MDX Built-in Functions - Aggregate and Numeric Functions

* [Aggregate](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-aggregate)
* [AVG](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-avg)
* [CoalesceEmpty](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-coalesceempty)
* [Count](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-count-)
* [DistinctCount](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-distinctcount)
* [IIF](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-iif)
* [LookupCube](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-lookupcube)
* [MAX](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-max)
* [Median](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-median)
* [MIN](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-min)
* [Ordinal](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-ordinal-)
* [Rank](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-rank-)
* [STDEV](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-stdev-)
* [STDEVP](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-stdevp)
* [StrToValue](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-strtovalue)
* [SUM](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-sum-)
* [VAR](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-var)
* [VARP](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#-varp)

FIXFIX - HREF ORPHANS

See Also: [Main\_Page](https://www.toadworld.com/platforms/sql-server/w/wiki/9386.sql-server-topics) - [Database Administration](https://www.toadworld.com/platforms/sql-server/w/wiki/9492.database-administration) - [Analysis Services](https://www.toadworld.com/platforms/sql-server/w/wiki/9493.analysis-services) - [MDX with Analysis Services](https://www.toadworld.com/platforms/sql-server/w/wiki/9551.mdx-with-analysis-services) - [MDX Built-In Functions](https://www.toadworld.com/platforms/sql-server/w/wiki/9556.mdx-built-in-functions)

Aggregate and numeric functions allow you to perform calculations on your measures. Following are the most commonly used numeric functions:

|  |  |
| --- | --- |
| <a href="ASMDX7C#AGGREGATE">Aggregate]] | <a href="ASMDX7C#MIN">MIN]] |
| <a href="ASMDX7C#AVG">AVG]] | <a href="ASMDX7C#ORDINAL">Ordinal]] |
| <a href="ASMDX7C#COALESCEEMPTY">CoalesceEmpty]] | <a href="ASMDX7C#RANK">Rank]] |
| <a href="ASMDX7C#COUNT">Count]] | <a href="ASMDX7C#STDEV">STDEV]] |
| <a href="ASMDX7C#DISTINCTCOUNT">DistinctCount]] | <a href="ASMDX7C#STDEVP">STDEVP]] |
| <a href="ASMDX7C#IIF">IIF]] | <a href="ASMDX7C#STRTOVALUE">StrToValue]] |
| <a href="ASMDX7C#LOOKUPCUBE">LookupCube]] | <a href="ASMDX7C#SUM">SUM]] |
| <a href="ASMDX7C#MAX">MAX]] | <a href="ASMDX7C#VAR">VAR]] |
| <a href="ASMDX7C#MEDIAN">Median]] | <a href="ASMDX7C#VARP">VARP]] |

<a name='Aggregate'>]]

|  |
| --- |
| Contents   * [1 Aggregate](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#Aggregate) * [2 AVG](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#AVG) * [3 CoalesceEmpty](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#CoalesceEmpty) * [4 Count](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#Count) * [5 DistinctCount](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#DistinctCount) * [6 IIF](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#IIF) * [7 LookupCube](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#LookupCube) * [8 MAX](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#MAX) * [9 Median](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#Median) * [10 MIN](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#MIN) * [11 Ordinal](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#Ordinal) * [12 Rank](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#Rank) * [13 STDEV](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#STDEV) * [14 STDEVP](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#STDEVP) * [15 StrToValue](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#StrToValue) * [16 SUM](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#SUM) * [17 VAR](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#VAR) * [18 VARP](https://www.toadworld.com/platforms/sql-server/w/wiki/10358.mdx-built-in-functions-aggregate-and-numeric-functions#VARP) |

Aggregate

Aggregate function performs a calculation based on the aggregation type of member and returns the result of the calculation. The syntax is AGGREGATE(SET, numeric expression) where "numeric expression" is a measure in the current cube; the numeric expression argument is optional. If the numeric expression is not specified then the AGGREGATE function uses the context of the query to decide which measure to aggregate. Furthermore, if you don't specify the numeric expression AGGREGATE will use the aggregation type that was specified when building the dimension. For example, if the aggregation type was MIN then the AGGREGATE function will return the least value. If a numeric expression is specified then the values are summed regardless of what aggregation type was used to build the dimension.

Supported aggregation types and their respective behavior is summarized in the following table:

|  |  |
| --- | --- |
| **Aggregation Type** | **AGGREGATE behavior** |
| MAX | Returns the greatest value in the set |
| MIN | Returns the least value in the set |
| COUNT | Returns the count of all values in the set |
| SUM | Returns the sum of all values in the set |
| DISTINCT COUNT | Generates an error |

**NOTE:** this function does NOT work on calculated members in MSAS 2000.

For example, the following query returns order counts for calendar years 2001 and 2002, as well as the sum of two years:

WITH member [date].[calendar].[2001 and 2002] AS 'AGGREGATE({[date].[calendar].[calendar year].[cy 2001], [date].[calendar].[calendar year].[cy 2002]})' SELECT {[date].[calendar].[2001 and 2002], [date].[calendar].[calendar year].[cy 2001], [date].[calendar].[calendar year].[cy 2002]} ON COLUMNS, {measures.[order count]} ON ROWS FROM [adventure works]

Results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **2001 and 2002** | **CY 2001** | **CY 2002** |
| Order Count | 5,071 | 1,379 | 3,692 |

The next example specifies the numeric expression as sales amount and aggregates it for calendar years 2001 and 2002:

WITH member [date].[calendar].[2001 and 2002] AS 'AGGREGATE({[date].[calendar].[calendar year].[cy 2001], [date].[calendar].[calendar year].[cy 2002]} , [measures].[sales amount] )' SELECT {[date].[calendar].[2001 and 2002], [date].[calendar].[calendar year].[cy 2001], [date].[calendar].[calendar year].[cy 2002]} ON COLUMNS, { measures.[sales amount]} ON ROWS FROM [adventure works]

Results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **2001 and 2002** | **CY 2001** | **CY 2002** |
| Sales Amount | $42,006,582.14 | $11,331,808.96 | $30,674,773.18 |

<a name='Aggregate\_END'>]] <a name='AVG'>]]

AVG

AVG function returns the average value of each measure referenced in the query, or average value of a specific measure as it applies to a given set. The function accepts two arguments: AVG(SET, numeric expression) where numeric expression is a measure in the current cube. If the numeric expression is NOT specified then average is derived for the measure referenced in the query. For example, the following query retrieves the average of sales amount as well as order count within "pumps" subcategory:

WITH member [product].[product categories].[average pump sales] AS 'AVG({[product].[product categories].[subcategory].[pumps]})' SELECT { [product].[product categories].[average pump sales]} ON COLUMNS FROM [adventure works] WHERE ([measures].[sales amount])

Results:

|  |
| --- |
| **average pump sales** |
| $13,514.69 |

The next example only retrieves the average order count for the "pumps" subcategory:

WITH member [measures].[average pump sales] AS 'AVG({[product].[product categories].[subcategory].[pumps]}, [measures].[order count])' SELECT { [measures].[average pump sales]} ON COLUMNS FROM [adventure works]

Results:

|  |
| --- |
| **average pump sales** |
| 267 |

The next example returns average gross profit margin for each calendar year:

WITH MEMBER Measures.[Avg Gross Profit Margin] AS Avg( Descendants( [Date].[Calendar].CurrentMember, [Date].[Calendar].[Date] ), Measures.[Gross Profit Margin] ) SELECT Measures.[Avg Gross Profit Margin] ON COLUMNS, [Date].[Calendar].[Calendar year].Members ON ROWS FROM [Adventure Works]

Results:

|  |  |
| --- | --- |
|  | **Avg Gross Profit Margin** |
| CY 2001 | 38.92% |
| CY 2002 | 39.20% |
| CY 2003 | 40.42% |
| CY 2004 | 42.33% |

<a name='AVG\_END'>]] <a name='CoalesceEmpty'>]]

CoalesceEmpty

CoalesceEmpty function evaluates multiple string or numeric values and returns the first non-empty value. The syntax is the same whether you use string or numeric values: CoalesceEmpty(expression1, expression2, â€¦ expressionN). Values are examined from left to right. You cannot mix string values with numeric values. If none of the values can be evaluated to non empty then the function will return an empty value. For example, the following query examines order count, reseller order count and internet order count measures; if all of these measures are empty then the number zero is returned instead of an empty value:

WITH member [measures].[non empty count] AS 'COALESCEEMPTY([measures].[order count], [measures].[reseller order count], [measures].[internet order count],0)' SELECT {[promotion].[end date].members} ON ROWS, {[measures].[order count], [measures].[non empty count]} ON COLUMNS FROM [adventure works]

Results:

|  |  |  |
| --- | --- | --- |
|  | **Order Count** | **non empty count** |
| All Promotions | 31455 | 31455 |
| 30-Jun-02 | 24 | 24 |
| 31-Jul-02 | 36 | 36 |
| 31-Aug-02 | 61 | 61 |
| 31-Jul-03 | 30 | 30 |
| 15-Aug-03 | (null) | 0 |
| 30-Aug-03 | (null) | 0 |
| 15-Sep-03 | (null) | 0 |
| 30-Sep-03 | 149 | 149 |
| 30-Jun-04 | 2831 | 2831 |
| 31-Dec-04 | 30901 | 30901 |

<a name='CoalesceEmpty\_END'>]] <a name='Count'>]]

Count

Count function returns the number of items in a collection: dimension, level, set or a tuple. Each collection has a different syntax, as follows:

**Dimensions.count** - returns the count of all dimensions in the current cube, including measures. For example, the following query will return the number of dimensions within the adventure works cube:

WITH member [measures].[dim count] AS '(dimensions.count)' SELECT {[measures].[dim count]} ON COLUMNS FROM [adventure works]

**Dimension.Hierarchy.Levels.Count** - returns the number of levels within a given dimension (if no hierarchy is specified) or within a specific hierarchy of the dimension. The count of levels includes the "ALL" level if one exists for the current dimension. For example, the following query returns the number of levels within product categories' hierarchy of the product dimension:

WITH member [measures].[level count] AS '([product].[product categories].levels.count)' SELECT {[measures].[level count]} ON COLUMNS FROM [adventure works]

The count function has two variations as it applies to sets. The first flavor allows you to include or exclude empty sets with the following syntax:

COUNT(SET, (ExcludeEmpty | IncludeEmpty)

If you don't specify either ExcludeEmpty or IncludeEmpty flags, the count function will include empty members. For example, the following expression returns the count of all members within product categories' hierarchy of the product dimension whether they had any sales or not:

WITH member [measures].[member count] AS 'COUNT( CROSSJOIN( {[product].[product categories].members}, {[measures].[sales amount]}) )' SELECT {[measures].[member count]} ON COLUMNS FROM [adventure works]

The next query excludes those members that did NOT have any sales:

WITH member [measures].[member count] AS 'COUNT( CROSSJOIN({[product].[product categories].members}, {[measures].[sales amount]}), ExcludeEMPTY )' SELECT {[measures].[member count]} ON COLUMNS FROM [adventure works]

The other variation of the count function that applies to sets is appended to the end of the set and always includes empty members. The syntax is SET.Count. For example, the following query will count all members within product categories' hierarchy of the product dimension:

WITH member [measures].[member count] AS 'CROSSJOIN( {[product].[product categories].members}, {[measures].[sales amount]}).Count' SELECT {[measures].[member count]} ON COLUMNS FROM [adventure works]

This query is equivalent to the first example referring to the count function as it applies to sets. There is no performance benefit to either set syntax for the COUNT function. You should use the former syntax in those cases when you want to exclude empty members.

**Tuple.Count** construct returns the number of dimensions within a tuple. For example, the following query returns 3 because the tuple contains three dimensions:

WITH member [measures].[tuple dim count] AS '([product].[product categories].[category].[bikes], [measures].[sales amount], [date].[calendar].[calendar year].[cy 2002]).Count' SELECT {[measures].[tuple dim count]} ON COLUMNS FROM [adventure works]

<a name='Count\_END'>]] <a name='DistinctCount'>]]

DistinctCount

DistinctCount function returns the count of distinct (that means no duplicates), nonempty tuples of the given set. This function accepts a set as a single argument: DistinctCount(SET). For example, the following query returns a distinct subcategory count that has had sales:

WITH member [measures].[distinct member count] AS 'DistinctCount( CROSSJOIN( [product].[product categories].[subcategory].members, {[measures].[sales amount]} ) )' SELECT {[measures].[distinct member count]} ON COLUMNS FROM [adventure works]

<a name='DistinctCount\_END'>]] <a name='IIF'>]]

IIF

IIF function returns one of two values as determined by logical test. IIF is sometimes referred to as "immediate IF". The syntax of the function is:

IIF(logical expression, True expression, False expression)

If the logical expression evaluates to TRUE then the true expression is returned; otherwise the false expression is returned. For example, the following expression checks whether the previous member within the fiscal hierarchy of the date dimension has a non-empty customer count; if the customer count is empty the function returns NULL, otherwise it returns the growth of customers over the prior period:

IIF(ISEMPTY( ([Date].[Fiscal].CurrentMember.PrevMember, [Measures].[Customer Count] ) ), NULL, ( [Date].[Fiscal].CurrentMember, [Measures].[Customer Count] ) - ( [Date].[Fiscal].PrevMember, [Measures].[Customer Count] ) ) / ( [Date].[Fiscal].PrevMember,[Measures].[Customer Count] )

IIF statements can be nested. For example, you might decide to extend the example above by checking to see if the current member is the topmost member of the fiscal hierarchy. The topmost level ("all periods") should not be compared with any other level; therefore you simply return "NA" if indeed the IIF function is checking the topmost member in the fiscal hierarchy. The rest of the query is identical to the one above:

//first check if we're on topmost level //if so, return "NA" IIF([Date].[Fiscal].CurrentMember.Level.Ordinal = 0, "NA", //nested IIF starts here. This part is only executed // if we're NOT on the topmost level: IIF(ISEMPTY( ([Date].[Fiscal].CurrentMember.PrevMember, [Measures].[Customer Count] ) ), //previous years customer count is empty so return NULL: NULL, ( [Date].[Fiscal].CurrentMember, [Measures].[Customer Count] ) - ( [Date].[Fiscal].PrevMember, [Measures].[Customer Count] ) / ( [Date].[Fiscal].PrevMember, [Measures].[Customer Count] ) ) )

IIF is the only way to evaluate logical conditions with MSAS 2000. With MSAS 2000 both "true expression" and "false expression" must have the same data type, that is, you cannot combine a string and a numeric expression in a single IIF statement. With MSAS 2005 IIF no longer has this limitation. Furthermore, MSAS 2005 supports CASE statements which are much easier to read than multiple nested IIF statements.

<a name='IIF\_END'>]] <a name='LookupCube'>]]

LookupCube

LookupCube function returns the value of a specified string or numeric MDX expression evaluated against a specified cube. The syntax is:

LookupCube(cube name, numeric OR string expression)

For example the following query looks up the count of active employees for the current facility by querying the "HR" cube:

LookupCube( "HR", "(" + MemberToStr([Facility].CurrentMember) + "," + MemberToStr([status].[active]) + ", [measures].[employee count])")

The cube referred to by this function must reside in the same database as the current cube. This function was useful with MSAS 2000 because each cube was limited to a single fact table. With MSAS 2005 you can have multiple measure groups (fact tables) in a single cube; therefore with MSAS 2005 the LookupCube function is deprecated but supported for backward compatibility.

<a name='LookupCube\_END'>]] <a name='MAX'>]]

MAX

MAX function returns the maximum value of a numeric expression evaluated over a set. The syntax is: MAX(SET, numeric expression) where "numeric expression" is a measure within the current cube. For example, the following query returns the largest sale among all product categories within 2001:

WITH member [measures].[max sales amount] AS 'MAX({[product].[product categories].members}, [measures].[sales amount] )' SELECT {[measures].[max sales amount]} ON COLUMNS FROM [adventure works] WHERE ([date].[calendar].[calendar year].[cy 2001])

<a name='MAX\_END'>]] <a name='Median'>]]

Median

Median function returns the median value of a numeric expression evaluated over a set. The syntax is: Median(SET, numeric expression) where "numeric expression" is a measure within the current cube. For example, the following query returns the median sale among all product categories within 2001:

WITH member [measures].[median sales amount] AS 'MEDIAN({[product].[product categories].members}, [measures].[sales amount] )', FORMAT\_STRING = 'currency' SELECT {[measures].[median sales amount]} ON COLUMNS FROM [adventure works] WHERE ([date].[calendar].[calendar year].[cy 2001])

<a name='Median\_END'>]] <a name='MIN'>]]

MIN

MIN function returns the minimum value of a numeric expression evaluated over a set. The syntax is: MIN(SET, numeric expression) where "numeric expression" is a measure within the current cube. For example, the following query returns the smallest sale among all product categories within 2001:

WITH member [measures].[min sales amount] AS 'MIN({[product].[product categories].members}, [measures].[sales amount] )' SELECT {[measures].[min sales amount]} ON COLUMNS FROM [adventure works] WHERE ([date].[calendar].[calendar year].[cy 2001])

<a name='MIN\_END'>]] <a name='Ordinal'>]]

Ordinal

Ordinal function returns the position number of a particular level within a dimension hierarchy. The syntax is Level.Ordinal. For example, the following query returns 2 because subcategory is the second level (below category) in the product categories' hierarchy:

WITH member [measures].[ordinal] AS '[product].[product categories].[subcategory].Ordinal' SELECT {[measures].[ordinal]} ON COLUMNS FROM [adventure works]

The following will return 5 because date level is the 5th level (below calendar year, calendar semester, calendar quarter, and month) in the calendar hierarchy:

WITH member [measures].[ordinal] AS '[date].[calendar].[date].Ordinal' SELECT {[measures].[ordinal]} ON COLUMNS FROM [adventure works]

Topmost levels (all) will have the ordinal of zero.

<a name='Ordinal\_END'>]] <a name='Rank'>]]

Rank

Rank function returns either an ordinal position of the tuple in the set or "rank" of a given tuple evaluated against the rest of the tuples in the set. The syntax is: RANK(TUPLE, SET, measure); measure argument is optional. If measure is not specified RANK simply returns the ordinal position of the tuple within the set. For example, the following query returns the ordinal position of "bike racks" within the accessories category of the product categories' hierarchy:

WITH member [measures].[rank] AS 'RANK( ([product].[product categories].[bike racks]), {[product].[product categories].[category].[accessories].children} )' SELECT {[measures].[rank]} ON COLUMNS FROM [adventure works]

Bike racks happens to be the first member so the query returns 1. If you execute the same query against "pumps" instead of "bike racks" you will get 11 because pumps happens to be in the 11th position.

If the measure is specified then the rank function will evaluate the tuple against the set to determine its rank. For example, the following query evaluates the ranking of bike racks by order count within the accessories category:

WITH member [measures].[rank] AS 'RANK( ([product].[product categories].[bike racks]), {[product].[product categories].[category].[accessories].children} ,measures.[order count] )' SELECT {[measures].[rank]} ON COLUMNS FROM [adventure works]

<a name='Rank\_END'>]] <a name='STDEV'>]]

STDEV

STDEV function returns the standard deviation of a measure evaluated over a set using the unbiased population formula. The syntax is STDEV(SET, numeric expression), where numeric expression is a measure within the current cube. For example, the following query returns a standard deviation of order counts across members of the category level of the product categories' hierarchy, along with actual order counts for each category:

WITH member [measures].[standard deviation] AS 'STDEV( [product].[product categories].[category].members, [measures].[order count] )' SELECT {[measures].[standard deviation], [measures].[order count]} ON ROWS, {[product].[product categories].[category].members} ON COLUMNS FROM [adventure works]

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accessories** | **Bikes** | **Clothing** | **Components** |
| Standard deviation | 7908.055998 | 7908.055998 | 7908.055998 | 7908.055998 |
| Order Count | 19,523 | 18,358 | 9,871 | 2,646 |

STDDEV function does the same thing as STDEV.

<a name='STDEV\_END'>]] <a name='STDEVP'>]]

STDEVP

STDEVP function returns the population standard deviation of a measure evaluated over a set using the biased population formula. The syntax of this function is identical to that of STDEV: STDEVP(SET, numeric expression) where numeric expression is a measure within the current cube. For example, the following query returns the population standard deviation of order counts across members of the category level of the product categories' hierarchy, along with actual order counts for each category:

WITH member [measures].[standard deviation] AS 'STDEVP( [product].[product categories].[category].members, [measures].[order count] )' SELECT {[measures].[standard deviation], [measures].[order count]} ON ROWS, {[product].[product categories].[category].members} ON COLUMNS FROM [adventure works]

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Accessories** | **Bikes** | **Clothing** | **Components** |
| standard deviation | 6848.577389 | 6848.577389 | 6848.577389 | 6848.577389 |
| Order Count | 19,523 | 18,358 | 9,871 | 2,646 |

STDDEVP does the same thing as STDEVP.

<a name='STDEVP\_END'>]] <a name='StrToValue'>]]

StrToValue

StrToValue function returns a value from an MDX

formatted string. This function can be used to translate a member property into a numeric data type; even though the member property might be a number (for instance, employee salary or store square feet) with MSAS 2000 member properties are exposed to MDX as a string. You can apply StrToValue to convert a member property string back to a number. The syntax is StrToValue(string expression).

For example, the following query returns the weight of products:

WITH member [measures].[weight] AS 'StrToValue([product].[product].CurrentMember.Properties("weight")) ' SELECT {[measures].[weight]} ON COLUMNS, {[product].[product].members} ON ROWS FROM [adventure works]

Results (abbreviated):

|  |  |
| --- | --- |
|  | **weight** |
| Mountain-100 Black, 38 | 20.35 |
| Mountain-100 Black, 42 | 20.77 |
| Mountain-100 Black, 44 | 21.13 |
| Mountain-100 Black, 48 | 21.42 |
| Mountain-200 Silver, 38 | 23.35 |
| Mountain-200 Silver, 38 | 23.35 |
| Mountain-200 Silver, 42 | 23.77 |
| Mountain-200 Silver, 42 | 23.77 |
| Mountain-200 Silver, 46 | 24.13 |

<a name='StrToValue\_END'>]] <a name='SUM'>]]

SUM

SUM function returns the sum of a measure evaluated over a set. The syntax is SUM(SET, numeric expression), where numeric expression is a measure. If the numeric expression isn't specified MDX will use the default measure or the measure from the current context of the query. For example, the following query returns the average of the default measure for bike rack sales:

WITH member [product].[product categories].[sum] AS 'SUM( {[product].[product categories].[subcategory].[bike racks]} )' SELECT {[product].[product categories].[sum]} ON COLUMNS FROM [adventure works]

The next query returns the average of order counts because that is the measure referenced in the WHERE clause:

WITH member [product].[product categories].[sum] AS 'SUM( {[product].[product categories].[subcategory].[bike racks]} )' SELECT {[product].[product categories].[sum]} ON COLUMNS FROM [adventure works] WHERE ([measures].[order count])

The next query returns the average of sales amount - the measure specified as the numeric expression within the SUM function:

WITH member [product].[product categories].[sum] AS 'SUM( {[product].[product categories].[subcategory].[bike racks]}, [measures].[sales amount] )' SELECT {[product].[product categories].[sum]} ON COLUMNS FROM [adventure works]

<a name='SUM\_END'>]] <a name='VAR'>]]

VAR

VAR function returns a sample variance of a measure evaluated over a set. This function uses the unbiased population formula. For example, the following query returns the variance of order counts for the subcategory level of the product categories' hierarchy:

WITH member [product].[product categories].[variance] AS 'VAR( {[product].[product categories].[subcategory].members}, [measures].[order count] )' SELECT {[product].[product categories].[variance]} ON COLUMNS FROM [adventure works]

<a name='VAR\_END'>]] <a name='VARP'>]]

VARP

VARP function returns the population variance of a measure evaluated over a set. This function uses the biased population formula. For example, the following query returns the population variance of order counts for the subcategory level of the product categories' hierarchy:

WITH member [product].[product categories].[variance] AS 'VARP( {[product].[product categories].[subcategory].members}, [measures].[order count] )' SELECT {[product].[product categories].[variance]} ON COLUMNS FROM [adventure works]

<a name='VARP\_END'>]]