN 'LARGE'
        ELSE 'VERY LARGE'
END
SELECT Calendar.[Calendar Year] on 0
, NON EMPTY [Geography].[Postal Code].Members ON 1
FROM [Adventure Works]
WHERE [Measures].x
```

Searched Case Expression

To use the case expression to perform more complex evaluations, use the searched case expression. This variation of the search expression allows you to evaluate whether an input expression is within a range of values. MDX evaluates the WHEN clauses in the order that these clauses appear in the CASE statement.

In the following example, the Reseller Order Count measure is evaluated against the specified Boolean_expression for each of several WHEN clauses. A result is returned based on the value of the Reseller Order Count measure for each year. Because WHEN clauses are evaluated in the order they appear, all values larger than 6 can easily be assigned the value of "VERY LARGE" without having to specify each value explicitly. For Reseller Order Count values that are not specified in a WHEN clause, the scalar value of the else_result_expression is returned.

```
WITH MEMBER [Measures].x AS
CASE
    WHEN [Measures].[Reseller Order Count] > 6 THEN 'VERY LARGE'
    WHEN [Measures].[Reseller Order Count] > 4 THEN 'LARGE'
    WHEN [Measures].[Reseller Order Count] > 2 THEN 'MEDIUM'
```

```

    WHEN [Measures].[Reseller Order Count] > 0 THEN 'SMALL'
    ELSE "NONE"
END
SELECT Calendar.[Calendar Year] on 0,
NON EMPTY [Geography].[Postal Code].Members on 1
FROM [Adventure Works]
WHERE [Measures].x

```

See Also

[MDX Scripting Statements \(MDX\)](#)

Children

Returns the set of children of a specified member.

Syntax

```
Member_Expression.Children
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **Children** function returns a naturally ordered set that contains the children of a specified member. If the specified member has no children, this function returns an empty set.

Example

The following example returns the children of the United States member of the Geography hierarchy in the Geography dimension.

```

SELECT [Geography].[Geography].[Country].&[United States].Children ON 0
FROM [Adventure Works]

```

The following example returns all members in the **Measures** dimension on the column axis, this includes all calculated members, and the set of all children of the [Product].[Model Name] attribute hierarchy on the row axis from the **Adventure Works** cube.

```

SELECT

```

```

Measures.AllMembers ON COLUMNS,
[Product].[Model Name].Children ON ROWS
FROM
[Adventure Works]

```

Release	History
	<p>Changed content:</p> <ul style="list-style-type: none"> • Updated syntax and arguments to improve clarity. • Added updated examples.

See Also

[MDX Function Reference \(MDX\)](#)

ClosingPeriod

Returns the member that is the last sibling among the descendants of a specified member at a specified level.

Syntax

```
ClosingPeriod( [ Level_Expression [ ,Member_Expression ] ] )
```

Arguments

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

This function is primarily intended to be used against a dimension with a type of Time, but can be used with any dimension.

- If a level expression is specified, the **ClosingPeriod** function uses the dimension that contains the specified level and returns the last sibling among the descendants of the default member at the specified level.
- If both a level expression and a member expression are specified, the **ClosingPeriod** function returns the last sibling among the descendants of specified member at the specified level.
- If neither a level expression nor a member expression is specified, the **ClosingPeriod** function uses the default level and member of the dimension (if any) in the cube with a type of Time.

The **ClosingPeriod** function is equivalent to the following MDX statement:

```
Tail(Descendants(Member_Expression, Level_Expression), 1).
```

Note

The **OpeningPeriod** function is similar to the **ClosingPeriod** function, except that the **OpeningPeriod** function returns the first sibling instead of the last sibling.

Examples

The following example returns the value for the default measure for FY2007 member of the Date dimension (which has a semantic type of Time). This member is returned because the Fiscal Year level is the first descendant of the [All] level, the Fiscal hierarchy is the default hierarchy because it is the first user-defined hierarchy in the hierarchy collection, and the FY 2007 member is the last sibling in this hierarchy at this level.

```
SELECT ClosingPeriod() ON 0
FROM [Adventure Works]
```

The following example returns the value for the default measure for November 30, 2006 member at the Date.Date.Date level for the Date.Date attribute hierarchy. This member is the last sibling of the descendant of [All] level in the Date.Date attribute hierarchy.

```
SELECT ClosingPeriod ([Date].[Date].[Date]) ON 0
FROM [Adventure Works]
```

The following example returns the value for the default measure for December, 2003 member, which is the last sibling of the descendant of the 2003 member at the year level in the Calendar user-defined hierarchy.

```
SELECT ClosingPeriod ([Date].[Calendar].[Month],[Date].[Calendar].[Calendar
Year].&[2003]) ON 0
FROM [Adventure Works]
```

The following example returns the value for the default measure for June, 2003 member, which is the last sibling of the descendant of the 2003 member at the year level in the Fiscal user-defined hierarchy.

```
SELECT ClosingPeriod ([Date].[Fiscal].[Month],[Date].[Fiscal].[Fiscal
Year].&[2003]) ON 0
FROM [Adventure Works]
```

See Also

[OpeningPeriod \(MDX\) \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

[LastSibling \(MDX\)](#)

CoalesceEmpty

Converts an empty cell value to a specified nonempty cell value, which can be either a number or string.

Syntax

Numeric syntax

CoalesceEmpty(**Numeric_Expression1** [,**Numeric_Expression2**,...n])

String syntax

CoalesceEmpty(**String_Expression1** [,**String_Expression2**,...n])

Arguments

Numeric_Expression1

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Numeric_Expression2

A valid numeric expression that is typically a specified numeric value.

String_Expression1

A valid string expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that returns a string.

String_Expression2

A valid string expression that is typically a specified string value that is substituted for a NULL returned by the first string expression.

Remarks

If one or more numeric expressions are specified, the **CoalesceEmpty** function returns the numeric value of the first numeric expression (from left to right) that can be resolved to a nonempty value. If none of the specified numeric expressions can be resolved to a nonempty value, the function returns the empty cell value. Typically, the value for the second numeric expression is the numeric value that is substituted for a NULL returned by the first numeric expression.

If one or more string expressions are specified, the function returns the string value of the first string expression (from left to right) that can be resolved to a nonempty value. If none of the specified string expressions can be resolved to a nonempty value, the function returns the empty cell value. Typically, the value for the second string expression value is the string value that is substituted for a NULL returned by the first string expression.

The **CoalesceEmpty** function can only take values of the same type. In other words, all specified value expressions must evaluate to only numeric data types or an empty cell value, or all specified value expressions must evaluate to string data types or to an empty cell value. A single call to this function cannot include both numeric and string expressions.

For more information about empty cells, see the OLE DB documentation.

Example

The following example queries the **Adventure Works** cube. This example returns the order quantity of each product and the percentage of order quantities by category. The **CoalesceEmpty** function ensures that null values are represented as zero (0) when formatting the calculated members.

```
WITH
    MEMBER [Measures].[Order Percent by Category] AS
        CoalesceEmpty
        (
            ([Product].[Product Categories].CurrentMember,
            Measures.[Order Quantity]) /
            (
                Ancestor
                ( [Product].[Product Categories].CurrentMember,
                [Product].[Product Categories].[Category]
                ), Measures.[Order Quantity]
            ), 0
        ), FORMAT_STRING='Percent'
SELECT
```

```
{Measures.[Order Quantity],
    [Measures].[Order Percent by Category]} ON COLUMNS,
{[Product].[Product].Members} ON ROWS
FROM [Adventure Works]
WHERE {[Date].[Calendar Year].[Calendar Year].&[2003]}
```

See Also

[MDX Function Reference \(MDX\)](#)

Correlation

Returns the correlation coefficient of x-y pairs of values evaluated over a set.

Syntax

```
Correlation( Set_Expression, Numeric_Expression_y [ ,Numeric_Expression_x ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

The **Correlation** function calculates the correlation coefficient of two pairs of values by first evaluating the specified set against the first numeric expression to obtain the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if present, to obtain the set of values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as the values for the x-axis.

Note

The **Correlation** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

See Also

[MDX Function Reference \(MDX\)](#)

Count (Dimension)

Returns the number of hierarchies in a cube.

Syntax

Dimensions.Count

Remarks

Returns the number of hierarchies in a cube, including the [Measures] . [Measures] hierarchy.

Example

The following example returns the number of hierarchies in the Adventure Works cube.

```
WITH MEMBER measures.X AS
    dimensions.count
SELECT Measures.X ON 0
FROM [Adventure Works]
```

See Also

[Count \(Tuple\) \(MDX\)](#)

[Count \(Hierarchy Levels\) \(MDX\)](#)

[Count \(Set\) \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Count (Hierarchy Levels)

Returns the number of levels in hierarchy.

Syntax

Hierarchy_Expression.Levels.Count

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

Returns the number of levels in a hierarchy, including the [All] level, if applicable.

Important

When a dimension contains only a single visible hierarchy, the hierarchy can be referred to either by the dimension name or by the hierarchy name, because the dimension name is resolved to its only visible hierarchy. For example, `Measures.Levels.Count` is a valid MDX expression because it resolves to the only hierarchy in the Measures dimension.

Example

The following example returns a count of the number of levels in the Product Categories user-defined hierarchy in the Adventure Works cube.

```
WITH MEMBER measures.X AS
    [Product].[Product Categories].Levels.Count
Select Measures.X ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Count \(Tuple\) \(MDX\)](#)

[Count \(Set\) \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Count (Set)

Returns the number of cells in a set.

Syntax

Standard syntax

```
Count(Set_Expression [ , ( EXCLUDEEMPTY | INCLUDEEMPTY ) ] )
```

Alternate syntax

```
Set_Expression.Count
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **Count (Set)** function includes or excludes empty cells, depending on the syntax used. If the standard syntax is used, empty cells can be excluded or included by using the **EXCLUDEEMPTY** or **INCLUDEEMPTY** flags, respectively. If the alternate syntax is used, the function always includes empty cells.

To exclude empty cells in the count of a set, use the standard syntax and the optional **EXCLUDEEMPTY** flag.

Note

The **Count (Set)** function counts empty cells by default. In contrast, the **Count** function in OLE DB that counts a set excludes empty cells by default.

Examples

The following example counts the number of cells in the set of members that consist of the children of the Model Name attribute hierarchy in the Product dimension.

```
WITH MEMBER measures.X AS
    [Product].[Model Name].children.count
SELECT Measures.X ON 0
FROM [Adventure Works]
```

The following example counts the number of products in the Product dimension by using the **DrilldownLevel** function in conjunction with the **Count** function.

```
Count(DrilldownLevel (
    [Product].[Product].[Product]))
```

The following example returns those resellers with declining sales compared to the previous calendar quarter, by using the **Count** function in conjunction with the **Filter** function and a number of other functions. This query uses the **Aggregate** function to support the selection of multiple geography members, such as for selection from within a drop-down list in a client application.

```
WITH MEMBER Measures.[Declining Reseller Sales] AS
    Count
    (Filter
        (Existing(Reseller.Reseller.Reseller),
         [Measures].[Reseller Sales Amount]
```

```

        < ([Measures].[Reseller Sales Amount],
          [Date].Calendar.PrevMember)
    )
)
MEMBER [Geography].[State-Province].x AS
    Aggregate
    ( { [Geography].[State-Province].&[WA]&[US],
      [Geography].[State-Province].&[OR]&[US] }
    )
SELECT NON EMPTY HIERARCHIZE
    (AddCalculatedMembers
      ({DrillDownLevel
        ({[Product].[All Products])
      })
    ) DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
      [Date].[Calendar].[Calendar Quarter].&[2003]&[4]
      , [Measures].[Declining Reseller Sales])

```

See Also

[MDX Function Reference \(MDX\)](#)

[Count \(Level\) \(MDX\)](#)

[Count \(Tuple\) \(MDX\)](#)

[DrilldownLevel \(MDX\)](#)

[AddCalculatedMembers \(MDX\)](#)

[Hierarchize \(MDX\)](#)

[Properties \(MDX\)](#)

[Aggregate \(MDX\)](#)

[Filter \(MDX\)](#)

[PrevMember \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Count (Tuple)

Returns the number of dimensions in a tuple.

Syntax

```
Tuple_Expression.Count
```

Arguments

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Remarks

Returns the number of dimensions in a tuple.

Example

The calculated measure in the following query returns the value 2, which is the number of hierarchies present in the tuple ([Measures].[Internet Sales Amount], [Date].[Calendar].[Calendar Year].&[2001]):

```
WITH MEMBER MEASURES.COUNTTUPLE AS
COUNT(([Measures].[Internet Sales Amount], [Date].[Calendar].[Calendar
Year].&[2001]))
SELECT MEASURES.COUNTTUPLE ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Count \(Level\) \(MDX\)](#)

[Count \(Set\) \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Cousin

Returns the child member with the same relative position under a parent member as the specified child member.

Syntax

Cousin(**Member_Expression** , **Ancestor_Member_Expression**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Ancestor_Member_Expression

A valid Multidimensional Expressions (MDX) member expression that returns an ancestor member.

Remarks

This function operates on the order and position of members within levels. If two hierarchies exist, in which the first one has four levels and the second one has five levels, the cousin of the third level of the first hierarchy is the third level of the second hierarchy.

Examples

The following example retrieves the cousin of the fourth quarter of fiscal year 2002, based on its ancestor at the year level in fiscal year 2003. The retrieved cousin is the fourth quarter of fiscal year 2003.

```
SELECT Cousin
  ( [Date].[Fiscal].[Fiscal Quarter].[Q4 FY 2002],
    [Date].[Fiscal].[FY 2003]
  ) ON 0
FROM [Adventure Works]
```

The following example retrieves the cousin of the month of July of fiscal year 2002 based on its ancestor at the quarter level in the second quarter of fiscal year 2004. The retrieved cousin is the month of October of 2003.

```
SELECT Cousin
  ([Date].[Fiscal].[Month].[July 2002] ,
   [Date].[Fiscal].[Fiscal Quarter].[Q2 FY 2004]
  ) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Covariance

Returns the population covariance of x-y pairs of values evaluated over a set, by using the biased population formula (dividing by the number of x-y pairs).

Syntax

```
Covariance(Set_Expression, Numeric_Expression_y [ , Numeric_Expression_x ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

The **Covariance** function evaluates the specified set against the first numeric expression, to get the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if specified, to get the set of values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as values for the x-axis.

The **Covariance** function uses the biased population formula. This is in contrast to the CovarianceN function that uses the unbiased population formula (dividing the number of x-y pairs, then subtracting 1).

Note

The **Covariance** function ignores empty cells or cells that contain text or logical values are ignored. However, the function includes cells with values of zero.

Example

The following example shows how to use the Covariance function:

```
WITH  
MEMBER [Measures].[CovarianceDemo] AS
```

```

COVARIANCE([Date].[Date].[Date].Members, [Measures].[Internet Sales Amount],
[Measures].[Internet Order Count])
SELECT {[Measures].[CovarianceDemo]} ON 0
FROM
[Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

CovarianceN

Returns the sample covariance of x-y pairs of values evaluated over a set, by using the unbiased population formula (dividing by the number of x-y pairs).

Syntax

```
CovarianceN(Set_Expression, Numeric_Expression_y [, Numeric_Expression_x ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

The **CovarianceN** function evaluates the specified set against the first numeric expression, to get the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if specified, to get the set of values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as values for the x-axis.

The **CovarianceN** function uses the unbiased population formula. This is in contrast to the Covariance function that uses the biased population formula (dividing by the number of x-y pairs).



Note

The **CovarianceN** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

See Also

[MDX Function Reference \(MDX\)](#)

Crossjoin

Returns the cross product of one or more sets.

Syntax

Standard syntax

`Crossjoin(Set_Expression1 ,Set_Expression2 [,...n])`

Alternate syntax

`Set_Expression1 * Set_Expression2 [* ...n]`

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **Crossjoin** function returns the cross product of two or more specified sets. The order of tuples in the resulting set depends on the order of the sets to be joined and the order of their members. For example, when the first set consists of {x1, x2,...,xn}, and the second set consists of {y1, y2, ..., yn}, the cross product of these sets is:

{(x1, y1), (x1, y2),..., (x1, yn), (x2, y1), (x2, y2),...,
(x2, yn),..., (xn, y1), (xn, y2),..., (xn, yn)}



Important

If the sets in the cross join are composed of tuples from different attribute hierarchies in the same dimension, this function will return only those tuples that actually exist. For more information, see [MDX Function Reference \(MDX\)](#).

Examples

The following query shows simple examples of the use of the Crossjoin function on the Columns and Rows axis of a query:

```
SELECT
    [Customer].[Country].Members *
        [Customer].[State-Province].Members
ON 0,
Crossjoin(
    [Date].[Calendar Year].Members,
    [Product].[Category].[Category].Members)
ON 1
FROM [Adventure Works]
WHERE Measures.[Internet Sales Amount]
```

The following example shows the automatic filtering that takes place when different hierarchies from the same dimension are crossjoined:

```
SELECT
Measures.[Internet Sales Amount]
ON 0,
//Only the dates in Calendar Years 2003 and 2004 will be returned here
Crossjoin(
    {[Date].[Calendar Year].&[2003], [Date].[Calendar Year].&[2004]},
    [Date].[Date].[Date].Members)
ON 1
FROM [Adventure Works]
```

The following three examples return the same results - the Internet Sales Amount by state for states within the United States. The first two use the two cross join syntaxes and the third demonstrates the use of the WHERE clause to return the same information.

Example 1

```
SELECT CROSSJOIN
    (
        {[Customer].[Country].[United States]},
        [Customer].[State-Province].Members
    ) ON 0
FROM [Adventure Works]
WHERE Measures.[Internet Sales Amount]
```

Example 2

```
SELECT
    [Customer].[Country].[United States] *
    [Customer].[State-Province].Members
ON 0
FROM [Adventure Works]
WHERE Measures.[Internet Sales Amount]
```

Example 3

```
SELECT
    [Customer].[State-Province].Members
ON 0
FROM [Adventure Works]
WHERE (Measures.[Internet Sales Amount],
    [Customer].[Country].[United States])
```

See Also

[MDX Function Reference \(MDX\)](#)

Current

Returns the current tuple from a set during iteration.

Syntax

```
Set_Expression.Current
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

At each step during an iteration, the tuple being operated upon is the current tuple. The **Current** function returns that tuple. This function is only valid during an iteration over a set. MDX functions that iterate through a set include the Generate function.

 **Note**

This function only works with sets that are named, either using a set alias or by defining a named set.

Examples

The following example shows how to use the **Current** function inside **Generate**:

```
WITH
//Creates a set of tuples consisting of all Calendar Years crossjoined with
//all Product Categories
SET MyTuples AS CROSSJOIN(
[Date].[Calendar Year].[Calendar Year].MEMBERS,
[Product].[Category].[Category].MEMBERS)
//Iterates through each tuple in the set and returns the name of the Calendar
//Year in each tuple
MEMBER MEASURES.CURRENTDEMO AS
GENERATE(MyTuples, MyTuples.CURRENT.ITEM(0).NAME, ", ")
SELECT MEASURES.CURRENTDEMO ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

CurrentMember

Returns the current member along a specified hierarchy during iteration.

Syntax

Hierarchy_Expression.CurrentMember

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

When iterating through a set of hierarchy members, at each step in the iteration, the member being operated upon is the current member. The **CurrentMember** function returns that member.

Important

When a dimension contains only a single visible hierarchy, the hierarchy can be referred to either by the dimension name or by the hierarchy name, because the dimension name is resolved to its only visible hierarchy. For example, `Measures.CurrentMember` is a valid MDX expression because it resolves to the only hierarchy in the Measures dimension.

Examples

The following query shows how **Currentmember** can be used to find the current member from hierarchies on the Columns, Rows and slice axis:

```
WITH MEMBER MEASURES.CURRENTDATE AS
[Date].[Calendar].CURRENTMEMBER.NAME
MEMBER MEASURES.CURRENTPRODUCT AS
[Product].[Product Categories].CURRENTMEMBER.NAME
MEMBER MEASURES.CURRENTMEASURE AS
MEASURES.CURRENTMEMBER.NAME
MEMBER MEASURES.CURRENTCUSTOMER AS
[Customer].[Customer Geography].CURRENTMEMBER.NAME
SELECT
[Product].[Product Categories].[Category].MEMBERS
*
{MEASURES.CURRENTDATE, MEASURES.CURRENTPRODUCT, MEASURES.CURRENTMEASURE,
MEASURES.CURRENTCUSTOMER}
ON 0,
[Date].[Calendar].MEMBERS
ON 1
FROM [Adventure Works]
WHERE ([Customer].[Customer Geography].[Country].&[Australia])
```

The current member changes on a hierarchy used on an axis in a query. Therefore, the current member on other hierarchies on the same dimension that are not used on an axis can also change; this behavior is called 'auto-exists' and more details can be found in [Key Concepts in MDX \(MDX\)](#). For example, the query below shows how the current member on the Calendar Year hierarchy of the Date dimension changes with the current member on the Calendar hierarchy, when the latter is displayed on the Rows axis:

```
WITH MEMBER MEASURES.CURRENTYEAR AS
[Date].[Calendar Year].CURRENTMEMBER.NAME
SELECT
```

```

{MEASURES.CURRENTYEAR}
ON 0,
[Date].[Calendar].MEMBERS
ON 1
FROM [Adventure Works]

```

CurrentMember is very important for making calculations aware of the context of the query they are being used in. The following example returns the order quantity of each product and the percentage of order quantities by category and model, from the **Adventure Works** cube. The **CurrentMember** function identifies the product whose order quantity is to be used during calculation.

```

WITH
    MEMBER [Measures].[Order Percent by Category] AS
    CoalesceEmpty
    (
        ([Product].[Product Categories].CurrentMember,
        Measures.[Order Quantity]) /
        (
            Ancestor
            ( [Product].[Product Categories].CurrentMember,
            [Product].[Product Categories].[Category]
            ), Measures.[Order Quantity]
        ), 0
    ), FORMAT_STRING='Percent'
SELECT
    {Measures.[Order Quantity],
    [Measures].[Order Percent by Category]} ON COLUMNS,
    {[Product].[Product].Members} ON ROWS
FROM [Adventure Works]
WHERE {[Date].[Calendar Year].[Calendar Year].&[2003]}

```

See Also

[MDX Function Reference \(MDX\)](#)

CurrentOrdinal

Returns the current iteration number within a set during iteration.

Syntax

```
Set_Expression.CurrentOrdinal
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

When iterating through a set, such as with the Filter (MDX) or Generate (MDX) functions, the **CurrentOrdinal** function returns the iteration number.

Examples

The following simple example shows how **CurrentOrdinal** can be used with **Generate** to return a string containing the name of each item in a set along with its position in the set:

```
WITH SET MySet AS [Customer].[Customer Geography].[Country].MEMBERS
MEMBER MEASURES.CURRENTORDINALDEMO AS
GENERATE(MySet, CSTR(MySet.CURRENTORDINAL) + ") " +
MySet.CURRENT.ITEM(0).NAME, ", ")
SELECT MEASURES.CURRENTORDINALDEMO ON 0
FROM [Adventure Works]
```

The practical use of CurrentOrdinal is limited to very complex calculations. The following example returns the number of products in the set that are unique, using the **Order** function to order the non-empty tuples before utilizing the **Filter** function. The **CurrentOrdinal** function is used to compare and eliminate ties.

```
WITH MEMBER [Measures].[PrdTies] AS Count
  (Filter
    (Order
      (NonEmpty
        ([Product].[Product].[Product].Members
        , {[Measures].[Reseller Order Quantity]}
        )
      , [Measures].[Reseller Order Quantity]
```

```

    , BDESC
  ) AS OrdPrds
, NOT((OrdPrds.CurrentOrdinal < OrdPrds.Count
  AND [Measures].[Reseller Order Quantity] =
    ( [Measures].[Reseller Order Quantity]
      , OrdPrds.Item
        (OrdPrds.CurrentOrdinal
          )
        )
    )
  )
OR (OrdPrds.CurrentOrdinal > 1
  AND [Measures].[Reseller Order Quantity] =
    ([Measures].[Reseller Order Quantity]
      , OrdPrds.Item
        (OrdPrds.CurrentOrdinal-2)
          )
    )
  ))
)
SELECT {[Measures].[PrdTies]} ON 0
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

CustomData

Returns the value of the **CustomData** connection string property if defined; otherwise, **null**.

Syntax

CustomData()

Return Value

The **CustomData** function can retrieve the **CustomData** connection string property and pass a configuration setting to be used by Multidimensional Expressions (MDX) functions and statements, such as UserName (MDX) and CALL Statement (MDX). For example, this function can be used in a dynamic security expression to select the allowed/denied set members for the string value in the **CustomData** connection string property.

Example

The following query displays the value returned by the **CustomData** function in a calculated measure:

```
WITH MEMBER [Measures].CUSTOMDATADEMO AS CUSTOMDATA()  
SELECT [Measures].CUSTOMDATADEMO ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

DataMember

Returns the system-generated data member that is associated with a nonleaf member of a dimension.

Syntax

Member_Expression.DataMember

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

This function operates on nonleaf members in any hierarchy and can be used by the UPDATE CUBE Statement (MDX) command to writeback data to a nonleaf member directly, rather than to the leaf member's descendants.

Note

Returns the specified member if the specified member is a leaf member, or if the nonleaf member does not have an associated data member.

Example

The following example uses the **DataMember** function in a calculated measure to show the sales quota for each individual employee:

```
WITH MEMBER measures.IndividualQuota AS
([Employee].[Employees].currentmember.datamember, [Measures].[Sales Amount
Quota])
,FORMAT_STRING='Currency'
SELECT {[Measures].[Sales Amount Quota],[Measures].IndividualQuota} ON COLUMNS,
[Employee].[Employees].MEMBERS ON ROWS
FROM [Adventure Works]
```

See Also

[Key Concepts in MDX \(MDX\)](#)

[Key Concepts in MDX \(MDX\)](#)

DefaultMember

Returns the default member of a hierarchy.

Syntax

Hierarchy_Expression.DefaultMember

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

The default member on an attribute is used to evaluate expressions when an attribute is not included in a query.

Example

The following example uses the **DefaultMember** function, in conjunction with the **Name** function, to return the default member for the Destination Currency dimension in the Adventure

Works cube. The example returns **US Dollar**. The **Name** function is used to return the name of the measure rather than the default property of the measure, which is **value**.

```
WITH MEMBER Measures.x AS
    [Destination Currency].[Destination Currency].DefaultMember.Name
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[Defining a Default Member](#)

[Defining a Default Member](#)

Descendants

Returns the set of descendants of a member at a specified level or distance, optionally including or excluding descendants in other levels.

Syntax

Member expression syntax using a level expression

```
Descendants(Member_Expression [ , Level_Expression [ ,Desc_Flag ] ] )
```

Member expression syntax using a numeric expression

```
Descendants(Member_Expression [ , Distance [ ,Desc_Flag ] ] )
```

Set expression syntax using a level expression

```
Descendants(Set_Expression [ , Level_Expression [ ,Desc_Flag ] ] )
```

Member expression syntax using a numeric expression

```
Descendants(Set_Expression [ , Distance [ ,Desc_Flag ] ] )
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Distance

A valid numeric expression that specifies the distance from the specified member.

Desc_Flag

A valid string expression specifying a description flag that distinguishes among possible sets of descendants.

Remarks

If a level is specified, the **Descendants** function returns a set that contains the descendants of the specified member or the members of the specified set, at a specified level, optionally modified by a flag specified in Desc_Flag.

If Distance is specified, the **Descendants** function returns a set that contains the descendants of the specified member or the members of the specified set that are the specified number of levels away in the hierarchy of the specified member, optionally modified by a flag specified in Desc_Flag. Typically, you use this function with the Distance argument to deal with ragged hierarchies. If the specified distance is zero (0), the function returns a set that consists only of the specified member or the specified set.

If a set expression is specified, the **Descendants** function is resolved individually for each member of the set, and the set is created again. In other words, the syntax used for the **Descendants** function is functionally equivalent to the MDX Generate function.

If no level or distance is specified, the default value for the level used by the function is determined by calling the Level function (<<Member>>.Level) for the specified member (if a member is specified) or by calling the **Level** function for each member of the specified set (if a set is specified). If no level expression, distance or flags are specified, the function performs as if the following syntax were used:

```
Descendants
(
    Member_Expression ,
    Member_Expression.Level ,
    SELF_BEFORE_AFTER
)
```

If a level is specified and a description flag is not specified, the function performs as if the following syntax were used.

```
Descendants
(
    Member_Expression ,
    Level_Expression,
```


SELF

)

By changing the value of the description flag, you can include or exclude descendants at the specified level or distance, the children before or after the specified level or distance (until the leaf node), and the leaf children regardless of the specified level or distance. The following table describes the flags allowed in the Desc_Flag argument.

Flag	Description
SELF	Returns only descendant members from the specified level or at the specified distance. The function includes the specified member, if the specified level is the level of the specified member.
AFTER	Returns descendant members from all levels subordinate to the specified level or distance.
BEFORE	Returns descendant members from all levels between the specified member and the specified level, or at the specified distance. It includes the specified member, but does not include members from the specified level or distance.
BEFORE_AND_AFTER	Returns descendant members from all levels subordinate to the level of the specified member. It includes the specified member, but does not include members from the specified level or at the specified distance.
SELF_AND_AFTER	Returns descendant members from the specified level or at the specified distance and all levels subordinate to the specified level, or at the specified distance.
SELF_AND_BEFORE	Returns descendant members from the specified level or at the specified distance, and from all levels between the specified member and the specified level, or at the specified distance, including the specified member.

Flag	Description
SELF_BEFORE_AFTER	Returns descendant members from all levels subordinate to the level of the specified member, and includes the specified member.
LEAVES	Returns leaf descendant members between the specified member and the specified level, or at the specified distance.

Examples

The following example returns the specified member (United States), and the members between the specified member (United States) and the members of the level before the specified level (City). The example returns the specified member itself (United States), and the members of the State-Province level (the level before the City level). This example includes commented arguments to enable you to easily test other arguments for this function.

```
SELECT Descendants
    ([Geography].[Geography].[Country].&[United States]
    //, [Geography].[Geography].[Country]
, [Geography].[Geography].[City]
    //, [Geography].[Geography].Levels (3)
    //, SELF
    //, AFTER
, BEFORE
    // BEFORE_AND_AFTER
    //, SELF_AND_AFTER
    //, SELF_AND_BEFORE
    //,SELF_BEFORE_AFTER
    //,LEAVES
) ON 0
FROM [Adventure Works]
```

The following example returns the daily average of the Measures.[Gross Profit Margin] measure, calculated across the days of each month in the 2003 fiscal year, from the **Adventure Works** cube. The **Descendants** function returns a set of days determined from the current member of the [Date].[Fiscal] hierarchy.

```
WITH MEMBER Measures.[Avg Gross Profit Margin] AS Avg
```

```

(
    Descendants
    ( [Date].[Fiscal].CurrentMember,
      [Date].[Fiscal].[Date]
    ),
    Measures.[Gross Profit Margin]
)
SELECT
    Measures.[Avg Gross Profit Margin] ON COLUMNS,
    [Date].[Fiscal].[Month].Members ON ROWS
FROM [Adventure Works]
WHERE ([Date].[Fiscal Year].&[2003])

```

The following example uses a level expression and returns the Internet Sales Amount for each State-Province in Australia, and returns the percentage of the total Internet Sales Amount for Australia for by each State-Province. This example uses the Item function to extract the first (and only) tuple from the set that is returned by the **Ancestors** function.

```

WITH MEMBER Measures.x AS
    [Measures].[Internet Sales Amount] /
    ( [Measures].[Internet Sales Amount],
      Ancestors
        ( [Customer].[Customer Geography].CurrentMember,
          [Customer].[Customer Geography].[Country]
        ).Item (0)
    ), FORMAT_STRING = '0%'
SELECT {[Measures].[Internet Sales Amount], Measures.x} ON 0,
{Descendants
  ( [Customer].[Customer Geography].[Country].&[Australia],
    [Customer].[Customer Geography].[State-Province], SELF
  )
} ON 1
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

Dimension

Returns the hierarchy that contains a specified member, level, or hierarchy.

Syntax

Hierarchy syntax

`Hierarchy_Expression.Dimension`

Level syntax

`Level_Expression.Dimension`

Member syntax

`Member_Expression.Dimension`

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Examples

The following example uses the **Dimension** function, in conjunction with the **Name** function, to return the hierarchy name of the specified member.

```
WITH member measures.x as [Product].[Product Model Lines].[Model].&[HL Road  
Tire].Dimension.Name  
SELECT measures.x on 0  
FROM [Adventure Works]
```

The following example uses the Dimension function, in conjunction with the Levels and the Count functions, to return the number of levels in the hierarchy containing the specified member.

```
WITH member measures.x as [Product].[Product Model Lines].[Model].&[HL Road  
Tire].Dimension.Levels.Count  
SELECT measures.x on 0  
FROM [Adventure Works]
```

The following example uses the **Dimension** function, in conjunction with the **Members** and the **Count** functions, to return the number of members in the hierarchy containing the specified member.

```
WITH member measures.x as [Product].[Product Model Lines].[Model].&[HL Road  
Tire].Dimension.Members.Count  
SELECT measures.x on 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Count \(Set\) \(MDX\)](#)

[Levels \(MDX\)](#)

[Members \(Set\) \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Dimensions

Returns a hierarchy specified by a numeric or string expression.

Syntax

Numeric expression syntax

Dimensions(**Hierarchy_Number**)

String expression syntax

Dimensions(**Hierarchy_Name**)

Arguments

Hierarchy_Number

A valid numeric expression that specifies a hierarchy number.

Hierarchy_Name

A valid string expression that specifies a hierarchy name

Remarks

If a hierarchy number is specified, the **Dimensions** function returns a hierarchy whose zero-based position within the cube is specified hierarchy number.

If a hierarchy name is specified, the **Dimensions** function returns the specified hierarchy. Typically, you use this string version of the **Dimensions** function with user-defined functions.

Note

The **Measures** dimension is always represented by `Dimensions(0)`.

Examples

The following examples use the **Dimensions** function to return the name, count of levels, and count of members of a specified hierarchy, using both a numeric expression and a string expression.

```
WITH MEMBER Measures.x AS Dimensions
    ('[Product].[Product Model Lines]').Name
SELECT Measures.x on 0
FROM [Adventure Works]
```

```
WITH MEMBER Measures.x AS Dimensions
    ('[Product].[Product Model Lines]').Levels.Count
SELECT Measures.x on 0
FROM [Adventure Works]
```

```
WITH MEMBER Measures.x AS Dimensions
    ('[Product].[Product Model Lines]').Members.Count
SELECT Measures.x on 0
FROM [Adventure Works]
```

```
WITH MEMBER Measures.x AS Dimensions(0).Name
SELECT Measures.x on 0
FROM [Adventure Works]
```

```
WITH MEMBER Measures.x AS Dimensions(0).Levels.Count
```

```
SELECT measures.x on 0  
FROM [Adventure Works]
```

```
WITH MEMBER Measures.x AS Dimensions(0).Members.Count  
SELECT measures.x on 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Distinct

Evaluates a specified set, removes duplicate tuples from the set, and returns the resulting set.

Syntax

```
Distinct(Set_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

If the **Distinct** function finds duplicate tuples in the specified set, the function keeps only the first instance of the duplicate tuple while leaving the order of the set intact.

Examples

The following example query shows how to use the Distinct function with a named set, as well as how to use it with the Count function to find the number of distinct tuples in a set:

```
WITH SET MySet AS  
{ [Customer].[Customer Geography].[Country].&[Australia], [Customer].[Customer  
Geography].[Country].&[Australia],  
[Customer].[Customer Geography].[Country].&[Canada], [Customer].[Customer  
Geography].[Country].&[France],  
[Customer].[Customer Geography].[Country].&[United  
Kingdom], [Customer].[Customer Geography].[Country].&[United Kingdom] }  
MEMBER MEASURES.SETCOUNT AS
```

```

COUNT (MySet)
MEMBER MEASURES.SETDISTINCTCOUNT AS
COUNT (DISTINCT (MySet) )
SELECT {MEASURES.SETCOUNT, MEASURES.SETDISTINCTCOUNT} ON 0,
DISTINCT (MySet) ON 1
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

DistinctCount

Returns the number of distinct, nonempty tuples in a set.

Syntax

```
DistinctCount(Set_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **DistinctCount** function is equivalent to `Count (Distinct (Set_Expression) , EXCLUDEEMPTY)`.

Examples

The following query shows how to use the DistinctCount function:

```

WITH SET MySet AS
{ [Customer].[Customer Geography].[Country].&[Australia], [Customer].[Customer
Geography].[Country].&[Australia],
[Customer].[Customer Geography].[Country].&[Canada], [Customer].[Customer
Geography].[Country].&[France],
[Customer].[Customer Geography].[Country].&[United
Kingdom], [Customer].[Customer Geography].[Country].&[United Kingdom] }
*
{ ([Date].[Calendar].[Date].&[20010701], [Measures].[Internet Sales Amount] ) }

```



```
//Returns the value 3 because Internet Sales Amount is null
//for the UK on the date specified
MEMBER MEASURES.SETDISTINCTCOUNT AS
DISTINCTCOUNT(MySet)
SELECT {MEASURES.SETDISTINCTCOUNT} ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

DrilldownLevel

Drills down the members of a set to one level below the lowest level represented in the set, or to one level below an optionally specified level of a member represented in the set.

Syntax

Level expression syntax

DrilldownLevel(Set_Expression [, Level_Expression])

Numeric expression syntax

DrilldownLevel(Set_Expression [, ,Index])

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Index

A valid numeric expression that specifies the hierarchy number to drill down into within the set.

Remarks

The **DrilldownLevel** function returns a set of child members in a hierarchical order, based on the members included in the specified set. Order is preserved among the original members in the specified set, except that all child members included in the result set of the function are included immediately under their parent member.

If a level expression is specified, the function constructs a set in a hierarchical order by retrieving the children of only those members that are at the specified level. If a level expression is specified and there is no member at the specified level represented in the specified set, the specified set is returned.

If an index value is specified, the function constructs a set in a hierarchical order by retrieving the children of only those members that are at the next lowest level of the specified hierarchy referenced in the specified set, based on a zero-based index.

If neither a level expression nor an index value is specified, the function constructs a set in a hierarchical order by retrieving the children of only those members that are at the lowest level of the first dimension referenced in the specified set.

Querying the XMLA property `MdpropMdxDrillFunctions` enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

Examples

The following example counts the number of products in the Product dimension by using the `DrilldownLevel` function in conjunction with the `Count` function.

```
Count(DrilldownLevel (
    [Product].[Product].[Product]))
```

The following example uses the numeric expression syntax to drilldown into the first hierarchy, the Customer Geography hierarchy.

```
SELECT DRILLDOWNLEVEL
    ( {[Customer].[Customer Geography].[Country].&[Canada]} *
    {[Customer].[Gender].[All Customers]},,0)
    ON 0
FROM [Adventure Works]
```

The following example uses the numeric expression syntax to drilldown into the second hierarchy, which is the Gender hierarchy.

```
SELECT DRILLDOWNLEVEL
    ( {[Customer].[Customer Geography].[Country].&[Canada]} *
    {[Customer].[Gender].[All Customers]},,1)
    ON 0
FROM [Adventure Works]
```

The following example returns the count of the resellers whose sales have declined over the previous time period, based on user-selected State-Province member values evaluated by using the `Aggregate` function. The `Hierarchize` and `DrilldownLevel` functions are used to return values for declining sales for product categories in the Product dimension. The `DrilldownLevel` function

is used to drill down to the next lowest level of the Product attribute hierarchy (because no level is specified).

```
WITH MEMBER Measures.[Declining Reseller Sales] AS
    Count (
        Filter(
            Existing(Reseller.Reseller.Reseller),
            [Measures].[Reseller Sales Amount] < ([Measures].[Reseller Sales
Amount]),
            [Date].Calendar.PrevMember)
        )
    )
MEMBER [Geography].[State-Province].x AS
    Aggregate (
        {[Geography].[State-Province].&[WA]&[US],
        [Geography].[State-Province].&[OR]&[US] }
    )
SELECT NON EMPTY Hierarchize (
    AddCalculatedMembers (
        {DrilldownLevel ({[Product].[All Products]})})
    ) )
    DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
    [Date].[Calendar].[Calendar Quarter].&[2003]&[4],
    [Measures].[Declining Reseller Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

DrilldownLevelBottom

Drills down the bottommost members of a set, at a specified level, to one level below.

Syntax

DrilldownLevelBottom(Set_Expression, Count [, Level_Expression [,Numeric_Expression]])

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the **DrilldownLevelBottom** function sorts, in ascending order, the children of each member in the specified set, according to the specified value, as evaluated over the set of child members. If a numeric expression is not specified, the function sorts, in ascending order, the children of each member in the specified set, according to the values of the cells represented by the set of child members, as determined by the query context; This behavior is similar to the **BottomCount** and **Tail (MDX)** functions which return a set of members in natural order, without any sorting.

After sorting, the **DrilldownLevelBottom** function returns a set that contains the parent members and the number of child members, specified in **Count**, with the lowest value.

The **DrilldownLevelBottom** function is similar to the **DrilldownLevel** function, but instead of including all children for each member at the specified level, the **DrilldownLevelBottom** function returns the bottom-most number of child members.

Querying the XMLA property `MdpropMdxDrillFunctions` enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

Example

The following example returns the bottom three children of the Product Category level, based on the default measure.

```
SELECT DrilldownLevelBottom
    ([Product].[Product Categories].children,
```

```
3,  
[Product].[Product Categories].[Category])  
ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

DrilldownLevelTop

Drills down the topmost members of a set, at a specified level, to one level below.

Syntax

```
DrilldownLevelTop(<set_expression>, <count> [, [<level_expression>]  
[, [<numeric_expression>][,INCLUDE_CALC_MEMBERS]]])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

INCLUDE_CALC_MEMBERS

A keyword to enable calculated members to be included in drilldown results

Remarks

If a numeric expression is specified, the **DrilldownLevelTop** function sorts, in descending order, the children of each member in the specified set according to the value of the numeric

expression, as evaluated over the set of child members. If a numeric expression is not specified, the function sorts, in descending order, the children of each member in the specified set according to the values of the cells represented by the set of child members, as determined by the query context.

After sorting, the **DrilldownLevelTop** function returns a set that contains the parent members and the number of child members, specified in Count, with the highest value.

The **DrilldownLevelTop** function is similar to the DrilldownLevel function, but instead of including all children for each member at the specified level, the **DrilldownLevelTop** function returns the topmost number of child members.

Querying the XMLA property MdpropMdxDrillFunctions enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

DrilldownMember

Drills down the members in a specified set that are present in a second specified set.

Alternatively, the function drills down on a set of tuples by using the first tuple hierarchy or the optionally specified hierarchy.

Syntax

```
DrillDownMember(<set_expression1>, <set_expression2> [, [<target_hierarchy>]]  
[, [RECURSIVE] [, INCLUDE_CALC_MEMBERS]])
```

Arguments

Term	Definition
Set_Expression1	A valid Multidimensional Expressions (MDX) expression that returns a set.
Set_Expression2	A valid Multidimensional Expressions (MDX) expression that returns a set.
Target_Hierarchy	A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

RECURSIVE	A keyword that indicates recursive comparison of sets.
INCLUDE_CALC_MEMBERS	A keyword to enable calculated members to be included in drilldown results.

Remarks

This function returns a set of child members that are ordered by hierarchy, and includes members specified in the first set that are also present in the second set. Parent members will not be drilled down if the first set contains the parent member and one or more children. The first set can have any dimensionality, but the second set must contain a one-dimensional set. Order is preserved among the original members in the first set, except that all child members included in the result set of the function are included immediately under their parent member. The function constructs the result set by retrieving the children for each member in the first set that is also present in the second set. If **RECURSIVE** is specified, the function continues to recursively compare the members of the result set against the second set, retrieving the children for each member in the result set that is also present in the second set until no more members from the result set can be found in the second set.

Querying the XMLA property `MdpropMdxDrillFunctions` enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

The first set can contain tuples instead of members. Tuple drilldown is an extension of OLE DB, and it returns a set of tuples instead of members.

Important

A member will not get drilled down into if it is immediately followed by one of its children. The order of members in the set matters for both the Drilldown* and Drillup* families of functions.

Examples

The following example drills down into Australia, which is the member of the first set which is also present in the second set.

```
SELECT DrilldownMember
    ( [Geography].[Geography].Children,
      { [Geography].[Geography].[Country].[Australia],
        [Geography].[Geography].[State-Province].[New South Wales] }
    )
ON 0
FROM [Adventure Works]
```

The following example drills down into Australia, which is the member of the first set which is also present in the second set. However, because the RECURSIVE argument is present, the function continues to recursively compare the members of the result set (members of the State-Province level) against the second set, retrieving the children for each member in the result set (members of the City level) that is also present in the second set until no more members from the result set can be found in the second set.

```
SELECT DrilldownMember
( [Geography].[Geography].Children,
  { [Geography].[Geography].[Country].[Australia],
    [Geography].[Geography].[State-Province].[New South Wales] }
, RECURSIVE)
ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

DrilldownMemberBottom

Drills down the members in a specified set that are present in a second specified set, limiting the result set to a specified number of members. Alternatively, this function also drills down on a set of tuples by using the first tuple hierarchy or the optionally specified hierarchy.

Syntax

```
DrillDownMemberBottom(<set_expression1>, <set_expression2>, <count>
[, [<numeric_expression>] [, [<hierarchy>]] [, [RECURSIVE][, INCLUDE_CALC_MEMBERS]])
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression

of cell coordinates that return a number.

Hierarchy	A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.
RECURSIVE	A keyword that indicates recursive comparison of sets.
INCLUDE_CALC_MEMBERS	A keyword to enable calculated members to be included in drilldown results.

Remarks

If a numeric expression is specified, the **DrilldownMemberBottom** function sorts, in ascending order, the children of each member in the first set, according to the value of the numeric expression, as evaluated over the set of child members. If a numeric expression is not specified, the function sorts, in ascending order, the children of each member in the first set according to the values of the cells represented by the set of child members, as determined by the query context. This behavior is similar to the **BottomCount** and **Tail (MDX)** functions which return a set of members in natural order, without any sorting.

After sorting, the **DrilldownMemberBottom** function returns a set that contains the parent members and the number of child members, specified in **Count**, with the lowest value and are contained by both sets.

If **RECURSIVE** is specified, the function sorts the first set as described previously, then recursively compares the members of the first set, as organized in a hierarchy, against the second set. The function retrieves the bottommost number of children for each member in the first set that is also present in the second set.

The first set can contain tuples instead of members. Tuple drilldown is an extension of OLE DB, and returns a set of tuples instead of members.

The **DrilldownMemberBottom** function is similar to the **DrilldownMember** function, but instead of including all children for each member in the first set that is also present in the second set, the **DrilldownMemberBottom** function returns the bottommost number of child members for each member.

Querying the XMLA property **MdpropMdxDrillFunctions** enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

See Also

[MDX Function Reference \(MDX\)](#)

DrilldownMemberTop

Drills down the members in a specified set that are present in a second specified set, limiting the result set to a specified number of members. Alternatively, this function drills down on a set of tuples by using the first tuple hierarchy or the optionally specified hierarchy.

Syntax

```
DrillDownMemberTop(<set_expression1>, <set_expression2>, <count>  
[, [<numeric_expression>] [, [<hierarchy>]] [, [RECURSIVE][, INCLUDE_CALC_MEMBERS]])
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Hierarchy	A valid Multidimensional Expressions (MDX) expression that returns a set.
RECURSIVE	A keyword that indicates recursive comparison of sets.
INCLUDE_CALC_MEMBERS	A keyword to enable calculated members to be included in drilldown results.

Remarks

If a numeric expression is specified, the **DrilldownMemberTop** function sorts, in descending order, the children of each member in the first set according to the value of the numeric expression, as evaluated over the set of child members. If a numeric expression is not specified, the function sorts, in descending order, the children of each member in the first set according to the values of the cells represented by the set of child members, as determined by the query

context. This behavior is similar to the TopCount and Head (MDX) functions which return a set of members in natural order, without any sorting.

After sorting, the **DrilldownMemberTop** function returns a set that contains the parent members and the number of child members, specified in Count, with the highest value and are contained in both sets.

If **RECURSIVE** is specified, the function sorts the first set as described previously, then recursively compares the members of the first set, as organized in a hierarchy, against the second set. The function retrieves the topmost number of children for each member in the first set that is also present in the second set.

The first set can contain tuples instead of members. Tuple drilldown is an extension of OLE DB, and returns a set of tuples instead of members.

The **DrilldownMemberTop** function is similar to the DrilldownMember function, but instead of including all children for each member in the first set that is also present in the second set, the **DrilldownMemberTop** function returns the topmost number of child members for each member.

Querying the XMLA property MdpropMdxDrillFunctions enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

See Also

[MDX Function Reference \(MDX\)](#)

DrillupLevel

Drills up the members of a set that are below a specified level.

Syntax

```
DrillupLevel(Set_Expression [ , Level_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Remarks

The **DrillupLevel** function returns a set of members organized hierarchically based on the members included in the specified set. Order is preserved among the members in the specified set.

If a level expression is specified, the **DrillupLevel** function constructs the set by retrieving only those members that are above the specified level. If a level expression is specified and there is no member of the specified level represented in the specified set, the specified set is returned.

If a level expression is not specified, the function constructs the set by retrieving only those members that are one level higher than the lowest level of the first dimension referenced in the specified set.

Example

The following example returns the set of members from the first set that are above the Subcategory level.

```
SELECT DrillUpLevel
    ( {[Product].[Product Categories].[All Products]
      , [Product].[Product Categories].[Subcategory].&[32],
      [Product].[Product Categories].[Product].&[215]} ,
      [Product].[Product Categories].[Subcategory]
    )
ON 0
FROM [Adventure Works]
WHERE [Measures].[Internet Order Quantity]
```

See Also

[MDX Function Reference \(MDX\)](#)

DrillupMember

Returns the members in a specified set that are not descendants of members in a second specified set.

Syntax

```
DrillupMember(Set_Expression1, Set_Expression2)
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **DrillupMember** function returns a set of members based on the members specified in the first set that are descendants of members in the second set. The first set can have any dimensionality, but the second set must contain a one-dimensional set. Order is preserved among the original members in the first set. The function constructs the set by including only those members in the first set that are immediate descendants of members in the second set. If the immediate ancestor of a member in the first set is not present in the second set, the member in the first set is included in the set returned by this function. Descendants in the first set that precede an ancestor member in the second set are also included.

The first set can contain tuples instead of members. Tuple drilldown is an extension of OLE DB, and returns a set of tuples instead of members.

Important

A member will get drilled up only if it is immediately followed by a child or a descendant. The order of members in the set matters for both the Drilldown* and Drillup* families of functions. Consider using the **Hierarchize** function to appropriately order the members of the first set.

Example

The following example drills up on the United States member, meaning that the member Colorado is not displayed on rows:

```
SELECT DrillUpMember
(
    { [Geography].[Geography].[Country].[Canada]
    , [Geography].[Geography].[Country].[United States]
    , [Geography].[Geography].[State-Province].[Colorado]
    , [Geography].[Geography].[State-Province].[Alberta]
    , [Geography].[Geography].[State-Province].[Brunswick]
    }
    , { [Geography].[Geography].[Country].[United States] }
)
ON 0
```

```
FROM [Adventure Works]
```

However, since DrillupMember only drills up on those members that are followed immediately by descendants in the first set, it does not drill up on the Canada member in the following example:

```
SELECT DrillUpMember
(
    { [Geography].[Geography].[Country].[Canada]
    , [Geography].[Geography].[Country].[United States]
    , [Geography].[Geography].[State-Province].[Colorado]
    , [Geography].[Geography].[State-Province].[Alberta]
    , [Geography].[Geography].[State-Province].[Brunswick]
    }
    , { [Geography].[Geography].[Country].[Canada] }
)
ON 0
FROM [Adventure Works]
```

The following example shows how the use of Hierarchize can avoid this issue, and drills up on the Canada member.

```
SELECT DrillUpMember
(
    Hierarchize
    (
        { [Geography].[Geography].[Country].[Canada]
        , [Geography].[Geography].[Country].[United States]
        , [Geography].[Geography].[State-Province].[Colorado]
        , [Geography].[Geography].[State-Province].[Alberta]
        , [Geography].[Geography].[State-Province].[Brunswick]
        }
        ), { [Geography].[Geography].[Country].[Canada] }
    )
ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Error

Raises an error, optionally providing a specified error message.

Syntax

```
Error( [ Error_Text ] )
```

Arguments

Error_Text

A valid string expression containing the error message to be returned.

Examples

The following query shows how to use the **Error** function inside a calculated measure:

```
WITH MEMBER MEASURES.ERRORDEMO AS ERROR("THIS IS AN ERROR")
SELECT
MEASURES.ERRORDEMO ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Except

Evaluates two sets and removes those tuples in the first set that also exist in the second set, optionally retaining duplicates.

Syntax

```
Except(Set_Expression1, Set_Expression2 [, ALL ] )
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

If **ALL** is specified, the function retains duplicates found in the first set; duplicates found in the second set will still be removed. The members are returned in the order they appear in the first set.

Examples

The following example demonstrates the use of this function.

```
//This query shows the quantity of orders for all products,  
//with the exception of Components, which are not  
//sold.  
  
SELECT  
    [Date].[Month of Year].Children ON COLUMNS,  
    Except  
        ([Product].[Product Categories].[All].Children ,  
         {[Product].[Product Categories].[Components]})  
    ) ON ROWS  
  
FROM  
    [Adventure Works]  
  
WHERE  
    ([Measures].[Order Quantity])
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Exists

Returns the set of tuples of the first set specified that exist with one or more tuples of the second set specified. This function performs manually what auto exists performs automatically. For more information about auto exists, see [IsEmpty \(MDX\)](#).

If the optional <Measure Group Name> is provided, the function returns tuples that exist with one or more tuples from the second set and those tuples that have associated rows in the fact table of the specified measure group.

Syntax

Exists(Set_Expression1 , Set_Expression2 [, MeasureGroupName])

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

MeasureGroupName

A valid string expression specifying a measure group name.

Remarks

Measure group rows with measures containing null values contribute to **Exists** when the MeasureGroupName argument is specified. This is the difference between this form of Exists and the Nonempty function: if the NullProcessing property of these measures is set to Preserve, this means the measures will show Null values when queries are run against that part of the cube; NonEmpty will always remove tuples from a set that that have Null measure values, whereas Exists with the MeasureGroupName argument will not filter tuples that have associated measure group rows, even if the measure values are Null.

Examples

Customers who live in California:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
EXISTS (
  [Customer].[Customer].[Customer].MEMBERS
, { [Customer].[State-Province].&[CA]&[US] }
) ON 1
FROM [Adventure Works]
```

Customers who live in California with sales:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
EXISTS (
  [Customer].[Customer].[Customer].MEMBERS
, { [Customer].[State-Province].&[CA]&[US] }
, "Internet Sales") ON 1
FROM [Adventure Works]
```

Customers with sales:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
EXISTS (
[Customer].[Customer].[Customer].MEMBERS
, , "Internet Sales") ON 1
FROM [Adventure Works]
```

Customers whom bought Bikes:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
EXISTS (
[Customer].[Customer].[Customer].MEMBERS
, {[Product].[Product Categories].[Category].&[1]}
, "Internet Sales") ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Crossjoin \(MDX\)](#)

[NonEmptyCrossjoin \(MDX\)](#)

[NonEmpty \(MDX\)](#)

[IsEmpty \(MDX\)](#)

Extract

Returns a set of tuples from extracted hierarchy elements.

Syntax

```
Extract(Set_Expression, Hierarchy_Expression1 [,Hierarchy_Expression2, ...n])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Hierarchy_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Hierarchy_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

The **Extract** function returns a set that consists of tuples from the extracted hierarchy elements. For each tuple in the specified set, the members of the specified hierarchies are extracted into new tuples in the result set. This function always removes duplicate tuples.

The **Extract** function performs the opposite action of the Crossjoin function.

Examples

The following query shows how to use the **Extract** function on a set of tuples returned by the **NonEmpty** function:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
//Returns the distinct combinations of Customer and Date for all purchases
//of Bike Racks or Bike Stands
EXTRACT (
NONEMPTY (
[Customer].[Customer].[Customer].MEMBERS
*
[Date].[Date].[Date].MEMBERS
*
{[Product].[Product Categories].[Subcategory].&[26],[Product].[Product
Categories].[Subcategory].&[27]}
*
{[Measures].[Internet Sales Amount]}
)
, [Customer].[Customer], [Date].[Date])
ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Filter

Returns the set that results from filtering a specified set based on a search condition.

Syntax

```
Filter(Set_Expression, Logical_Expression )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Logical_Expression

A valid Multidimensional Expressions (MDX) logical expression that evaluates to true or false.

Remarks

The **Filter** function evaluates the specified logical expression against each tuple in the specified set. The function returns a set that consists of each tuple in the specified set where the logical expression evaluates to **true**. If no tuples evaluate to **true**, an empty set is returned.

The **Filter** function works in a fashion similar to that of the **IIf** function. The **IIf** function returns only one of two options based on the evaluation of an MDX logical expression, while the **Filter** function returns a set of tuples that meet the specified search condition. In effect, the **Filter** function executes `IIf(Logical_Expression, Set_Expression.Current, NULL)` on each tuple in the set, and returns the resulting set.

Examples

The following example shows the use of the **Filter** function on the Rows axis of a query, to return only the Dates where Internet Sales Amount is greater than \$10000:

```
SELECT [Measures].[Internet Sales Amount] ON 0,  
FILTER (  
  [Date].[Date].[Date].MEMBERS  
, [Measures].[Internet Sales Amount]>10000)  
ON 1  
FROM  
  [Adventure Works]
```

The **Filter** function can also be using inside calculated member definitions. The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first nine months of 2003 contained in the `Date` dimension, from the **Adventure Works** cube. The **PeriodsToDate** function defines the tuples in the set over which the **Aggregate** function

operates. The **Filter** function limits those tuples being returned to those with lower values for the Reseller Sales Amount measure for the previous time period.

```
WITH MEMBER Measures.[Declining Reseller Sales] AS Count
    (Filter
        (Existing
            (Reseller.Reseller.Reseller),
            [Measures].[Reseller Sales Amount] <
                ([Measures].[Reseller Sales
Amount], [Date].Calendar.PrevMember)
        )
    )
MEMBER [Geography].[State-Province].x AS Aggregate
( {[Geography].[State-Province].&[WA]&[US],
    [Geography].[State-Province].&[OR]&[US] }
)
SELECT NON EMPTY HIERARCHIZE
    (AddCalculatedMembers
        ({DrillDownLevel
            ({[Product].[All Products]})})
    )
    ) DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
    [Date].[Calendar].[Calendar Quarter].&[2003]&[4],
    [Measures].[Declining Reseller Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

FirstChild

Returns the first child of a specified member.

Syntax

Member_Expression.FirstChild

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Example

The following query returns the first child of fiscal year 2003 in the Fiscal hierarchy, which is the first semester of Fiscal Year 2003.

```
SELECT [Date].[Fiscal].[Fiscal Year].&[2003].FirstChild ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

FirstSibling

Returns the first child of the parent of a member.

Syntax

Member_Expression.FirstSibling

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Example

The following query returns the first sibling of fiscal year 2003 in the Fiscal hierarchy, which is Fiscal Year 2002.

```
SELECT [Date].[Fiscal].[Fiscal Year].&[2003].FirstSibling ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Generate

Applies a set to each member of another set, and then joins the resulting sets by union. Alternatively, this function returns a concatenated string created by evaluating a string expression over a set.

Syntax

Set expression syntax

```
Generate( Set_Expression1 , Set_Expression2 [ , ALL ] )
```

String expression syntax

```
Generate( Set_Expression1 , String_Expression [ ,Delimiter ] )
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

String_Expression

A valid string expression that is typically the name of the current member (CurrentMember.Name) of each tuple in the specified set.

Delimiter

A valid delimiter expressed as a string expression.

Remarks

If a second set is specified, the **Generate** function returns a set generated by applying the tuples in the second set to each tuple in the first set, and then joining the resulting sets by union. If **ALL** is specified, the function retains duplicates in the resulting set.

If a string expression is specified, the **Generate** function returns a string generated by evaluating the specified string expression against each tuple in the first set, and then concatenating the results. Optionally, the string can be delimited, separating each result in the resulting concatenated string.

Examples

Set

In the following example, the query returns a set containing the Measure Internet Sales amount four times, because there are four members in the set [Date].[Calendar Year].[Calendar Year].MEMBERS:

```
SELECT
GENERATE( [Date].[Calendar Year].[Calendar Year].MEMBERS
, {[Measures].[Internet Sales Amount]}, ALL)
ON 0
FROM [Adventure Works]
```

Removing the ALL changes the query so that the Internet Sales Amount is returned once only:

```
SELECT
GENERATE( [Date].[Calendar Year].[Calendar Year].MEMBERS
, {[Measures].[Internet Sales Amount]})
ON 0
FROM [Adventure Works]
```

The most common practical use of **Generate** is to evaluate a complex set expression, such as TopCount, over a set of members. The following example query displays the top 10 Products for each Calendar Year on Rows:

```
SELECT
{[Measures].[Internet Sales Amount]}
ON 0,
GENERATE(
[Date].[Calendar Year].[Calendar Year].MEMBERS
, TOPCOUNT(
[Date].[Calendar Year].CURRENTMEMBER
*
[Product].[Product].[Product].MEMBERS
,10, [Measures].[Internet Sales Amount]))
ON 1
FROM [Adventure Works]
```

Note that a different top 10 is displayed for each year, and that the use of **Generate** is the only way to get this result. Simply crossjoining Calendar Years and the set of top 10 Products will display the top 10 Products for all time, repeated for each year, as shown in the following example:

```
SELECT
{[Measures].[Internet Sales Amount]}
```



```

ON 0,
[Date].[Calendar Year].[Calendar Year].MEMBERS
*
TOPCOUNT (
[Product].[Product].[Product].MEMBERS
,10, [Measures].[Internet Sales Amount])
ON 1
FROM [Adventure Works]

```

String

The following example shows the use of **Generate** to return a string:

```

WITH
MEMBER MEASURES.GENERATESTRINGDEMO AS
GENERATE (
[Date].[Calendar Year].[Calendar Year].MEMBERS,
[Date].[Calendar Year].CURRENTMEMBER.NAME)
MEMBER MEASURES.GENERATEDELIMITEDSTRINGDEMO AS
GENERATE (
[Date].[Calendar Year].[Calendar Year].MEMBERS,
[Date].[Calendar Year].CURRENTMEMBER.NAME, " AND ")
SELECT
{MEASURES.GENERATESTRINGDEMO, MEASURES.GENERATEDELIMITEDSTRINGDEMO}
ON 0
FROM [Adventure Works]

```



Note

- This form of the **Generate** function can be useful when debugging calculations, as it enables you to return a string displaying the names of all the members in a set. This might be easier to read than the strict MDX representation of a set that the [SetToStr \(MDX\)](#) function returns.

See Also

[MDX Function Reference \(MDX\)](#)

Head

Returns the first specified number of elements in a set, while retaining duplicates.

Syntax

```
Head(Set_Expression [ ,Count ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Remarks

The **Head** function returns the specified number of tuples from the beginning of the specified set. The order of elements is preserved. The default value of Count is 1. If the specified number of tuples is less than 1, the **Head** function returns an empty set. If the specified number of tuples exceeds the number of tuples in the set, the function returns the original set.

Example

The following example returns top five selling subcategories of products, irrespective of hierarchy, based on Reseller Gross Profit. The **Head** function is used to return only the first 5 sets in the result after the result is ordered using the **Order** function.

```
SELECT
[Measures].[Reseller Gross Profit] ON 0,
Head
    (Order
        ([Product].[Product Categories].[SubCategory].members
        , [Measures].[Reseller Gross Profit]
        , BDESC
        )
    , 5
    ) ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[Item \(Tuple\) \(MDX\)](#)

[Item \(Member\) \(MDX\)](#)

[Rank \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Hierarchize

Orders the members of a set in a hierarchy.

Syntax

```
Hierarchize(Set_Expression [ , POST ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **Hierarchize** function organizes the members of specified set into hierarchical order. The function always retains duplicates.

- If **POST** is not specified, the function sorts members in a level in their natural order. Their natural order is the default ordering of the members along the hierarchy when no other sort conditions are specified. Child members immediately follow their parent members.
- If **POST** is specified, the **Hierarchize** function sorts the members in a level using a post-natural order. In other words, child members precede their parents.

Example

The following example drills up on the Canada member. The **Hierarchize** function is used to organize the specified set members in hierarchical order, which is required by the **DrillUpMember** function.

```
SELECT DrillUpMember
(
    Hierarchize
    (
        { [Geography].[Geography].[Country].[Canada]
        , [Geography].[Geography].[Country].[United States]
        , [Geography].[Geography].[State-Province].[Alberta]
        , [Geography].[Geography].[State-Province].[Brunswick]
```

```

        , [Geography].[Geography].[State-Province].[Colorado]
    }
), {[Geography].[Geography].[Country].[United States]}
)
ON 0
FROM [Adventure Works]

```

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first nine months of 2003 contained in the `Date` dimension, from the **Adventure Works** cube. The **PeriodsToDate** function defines the tuples in the set over which the `Aggregate` function operates. The **Hierarchize** function organizes the members of the specified set of members from the `Product` dimension in hierarchical order.

```

WITH MEMBER Measures.[Declining Reseller Sales] AS Count
    (Filter
        (Existing
            (Reseller.Reseller.Reseller),
            [Measures].[Reseller Sales Amount] <
                ([Measures].[Reseller Sales
Amount], [Date].Calendar.PrevMember)
        )
    )
MEMBER [Geography].[State-Province].x AS Aggregate
    ( {[Geography].[State-Province].&[WA]&[US],
        [Geography].[State-Province].&[OR]&[US] }
    )
SELECT NON EMPTY HIERARCHIZE
    (AddCalculatedMembers
        ({{DrillDownLevel
            ({{[Product].[All Products]}})}
        )
    ) DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
    [Date].[Calendar].[Calendar Quarter].&[2003]&[4],
    [Measures].[Declining Reseller Sales])

```

See Also

[MDX Function Reference \(MDX\)](#)

Hierarchy

Returns the hierarchy that contains a specified member or level.

Syntax

Member expression syntax

`Member_Expression.Hierarchy`

Level expression syntax

`Level_Expression.Hierarchy`

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Examples

The following example returns the name of the Calendar hierarchy in the Data dimension in the AdventureWorks cube.

```
WITH
MEMBER Measures.HierarchyName as
[Date].[Calendar].Currentmember.Hierarchy.Name
SELECT {Measures.HierarchyName} ON 0,
{[Date].[Calendar].[All Periods]} ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

IIf

Returns one of two values determined by a logical test.

Syntax

Iif(Logical_Expression, Expression1, Expression2)

Arguments

Logical_Expression

A valid Multidimensional Expressions (MDX) logical expression that evaluates to true or false.

Expression1

A valid Multidimensional Expressions (MDX) expression.

Expression2

A valid Multidimensional Expressions (MDX) expression.

Remarks

The expression specified by the logical expression evaluates to **false** only if the value of this expression is zero. Any other value evaluates to **true**.

If the specified logical expression evaluates to **true**, the **Iif** function returns the first expression. Otherwise, the function returns the second expression.

The specified expressions can return values or MDX objects. Furthermore, the specified expressions need not match in type.

The **Iif** function is not recommended for creating a set of members based on search criteria. Instead, use the Filter function to evaluate each member in a specified set against a logical expression and return a subset of members.

Note

If either expression evaluates to NULL, the result set will be NULL when that condition is met.

Examples

The following query shows a simple use of **IIF** inside a calculated measure to return one of two different string values when the measure Internet Sales Amount is greater or less than \$10000:

```
WITH MEMBER MEASURES.IIFDEMO AS
IIF([Measures].[Internet Sales Amount]>10000
, "Sales Are High", "Sales Are Low")
SELECT { [Measures].[Internet Sales Amount],MEASURES.IIFDEMO } ON 0,
[Date].[Date].[Date].MEMBERS ON 1
FROM [Adventure Works]
```

A very common use of IIF is to handle 'division by zero' errors within calculated measures, as in the following example:

```
WITH
//Returns 1.#INF when the previous period contains no value
//but the current period does
MEMBER MEASURES.[Previous Period Growth With Errors] AS
([Measures].[Internet Sales Amount]-([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER))
/
([Measures].[Internet Sales Amount], [Date].[Date].CURRENTMEMBER.PREVMEMBER)
,FORMAT_STRING='PERCENT'
//Traps division by zero and returns null when the previous period contains
//no value but the current period does
MEMBER MEASURES.[Previous Period Growth] AS
IIF(([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER)=0,
NULL,
([Measures].[Internet Sales Amount]-([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER))
/
([Measures].[Internet Sales Amount], [Date].[Date].CURRENTMEMBER.PREVMEMBER)
),FORMAT_STRING='PERCENT'
SELECT {[Measures].[Internet Sales Amount],MEASURES.[Previous Period Growth
With Errors], MEASURES.[Previous Period Growth]} ON 0,
DESCENDANTS (
[Date].[Calendar].[Calendar Year].&[2004],
[Date].[Calendar].[Date])
ON 1
FROM [Adventure Works]
WHERE ([Product].[Product Categories].[Subcategory].&[26])
```

The following is an example of **IIF** returning one of two sets inside the Generate function to create a complex set of tuples on Rows:

```
SELECT {[Measures].[Internet Sales Amount]} ON 0,
//If Internet Sales Amount is zero or null
//returns the current year and the All Customers member
//else returns the current year broken down by Country
```

```

GENERATE (
  [Date].[Calendar Year].[Calendar Year].MEMBERS
, IIF([Measures].[Internet Sales Amount]=0,
  {[Date].[Calendar Year].CURRENTMEMBER, [Customer].[Country].[All
Customers]})
, {[Date].[Calendar Year].CURRENTMEMBER} *
[Customer].[Country].[Country].MEMBERS}
))
ON 1
FROM [Adventure Works]
WHERE([Product].[Product Categories].[Subcategory].&[26])

```

Lastly, this example shows how to use Plan Hints:

```

WITH MEMBER MEASURES.X AS
IIF(
  [Measures].[Internet Sales Amount]=0
, NULL
, (1/[Measures].[Internet Sales Amount]) HINT EAGER)
SELECT {[Measures].x} ON 0,
[Customer].[Customer Geography].[Country].MEMBERS ON 1
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

Instr

Returns the position of the first occurrence of one string within another.

Syntax

InStr([start,]ssearched_string, search_string[, compare])

Arguments

start

(Optional) A numeric expression that sets the starting position for each search. If this value is omitted, the search begins at the first character position. If start is null, the function return value is undefined.

searched_string

The string expression to be searched.

search_string

The string expression that is to be searched for.

Compare

(optional) An integer value. This argument is always ignored. It is defined for compatibility with other **Instr** functions in other languages.

Return Value

An integer value with the starting position of String2 in String1.

Also, **InStr** function returns the values listed in the following table depending on the condition:

Condition	Return value
String1 is zero-length	zero (0)
String1 is null	undefined
String2 is zero-length	start
String2 is null	undefined
String2 is not found	zero (0)
start is greater than Len(String2)	zero (0)

Remarks**⚠ Warning**

Instr always performs a case-insensitive comparison.

Example**Description**

The following example shows the usage of the **Instr** function and shows different result scenarios.

Code

```
with
    member [Date].[Date].[Results] as "Results"
```

```

member measures.[lowercase found in lowercase string] as InStr(
"abcdefghijklmnopqrstuvwxyz", "o")
member measures.[uppercase found in lowercase string] as InStr(
"abcdefghijklmnopqrstuvwxyz", "O")
member measures.[searched string is empty] as InStr( "", "o")
member measures.[searched string is null] as
iif(IsError(InStr( null, "o")), "Is Error", iif(IsNull(InStr( null, "o")),
"Is Null","Is undefined"))
member measures.[search string is empty] as InStr(
"abcdefghijklmnopqrstuvwxyz", "")
member measures.[search string is empty start 10] as InStr(10,
"abcdefghijklmnopqrstuvwxyz", "")
member measures.[search string is null] as
iif(IsError(InStr( null, "o")), "Is Error", iif(IsNull(InStr( null, "o")),
"Is Null","Is undefined"))
member measures.[found from start 10] as InStr( 10,
"abcdefghijklmnopqrstuvwxyz", "o")
member measures.[NOT found from start 17] as InStr( 17,
"abcdefghijklmnopqrstuvwxyz", "o")
member measures.[NULL start] as
iif(IsError(InStr( null, "abcdefghijklmnopqrstuvwxyz", "o")), "Is Error",
iif(IsNull(InStr( null, "abcdefghijklmnopqrstuvwxyz", "o")), "Is Null","Is
undefined"))
member measures.[start greater than searched length] as InStr( 170,
"abcdefghijklmnopqrstuvwxyz", "o")

```

```

select [Results] on columns,
{ measures.[lowercase found in lowercase string]
, measures.[uppercase found in lowercase string]
, measures.[searched string is empty]
, measures.[searched string is null]
, measures.[search string is empty]
, measures.[search string is empty start 10]
, measures.[search string is null]
, measures.[found from start 10]

```

```

, measures.[NOT found from start 17]
, measures.[NULL start]
, measures.[start greater than searched length]
} on rows

```

from [Adventure Works]

Comments

The following table displays the obtained results.

	Results
lowercase found in lowercase string	16
uppercase found in lowercase string	16
searched string is empty	0
searched string is null	Is undefined
search string is empty	1
search string is empty start 10	10
search string is null	Is undefined
found from start 10	16
NOT found from start 17	0
NULL start	Is undefined
start greater than searched length	0

Intersect

Returns the intersection of two input sets, optionally retaining duplicates.

Syntax

```
Intersect(Set_Expression1 , Set_Expression2 [, ALL ])
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **Intersect** function returns the intersection of two sets. By default, the function removes duplicates from both sets prior to intersecting the sets. The two sets specified must have the same dimensionality.

The optional **ALL** flag retains duplicates. If **ALL** is specified, the **Intersect** function intersects nonduplicated elements as usual, and also intersects each duplicate in the first set that has a matching duplicate in the second set. The two sets specified must have the same dimensionality.

Example

The following query returns the Years 2003 and 2004, the two members that appear in both the sets specified:

```
SELECT
INTERSECT (
{[Date].[Calendar Year].&[2001], [Date].[Calendar
Year].&[2002],[Date].[Calendar Year].&[2003]}
, {[Date].[Calendar Year].&[2002],[Date].[Calendar Year].&[2003],
[Date].[Calendar Year].&[2004]})
ON 0
FROM
[Adventure Works]
```

The following query fails because the two sets specified contain members from different hierarchies:

```
SELECT
INTERSECT (
{[Date].[Calendar Year].&[2001]}
, {[Customer].[City].&[Abingdon]&[ENG]})
ON 0
FROM
[Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

IsAncestor

Returns whether a specified member is an ancestor of another specified member.

Syntax

IsAncestor(**Member_Expression1**, **Member_Expression2**)

Arguments

Member_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a member.

Member_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **IsAncestor** function returns **true** if the first member specified is an ancestor of the second member specified. Otherwise, the function returns **false**.

Example

The following example returns **true** if [Date].[Fiscal].CurrentMember is an ancestor of January 2003:

```
WITH MEMBER MEASURES.ISANCESTORDEMO AS
IsAncestor([Date].[Fiscal].CurrentMember,
[Date].[Fiscal].[Month].&[2003]&[1])
SELECT MEASURES.ISANCESTORDEMO ON 0,
[Date].[Fiscal].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

IsEmpty

Returns whether the evaluated expression is the empty cell value.

Syntax

```
IsEmpty(value_Expression)
```

Arguments

Value_Expression

A valid Multidimensional Expressions (MDX) expression that typically returns the cell coordinates of a member or a tuple.

Remarks

The **IsEmpty** function returns **true** if the evaluated expression is an empty cell value. Otherwise, this function returns **false**.



Note

The default property for a member is the value of the member.

The **IsEmpty** function is the only way to reliably test for an empty cell because the empty cell value has special meaning in Microsoft SQL Server Analysis Services.



Important

If the evaluation of the value expression returns an error, the function will return **false**. A value expression can return an error, for example, if a properties reference refers to an invalid or non-existent property.

For more information about empty cells, see the OLE DB documentation.

Example

The following example returns TRUE if the Internet Sales Amount for the current member on the Fiscal hierarchy of the Date dimension returns an empty cell:

```
WITH MEMBER MEASURES.ISEMPYDEMO AS  
IsEmpty([Measures].[Internet Sales Amount])  
SELECT {[Measures].[Internet Sales Amount],MEASURES.ISEMPYDEMO} ON 0,  
[Date].[Fiscal].MEMBERS ON 1  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

IsGeneration

Returns whether a specified member is in a specified generation.

Syntax

IsGeneration(**Member_Expression**, **Generation_Number**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Generation_Number

A valid numeric expression that specifies the generation against which the specified member is evaluated.

Remarks

The **IsGeneration** function returns **true** if the specified member is in the specified generation number. Otherwise, the function returns **false**. Also, if the specified member evaluates to an empty member, the **IsGeneration** function returns **false**.

For the purposes of generation indexing, leaf members are generation index 0. The generation index of nonleaf members is determined by first getting the highest generation index from the union of all child members for the specified member, then adding 1 to that index. Because of how the generation index of nonleaf members is determined, a specific nonleaf member could belong to more than one generation.

Example

The following example returns TRUE if [Date].[Fiscal].CurrentMember is part of the second generation:

```
WITH MEMBER MEASURES.ISGENERATIONDEMO AS  
IsGeneration([Date].[Fiscal].CURRENTMEMBER, 2)  
SELECT {MEASURES.ISGENERATIONDEMO} ON 0,  
[Date].[Fiscal].MEMBERS ON 1  
FROM [Adventure Works]
```

See Also

IsLeaf

Returns whether a specified member is a leaf member.

Syntax

```
IsLeaf(Member_Expression)
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **IsLeaf** function returns **true** if the specified member is a leaf member. Otherwise, the function returns **false**.

Example

The following example returns TRUE if [Date].[Fiscal].CurrentMember is a leaf member:

```
WITH MEMBER MEASURES.ISLEAFDEMO AS  
IsLeaf([Date].[Fiscal].CURRENTMEMBER)  
SELECT {MEASURES.ISLEAFDEMO} ON 0,  
[Date].[Fiscal].MEMBERS ON 1  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

IsSibling

Returns whether a specified member is a sibling of another specified member.

Syntax

```
IsSibling(Member_Expression1, Member_Expression2)
```


Arguments

Member_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a member.

Member_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **IsSibling** function returns **true** if the first specified member is a sibling of the second specified member. Otherwise, the function returns **false**.

Example

The following example returns TRUE if the current member on the Fiscal hierarchy of the Date dimension is a sibling of July 2002:

```
WITH MEMBER MEASURES.ISSIBLINGDEMO AS
IsSibling([Date].[Fiscal].CURRENTMEMBER, [Date].[Fiscal].[Month].&[2002]&[7])
SELECT {MEASURES.ISSIBLINGDEMO} ON 0,
[Date].[Fiscal].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Item (Member)

Returns a member from a specified tuple.

Syntax

```
Tuple_Expression.Item( Index )
```

Arguments

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Index

A valid numeric expression that specifies the specific member by position within the tuple to be returned.

Remarks

The **Item** function returns a member from the specified tuple. The function returns the member found at the zero-based position specified by Index.

Example

The following example returns the member [2003] - the first item in the tuple [Date].[Calendar Year].&[2003], [Measures].[Internet Sales Amount]). - on columns :

```
SELECT
{( [Date].[Calendar Year].&[2003], [Measures].[Internet Sales Amount]
).Item(0) } ON 0
, {[Measures].[Reseller Sales Amount]} ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Item (Tuple)

Returns a tuple from a set.

Syntax

Index syntax

```
Set_Expression.Item(Index)
```

String expression syntax

```
Set_Expression.Item(String_Expression1 [,String_Expression2,...n])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

String_Expression1

A valid string expression that is a typically a tuple expressed in a string.

String_Expression2

A valid string expression that is a typically a tuple expressed in a string.

Index

A valid numeric expression that specifies the specific tuple by position within the set to be returned.

Remarks

The **Item** function returns a tuple from the specified set. There are three possible ways to call the **Item** function:

- If a single string expression is specified, the **Item** function returns the specified tuple. For example, "([2005].Q3, [Store05])".
- If more than one string expression is specified, the **Item** function returns the tuple defined by the specified coordinates. The number of strings must match the number of axis, and each string must identify a unique hierarchy. For example, "[2005].Q3", "[Store05]".
- If an integer is specified, the **Item** function returns the tuple that is in the zero-based position specified by Index.

Examples

The following example returns ([1996],Sales):

```
{([1996],Sales), ([1997],Sales), ([1998],Sales)}.Item(0)
```

The following example uses a level expression and returns the Internet Sales Amount for each State-Province in Australia and its percent of the total Internet Sales Amount for Australia. This example uses the Item function to extract the first (and only tuple) from the set returned by the **Ancestors** function.

```
WITH MEMBER Measures.x AS [Measures].[Internet Sales Amount] /  
    ( [Measures].[Internet Sales Amount],  
      Ancestors  
        ( [Customer].[Customer Geography].CurrentMember,  
          [Customer].[Customer Geography].[Country]  
        ).Item (0)  
    ), FORMAT_STRING = '0%'  
SELECT {[Measures].[Internet Sales Amount], Measures.x} ON 0,  
{ Descendants  
    ( [Customer].[Customer Geography].[Country].&[Australia],  
      [Customer].[Customer Geography].[State-Province], SELF  
    )  
} ON 1  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

KPIGoal

Returns the member that calculates the value for the goal portion of the specified Key Performance Indicator (KPI).

Syntax

```
KPIGoal(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of a KPI.

Remarks

Example

The following example returns the KPI value, KPI goal, KPI status, and KPI trend for the channel revenue measure for the descendants of three members of the Fiscal Year attribute hierarchy:

```
SELECT
    { KPIValue("Channel Revenue"),
      KPIGoal("Channel Revenue"),
      KPIStatus("Channel Revenue"),
      KPI_Trend("Channel Revenue")
    } ON Columns,
Descendants
    ( { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003],
        [Date].[Fiscal].[Fiscal Year].&[2004]
      }, [Date].[Fiscal].[Fiscal Quarter]
    ) ON Rows
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

KPIStatus

Returns a normalized value that represents the status portion of the specified Key Performance Indicator (KPI).

Syntax

```
KPIStatus(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of the KPI.

Remarks

The status value is generally a normalized value between -1 and 1.

Example

The following example returns the KPI value, KPI goal, KPI status, and KPI trend for the channel revenue measure for the descendants of three members of the Fiscal Year attribute hierarchy:

```
SELECT
    { KPIValue("Channel Revenue"),
      KPIGoal("Channel Revenue"),
      KPIStatus("Channel Revenue"),
      KPIITrend("Channel Revenue")
    } ON Columns,
Descendants
    ( { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003],
        [Date].[Fiscal].[Fiscal Year].&[2004]
      }, [Date].[Fiscal].[Fiscal Quarter]
    ) ON Rows
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

KPITrend

Returns the normalized value that represents the trend portion of the specified Key Performance Indicator (KPI).

Syntax

```
KPITrend(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of the KPI.

Remarks

The trend value is generally a normalized value between -1 and 1.

Example

The following example returns the KPI value, KPI goal, KPI status, and KPI trend for the channel revenue measure for the descendants of three members of the Fiscal Year attribute hierarchy:

```
SELECT
    { KPIValue("Channel Revenue"),
      KPIGoal("Channel Revenue"),
      KPIStatus("Channel Revenue"),
      KPITrend("Channel Revenue")
    } ON Columns,
Descendants
    ( { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003],
        [Date].[Fiscal].[Fiscal Year].&[2004]
      }, [Date].[Fiscal].[Fiscal Quarter]
    ) ON Rows
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

KPIWeight

Returns the weight of the specified Key Performance Indicator (KPI).

Syntax

```
KPIWeight(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of the KPI.

Remarks

The value returned is the contribution of the KPI to the parent.

See Also

[MDX Function Reference \(MDX\)](#)

KPICurrentTimeMember

Returns the current time member of the specified Key Performance Indicator (KPI).

Syntax

```
KPICurrentTimeMember(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of the KPI.

Remarks

A KPI can have a different time member from the default member of the time dimension.

See Also

[MDX Function Reference \(MDX\)](#)

KPIValue

Returns the member that calculates the value of the specified Key Performance Indicator (KPI).

Syntax

```
KPIValue(KPI_Name)
```

Arguments

KPI_Name

A valid string expression that specifies the name of the KPI.

Remarks

Example

The following example returns the KPI value, KPI goal, KPI status, and KPI trend for the channel revenue measure for the descendants of three members of the Fiscal Year attribute hierarchy.

```
SELECT
    { KPIValue("Channel Revenue"),
      KPIGoal("Channel Revenue"),
      KPIStatus("Channel Revenue"),
      KPI_Trend("Channel Revenue")
    } ON Columns,
Descendants
    ( { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003],
        [Date].[Fiscal].[Fiscal Year].&[2004]
      }, [Date].[Fiscal].[Fiscal Quarter]
    ) ON Rows
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Lag

Returns the member that is a specified number of positions before a specified member at the member's level.

Syntax

```
Member_Expression.Lag(Index)
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Index

A valid numeric expression that specifies the number of member positions to lag.

Remarks

Member positions within a level are determined by the attribute hierarchy's natural order. The numbering of the positions is zero-based.

If the specified lag is zero, the **Lag** function returns the specified member itself.

If the specified lag is negative, the **Lag** function returns a subsequent member.

`Lag(1)` is equivalent to the `PrevMember` function. `Lag(-1)` is equivalent to the `NextMember` function.

The **Lag** function is similar to the `Lead` function, except that the **Lead** function looks in the opposite direction to the **Lag** function. That is, `Lag(n)` is equivalent to `Lead(-n)`.

Example

The following example returns the value for December 2001:

```
SELECT [Date].[Fiscal].[Month].[February 2002].Lag(2) ON 0  
FROM [Adventure Works]
```

The following example returns the value for March 2002:

```
SELECT [Date].[Fiscal].[Month].[February 2002].Lag(-1) ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

LastChild

Returns the last child of a specified member.

Syntax

`Member_Expression.LastChild`

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Example

The following example returns the value for September 2001, which is the last child of the first fiscal quarter of fiscal year 2002.

```
SELECT [Date].[Fiscal].[Fiscal Quarter].[Q1 FY 2002].LastChild ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

LastPeriods

Returns a set of members up to and including a specified member.

Syntax

`LastPeriods(Index [,Member_Expression])`

Arguments

Index

A valid numeric expression that specifies a number of periods.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

If the specified number of periods is positive, the **LastPeriods** function returns a set of members that start with the member that lags Index - 1 from the specified member expression, and ends with the specified member. The number of members returned by the function is equal to Index.

If the specified number of periods is negative, the **LastPeriods** function returns a set of members that start with the specified member and ends with the member that leads (- Index - 1) from the specified member. The number of members returned by the function is equal to the absolute value of Index.

If the specified number of periods is zero, the **LastPeriods** function returns the empty set. This is unlike the **Lag** function, which returns the specified member if 0 is specified.

If a member is not specified, the **LastPeriods** function uses **Time.CurrentMember**. If no dimension is marked as a Time dimension, the function will parse and execute without an error, but will cause a cell error in the client application.

Examples

The following example returns the default measure value for the second third, and fourth fiscal quarters of fiscal year 2002.

```
SELECT LastPeriods(3,[Date].[Fiscal].[Fiscal Quarter].[Q4 FY 2002]) ON 0
FROM [Adventure Works]
```

Note

- This example can also be written using the : (colon) operator:
- [Date].[Fiscal].[Fiscal Quarter].[Q4 FY 2002]: [Date].[Fiscal].[Fiscal Quarter].[Q2 FY 2002]

The following example returns the default measure value for the first fiscal quarter of fiscal year 2002. Although the specified number of periods is three, only one can be returned because there are no earlier periods in the fiscal year.

```
SELECT LastPeriods
    (3,[Date].[Fiscal].[Fiscal Quarter].[Q1 FY 2002]
    ) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

LastSibling

Returns the last child of the parent of a specified member.

Syntax

```
Member_Expression.LastSibling
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Example

The following example returns the default measure for the last day in July 2002.

```
SELECT [Date].[Fiscal].[Date].&[20020717].LastSibling
    ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Lead

Returns the member that is a specified number of positions following a specified member along the member's level.

Syntax

```
Member_Expression.Lead( Index )
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Index

A valid numeric expression that specifies a number of member positions.

Remarks

Member positions within a level are determined by the attribute hierarchy's natural order. The numbering of the positions is zero-based.

If the specified lead is zero (0), the **Lead** function returns the specified member.

If the specified lead is negative, the **Lead** function returns a prior member.

`Lead(1)` is equivalent to the `NextMember` function. `Lead(-1)` is equivalent to the `PrevMember` function.

The **Lead** function is similar to the `Lag` function, except that the **Lag** function looks in the opposite direction to the **Lead** function. That is, `Lead(n)` is equivalent to `Lag(-n)`.

Example

The following example returns the value for December 2001:

```
SELECT [Date].[Fiscal].[Month].[February 2002].Lead(-2) ON 0
FROM [Adventure Works]
```

The following example returns the value for March 2002:

```
SELECT [Date].[Fiscal].[Month].[February 2002].Lead(1) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Leaves

Returns a set composed of all attributes (optionally limited to those belonging to a specific dimension). For each attribute *x* in the return set, if *x* is the granularity attribute or is directly or indirectly related to the granularity attribute, the granularity is set on attribute *x* without affecting the slice. The **Leaves** function is designed for use inside a `SCOPE` statement or at the left side of an assignment.

Syntax

```
Leaves( [ Dimension_expression ] )
```

Arguments

Dimension_Expression

A valid Multidimensional Expressions (MDX) expression that returns a dimension.

Remarks

Leaf members are tuples that are formed by the cross join of the lowest level of all attribute hierarchies. Calculated members are excluded.

- If a dimension name is specified, the **Leaves** function returns a set that contains the leaf members of the key attribute for the specified dimension.
- If the dimension is associated with multiple measure groups, that of the measure in the current scope is used.
- If a dimension name is not specified, the function returns a set that contains the leaf members of the entire cube.



Note

If the dimension expression resolves to a hierarchy, and the hierarchy unique name is the same as the dimension unique name (cube dimension property `HierarchyUniqueNameStyle=ExcludeDimensionName`, and the hierarchy name=dimension name), then the dimension is used.



Important

An error is generated if not all attributes have same granularity on measure groups in current scope.

See Also

[MDX Function Reference \(MDX\)](#)

Level

Returns the level of a member.

Syntax

`Member_Expression.Level`

Arguments

Member_Expression

A valid Multidimensional Expression (MDX) that returns a member.

Examples

The following example uses the **Level** function to return all months in the Adventure Works cube.

```
SELECT [Date].[Fiscal].[Month].[February 2002].Level.Members ON 0,  
[Measures].[Internet Sales Amount] ON 1  
FROM [Adventure Works]
```

The following example uses the **Level** function to return the name of the level for the All-Purpose Bike Stand in the Model Name attribute hierarchy in the Adventure Works cube.

```
WITH MEMBER Measures.x AS  
    [Product].[Model Name].[All-Purpose Bike Stand].Level.Name  
SELECT Measures.x ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Levels

Returns the level whose position in a dimension or hierarchy is specified by a numeric expression or whose name is specified by a string expression.

Syntax

Numeric expression syntax

```
Hierarchy_Expression.Levels( Level_Number )
```

String expression syntax

```
Hierarchy_Expression.Levels( Level_Name )
```

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Number

A valid numeric expression that specifies a level number.

Level_Name

A valid string expression that specifies a level name.

Remarks

If a level number is specified, the **Levels** function returns the level associated with the specified zero-based position.

If a level name is specified, the **Levels** function returns the specified level.



Note

Use the string expression syntax for user-defined functions.

Examples

The following examples illustrate each of the **Levels** function syntaxes.

Numeric

The following example returns the Country level:

```
SELECT [Geography].[Geography].Levels(1) ON 0
FROM [Adventure Works]
```

String

The following example returns the Country level:

```
SELECT [Geography].[Geography].Levels('Country') ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

LinkMember

Returns the member equivalent to a specified member in a specified hierarchy.

Syntax

```
LinkMember(Member_Expression, Hierarchy_Expression)
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

The **LinkMember** function returns the member from the specified hierarchy that matches the key values at each level of the specified member in a related hierarchy. Attributes at each level must have the same key cardinality and data type. In unnatural hierarchies, if there is more than one match for an attribute's key value, the result will be an error or indeterminate.

Examples

The following example uses the **LinkMember** function to return the default measure in the Adventure Works cube for the ascendants of the July 1, 2002 member of the Date.Date attribute hierarchy in the Calendar hierarchy.

```
SELECT Hierarchize
    (Ascendants
        (LinkMember
            ([Date].[Date].[July 1, 2002], [Date].[Calendar]
        )
    )
) ON 0
FROM [Adventure Works]
```

See Also

[Hierarchize \(MDX\)](#)

[Ascendants \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

LinRegIntercept

Calculates the linear regression of a set and returns the value of the x-intercept in the regression line, .

Syntax

```
LinRegIntercept(Set_Expression, Numeric_Expression_y [, Numeric_Expression_x ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

Linear regression, that uses the least-squares method, calculates the equation of a regression line (that is, the best-fit line for a series of points). The regression line has the following equation, where m is the slope and b is the intercept:

The **LinRegIntercept** function evaluates the specified set against the first numeric expression to obtain the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if specified, to obtain the values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as values for the x-axis. Not specifying the the x-axis argument is frequently used with the Time dimension.

After obtaining the set of points, the **LinRegIntercept** function returns the intercept of the regression line (b in the previous equation).

Note

The **LinRegIntercept** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

Example

The following example returns the intercept of a regression line for the unit sales and the store sales measures.

```
LinRegIntercept (LastPeriods(10), [Measures].[Unit Sales], [Measures].[Store Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

LinRegPoint

Calculates the linear regression of a set, and returns the value of the y-intercept in the regression line, b , for a particular value of x .

Syntax

```
LinRegPoint(Slice_Expression_x, Set_Expression, Numeric_Expression_y [  
,Numeric_Expression_x ] )
```

Arguments

Slice_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the slicer axis.

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

Linear regression, that uses the least-squares method, calculates the equation of a regression line (that is, the best-fit line for a series of points). The regression line has the following equation, where m is the slope and b is the intercept:

The **LinRegPoint** function evaluates the specified set against the second numeric expression to obtain the values for the y-axis. The function then evaluates the specified set against the third numeric expression, if specified, to get the values for the x-axis. If the third numeric expression is not specified, the function uses the current context of the cells in the specified set as the values for the x-axis. Not specifying the x-axis argument is frequently used with the Time dimension. Once the linear regression line has been calculated, the value of the equation is calculated for the first numeric expression and then returned.

Note

The **LinRegPoint** function ignores empty cells or cells that contain text. However, the function includes cells with values of zero.

Example

The following example returns the predicted value of Unit Sales over the past ten periods based on the statistical relationship between Unit Sales and Store Sales.

```
LinRegPoint([Measures].[Unit Sales],LastPeriods(10),[Measures].[Unit Sales],[Measures].[Store Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

LinRegR2

Calculates the linear regression of a set and returns the coefficient of determination, R^2 .

Syntax

```
LinRegR2(Set_Expression, Numeric_Expression_y [ ,Numeric_Expression_x ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

Linear regression, that uses the least-squares method, calculates the equation of a regression line (that is, the best-fit line for a series of points). The regression line has the following equation, where m is the slope and b is the intercept:

The **LinRegR2** function evaluates the specified set against the first numeric expression to obtain the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if specified, to obtain the values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as the values for the x-axis. Not specifying the x-axis argument is frequently used with the Time dimension.

After obtaining the set of points, the **LinRegR2** function returns the statistical R^2 that describes the fit of the linear equation to the points.



Note

The **LinRegR2** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

Example

The following example returns the statistical R^2 that describes the goodness of fit of the linear regression equation to the points for the unit sales and the store sales measures.

```
LinRegR2 (LastPeriods (10), [Measures].[Unit Sales],[Measures].[Store Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

LinRegSlope

Calculates the linear regression of a set, and returns the value of the slope in the regression line,

Syntax

```
LinRegSlope(Set_Expression, Numeric_Expression_y [ ,Numeric_Expression_x ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

Linear regression, that uses the least-squares method, calculates the equation of a regression line (that is, the best-fit line for a series of points). The regression line has the following equation, where m is the slope and b is the intercept:

The **LinRegSlope** function evaluates the specified set against the first numeric expression to obtain the values for the y-axis. The function then evaluates the specified set expression against the second numeric expression, if specified, to get the values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as the values for the x-axis. Not specifying the x-axis argument is frequently used with the Time dimension.

After obtaining the set of points, the **LinRegSlope** function returns the slope of the regression line (in the previous equation).

Note

The **LinRegSlope** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

Example

The following example returns the slope of a regression line for the unit sales and the store sales measures.

```
LinRegSlope (LastPeriods(10), [Measures].[Unit Sales], [Measures].[Store Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

LinRegVariance

Calculates the linear regression of a set, and returns the variance associated with the regression line, .

Syntax

```
LinRegVariance(Set_Expression, Numeric_Expression_y [,Numeric_Expression_x ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression_y

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the y-axis.

Numeric_Expression_x

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number that represents values for the x-axis.

Remarks

Linear regression, that uses the least-squares method, calculates the equation of a regression line (that is, the best-fit line for a series of points). The regression line has the following equation, where m is the slope and b is the intercept:

The **LinRegVariance** function evaluates the specified set against the first numeric expression to obtain the values for the y-axis. The function then evaluates the specified set against the second numeric expression, if specified, to obtain the values for the x-axis. If the second numeric expression is not specified, the function uses the current context of the cells in the specified set as the values for the x-axis. Not specifying the x-axis argument is frequently used with the Time dimension.

After obtaining the set of points, the **LinRegVariance** function returns the statistical variance that describes the fit of the linear equation to the points.



Note

The **LinRegVariance** function ignores empty cells or cells that contain text or logical values. However, the function includes cells with values of zero.

Example

The following example returns the statistical variance that describes the fit of the linear equation to the points for the unit sales and the store sales measures.

```
LinRegVariance (LastPeriods (10), [Measures].[Unit Sales], [Measures].[Store Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

LookupCube

Returns the value of a Multidimensional Expressions (MDX) expression evaluated over another specified cube in the same database.

Syntax

Numeric expression syntax

```
LookupCube(Cube_Name, Numeric_Expression)
```

String expression syntax

```
LookupCube(Cube_Name, String_Expression)
```

Arguments

Cube_Name

A valid string expression that specifies the name of a cube.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

String_Expression

A valid string expression that is typically a valid Multidimensional Expressions (MDX) expression of cell coordinates that returns a string.

Remarks

If a numeric expression is specified, the **LookupCube** function evaluates the specified numeric expression in the specified cube and returns the resulting numeric value.

If a string expression is specified, the **LookupCube** function evaluates the specified string expression in the specified cube and returns the resulting string value.

The **LookupCube** function works on cubes within the same database as the source cube on which the MDX query that contains the **LookupCube** function is running.

Important

You must provide any necessary current members in the numeric or string expression because the context of the current query does not carry over to the cube being queried.

Any calculation using the **LookupCube** function is likely to suffer from poor performance. Instead of using this function, consider redesigning your solution so that all of the data you need is present in one cube.

Examples

The following query demonstrates the use of **LookupCube**:

```
WITH MEMBER MEASURES.LOOKUPCUBEDEMO AS
LOOKUPCUBE("Adventure Works", "[Measures].[In" + "ternet Sales Amount]")
SELECT MEASURES.LOOKUPCUBEDEMO ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Max

Returns the maximum value of a numeric expression that is evaluated over a set.

Syntax

Max(Set_Expression [, Numeric_Expression])

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the specified numeric expression is evaluated across the set and then returns the maximum value from that evaluation. If a numeric expression is not specified, the specified set is evaluated in the current context of the members of the set and then returns the maximum value from that evaluation.



Note

Analysis Services ignores nulls when calculating the maximum value in a set of numbers.

Example

The following example returns the maximum monthly sales for each quarter, subcategory, and country in the Adventure Works cube.

```
WITH MEMBER Measures.x AS Max
    ([Date].[Calendar].CurrentMember.Children
    , [Measures].[Reseller Order Quantity]
    )
SELECT Measures.x ON 0
, NON EMPTY [Date].[Calendar].[Calendar Quarter]*
    [Product].[Product Categories].[Subcategory].members *
    [Geography].[Geography].[Country].Members
ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

MeasureGroupMeasures

Returns a set of measures that belongs to the specified measure group.

Syntax

```
MEASUREGROUPMEASURES(MeasureGroupName)
```

Arguments

MeasureGroupName

A valid string expression that contains the name of the measure group from which to retrieve the set of measures.

Remarks

The specified string must match the measure group name exactly. Square brackets for measure group names with spaces are not required.

Example

The following example returns all of the measures in the Internet Sales measure group in the Adventure Works cube.

```
SELECT MeasureGroupMeasures('Internet Sales') ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Median

Returns the median value of a numeric expression that is evaluated over a set.

Syntax

```
Median(Set_Expression [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the specified numeric expression is evaluated across the set and then returns the median value from that evaluation. If a numeric expression is not specified, the specified set is evaluated in the current context of the members of the set and returns the median value from the evaluation.

The median value is the middle value in a set of ordered numbers. (The medial value is unlike the mean value, which is the sum of a set of numbers divided by the count of numbers in the set). The median value is determined by choosing the smallest value such that at least half of the values in the set are no greater than the chosen value. If the number of values within the set is odd, the median value corresponds to a single value. If the number of values within the set is even, the median value corresponds to the sum of the two middle values divided by two.

Note

Analysis Services ignores nulls when calculating the median value in a set of ordered numbers.

Example

The following example returns the median monthly sales for each quarter, each subcategory, and each country in the Adventure Works cube.

```
WITH MEMBER Measures.x AS Median
    ([Date].[Calendar].CurrentMember.Children
    , [Measures].[Reseller Order Quantity]
    )
SELECT Measures.x ON 0
, NON EMPTY [Date].[Calendar].[Calendar Quarter]*
    [Product].[Product Categories].[Subcategory].members *
    [Geography].[Geography].[Country].Members
ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Members (Set)

Returns the set of members in a dimension, level, or hierarchy.

Syntax

Hierarchy expression syntax

```
Hierarchy_Expression.Members
```

Level expression Syntax

```
Level_Expression.Members
```

Arguments

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Remarks

If a hierarchy expression is specified, the **Members (Set)** function returns the set of all members within the specified hierarchy, not including calculated members. To obtain the set of all members, calculated or otherwise, on a hierarchy use the [AllMembers \(MDX\)](#) function

If a level expression is specified, the **Members (Set)** function returns the set of all members within the specified level.

Important

When a dimension contains only a single visible hierarchy, the hierarchy can be either referred to either by the dimension name or by the hierarchy name, because the dimension name in this scenario is resolved to its only visible hierarchy. For example, Measures.Members is a valid MDX expression because it resolves to the only hierarchy in the Measures dimension.

Examples

The following example returns the set of all members of the Calendar Year hierarchy in the Adventure Works cube.

```

SELECT
    [Date].[Calendar].[Calendar Year].Members ON 0
FROM
    [Adventure Works]

```

The following example returns the 2003 order quantities for each member in the [Product].[Products].[Product Line] level. The **Members** function returns a set that represents all of the members in the level.

```

SELECT
    {Measures.[Order Quantity]} ON COLUMNS,
    [Product].[Product Line].[Product Line].Members ON ROWS
FROM
    [Adventure Works]
WHERE
    {[Date].[Calendar Year].[Calendar Year].&[2003]}

```

See Also

[MDX Function Reference \(MDX\)](#)

[AllMembers](#)

Members (String)

Returns a member specified by a string expression.

Syntax

Members(**Member_Name**)

Arguments

Member_Name

A valid string expression that specifies a member name.

Remarks

The **Members (String)** function returns a single member whose name is specified. Typically, you use the **Members (String)** function with external functions, providing to the **Members (String)**

function a string that identifies a member, and the **Members (String)** function returns the value for this specified member.

Example

The following example uses the **Members (String)** function to convert the specified string to a valid member, and then returns the default measure for the member specified in the string. The specified string is in single quotes. The default measure is the Reseller Sales Amount measure.

```
SELECT Members ('[Geography].[Geography].[Country].&[United States] ') ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

MemberToStr

Returns a Multidimensional Expressions (MDX)-formatted string that corresponds to a specified member.

Syntax

MemberToStr(**Member_Expression**)

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

This function returns a string containing the uniqueness of a member. It is usually used to pass a member's uniqueness to an external function.

Example

The following example returns the string [Geography].[Geography].[Country].&[United States] :

```
WITH MEMBER Measures.x AS MemberToStr  
    ([Geography].[Geography].[Country].[United States])  
SELECT Measures.x ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

MemberValue

Returns the value of a member.

Syntax

Member_Expression.MemberValue

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that evaluates to a member.

Return Value

The member value returned contains the following information, listed in the order that this information appears in the return value:

- The value binding, if it has been defined.
- The key with the original data type if either there is no name binding, or the key and the caption are bound to the same column.
- The caption of the member.

Example

The following example returns the value binding, the member key, and the caption for the first date in the Date dimension in the Adventure Works cube.

```
WITH MEMBER Measures.ValueColumn as [Date].[Calendar].[July 1, 2001].MemberValue
MEMBER Measures.KeyColumn as [Date].[Calendar].[July 1, 2001].Member_Key
MEMBER Measures.NameColumn as [Date].[Calendar].[July 1, 2001].Member_Name

SELECT {Measures.ValueColumn, Measures.KeyColumn, Measures.NameColumn} ON 0
from [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Min

Returns the minimum value of a numeric expression that is evaluated over a set.

Syntax

```
Min( Set_Expression [ , Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the specified numeric expression is evaluated across the set and then returns the minimum value from that evaluation. If a numeric expression is not specified, the specified set is evaluated in the current context of the members of the set and then returns the minimum value from that evaluation.



Note

Analysis Services ignores nulls when calculating the minimum value in a set of numbers.

Example

The following example returns the minimum quarterly sales for each subcategory and each country in the Adventure Works cube.

```
WITH MEMBER Measures.x AS Min
    ([Date].[Calendar].CurrentMember.Children
    , [Measures].[Reseller Order Quantity]
    )
SELECT Measures.x ON 0
, NON EMPTY [Date].[Calendar].[Calendar Quarter]*
    [Product].[Product Categories].[Subcategory].members *
    [Geography].[Geography].[Country].Members
ON 1
FROM [Adventure Works]
```


See Also

[MDX Function Reference \(MDX\)](#)

Mtd

Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level in the Time dimension.

Syntax

```
Mtd( [ Member_Expression ] )
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

If a member expression is not specified, the default is the current member of the first hierarchy with a level of type Months in the first dimension of type Time in the measure group.

The **Mtd** function is a shortcut function for the PeriodsToDate function when the Type property of the attribute hierarchy on which a level is based is set to Months. That is,

`Mtd(Member_Expression)` is equivalent to
`PeriodsToDate(Month_Level_Expression, Member_Expression)`.

Example

The following example returns the sum of the month to date freight costs for Internet sales for the month of July, 2002 through the 20th day of July.

```
WITH MEMBER Measures.x AS SUM
(
    MTD([Date].[Calendar].[Date].[July 20, 2002])
    , [Measures].[Internet Freight Cost]
)
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Name

Returns the name of a dimension, hierarchy, level, or member.

Syntax

Dimension expression syntax

`Dimension_Expression.Name`

Hierarchy expression syntax

`Hierarchy_Expression.Name`

Level expression syntax

`Level_Expression.Name`

Member expression syntax

`Member_Expression.Name`

Arguments

Dimension_Expression

A valid Multidimensional Expressions (MDX) expression that returns a dimension.

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **Name** function returns the name of the object, not the unique name.

Examples

Dimension, Hierarchy and Level Expression Example

The following example returns the dimension name for the Date dimension and the hierarchy and level names for the July 2001 member.

```
WITH MEMBER Measures.DimensionName AS [Date].Name
MEMBER Measures.HierarchyName AS [Date].[Calendar].[July 2001].Hierarchy.Name
MEMBER Measures.LevelName as [Date].[Calendar].[July 2001].Level.Name

SELECT {Measures.DimensionName, Measures.HierarchyName, Measures.LevelName}
ON 0
from [Adventure Works]
```

Member Expression Example

The following example returns the member name, along with the member value, member key and member caption.

```
WITH MEMBER MemberName AS [Date].[Calendar].[July 1, 2001].Name
MEMBER Measures.ValueColumn as [Date].[Calendar].[July 1, 2001].MemberValue
MEMBER Measures.KeyColumn as [Date].[Calendar].[July 1, 2001].Member_Key
MEMBER Measures.NameColumn as [Date].[Calendar].[July 1, 2001].Member_Name

SELECT {Measures.MemberName, Measures.ValueColumn, Measures.KeyColumn,
Measures.NameColumn} ON 0
from [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

NameToSet

Returns a set that contains the member specified by a Multidimensional Expressions (MDX)-formatted string.

Syntax

NameToSet(**Member_Name**)

Arguments

Member_Name

A valid string expression that represents the name of a member.

Remarks

If the specified member name exists, the **NameToSet** function returns a set containing that member. Otherwise, the function returns an empty set.



Note

The specified member name must only be a member name; it cannot be a member expression. To use a member expression, see [MDX Function Reference \(MDX\)](#).

Example

The following returns the default measure value for the specified member name.

```
SELECT NameToSet('[Date].[Calendar].[July 2001]') ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

NextMember

Returns the next member in the level that contains a specified member.

Syntax

```
Member_Expression.NextMember
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **NextMember** function returns the next member, in the same level, that contains the specified member.

Example

The following example returns the August 2001 member as the next member to the July 2001 member.

```
SELECT [Date].[Calendar].[Month].[July 2001].NextMember ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

NonEmpty

Returns the set of tuples that are not empty from a specified set, based on the cross product of the specified set with a second set.

Syntax

```
NONEMPTY(set_expression1 [,set_expression2])
```

Arguments

set_expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

set_expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

This function returns the tuples in the first specified set that are non-empty when evaluated across the tuples in the second set. The **NonEmpty** function takes into account calculations and preserves duplicate tuples. If a second set is not provided, the expression is evaluated in the context of the current coordinates of the members of the attribute hierarchies and the measures in the cube.



Note

Use this function rather than the deprecated [NonEmptyCrossjoin \(MDX\)](#) function.



Important

Non-empty is a characteristic of the cells references by the tuples, not the tuples themselves.

Examples

The following query shows a simple example of **NonEmpty**, returning all the Customers who had a non-null value for Internet Sales Amount on July 1st 2001:

```
SELECT [Measures].[Internet Sales Amount] ON 0,
NONEMPTY (
  [Customer].[Customer].[Customer].MEMBERS
, {([Date].[Calendar].[Date].&[20010701], [Measures].[Internet Sales
Amount])}
)
ON 1
FROM [Adventure Works]
```

The following example returns the set of tuples containing customers and purchase dates, using the **Filter** function and the **NonEmpty** functions to find the last date that each customer made a purchase:

```
WITH SET MYROWS AS FILTER
  (NONEMPTY
    ([Customer].[Customer Geography].[Customer].MEMBERS
     * [Date].[Date].[Date].MEMBERS
     , [Measures].[Internet Sales Amount]
    ) AS MYSET
, NOT (MYSET.CURRENT.ITEM(0)
      IS MYSET.ITEM(RANK (MYSET.CURRENT, MYSET)).ITEM(0))
)
SELECT [Measures].[Internet Sales Amount] ON 0,
MYROWS ON 1
FROM [Adventure Works]
```

See Also

[DefaultMember \(MDX\)](#)

[Filter \(MDX\)](#)

[IsEmpty \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

[NonEmptyCrossjoin \(MDX\)](#)

NonEmptyCrossjoin

Returns a set that contains the cross product of one or more sets, excluding empty tuples and tuples without associated fact table data.

Syntax

```
NonEmptyCrossjoin( Set_Expression1 [ ,Set_Expression2, ...] [,Count ] )
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of sets to be returned.

Remarks

The **NonEmptyCrossjoin** function returns the cross product of two or more sets as a set, excluding empty tuples or tuples without data supplied by underlying fact tables. Because of how the **NonEmptyCrossjoin** function works, all calculated members are automatically excluded.

If Count is not specified, the function cross joins all specified sets and excludes empty members from the resulting set. If a number of sets is specified, the function cross joins the numbers of sets specified, starting with the first specified set. The **NonEmptyCrossjoin** function uses any remaining sets that are specified in subsequent specified sets, but which have not been cross joined to determine which members are considered nonempty in the resulting cross joined set. The **NonEmptyCrossjoin** function respects the **NON_EMPTY_BEHAVIOR** setting of calculated measures.

Important

This function is deprecated and you should not use it; it is retained only to maintain backwards compatibility. Instead, you should use the Exists (MDX) function with the measure group name argument or the NonEmpty (MDX) function.

See Also

[MDX Function Reference \(MDX\)](#)

OpeningPeriod

Returns the first sibling among the descendants of a specified level, optionally at a specified member.

Syntax

```
OpeningPeriod( [ Level_Expression [ , Member_Expression ] ] )
```

Arguments

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

This function is primarily intended to be used the Time dimension, but can be used with any dimension.

- If a level expression is specified, the **OpeningPeriod** function uses the hierarchy that contains the specified level and returns the first sibling among the descendants of the default member at the specified level.
- If both a level expression and a member expression are specified, the **OpeningPeriod** function returns the first sibling among the descendants of specified member at the specified level within the hierarchy containing the specified level.
- If neither a level expression nor a member expression are specified, the **OpeningPeriod** function uses the default level and member of the dimension with a type of Time.



Note

The **ClosingPeriod** function is similar to the **OpeningPeriod** function, except that the **ClosingPeriod** function returns the last sibling instead of the first sibling.

Examples

The following example returns the value for the default measure for the FY2002 member of the Date dimension (which has a type of Time). This member is returned because the Fiscal Year level is the first descendant of the [All] level, the Fiscal hierarchy is the default hierarchy because it is the first user-defined hierarchy in the hierarchy collection, and the FY2002 member is the first sibling in this hierarchy at this level.

```
SELECT OpeningPeriod() ON 0  
FROM [Adventure Works]
```


The following example returns the value for the default measure for July 1, 2001 member at the Date.Date.Date level for the Date.Date attribute hierarchy. This member is the first sibling of the descendant of [All] level in the Date.Date attribute hierarchy.

```
SELECT OpeningPeriod([Date].[Date].[Date]) ON 0
FROM [Adventure Works]
```

The following example returns the value for the default measure for January, 2003 member, which is the first sibling of the descendant of the 2003 member at the year level in the Calendar user-defined hierarchy.

```
SELECT OpeningPeriod([Date].[Calendar].[Month],[Date].[Calendar].[Calendar
Year].&[2003]) ON 0
FROM [Adventure Works]
```

The following example returns the value for the default measure for July, 2002 member, which is the first sibling of the descendant of the 2003 member at the year level in the Fiscal user-defined hierarchy.

```
SELECT OpeningPeriod([Date].[Fiscal].[Month],[Date].[Fiscal].[Fiscal
Year].&[2003]) ON 0
FROM [Adventure Works]
```

See Also

[FirstSibling \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

[FirstSibling \(MDX\)](#)

Order

Arranges members of a specified set, optionally preserving or breaking the hierarchy.

Syntax

Numeric expression syntax

```
Order(Set_Expression, Numeric_Expression
[, { ASC | DESC | BASC | BDESC }])
```

String expression syntax

```
Order(Set_Expression, String_Expression  
[, { ASC | DESC | BASC | BDESC } ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

String_Expression

A valid string expression that is typically a valid Multidimensional Expressions (MDX) expression of cell coordinates that return a number expressed as a string.

Remarks

The **Order** function can either be hierarchical (as specified by using the **ASC** or **DESC** flag) or nonhierarchical (as specified by using the **BASC** or **BDESC** flag; the **B** stands for "break hierarchy"). If **ASC** or **DESC** is specified, the **Order** function first arranges the members according to their position in the hierarchy, and then orders each level. If **BASC** or **BDESC** is specified, the **Order** function arranges members in the set without regard to the hierarchy. In no flag is specified, **ASC** is the default.

If the **Order** function is used with a set where two or more hierarchies are crossjoined, and the **DESC** flag is used, only the members of the last hierarchy in the set are ordered. This is a change from Analysis Services 2000 where all hierarchies in the set were ordered.

Examples

The following example returns, from the **Adventure Works** cube, the number of reseller orders for all Calendar Quarters from the Calendar hierarchy on the Date dimension. The **Order** function reorders the set for the ROWS axis. The **Order** function orders the set by [Reseller Order Count] in descending hierarchical order as determined by the [Calendar] hierarchy.

```
SELECT  
    Measures.[Reseller Order Count] ON COLUMNS,  
    Order(  
[Date].[Calendar].[Calendar Quarter].MEMBERS  
        ,Measures.[Reseller Order Count]  
        ,DESC
```

```

    ) ON ROWS
FROM [Adventure Works]

```

Notice how in this example, when the **DESC** flag is changed to **BDESC**, the hierarchy is broken and the list of Calendar Quarters is returned with no regard for the hierarchy:

```

SELECT
    Measures.[Reseller Order Count] ON COLUMNS,
    Order(
[Date].[Calendar].[Calendar Quarter].MEMBERS
    ,Measures.[Reseller Order Count]
    ,BDESC
    ) ON ROWS
FROM [Adventure Works]

```

The following example returns the Reseller Sales Measure for the top five selling subcategories of products, irrespective of hierarchy, based on Reseller Gross Profit. The **Subset** function is used to return only the first 5 tuples in the set after the result is ordered using the **Order** function.

```

SELECT Subset
    (Order
        ([Product].[Product Categories].[SubCategory].members
        , [Measures].[Reseller Gross Profit]
        , BDESC
        )
    , 0
    , 5
    ) ON 0
FROM [Adventure Works]

```

The following example uses the **Rank** function to rank the members of the City hierarchy, based on the Reseller Sales Amount measure, and then displays them in ranked order. By using the **Order** function to first order the set of members of the City hierarchy, the sorting is done only once and then followed by a linear scan before being presented in sorted order.

```

WITH
SET OrderedCities AS Order
    ([Geography].[City].[City].members
    , [Measures].[Reseller Sales Amount], BDESC
    )
MEMBER [Measures].[City Rank] AS Rank
    ([Geography].[City].CurrentMember, OrderedCities)

```

```

SELECT {[Measures].[City Rank],[Measures].[Reseller Sales Amount]} ON 0
,Order
  ([Geography].[City].[City].MEMBERS
  ,[City Rank], ASC)
  ON 1
FROM [Adventure Works]

```

The following example returns the number of products in the set that are unique, using the **Order** function to order the non-empty tuples before utilizing the **Filter** function. The **CurrentOrdinal** function is used to compare and eliminate ties.

```

WITH MEMBER [Measures].[PrdTies] AS Count
  (Filter
    (Order
      (NonEmpty
        ([Product].[Product].[Product].Members
        , {[Measures].[Reseller Order Quantity]}
        )
      , [Measures].[Reseller Order Quantity]
      , BDESC
      ) AS OrdPrds
    , (OrdPrds.CurrentOrdinal < OrdPrds.Count
      AND [Measures].[Reseller Order Quantity] =
        ( [Measures].[Reseller Order Quantity]
        , OrdPrds.Item
          (OrdPrds.CurrentOrdinal
          )
        )
      )
    )
  OR (OrdPrds.CurrentOrdinal > 1
      AND [Measures].[Reseller Order Quantity] =
        ([Measures].[Reseller Order Quantity]
        , OrdPrds.Item
          (OrdPrds.CurrentOrdinal-2)
          )
      )
  )

```

```

    )
)
SELECT {[Measures].[PrdTies]} ON 0
FROM [Adventure Works]

```

To understand how the **DESC** flag works with sets of tuples, first consider the results of the following query:

```

SELECT
{[Measures].[Tax Amount]} ON 0,
ORDER(
[Sales Territory].[Sales Territory].[Group].MEMBERS
,[Measures].[Tax Amount], DESC)
ON 1
FROM [Adventure Works]

```

On the Rows axis you can see that the Sales Territory Groups have been ordered in descending order by Tax Amount, as follows: North America, Europe, Pacific, NA. Now see what happens if we crossjoin the set of Sales Territory Groups with the set of Product Subcategories and apply the **Order** function in the same way, as follows:

```

SELECT
{[Measures].[Tax Amount]} ON 0,
ORDER(
[Sales Territory].[Sales Territory].[Group].MEMBERS
*
{[Product].[Product Categories].[subCategory].Members}
,[Measures].[Tax Amount], DESC)
ON 1
FROM [Adventure Works]

```

While the set of Product Subcategories has been ordered in descending, hierarchical order, the Sales Territory Groups are now not sorted and appear in the order they appear on the hierarchy: Europe, NA, North America and Pacific. This is because only the last hierarchy in the set of tuples, Product Subcategories, is sorted. To reproduce the behavior of Analysis Services 2000, use a series of nested **Generate** functions to sort each set before it is crossjoined, for example:

```

SELECT
{[Measures].[Tax Amount]} ON 0,
GENERATE (
ORDER (
[Sales Territory].[Sales Territory].[Group].MEMBERS
,[Measures].[Tax Amount], DESC)
,
ORDER (
[Sales Territory].[Sales Territory].CURRENTMEMBER
*
{[Product].[Product Categories].[subCategory].Members}
,[Measures].[Tax Amount], DESC))
ON 1
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

Ordinal

Returns the zero-based ordinal value associated with a level.

Syntax

Level_Expression.Ordinal

Arguments

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Remarks

The **Ordinal** function is frequently used in conjunction with the **IIF** and **CurrentMember** functions to conditionally display different values at different hierarchy levels, based on the ordinal position of each specific cell in the query result. For example, you can use the **Ordinal**

function to perform calculations at certain levels and display a default value of "N/A" at other levels.

Example

The following example returns the ordinal number for the Calendar Quarter level in the Calendar hierarchy.

```
WITH MEMBER Measures.x AS [Date].[Calendar].[Calendar Quarter].Ordinal
SELECT Measures.x on 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

ParallelPeriod

Returns a member from a prior period in the same relative position as a specified member.

Syntax

```
ParallelPeriod( [ Level_Expression [ ,Index [ , Member_Expression ] ] ] )
```

Arguments

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Index

A valid numeric expression that specifies the number of parallel periods to lag.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

Although similar to the **Cousin** function, the **ParallelPeriod** function is more closely related to time series. The **ParallelPeriod** function takes the ancestor of the specified member at the specified level, finds the ancestor's sibling with the specified lag, and finally returns the parallel period of the specified member among the descendants of the sibling.

The **ParallelPeriod** function has the following defaults:

- If neither a level expression nor a member expression is specified, the default member value is the current member of the first hierarchy on the first dimension with a type of Time in the measure group.
- If a level expression is specified, but a member expression is not specified, the default member value is `Level_Expression.Hierarchy.CurrentMember`.
- The default index value is 1.
- The default level is the level of the parent of the specified member.

The **ParallelPeriod** function is equivalent to the following MDX statement:

```
Cousin(Member_Expression, Ancestor(Member_Expression, Level_Expression)
.Lag(Numeric_Expression))
```

Example

The following example returns the parallel period for the month of October 2003 with a lag of three periods, based on the quarter level, which returns the month of January, 2003.

```
SELECT ParallelPeriod ([Date].[Calendar].[Calendar Quarter]
, 3
, [Date].[Calendar].[Month].[October 2003])
ON 0
FROM [Adventure Works]
```

The following example returns the parallel period for the month of October 2003 with a lag of three periods, based on the semester level, which returns the month of April, 2002.

```
SELECT ParallelPeriod ([Date].[Calendar].[Calendar Semester]
, 3
, [Date].[Calendar].[Month].[October 2003])
ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Parent

Returns the parent of a member.

Syntax

`Member_Expression.Parent`

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **Parent** function returns the parent member of the specified member.

Examples

The following examples return the parent of the July 1, 2001 member. The first example specifies this member in the context of the Date attribute hierarchy and returns the All Periods member.

```
SELECT [Date].[Date].[July 1, 2001].Parent ON 0  
FROM [Adventure Works]
```

The following example specifies the July 1, 2001 member in the context of the Calendar hierarchy.

```
SELECT [Date].[Calendar].[July 1, 2001].Parent ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

PeriodsToDate

Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by a specified level in the Time dimension.

Syntax

```
PeriodsToDate( [ Level_Expression [ ,Member_Expression ] ] )
```

Arguments

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

Within the scope of the specified level, the **PeriodsToDate** function returns the set of periods on the same level as the specified member, starting with the first period and ending with specified member.

- If a level is specified, the current member of the hierarchy is inferred *hierarchy.CurrentMember*, where *hierarchy* is the hierarchy of the specified level.
- If neither a level nor a member is specified, the level is the parent level of the current member of the first hierarchy on the first dimension of type Time in the measure group.

`PeriodsToDate(Level_Expression, Member_Expression)` is functionally equivalent to the following MDX expression:

```
TopCount (Descendants (Ancestor (Member_Expression, Level_Expression),  
Member_Expression.Level), 1):Member_Expression
```

Examples

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first eight months of calendar year 2003 that are contained in the `Date` dimension, from the **Adventure Works** cube.

```
WITH MEMBER [Date].[Calendar].[First8Months2003] AS  
    Aggregate (  
        PeriodsToDate (  
            [Date].[Calendar].[Calendar Year],  
            [Date].[Calendar].[Month].[August 2003]  
        )  
    )  
SELECT  
    [Date].[Calendar].[First8Months2003] ON COLUMNS,  
    [Product].[Category].Children ON ROWS  
FROM  
    [Adventure Works]  
WHERE  
    [Measures].[Order Quantity]
```

The following example aggregates over the first two months of the second semester of calendar year 2003.

```
WITH MEMBER [Date].[Calendar].[First2MonthsSecondSemester2003] AS  
    Aggregate (  
        PeriodsToDate (  
            [Date].[Calendar].[Month].[January 2003],  
            [Date].[Calendar].[Month].[February 2003]  
        )  
    )
```

```

        [Date].[Calendar].[Calendar Semester],
        [Date].[Calendar].[Month].[August 2003]
    )
)
SELECT
    [Date].[Calendar].[First2MonthsSecondSemester2003] ON COLUMNS,
    [Product].[Category].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    [Measures].[Order Quantity]

```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Predict

Caution

- This function is in the process of being removed due to internal inconsistencies.
- Review the example section for a workaround using a DMX expression.

Returns a value of a numeric expression evaluated over a data mining model.

Syntax

```
Predict(Mining_Model_Name,String_Expression)
```

Arguments

Mining_Model_Name

A valid string expression that represents the name of a mining model.

String_Expression

A valid string expression that evaluates to a valid DMX expression for the specified mining model.

Remarks

The **Predict** function evaluates the specified string expression within the context of the specified mining model.

Data mining syntax and functions are documented in Data Mining Expressions (DMX) reference.

Example

The following example predicts name of the cluster and the distance from it of a particular customer using the Customer Clusters mining model:

```
WITH MEMBER MEASURES.CLNAME AS
PREDICT("Customer Clusters", "Cluster()")
MEMBER MEASURES.CLDISTANCE AS
PREDICT("Customer Clusters", "ClusterDistance(Cluster())")
SELECT {MEASURES.CLNAME, MEASURES.CLDISTANCE} ON 0
FROM [Adventure Works]
WHERE ([Customer].[Customer Geography].[Customer].&[12012])
```

See Also

[MDX Function Reference \(MDX\)](#)

PrevMember

Returns the previous member in the level that contains a specified member.

Syntax

```
Member_Expression.PrevMember
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **PrevMember** function returns the previous member in the same level as the specified member.

Example

The following example shows a simple query that uses the **PrevMember** function to display the name of the member immediately before the current member on the rows axis:

```
WITH MEMBER MEASURES.PREVMEMBERDEMO AS
[Date].[Calendar].CURRENTMEMBER.PREVMEMBER.NAME
SELECT MEASURES.PREVMEMBERDEMO ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```

The following example returns the count of the resellers whose sales have declined over the previous time period, based on user-selected State-Province member values evaluated using the **Aggregate** function. The **Hierarchize** and **DrillDownLevel** functions are used to return values for declining sales for product categories in the Product dimension. The **PrevMember** function is used to compare the current time period with the previous time period.

```
WITH MEMBER Measures.[Declining Reseller Sales] AS
    Count (
        Filter(
            Existing(Reseller.Reseller.Reseller),
            [Measures].[Reseller Sales Amount] < ([Measures].[Reseller Sales
Amount]),
            [Date].Calendar.PrevMember)
        )
    )
MEMBER [Geography].[State-Province].x AS
    Aggregate (
        {[Geography].[State-Province].&[WA]&[US]},
        [Geography].[State-Province].&[OR]&[US] }
    )
SELECT NON EMPTY Hierarchize (
    AddCalculatedMembers (
        {DrillDownLevel({[Product].[All Products]})}
    )
)
    DIMENSION PROPERTIES PARENT_UNIQUE_NAME ON COLUMNS
FROM [Adventure Works]
WHERE ([Geography].[State-Province].x,
```

```
[Date].[Calendar].[Calendar Quarter].&[2003]&[4],  
[Measures].[Declining Reseller Sales])
```

See Also

[MDX Function Reference \(MDX\)](#)

Properties

Returns a string, or a strongly-typed value, that contains a member property value.

Syntax

```
Member_Expression.Properties(Property_Name [, TYPED])
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Property_Name

A valid string expression of a member property name.

Remarks

The **Properties** function returns the value of the specified member for the specified member property. The member property can be any of the intrinsic member properties, such as **NAME**, **ID**, **KEY**, or **CAPTION**, or it can be a user-defined member property. For more information, see [Intrinsic Member Properties \(MDX\)](#) and [User-defined Member Properties \(MDX\)](#).

By default, the value is coerced to be a string. If **TYPED** is specified, the return value is strongly typed.

- If the property type is intrinsic, the function returns the original type of the member.
- If the property type is user defined, the type of the return value is the same as the type of the return value of the **MemberValue** function.

Note

Properties ('Key') returns the same result as Key0 except for composite keys. Properties ('Key') will return null for composite keys. Use the Keyx syntax for composite keys, as illustrated in the example. Properties ('Key0'), Properties('Key1'), Properties('Key2'), etc collectively form the composite key.

Example

The following example returns both intrinsic and user-defined member properties, utilizing the TYPED argument to return the strongly typed value for the Day Name member property.

```
WITH MEMBER Measures.MemberName AS
    [Date].[Calendar].[July 1, 2003].Properties('Name')
MEMBER Measures.MemberVal AS
    [Date].[Calendar].[July 1, 2003].Properties('Member_Value')
MEMBER Measures.MemberKey AS
    [Date].[Calendar].[July 1, 2003].Properties('Key')
MEMBER Measures.MemberID AS
    [Date].[Calendar].[July 1, 2003].Properties('ID')
MEMBER Measures.MemberCaption AS
    [Date].[Calendar].[July 1, 2003].Properties('Caption')
MEMBER Measures.DayName AS
    [Date].[Calendar].[July 1, 2003].Properties('Day Name', TYPED)
MEMBER Measures.DayNameTyped AS
    [Date].[Calendar].[July 1, 2003].Properties('Day Name')
MEMBER Measures.DayofWeek AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Week')
MEMBER Measures.DayofMonth AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Month')
MEMBER Measures.DayofYear AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Year')

SELECT {Measures.MemberName
    , Measures.MemberVal
    , Measures.MemberKey
    , Measures.MemberID
    , Measures.MemberCaption
    , Measures.DayName
    , Measures.DayNameTyped
    , Measures.DayofWeek
    , Measures.DayofMonth
    , Measures.DayofYear
```

```

    } ON 0
FROM [Adventure Works]
The following example shows the use of the KEYx property.
WITH
MEMBER Measures.MemberKey AS
    [Customer].[Customer Geography].[State-
Province].&[QLD]&[AU].Properties('Key')
MEMBER Measures.MemberKey0 AS
    [Customer].[Customer Geography].[State-
Province].&[QLD]&[AU].Properties('Key0')
MEMBER Measures.MemberKey1 AS
    [Customer].[Customer Geography].[State-
Province].&[QLD]&[AU].Properties('Key1')

SELECT {Measures.MemberKey
    , Measures.MemberKey0
    , Measures.MemberKey1
    } ON 0
FROM [Adventure Works]

```

See Also

[Using Member Properties \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Qtd

Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Quarter level in the Time dimension.

Syntax

Qtd([**Member_Expression**])

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

If a member expression is not specified, the default is the current member of the first hierarchy with a level of type Quarters in the first dimension of type Time in the measure group.

The **Qtd** function is a shortcut function for the [MDX Function Reference \(MDX\)](#) function whose level expression argument is set to Quarter. That is, `Qtd(Member_Expression)` is functionally equivalent to `PeriodsToDate(Quarter_Level_Expression, Member_Expression)`.

Example

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first two months of the third quarter of calendar year 2003 that are contained in the `Date` dimension, from the **Adventure Works** cube.

```
WITH MEMBER [Date].[Calendar].[First2MonthsSecondSemester2003] AS
    Aggregate (
        QTD([Date].[Calendar].[Month].[August 2003])
    )
SELECT
    [Date].[Calendar].[First2MonthsSecondSemester2003] ON COLUMNS,
    [Product].[Category].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    [Measures].[Order Quantity]
```

See Also

[MDX Function Reference \(MDX\)](#)

Rank

Returns the one-based rank of a specified tuple in a specified set.

Syntax

```
Rank(Tuple_Expression, Set_Expression [ ,Numeric Expression ] )
```

Arguments

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the **Rank** function determines the one-based rank for the specified tuple by evaluating the specified numeric expression against the tuple. If a numeric expression is specified, the **Rank** function assigns the same rank to tuples with duplicate values in the set. This assignment of the same rank to duplicate values affects the ranks of subsequent tuples in the set. For example, a set consists of the following tuples, $\{(a, b), (e, f), (c, d)\}$. The tuple (a, b) has the same value as the tuple (c, d) . If the tuple (a, b) has a rank of 1, then both (a, b) and (c, d) would have a rank of 1. However, the tuple (e, f) would have a rank of 3. There could be no tuple in this set with a rank of 2.

If a numeric expression is not specified, the **Rank** function returns the one-based ordinal position of the specified tuple.

The **Rank** function does not order the set.

Example

The following example returns the set of tuples containing customers and purchase dates, by using the **Filter**, **NonEmpty**, **Item**, and **Rank** functions to find the last date that each customer made a purchase.

```
WITH SET MYROWS AS FILTER
    (NONEMPTY
        ([Customer].[Customer Geography].MEMBERS
            * [Date].[Date].[Date].MEMBERS
            , [Measures].[Internet Sales Amount]
        ) AS MYSET
    , NOT (MYSET.CURRENT.ITEM(0)
        IS MYSET.ITEM(RANK(MYSET.CURRENT, MYSET)).ITEM(0))
    )
SELECT [Measures].[Internet Sales Amount] ON 0,
```

```
MYROWS ON 1
```

```
FROM [Adventure Works]
```

The following example uses the **Order** function, rather than the **Rank** function, to rank the members of the City hierarchy based on the Reseller Sales Amount measure and then displays them in ranked order. By using the **Order** function to first order the set of members of the City hierarchy, the sorting is done only once and then followed by a linear scan before being presented in sorted order.

```
WITH
```

```
SET OrderedCities AS Order
```

```
    ([Geography].[City].[City].members  
    , [Measures].[Reseller Sales Amount], BDESC  
    )
```

```
MEMBER [Measures].[City Rank] AS Rank
```

```
    ([Geography].[City].CurrentMember, OrderedCities)
```

```
SELECT {[Measures].[City Rank],[Measures].[Reseller Sales Amount]} ON 0  
,Order
```

```
    ([Geography].[City].[City].MEMBERS  
    , [City Rank], ASC)
```

```
    ON 1
```

```
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

RollupChildren

Returns a value generated by rolling up the values of the children of a specified member using the specified unary operator.

Syntax

```
RollupChildren(Member_Expression, Unary_Operator)
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Unary_Operator

A valid string expression that specifies a unary operator.

Remarks

The **RollupChildren** function rolls up the values of the children of the specified member using the specified unary operator.

The following table describes the valid unary operators for this function.

Operator	Result
+	total = total + current child
-	total = total - current child
*	total = total * current child
/	total = total / current child
%	total = (total / current child) * 100
~	The child is not used in the rollup. Its value is ignored.

If the operator in the member property does not appear in the list, an error occurs. The order of evaluation is determined by the order of the siblings, not by the precedence of the operators.

Example

The following example uses a member property called "Alternate Rollup Operator" that contains alternate values for unary operators to rollup up children of the Net Profit hierarchy in the Account dimension in an alternate manner. This member property does not exist in the Adventure Works cube, but could be created. This use of the **RollupChildren** function could be used in a budgeting application for what-if analysis.

```
RollupChildren  
    ( [Account].[Net Profit]  
    , [Account].CurrentMember.Properties ('Alternate Rollup Operator') )
```

See Also

[MDX Function Reference \(MDX\)](#)

Root

Returns a tuple that consists of the **All** members from each attribute hierarchy within the current scope in a cube, dimension, or tuple. For more information about Scope, see [MDX Function Reference \(MDX\)](#).

Note

If an attribute hierarchy does not have an **All** member, the tuple contains the default member for that hierarchy.

Syntax

Cube syntax

Root ()

Dimension syntax

Root(*Dimension_Name*)

Tuple syntax

Root(*Tuple_Expression*)

Arguments

Dimension_Name

A valid string expression specifying a dimension name.

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Remarks

If neither a dimension name nor a tuple expression is specified, the **Root** function returns a tuple that contains the **All** member (or the default member if the **All** member does not exist) from each attribute hierarchy in the cube. The order of members in the tuple is based on the sequence in which the attribute hierarchies are defined within the cube.

If a dimension name is specified, the **Root** function returns a tuple that contains the **All** member (or the default member if the **All** member does not exist) from each attribute hierarchy in the specified dimension based on the context of the current member. The order of members in the tuple is based on the sequence in which the attribute hierarchies are defined within the dimension.

Note

If a hierarchy name is specified, the **Tuple** function will pick the dimension name from the hierarchy name specified.

If a tuple expression is specified, the **Root** function returns a tuple that contains the intersection of the specified tuple and the **All** members of all other dimension attributes not explicitly included in the specified tuple. The specified tuple must reference only one dimension, or an error occurs.

Examples

The following example returns the tuple containing the **All** member (or the default if the **All** member does not exist) from each hierarchy in the Adventure Works cube.

```
SELECT Root () ON 0
FROM [Adventure Works]
```

The following example returns the tuple containing the **All** member (or the default if the **All** member does not exist) from each hierarchy in the Date dimension in the Adventure Works cube and the value for the specified member of Measures dimension that intersects with these default members.

```
SELECT Root ([Date]) ON 0
FROM [Adventure Works]
WHERE [Measures].[Order Count]
```

The following example returns the tuple containing specified tuple member (July 1, 2001, along with the **All** member (or the default if the **All** member does not exist) from each non-specified hierarchy in the Date dimension Adventure Works cube and the value for the specified member of Measures dimension that intersects with these members.

```
SELECT Root ([Date].[July 1, 2001]) ON 0
FROM [Adventure Works]
WHERE [Measures].[Order Count]
```

See Also

[MDX Function Reference \(MDX\)](#)

SetToArray

Converts one or more sets to an array for use in a user-defined function.

Syntax

```
SetToArray(Set_Expression1 [ ,Set_Expression2,...n ][ ,Numeric_Expression ] )
```

Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **SetToArray** function converts one or more sets to an array for use in a user-defined function. The number of dimensions in the resulting array is the same as the number of sets specified.

The optional numeric expression can provide the values in the array cells. If a numeric expression is not specified, the cross join of the sets is evaluated in the current context.

The cell coordinates in the resulting array correspond to the position of the sets in the list. For example, there are three sets, SA, SB, and SC. Each of these sets has two elements. The MDX statement, `SetToArray(SA, SB, SC)`, creates the following three-dimensional array:

```
(SA1, SB1, SC1) (SA2, SB1, SC1) (SA1, SB2, SC1) (SA2, SB2, SC1)
(SA1, SB1, SC2) (SA2, SB1, SC2) (SA1, SB2, SC2) (SA2, SB2, SC2)
```

Note

The return type of the **SetToArray** function is the VARIANT type, VT_ARRAY. Therefore, the output of the **SetToArray** function should be used only as input to a user-defined function.

Example

The following example returns an array.

```
SetToArray([Geography].[Geography].Members, [Measures].[Internet Sales Amount])
```

See Also

[MDX Function Reference \(MDX\)](#)

SetToStr

Returns a Multidimensional Expressions (MDX)-formatted string of that corresponds to a specified set.

Syntax

SetToStr(**Set_Expression**)

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

This function is used to transfer a string-representation of a set to an external function for parsing. The string that is returned is enclosed in braces {}, with each item in the set separated by a comma.

Example

The following example returns a string containing all of the members of the Geography.Country attribute hierarchy.

```
WITH MEMBER Measures.x AS SetToStr (Geography.Geography.Children)
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Siblings

Returns the siblings of a specified member, including the member itself.

Syntax

Member_Expression.Siblings

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Example

The following example returns the default measure for the siblings of March of 2003, which are January 2003 and February 2003, and including March 2003.

```
SELECT [Date].[Calendar].[Month].[March 2003].Siblings ON 0  
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Stddev

Alias for the Stdev function.

See Also

[MDX Function Reference \(MDX\)](#)

StddevP

Alias for the StdevP function

See Also

[MDX Function Reference \(MDX\)](#)

Stdev

Returns the sample standard deviation of a numeric expression evaluated over a set, using the unbiased population formula (dividing by n-1).

Syntax

```
Stdev(Set_Expression [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **Stdev** function uses the unbiased population formula, while the StdevP function uses the biased population formula.

Example

The following example returns the standard deviation for Internet Order Quantity, evaluated over the first three months of calendar year 2003, using the unbiased population formula.

```
WITH MEMBER Measures.x AS
    Stdev
    ( { [Date].[Calendar].[Month].[January 2003],
        [Date].[Calendar].[Month].[February 2003],
        [Date].[Calendar].[Month].[March 2003] },
        [Measures].[Internet Order Quantity])
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

StdevP

Returns the population standard deviation of a numeric expression evaluated over a set, using the biased population formula (dividing by n).

Syntax

```
StdevP(Set_Expression [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **StdevP** function uses the biased population formula, while the Stdev function uses the unbiased population formula.

Example

The following example returns the standard deviation for Internet Order Quantity evaluated over the first three months of calendar year 2003 using the biased population formula.

```
WITH MEMBER Measures.x AS
    StdevP
    ( { [Date].[Calendar].[Month].[January 2003],
        [Date].[Calendar].[Month].[February 2003],
        [Date].[Calendar].[Month].[March 2003] },
        [Measures].[Internet Order Quantity])
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

StripCalculatedMembers

Returns a set generated by removing calculated members from a specified set.

Syntax

```
StripCalculatedMembers(Set_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **StripCalculatedMembers** function removes calculated members from a set. Calculated members can be added to a set by using the AddCalculatedMembers function, which returns calculated members that are defined on the server, or calculated members that were added within the query itself by using the WITH MEMBER syntax.

Example

The following example removes all calculated members from the query.

```
WITH MEMBER Measures.MemberName AS
    [Date].[Calendar].[July 1, 2003].Properties('Name')
MEMBER Measures.MemberVal AS
    [Date].[Calendar].[July 1, 2003].Properties('Member_Value')
MEMBER Measures.MemberKey AS
    [Date].[Calendar].[July 1, 2003].Properties('Key')
MEMBER Measures.MemberID AS
    [Date].[Calendar].[July 1, 2003].Properties('ID')
MEMBER Measures.MemberCaption AS
    [Date].[Calendar].[July 1, 2003].Properties('Caption')
MEMBER Measures.DayName AS
    [Date].[Calendar].[July 1, 2003].Properties('Day Name', TYPED)
MEMBER Measures.DayNameTyped AS
    [Date].[Calendar].[July 1, 2003].Properties('Day Name')
MEMBER Measures.DayOfWeek AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Week')
MEMBER Measures.DayOfMonth AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Month')
MEMBER Measures.DayOfYear AS
    [Date].[Calendar].[July 1, 2003].Properties('Day of Year')

SELECT StripCalculatedMembers(
    { Measures.DefaultMember
      , Measures.MemberName
      , Measures.MemberVal
      , Measures.MemberKey
      , Measures.MemberID
      , Measures.MemberCaption
      , Measures.DayName
      , Measures.DayNameTyped
      , Measures.DayOfWeek
```

```
    , Measures.DayofMonth
    , Measures.DayofYear
  }
) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

StrToMember

Returns the member specified by a Multidimensional Expressions (MDX)-formatted string.

Syntax

```
StrToMember(Member_Name [,CONSTRAINED] )
```

Arguments

Member_Name

A valid string expression specifying, directly or indirectly, a member.

Remarks

The **StrToMember** function returns the member specified in the string expression. The **StrToMember** function is typically used with user-defined functions to return a member specification from an external function back to an MDX statement, or when an MDX query is parameterized.

- When the CONSTRAINED flag is used, the member name must be directly resolvable to a qualified or unqualified member name. This flag is used to reduce the risk of injection attacks via the specified string. If a string is provided that is not directly resolvable to a qualified or unqualified member name, the following error appears: "The restrictions imposed by the CONSTRAINED flag in the STRTOMEMBER function were violated."
- When the CONSTRAINED flag is not used, the specified member can resolve either directly to a member name or can resolve to an MDX expression that resolves to a name.
- To better understand the differences between sets and members, see [Using Set Expressions](#) and [Using Member Expressions](#).

Examples

The following example returns the Reseller Sales Amount measure for the Bayern member in the State-Province attribute hierarchy using the **StrToMember** function. The specified string provided the qualified member name.

```
SELECT {StrToMember ('[Geography].[State-Province].[Bayern]')}
ON 0,
{[Measures].[Reseller Sales Amount]} ON 1
FROM [Adventure Works]
```

The following example returns the Reseller Sales Amount measure for the Bayern member using the **StrToMember** function. Since the member name string provided only an unqualified member name, the query returns the first instance of the specified member, which happens to be in the Customer Geography hierarchy in the Customer dimension, which does not intersect with the Reseller Sales. Best practices dictate specifying the qualified name to ensure expected results.

```
SELECT {StrToMember ('[Bayern]').Parent}
ON 0,
{[Measures].[Reseller Sales Amount]} ON 1
FROM [Adventure Works]
```

The following example returns the Reseller Sales Amount measure for the Bayern member in the State-Province attribute hierarchy using the **StrToMember** function. The member name string provided resolves to a qualified member name.

```
SELECT {StrToMember ('[Geography].[Geography].[Country].[Germany].FirstChild',
CONSTRAINED)}
ON 0,
{[Measures].[Reseller Sales Amount]} ON 1
FROM [Adventure Works]
```

The following example returns an error due to the CONSTRAINED flag. While the member name string provided contains a valid MDX member expression that resolves to a qualified member name, the CONSTRAINED flag requires qualified or unqualified member names in the member name string.

```
SELECT StrToMember ('[Geography].[Geography].[Country].[Germany].FirstChild',
CONSTRAINED)
ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

StrToSet

Returns the set specified by a Multidimensional Expressions (MDX)-formatted string.

Syntax

```
StrToSet(Set_Specification [,CONSTRAINED] )
```

Arguments

Set_Specification

A valid string expression specifying, directly or indirectly, a set.

Remarks

The **StrToSet** function returns the set specified in the string expression. The **StrToSet** function is typically used with user-defined functions to return a set specification from an external function back to an MDX statement, or when an MDX query is parameterized.

- When the CONSTRAINED flag is used, the set specification must contain qualified or unqualified member names or a set of tuples containing qualified or unqualified member names enclosed by braces {}. This flag is used to reduce the risk of injection attacks via the specified string. If a string is provided that is not directly resolvable to qualified or unqualified member names, the following error appears: "The restrictions imposed by the CONSTRAINED flag in the STRTOSET function were violated."
- When the CONSTRAINED flag is not used, the specified set specification can resolve to a valid Multidimensional Expressions (MDX) expression that returns a set.
- To better understand the differences between sets and members, see [Using Set Expressions](#) and [Using Member Expressions](#).

Examples

The following example returns the set of members of the State-Province attribute hierarchy using the **StrToSet** function. The set specification provides a valid MDX set expression.

```
SELECT StrToSet ('[Geography].[State-Province].Members')  
ON 0  
FROM [Adventure Works]
```

The following example returns an error due to the CONSTRAINED flag. While the set specification provides a valid MDX set expression, the CONSTRAINED flag requires qualified or unqualified member names in the set specification.

```
SELECT StrToSet ('[Geography].[State-Province].Members', CONSTRAINED)
ON 0
FROM [Adventure Works]
```

The following example returns the Reseller Sales Amount measure for the countries of Germany and Canada. The set specification provided in the specified string contains qualified member names, as required by the CONSTRAINED flag.

```
SELECT StrToSet
('{[Geography].[Geography].[Country].[Germany],[Geography].[Geography].[Country].[Canada]}', CONSTRAINED)
ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

StrToTuple

Returns the tuple specified by a Multidimensional Expressions (MDX)-formatted string.

Syntax

```
StrToTuple(tuple_specification [,CONSTRAINED] )
```

Arguments

tuple_specification

A valid string expression specifying, directly or indirectly, a tuple.

Remarks

The **StrToTuple** function returns the specified set. The **StrToTuple** function is typically used with user-defined functions to return a tuple specification from an external function back to an MDX statement.

- When the CONSTRAINED flag is used, the tuple specification must contain qualified or unqualified member names. This flag is used to reduce the risk of injection attacks via the

specified string. If a string is provided that is not directly resolvable to qualified or unqualified member names, the following error appears: "The restrictions imposed by the CONSTRAINED flag in the STRTOTUPLE function were violated."

- When the CONSTRAINED flag is not used, the specified tuple can resolve to a valid MDX expression that returns a tuple.

Examples

The following example returns the Reseller Sales Amount measure for the Bayern member for calendar year 2004. The tuple specification that is provided contains a valid MDX tuple expression.

```
SELECT StrToTuple ('([Geography].[State-Province].[Bayern],[Date].[Calendar Year].[CY 2004],[Measures].[Reseller Sales Amount])')
ON 0
FROM [Adventure Works]
```

The following example returns the Reseller Sales Amount measure for the Bayern member for calendar year 2004. The tuple specification that is provided contains qualified member names, as required by the CONSTRAINED flag.

```
SELECT StrToTuple ('([Geography].[State-Province].[Bayern],[Date].[Calendar Year].[CY 2004],[Measures].[Reseller Sales Amount]'),' , CONSTRAINED)
ON 0
FROM [Adventure Works]
```

The following example returns the Reseller Sales Amount measure for the Bayern member for calendar year 2004. The tuple specification that is provided contains a valid MDX tuple expression.

```
SELECT StrToTuple ('([Geography].[State-Province].[Bayern],[Date].[Calendar Year].&[2003].NEXTMEMBER,[Measures].[Reseller Sales Amount])')
ON 0
FROM [Adventure Works]
```

The following example returns an error due to the CONSTRAINED flag. While the tuple specification provided contains a valid MDX tuple expression, the CONSTRAINED flag requires qualified or unqualified member names in the tuple specification.

```
SELECT StrToTuple ('([Geography].[State-Province].[Bayern],[Date].[Calendar Year].&[2003].NEXTMEMBER,[Measures].[Reseller Sales Amount]'),' , CONSTRAINED)
ON 0
```

FROM [Adventure Works]

See Also

[MDX Function Reference \(MDX\)](#)

StrToValue

Returns the numeric value specified by a Multidimensional Expressions (MDX)–formatted string.

Syntax

```
StrToValue(MDX_Expression [,CONSTRAINED] )
```

Arguments

MDX_Expression

A valid string expression that resolves, directly or indirectly, to a single cell.

Remarks

The **StrToValue** function returns the numeric value specified by the MDX expression. The **StrToValue** function is typically used with user-defined functions to return an MDX expression from an external function back to an MDX statement that can be resolved to a single cell.

- When the CONSTRAINED flag is used, the MDX expression must contain only a scalar value. The CONSTRAINED flag is used to reduce the risk of injection attacks via the specified string. If a MDX expression is provided that is not directly resolvable to a scalar value, the following error appears: "The restrictions imposed by the CONSTRAINED flag in the STRTOVALUE function were violated."
- When the CONSTRAINED flag is not used, the specified MDX expression can be as complex as desired as long as it resolves to a valid Multidimensional Expressions (MDX) expression that returns a single cell.

Note

Returning the result of an MDX expression as a numeric value can be useful if the value is stored as text and you want to perform arithmetic operations on the returned values.

Example

The following example uses the **StrToValue** function to return the weight of each bicycle as a value.

```
WITH MEMBER Measures.x AS  
StrToValue
```

```

    ([Product].[Product].CurrentMember.Properties ('Weight')
,CONSTRAINED
)
SELECT Measures.x ON 0
,[Product].[Product].[Product].Members ON 1
FROM [Adventure Works]
WHERE [Product].[Product Categories].[Bikes]

```

See Also

[MDX Function Reference \(MDX\)](#)

Subset

Returns a subset of tuples from a specified set.

Syntax

```
Subset(Set_Expression, Start [,Count ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Start

A valid numeric expression that specifies the position of the first tuple to be returned.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Remarks

From the specified set, the **Subset** function returns a subset that contains the specified number of tuples, beginning at the specified start position. The start position is based on a zero-based index; that is, zero (0) corresponds to the first tuple in the specified set, 1 corresponds to the second, and so on.

If Count is not specified, the function returns all tuples from Start to the end of the set.

Example

The following example returns the Reseller Sales Measure for the top five selling subcategories of products, irrespective of hierarchy, based on Reseller Gross Profit. The **Subset** function is used to return only the first five sets in the result after the result is ordered using the **Order** function.

```
SELECT Subset
    (Order
        ([Product].[Product Categories].[SubCategory].members
        , [Measures].[Reseller Gross Profit]
        , BDESC
        )
    , 0
    , 5
    ) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Sum

Returns the sum of a numeric expression evaluated over a specified set.

Syntax

```
Sum( Set_Expression [ , Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) set expression.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the specified numeric expression is evaluated across the set and then summed. If a numeric expression is not specified, the specified set is evaluated in the

current context of the members of the set and then summed. If the SUM function is applied to a non-numeric expression, the results are undefined.



Note

Analysis Services ignores nulls when calculating the sum of a set of numbers.

Examples

The following example returns the sum of Reseller Sales Amounts for all members of the Product.Category attribute hierarchy for calendar years 2001 and 2002.

```
WITH MEMBER Measures.x AS SUM
    ( { [Date].[Calendar Year].&[2001]
      , [Date].[Calendar Year].&[2002] }
      , [Measures].[Reseller Sales Amount]
    )
SELECT Measures.x ON 0
, [Product].[Category].Members ON 1
```

```
FROM [Adventure Works]
```

The following example returns the sum of the month-to-date freight costs for Internet sales for the month of July, 2002 through the 20th day of July.

```
WITH MEMBER Measures.x AS SUM
    (
      MTD([Date].[Calendar].[Date].[July 20, 2002])
      , [Measures].[Internet Freight Cost]
    )
SELECT Measures.x ON 0
FROM [Adventure Works]
```

The following example uses the WITH MEMBER keyword and the **SUM** function to define a calculated member in the Measures dimension that contains the sum of the Reseller Sales Amount measure for the Canada and United States members of the Country attribute hierarchy in the Geography dimension.

```
WITH MEMBER Measures.NorthAmerica AS SUM
    (
      { [Geography].[Country].&[Canada]
      , [Geography].[Country].&[United States] }
      , [Measures].[Reseller Sales Amount]
    )
```

```
SELECT {[Measures].[NorthAmerica]} ON 0,
[Product].[Category].members ON 1
FROM [Adventure Works]
```

Often, the **SUM** function is used with the **CURRENTMEMBER** function or functions like **YTD** that return a set that varies depending on the currentmember of a hierarchy. For example, the following query returns the sum of the Internet Sales Amount measure for all dates from the beginning of the calendar year to the date displayed on the Rows axis:

```
WITH MEMBER MEASURES.YTDSUM AS
SUM(YTD(), [Measures].[Internet Sales Amount])
SELECT {[Measures].[Internet Sales Amount], MEASURES.YTDSUM} ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Tail

Returns a subset from the end of a set.

Syntax

```
Tail(Set_Expression [ ,Count ])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Remarks

The **Tail** function returns the specified number of tuples from the end of the specified set. The order of elements is preserved. The default value of Count is 1. If the specified number of tuples is less than 1, the function returns the empty set. If the specified number of tuples exceeds the number of tuples in the set, the function returns the original set.

Example

The following example returns the Reseller Sales Measure for the top five selling subcategories of products, irrespective of hierarchy, based on Reseller Gross Profit. The **Tail** function is used to return only the last five sets in the result after the result is reverse-ordered using the **Order** function.

```
SELECT Tail
    (Order
        ([Product].[Product Categories].[SubCategory].members
            , [Measures].[Reseller Gross Profit]
            ,BASC
        )
    ,5
    ) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

This

Returns the current subcube for use with assignments in the Multidimensional Expressions (MDX) calculation script.

Syntax

This

Remarks

The **This** function can be used in the place of any subcube expression to provide the current subcube within the current scope within the MDX calculation script. The **This** function must be used on the left side of an assignment.

Examples

The following MDX Script fragment shows how the This keyword can be used with SCOPE statements to make assignments to subcubes:

```
Scope
    (
```

```

    [Date].[Fiscal Year].&[2005],
    [Date].[Fiscal].[Fiscal Quarter].Members,
    [Measures].[Sales Amount Quota]
) ;

This = ParallelPeriod
(
    [Date].[Fiscal].[Fiscal Year], 1,
    [Date].[Fiscal].CurrentMember
) * 1.35 ;

/*-- Allocate equally to months in FY 2002 -----*/

Scope
(
    [Date].[Fiscal Year].&[2002],
    [Date].[Fiscal].[Month].Members
) ;

    This = [Date].[Fiscal].CurrentMember.Parent / 3 ;

End Scope ;
End Scope;

```

See Also

[MDX Function Reference \(MDX\)](#)

[Calculations](#)

ToggleDrillState

Toggles the drill state of members.

Syntax

```
ToggleDrillState(Set_Expression1,Set_Expression2 [ , RECURSIVE ] )
```


Arguments

Set_Expression1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set_Expression2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **ToggleDrillState** function toggles the drill state of each member of the second set that is present in the first set. The first set can contain tuples with any dimensionality, but the second set must contain members of a single dimension. The **ToggleDrillState** function is a combination of the **DrillupMember** and **DrilldownMember** functions. If the member, *m*, of the second set is present in the first set, and that member is drilled down (that is, has a descendant immediately following it), then `DrillupMember(Set_Expression1, {m})` is applied to the member or tuple in the first set. If that *m* member is drilled up (that is, there is no descendant of *m* that immediately follows *m*), `DrilldownMember(Set_Expression1, {m}[, RECURSIVE])` is applied to the first set.

If the optional **RECURSIVE** flag is used, drill up and drill down are applied recursively. For more information about the recursive flag, see the `DrillupMember` and `DrilldownMember` functions.

Querying the XMLA property `MdpropMdxDrillFunctions` enables you to verify the level of support that the server provides for the drilling functions; see [Supported XMLA Properties \(XMLA\)](#) for details.

Example

The following example drills down on the Australia member of the first set, and drills up on the United States member of the first set.

```
SELECT ToggleDrillState
    ({ [Geography].[Geography].[Country].Members,
      [Geography].[Geography].[Country].&[United States].Children},
     { [Geography].[Geography].[Country].[Australia]
       , [Geography].[Geography].[Country].&[United States] }
     --, RECURSIVE
) ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

TopCount

Sorts a set in descending order and returns the specified number of elements with the highest values.

Syntax

```
TopCount(Set_Expression,Count [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Count

A valid numeric expression that specifies the number of tuples to be returned.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

If a numeric expression is specified, the **TopCount** function sorts, in descending order, the tuples in the set specified by the specified set according to the value specified by the numeric expression, as evaluated over the specified set. After sorting the set, the **TopCount** function then returns the specified number of tuples with the highest value.

Important

Like the BottomCount function, the **TopCount** function always breaks the hierarchy.

If a numeric expression is not specified, the function returns the set of members in natural order, without any sorting, behaving like the Head (MDX) function.

Examples

The following example returns the top 10 dates by Internet Sales Amount:

```
SELECT [Measures].[Internet Sales Amount] ON 0,  
TOPCOUNT([Date].[Date].[Date].MEMBERS, 10, [Measures].[Internet Sales  
Amount])  
ON 1  
FROM [Adventure Works]
```

The following example returns, for the Bike category, the first five members in the set containing all combinations of members of the City level in the Geography hierarchy in the Geography

dimension and all fiscal years from the Fiscal hierarchy of the Date dimension, ordered by the Reseller Sales Amount measure (beginning with the members of this set with the largest number of sales).

```
SELECT [Measures].[Reseller Sales Amount] ON 0,
TopCount
    ( {[Geography].[Geography].[City].Members
      *[Date].[Fiscal].[Fiscal Year].Members}
    , 5
    , [Measures].[Reseller Sales Amount]
    ) ON 1
FROM [Adventure Works]
WHERE ([Product].[Product Categories].Bikes)
```

See Also

[MDX Function Reference \(MDX\)](#)

TopPercent

Sorts a set in descending order, and returns a set of tuples with the highest values whose cumulative total is equal to or greater than a specified percentage.

Syntax

```
TopPercent(Set_Expression, Percentage, Numeric_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Percentage

A valid numeric expression that specifies the percentage of tuples to be returned.



Important

Percentage needs to be a positive value; negative values generate an error.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **TopPercent** function calculates the sum of the specified numeric expression evaluated over the specified set, sorting the set in descending order. The function then returns the elements with the highest values whose cumulative percentage of the total summed value is at least the specified percentage. This function returns the smallest subset of a set whose cumulative total is at least the specified percentage. The returned elements are ordered largest to smallest.

Warning

- If Numeric_Expression returns any negative value then **TopPercent** returns only one (1) row.
- See the second example for a detailed presentation of this behavior.

Important

Like the BottomPercent function, the **TopPercent** function always breaks the hierarchy.

Example

The following example returns the best cities that help make the top 10% of the resellers' sales, for the Bike category. The result is sorted in descending order, beginning with the city that has the highest value of sales.

```
SELECT [Measures].[Reseller Sales Amount] ON 0,
TopPercent
  ({[Geography].[Geography].[City].Members}
, 10
, [Measures].[Reseller Sales Amount]
) ON 1
FROM [Adventure Works]
WHERE ([Product].[Product Categories].[Bikes])
```

The above expression produces the following results:

	Reseller Sales Amount
Toronto	\$3,508,904.84
London	\$1,521,530.09
Seattle	\$1,209,418.16
Paris	\$1,170,425.18

The original set of data can be obtained with the following query and returns 588 rows:

```

SELECT [Measures].[Reseller Sales Amount] ON 0,
Order
    ( {[Geography].[Geography].[City].Members}
    , [Measures].[Reseller Sales Amount]
    , BDESC
    ) ON 1
FROM [Adventure Works]
WHERE([Product].[Product Categories].[Bikes])

```

Example

The following walkthrough will help understand the effect of negative values in the Numeric_Expression. First let's build some context where we can present the behavior.

The following query returns a table of Resellers 'Sales Amount', 'Total Product Cost' and 'Gross Profit', sorted in descending order of profit. Please note there are only negative values for profit; so, the smallest loss appears at the top.

```

SELECT { [Measures].[Reseller Sales Amount], [Measures].[Reseller Total
Product Cost], [Measures].[Reseller Gross Profit] } ON columns
    , ORDER( [Product].[Product Categories].[Bikes].[Touring
Bikes].children, [Measures].[Reseller Gross Profit], BDESC ) ON rows
FROM [Adventure Works]

```

The above query returns the following results; rows from the middle section were removed for readability.

	Reseller Sales Amount	Reseller Total Product Cost	Reseller Gross Profit
Touring-2000 Blue, 50	\$157,444.56	\$163,112.57	(\$5,668.01)
Touring-2000 Blue, 46	\$321,027.03	\$333,021.50	(\$11,994.47)
Touring-3000 Blue, 62	\$87,773.61	\$100,133.52	(\$12,359.91)
...
Touring-1000	\$1,016,312.83	\$1,234,454.27	(\$218,141.44)

	Reseller Sales Amount	Reseller Total Product Cost	Reseller Gross Profit
Yellow, 46			
Touring-1000 Yellow, 60	\$1,184,363.30	\$1,443,407.51	(\$259,044.21)

Now, if you were asked to present the top 100% bikes by profit you would write a query like:

```
SELECT { [Measures].[Reseller Sales Amount], [Measures].[Reseller Total
Product Cost], [Measures].[Reseller Gross Profit] } ON columns
      , TOPPERCENT( [Product].[Product Categories].[Bikes].[Touring
Bikes].children, 100, [Measures].[Reseller Gross Profit] ) ON rows
FROM [Adventure Works]
```

Please note that the query asks for one hundred percent (100%); that means all rows should be returned. However, because there are negative values in the Numeric_Expression , only one row is returned.

	Reseller Sales Amount	Reseller Total Product Cost	Reseller Gross Profit
Touring-2000 Blue, 50	\$157,444.56	\$163,112.57	(\$5,668.01)

See Also

[MDX Function Reference \(MDX\)](#)

TopSum

Sorts a set and returns the topmost elements whose cumulative total is at least a specified value.

Syntax

TopSum(**Set_Expression**, **Value**, **Numeric_Expression**)

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Value

A valid numeric expression that specifies the value against which each tuple is compared.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression that returns a measure.

Remarks

The **TopSum** function calculates the sum of a specified measure evaluated over a specified set, sorting the set in descending order. The function then returns the elements with the highest values whose total of the specified numeric expression is at least the specified value. This function returns the smallest subset of a set whose cumulative total is at least the specified value. The returned elements are ordered largest to smallest.

Important

Like the BottomSum function, the **TopSum** function always breaks the hierarchy.

Example

The following example returns, for the Bike category, the smallest set of members of the City level in the Geography hierarchy in the Geography dimension whose cumulative total using the Reseller Sales Amount measure is at least the sum of 6,000,000 (beginning with the members of this set with the largest number of sales).

```
SELECT [Measures].[Reseller Sales Amount] ON 0,
TopSum
    ({[Geography].[Geography].[City].Members}
    , 6000000
    , [Measures].[Reseller Sales Amount]
    ) ON 1
FROM [Adventure Works]
WHERE ([Product].[Product Categories].Bikes)
```

See Also

[MDX Function Reference \(MDX\)](#)

TupleToStr

Returns a Multidimensional Expressions (MDX)–formatted string that corresponds to a specified tuple.

Syntax

```
TupleToStr(Tuple_Expression)
```

Arguments

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Remarks

This function is used to transfer a string-representation of a tuple to an external function for parsing. The string that is returned is enclosed in braces {} and each member, if more than one is expressly defined in the tuple, is separated by a comma.

Examples

The following example returns the string ([Date].[Calendar Year].&[2001],[Geography].[Geography].[Country].&[United States]) :

```
WITH MEMBER Measures.x AS TupleToStr
    (
        ([Date].[Calendar Year].&[2001]
        , [Geography].[Geography].[Country].&[United States]
        )
    )
SELECT Measures.x ON 0
FROM [Adventure Works]
```

The following example returns the same string as the previous example.

```
WITH SET s AS
    {
        ([Date].[Calendar Year].&[2001],
        [Geography].[Geography].[Country].&[United States]
        )
    }

```



```
MEMBER Measures.x AS TupleToStr ( s.Item(0) )
SELECT Measures.x ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Union

Returns a set that is generated by the union of two sets, optionally retaining duplicate members.

Syntax

Standard syntax

Union(**Set_Expression1**, **Set_Expression2** [...n][, ALL])

Alternate syntax 1

Set_Expression1 + **Set_Expression2** [...n]

Alternate syntax 2

{**Set_Expression1** , **Set_Expression2** [...n]}

Arguments

Set Expression 1

A valid Multidimensional Expressions (MDX) expression that returns a set.

Set Expression 2

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

This function returns the union of two or more specified sets. With the standard syntax and with alternate syntax 1, duplicates are eliminated by default. With the standard syntax, using the **ALL** flag keeps duplicates in the joined set. Duplicates are deleted from the tail of the set. With alternate syntax 2, duplicates are always retained.

Examples

The following examples demonstrate the behavior of the **Union** function using each syntax.

Standard syntax, duplicates eliminated

```
SELECT Union
    ([Date].[Calendar Year].children
    , {[Date].[Calendar Year].[CY 2002]}
    , {[Date].[Calendar Year].[CY 2003]}
    ) ON 0
FROM [Adventure Works]
```

Standard syntax, duplicates retained

```
SELECT Union
    ([Date].[Calendar Year].children
    , {[Date].[Calendar Year].[CY 2002]}
    , {[Date].[Calendar Year].[CY 2003]}
    , ALL
    ) ON 0
FROM [Adventure Works]
```

Alternate syntax 1, duplicates eliminated

```
SELECT
    [Date].[Calendar Year].children
    + {[Date].[Calendar Year].[CY 2002]}
    + {[Date].[Calendar Year].[CY 2003]} ON 0
FROM [Adventure Works]
```

Alternate syntax 2, duplicates retained

```
SELECT
    {[Date].[Calendar Year].children
    , [Date].[Calendar Year].[CY 2002]
    , [Date].[Calendar Year].[CY 2003]} ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)Union \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

UniqueName

Returns the unique name of a specified dimension, hierarchy, level, or member.

Syntax

Dimension expression syntax

`Dimension_Expression.UniqueName`

Hierarchy expression syntax

`Hierarchy_Expression.UniqueName`

Level expression syntax

`Level_Expression.UniqueName`

Member expression syntax

`Member_Expression.UniqueName`

Arguments

Dimension_Expression

A valid Multidimensional Expressions (MDX) expression that resolves to a dimension.

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Level_Expression

A valid Multidimensional Expressions (MDX) expression that returns a level.

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **UniqueName** function returns the unique name of the object, not the name returned by the Name function. The returned name does not include the name of the cube. The results returned

depend upon the server-side settings or the MDX Unique Name Style connection string property.

Example

The following example returns the unique name value for the Product dimension, the Product Categories hierarchy, the Subcategory level, and the Bike Racks member in the Adventure Works cube.

```
WITH MEMBER DimensionUniqueName
    AS [Product].UniqueName
MEMBER HierarchyUniqueName
    AS [Product].[Product Categories].UniqueName
MEMBER LevelUniqueName
    AS [Product].[Product Categories].[Subcategory].UniqueName
MEMBER MemberUniqueName
    AS [Product].[Product Categories].[Subcategory].[Bike Racks]
SELECT
    {DimensionUniqueName
    , HierarchyUniqueName
    , LevelUniqueName
    , MemberUniqueName }
    ON 0
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

UnknownMember

Returns the unknown member associated with a level or member.

Syntax

Member expression syntax

Member_Expression.UnknownMember

Hierarchy_expression syntax

Hierarchy_Expression.UnknownMember

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Hierarchy_Expression

A valid Multidimensional Expressions (MDX) expression that returns a hierarchy.

Remarks

Microsoft SQL Server Analysis Services creates an unknown member to associate fact table data with a hierarchy when the hierarchy is not known. The unknown member can be at one of the following levels:

- At the top level for attribute hierarchies that are not aggregated.
- At the first level below the **All** level for natural hierarchies.
- At any level for unnatural hierarchies.

If a member expression is specified, the **UnknownMember** function returns the unknown member child of the specified member. If the specified member does not exist, the function returns null.

If a hierarchy expression is specified, the **UnknownMember** function returns the unknown member at the top level if one exists.

If the unknown member does not exist on the level or member, the **UnknownMember** function creates a null member.

Note

If the unknown member does not exist on the hierarchy or member, an error is generated.

Examples

The following example returns the unknown member for the All Products member in the Product attribute hierarchy for all members of the Measures dimension.

```
SELECT [Product].[Product].[All Products].UnknownMember
       ON Columns,
[Measures].Members
       ON Rows
FROM [Adventure Works]
```

The following example returns the unknown member for the Product Categories hierarchy for all members of the Measures dimension.

```
SELECT [Product].[Product Categories].UnknownMember
       ON Columns,
[Measures].Members
       ON Rows
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Unorder

Removes any enforced ordering from a specified set.

Syntax

```
Unorder(Set_Expression)
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Remarks

The **Unorder** function removes any ordering imposed on the tuples contained in the set by any other function or statement, such as the Order function. The ordering of the tuples in the set returned by the **Unorder** function is indeterminate.

The **Unorder** function is used as a hint to Microsoft SQL Server Analysis Services for query optimization for set processing. If the order of tuples within a set is unimportant to a calculation or query, using the **Unorder** function can provide a performance benefit in such cases. For example, the NonEmpty (MDX) function may perform better when the set provided to this function is unordered than if Analysis Services needs to preserve order, although with SQL Server 2012 Analysis Services (SSAS), the query processor attempts to perform this function automatically for many functions, such as **Sum** and **Aggregate**. The performance benefit of using **Unorder** is only likely to be noticeable on very large sets consisting of millions of tuples.

Example

The following pseudo-code illustrates the syntax for this function.

```
NonEmpty (UnOrder (<set_expression>))
```

See Also

[MDX Function Reference \(MDX\)](#)

UserName

Returns the domain name and user name of the current connection.

Syntax

```
UserName [ ( ) ]
```

Remarks

The returned value is a string with the following format:

domain-name\user-name

Example

The following example returns the user name of the user that is executing the query.

```
WITH MEMBER Measures.x AS UserName
SELECT Measures.x ON COLUMNS
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

ValidMeasure

Returns the value of a measure in a cube by forcing inapplicable dimensions to their All level (or default member if not aggregatable) when returning the result for a specified tuple.

Syntax

```
ValidMeasure(Tuple_Expression)
```

Arguments

Tuple_Expression

A valid Multidimensional Expressions (MDX) expression that returns a tuple.

Remarks

The **ValidMeasure** function returns the value of a tuple, ignoring attributes that have no relationship with the measure group of the Measure whose value the tuple returns. An attribute can be unrelated to a measure for two reasons:

- The attribute's dimension has no relationship with the measure group of the measure in the tuple.
- The attribute's dimension does not have a relationship with the measure group of the measure, but the granularity attribute is not the key attribute, and the granularity attribute does not have a direct relationship with the attribute in the tuple.

The behavior specified by this function is the default server-side behavior and is controlled by the **IgnoreUnrelatedDimensions** property on the measure group object.

For each attribute in the specified tuple with granularity (that is to say, where the member in the tuple is not the All member), the current coordinate for each such attribute is moved as follows:

- Related attributes to the specified attribute member are moved to the member that exists with the current member.
- Relating attributes to the specified attribute member are moved to the All member (or the default member if the hierarchy is not aggregatable).
- Unrelated attributes are moved to the All member (based on measure).

Example

The following query shows how the ValidMeasure function can be used to override the behavior of the IgnoreUnrelatedDimensions property. In the Adventure Works cube, the Sales Targets measure group has IgnoreUnrelatedDimensions set to False; since the Date dimension joins to this measure group at the Calendar Quarter granularity, this means that the Sales Quota measure will, by default, return null below Calendar Quarter (although there is also a calculation in the MDX Script which allocates values down to the Month level too). Using the ValidMeasure function in a calculated measure can be used to make the Sales Quota measure behave as if IgnoreUnrelatedDimensions was set to True and force Sales Quota to display the value of the current Calendar Quarter.

```
WITH MEMBER MEASURES.VTEST AS VALIDMEASURE([Measures].[Sales Amount Quota])
SELECT {[Measures].[Sales Amount Quota], MEASURES.VTEST} ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```


Similarly, the Sales Targets measure group has no relationship at all with the Promotion dimension, so below the All Member of any hierarchy on Promotion it will return null. Again, this behavior can be changed by using ValidMeasure:

```
WITH MEMBER MEASURES.VTEST AS VALIDMEASURE([Measures].[Sales Amount Quota])
SELECT {[Measures].[Sales Amount Quota], MEASURES.VTEST} ON 0,
[Promotion].[Promotions].members ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

Value

Returns the value of the current member of the Measures dimension that intersects with the current member of the attribute hierarchies in the context of the query.

Syntax

```
Member_Expression[.Value]
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

The **Value** function returns the value of the specified member as a string. The **Value** argument is optional because the value of a member is the default property of a member, and is value that is returned for a member if no other value is specified. For more information about properties of members, see [MDX Function Reference \(MDX\)](#) and [User-defined Member Properties \(MDX\)](#).

Examples

The following example returns the value of a member as well explicitly returning the name of the member.

```
WITH MEMBER [Date].[Calendar].NumericValue as [Date].[Calendar].[July 1,
2001].Value
MEMBER [Date].[Calendar].MemberName AS [Date].[Calendar].[July 1, 2001].Name

SELECT {[Date].[Calendar].NumericValue, [Date].[Calendar].MemberName} ON 0
```

```
from [Adventure Works]
```

The following example returns the value of a member, as the default value that is returned for a member on an axis.

```
SELECT {[Date].[Calendar].[July 1, 2001]} ON 0  
from [Adventure Works]
```

See Also

[MemberValue \(MDX\)](#)

[Properties \(MDX\)](#)

[Name \(MDX\)](#)

[UniqueName \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Var

Returns the sample variance of a numeric expression evaluated over a set, using the unbiased population formula (dividing by n).

Syntax

```
Var(Set_Expression [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **Var** function returns the unbiased variance of a specified numeric expression evaluated over a specified set.

The **Var** function uses the unbiased population formula, and the **VarP** function uses the biased population formula.

See Also

[MDX Function Reference \(MDX\)](#)

Variance

Alias for the Var function.

See Also

[MDX Function Reference \(MDX\)](#)

VarianceP

Alias for the VarP function.

See Also

[MDX Function Reference \(MDX\)](#)

VarP

Returns the population variance of a numeric expression evaluated over a set, using the biased population formula (dividing by $n-1$).

Syntax

```
VarP(Set_Expression [ ,Numeric_Expression ] )
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Numeric_Expression

A valid numeric expression that is typically a Multidimensional Expressions (MDX) expression of cell coordinates that return a number.

Remarks

The **VarP** function returns the biased variance of a specified numeric expression, evaluated over a specified set.

The **VarP** function uses the biased population formula, while the Var function uses the unbiased population formula.

See Also

[MDX Function Reference \(MDX\)](#)

VisualTotals

Returns a set generated by dynamically totaling child members in a specified set, optionally using a pattern for the name of the parent member in the result set.

Syntax

```
VisualTotals(Set_Expression[,Pattern])
```

Arguments

Set_Expression

A valid Multidimensional Expressions (MDX) expression that returns a set.

Pattern

A valid string expression for the parent member of the set, that contains an asterisk (*) as the substitution character for the parent name.

Remarks

The specified set expression can specify a set that contains members at any level within a single dimension, generally members with an ancestor-descendant relationship. The **VisualTotals** function totals the values of the child members in the specified set and ignores child members that are not in the set in calculating the result totals. Totals are visually totaled for sets ordered in hierarchy order. If the order of members in sets breaks the hierarchy, results are not visual totals. For example, `VisualTotals (USA, WA, CA, Seattle)` does not return WA as Seattle, but rather returns the values for WA, CA, and Seattle, then totals these values as the visual total for USA, counting the sales for Seattle twice.

Note

Applying the **VisualTotals** function to dimension members that are not related to a measure or are under the measure group granularity will cause values to be replaced with null.

Pattern, which is optional, specifies the format for the totals label. Pattern requires an asterisk (*) as the substitution character for the parent member and the remainder of the text in the string appears in the result concatenated with the parent name. To display a literal asterisk, use two asterisks (**).

Examples

The following example returns the visual total for the third quarter of the 2001 calendar year based on the single descendant specified - the month of July.

```
SELECT VisualTotals
```

```

    ( {[Date].[Calendar].[Calendar Quarter].&[2001]&[3]
      , [Date].[Calendar].[Month].&[2001]&[7]} ) ON 0
FROM [Adventure Works]

```

The following example returns the [All] member of the Category attribute hierarchy in the Product dimension together with two of its four children. The total returned for the [All] member for the Internet Sales Amount measure is the total for the Accessories and Clothing members only. Also, the pattern argument is used to specify the label for the [All Products] column.

```

SELECT
    VisualTotals
    ( {[Product].[Category].[All Products]
      , [Product].[Category].[Accessories]
      , [Product].[Category].[Clothing]}
      , '* - Visual Total'
    ) ON Columns
, [Measures].[Internet Sales Amount] ON Rows
FROM [Adventure Works]

```

See Also

[MDX Function Reference \(MDX\)](#)

Wtd

Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Week level in the Time dimension.

Syntax

```
Wtd( [ Member_Expression ] )
```

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

If a member expression is not specified, the default is the current member of the first hierarchy with a level of type Weeks in the first dimension of type Time (**Time.CurrentMember**) in the measure group.

The **Wtd** function is a shortcut function for the PeriodsToDate function where the level is set to Weeks. That is, `Wtd(Member_Expression)` is equivalent to `PeriodsToDate(Week_Level_Expression,Member_Expression)`.

See Also

[MDX Function Reference \(MDX\)](#)

[Mtd \(MDX\)](#)

[Ytd \(MDX\)](#)

[MDX Function Reference \(MDX\)](#)

Ytd

Returns a set of sibling members from the same level as a given member, starting with the first sibling and ending with the given member, as constrained by the Year level in the Time dimension.

Syntax

`Ytd([Member_Expression])`

Arguments

Member_Expression

A valid Multidimensional Expressions (MDX) expression that returns a member.

Remarks

If a member expression is not specified, the default is the current member of the first hierarchy with a level of type Years in the first dimension of type Time in the measure group.

The **Ytd** function is a shortcut function for the PeriodsToDate function where the Type property of the attribute hierarchy on which the level is based is set to Years. That is,

`Ytd(Member_Expression)` is equivalent to

`PeriodsToDate(Year_Level_Expression,Member_Expression)`. Note that this function will not work when the Type property is set to FiscalYears.

Example

The following example returns the sum of the `Measures.[Order Quantity]` member, aggregated over the first eight months of calendar year 2003 that are contained in the `Date` dimension, from the **Adventure Works** cube.

```
WITH MEMBER [Date].[Calendar].[First8MonthsCY2003] AS
    Aggregate (
        YTD([Date].[Calendar].[Month].[August 2003])
    )
SELECT
    [Date].[Calendar].[First8MonthsCY2003] ON COLUMNS,
    [Product].[Category].Children ON ROWS
FROM
    [Adventure Works]
WHERE
    [Measures].[Order Quantity]
```

Ytd is frequently used in combination with no parameters specified, meaning that the [CurrentMember \(MDX\)](#) function will display a running cumulative year-to-date total in a report, as shown in the following query:

```
WITH MEMBER MEASURES.YTDDEMO AS
AGGREGATE(YTD(), [Measures].[Internet Sales Amount])
SELECT {[Measures].[Internet Sales Amount], MEASURES.YTDDEMO} ON 0,
[Date].[Calendar].MEMBERS ON 1
FROM [Adventure Works]
```

See Also

[MDX Function Reference \(MDX\)](#)

MDX Reserved Words

The following table contains words reserved for use by Multidimensional Expressions (MDX). You should not use these words as part of any identifier, such as a cube name, or user-defined function name, in MDX.

ABSOLUTE	DESC	LEAVES	SELF_BEFORE_AFTER
ACTIONPARAMETERSE T	DESCENDANTS	LEVEL	SESSION

ADDCALCULATEDMEMBERS	DESCRIPTION	LEVELS	SET
AFTER	DIMENSION	LINKMEMBER	SETTOARRAY
AGGREGATE	DIMENSIONS	LINREGINTERCEPT	SETTOSTR
ALL	DISTINCT	LINREGPOINT	SORT
ALLMEMBERS	DISTINCTCOUNT	LINREGR2	STDDEV
ANCESTOR	DRILLDOWNLEVEL	LINREGSLOPE	STDDEVP
ANCESTORS	DRILLDOWNLEVELBOTTOM	LINREGVARIANCE	STDEV
AND	DRILLDOWNLEVELTOP	LOOKUPCUBE	STDEVP
AS	DRILLDOWNMEMBER	MAX	STORAGE
ASC	DRILLDOWNMEMBERBOTTOM	MEASURE	STRIPCALCULATEDMEMBERS
ASCENDANTS	DRILLDOWNMEMBERTOP	MEDIAN	STRTOMEMBER
AVERAGE	DRILLUPLEVEL	MEMBER	STRTOSET
AXIS	DRILLUPMEMBER	MEMBERS	STRTOTUPLE
BASC	DROP	MEMBERTOSTR	STRTOVAL
BDESC	EMPTY	MIN	STRTOVALUE
BEFORE	END	MTD	SUBSET
BEFORE_AND_AFTER	ERROR	NAME	SUM
BOTTOMCOUNT	EXCEPT	NAMETOSET	TAIL
BOTTOMPERCENT	EXCLUDEEMPTY	NEST	THIS
BOTTOMSUM	EXTRACT	NEXTMEMBER	TOGGLEDRIILLSTATE
BY	FALSE	NO_ALLOCATION	TOPCOUNT
CACHE	FILTER	NO_PROPERTIES	TOPPERCENT
CALCULATE	FIRSTCHILD	NON	TOPSUM
CALCULATION	FIRSTSIBLING	NONEMPTYCROSSJOIN	TOTALS
CALCULATIONCURRENTPASS	FOR	NOT_RELATED_TO_FACTS	TREE

CALCULATIONPASSVALUE	FREEZE	NULL	TRUE
CALCULATIONS	FROM	ON	TUPLETOSTR
CALL	GENERATE	OPENINGPERIOD	TYPE
CELL	GLOBAL	OR	UNION
CELLFORMULASETLIST	GROUP	PAGES	UNIQUE
CHAPTERS	GROUPING	PARALLELPERIOD	UNIQUENAME
CHILDREN	HEAD	PARENT	UPDATE
CLEAR	HIDDEN	PASS	USE
CLOSINGPERIOD	HIERARCHIZE	PERIODSTODATE	USE_EQUAL_ALLOCATION
COALESCEEMPTY	HIERARCHY	POST	USE_WEIGHTED_ALLOCATION
COLUMN	IGNORE	PREDICT	USE_WEIGHTED_INCREMENT
COLUMNS	IIF	PREVMEMBER	USERNAME
CORRELATION	INCLUDEEMPTY	PROPERTIES	VALIDMEASURE
COUNT	INDEX	PROPERTY	VALUE
COUSIN	INTERSECT	QTD	VAR
COVARIANCE	IS	RANK	VARIANCE
COVARIANCEN	ISANCESTOR	RECURSIVE	VARIANCEP
CREATE	ISEMPTY	RELATIVE	VARP
CREATEPROPERTYSET	ISGENERATION	ROLLUPCHILDREN	VISUAL
CREATEVIRTUALDIMENSION	ISLEAF	ROOT	VISUALTOTALS
CROSSJOIN	ISSIBLING	ROWS	WHERE
CUBE	ITEM	SCOPE	WITH
CURRENT	LAG	SECTIONS	WTD
CURRENTCUBE	LASTCHILD	SELECT	XOR
CURRENTMEMBER	LASTPERIODS	SELF	YTD

DEFAULT_MEMBER	LASTSIBLING	SELF_AND_AFTER	
DEFAULTMEMBER	LEAD	SELF_AND_BEFORE	

See Also

[MDX Language Reference \(MDX\)](#)

[MDX Language Reference \(MDX\)](#)