# Table of Contents

## About This Book ............................................................................................................................................................................ 1

## Basic Rules ...................................................................................................................................................................................... 3

- Identifiers .......................................................................................................................................................................................... 4
- Comments ........................................................................................................................................................................................... 5

## Expression Types .............................................................................................................................................................................. 7

- Numeric Expressions .......................................................................................................................................................................... 8
- String Expressions .............................................................................................................................................................................. 9
- Logical Expressions ............................................................................................................................................................................... 10
- Tuple Expressions ............................................................................................................................................................................... 11
- Member Expressions .......................................................................................................................................................................... 12
- Level Expressions .......................................................................................................................................................... 14
- Hierarchy Expressions ....................................................................................................................................................................... 15
- Set Expressions .................................................................................................................................................................................. 16
- Measure Search Expressions ................................................................................................................................................................ 18
- Quality Measure Expressions ........................................................................................................................................................ 20

## MDX Statements and Clauses ......................................................................................................................................................... 21

- `%FILTER` Clause ........................................................................................................................................................................... 22
- `CREATE MEMBER` Statement .................................................................................................................................................. 23
- `CREATE SET` Statement ........................................................................................................................................................... 25
- `DRILLFACTS` Statement .............................................................................................................................................................. 26
- `DRILLTHROUGH` Statement ..................................................................................................................................................... 28
- `DROP MEMBER` Statement ....................................................................................................................................................... 30
- `DROP SET` Statement ................................................................................................................................................................. 31
- `FORMAT_STRING` Clause .......................................................................................................................................................... 32
- `SELECT` Statement ...................................................................................................................................................................... 34
- `SOLVE_ORDER` Clause ............................................................................................................................................................ 36
- `WHERE` Clause ............................................................................................................................................................................. 38
- `WITH` Clause ................................................................................................................................................................................ 41

## MDX Functions ................................................................................................................................................................................. 45

- `%ALL` .......................................................................................................................................................................................... 46
- `%CELL` .......................................................................................................................................................................................... 48
- `%FIRST` ........................................................................................................................................................................................ 49
- `%KPI` ............................................................................................................................................................................................ 50
- `%LABEL` ........................................................................................................................................................................................ 52
- `%LAST` ........................................................................................................................................................................................... 53
- `%LIST` ........................................................................................................................................................................................... 54
- `%LOOKUP` ...................................................................................................................................................................................... 55
- `%MDX` ........................................................................................................................................................................................... 56
- `%NOT` ............................................................................................................................................................................................ 58
- `%OR` .............................................................................................................................................................................................. 59
- `%SEARCH` .................................................................................................................................................................................... 60
- `%SPACE` ...................................................................................................................................................................................... 61
- `%TERMLIST` .................................................................................................................................................................................. 62
- `%TIMERANGE` ............................................................................................................................................................................... 64
- `%TIMEWINDOW` ............................................................................................................................................................................ 65
%TOPMEMBERS ................................................................. 67
AGGREGATE ........................................................................... 69
ALLMEMBERS ........................................................................ 71
ANCESTOR ........................................................................... 73
AVG ...................................................................................... 74
BOTTOMCOUNT ..................................................................... 76
BOTTOMPERCENT ................................................................. 78
BOTTOMSUM ......................................................................... 80
CHILDREN ........................................................................... 81
CLOSINGPERIOD .................................................................. 82
COUNT ................................................................................ 83
COUSIN ............................................................................... 85
CROSSJOIN ......................................................................... 86
CURRENTMEMBER .............................................................. 88
DISTINCT ............................................................................. 90
EXCEPT ................................................................................ 91
FILTER ............................................................................... 92
FIRSTCHILD ......................................................................... 94
FIRSTSIBLING ..................................................................... 96
HEAD .................................................................................. 97
HIERARCHIZE ...................................................................... 99
HIERARCHICHIZE ................................................................. 99
IIF ...................................................................................... 101
INTERSECT .......................................................................... 103
ISNULL ............................................................................... 104
LAG ..................................................................................... 105
LASTCHILD .......................................................................... 107
LASTSIBLING ..................................................................... 108
LEAD ................................................................................... 109
LOG .................................................................................... 111
LOOKUP ............................................................................... 112
MAX ................................................................................... 114
MEDIAN .............................................................................. 116
MEMBERS .......................................................................... 118
MIN .................................................................................... 120
NEXTMEMBER ..................................................................... 122
NONEMPTYCROSSJOIN ......................................................... 124
OPENINGPERIOD ................................................................. 125
ORDER ............................................................................... 126
PARALLELPERIOD ............................................................... 128
PARENT .............................................................................. 130
PERCENTILE ....................................................................... 131
PERCENTILERANK ............................................................... 133
PERIODSTODATE ................................................................. 134
POWER ............................................................................... 136
PREVMEMBER ..................................................................... 137
PROPERTIES ......................................................................... 139
RANK ................................................................................ 141
ROUND .............................................................................. 142
SIBLINGS ........................................................................... 143
SQRT ................................................................................. 144
About This Book

This book provides information on the MDX expressions, statements, and functions supported in InterSystems DeepSee. It contains the following sections:

- Basic Rules
- Expression Types
- MDX Statements and Clauses
- MDX Functions
- Intrinsic Properties
- NOW Member for Date/Time Levels
- Quick Function Reference

For a detailed outline, see the table of contents.

The other developer books for DeepSee are as follows:

- Getting Started with DeepSee briefly introduces DeepSee and the tools that it provides.
- DeepSee Developer Tutorial guides developers through the process of creating a sample that consists of a cube, subject areas, pivot tables, and dashboards.
- DeepSee Implementation Guide describes how to implement DeepSee, apart from creating the model.
- Defining DeepSee Models describes how to define all the elements used in DeepSee queries: DeepSee cubes, subject areas, worksheets, quality measures, KPIs, and plugins.
- Using MDX with DeepSee introduces MDX and describes how to write MDX queries manually for use with DeepSee cubes.

The following books are for both developers and users:

- Creating DeepSee Dashboards describes how to create and modify dashboards in DeepSee.
- Using the DeepSee Analyzer describes how to create and modify pivot tables, as well as use the Analyzer in general.

For general information, see the InterSystems Documentation Guide.

Note: DeepSee provides an implementation of MDX. Results may differ from other implementations.
Basic Rules

This reference section provides information on the most basic rules in MDX — rules for identifiers and comments.
Identifiers

This section discusses identifiers in DeepSee MDX.

MDX Identifiers and Allowed Variations

Every cube, subject area, relationship, dimension, hierarchy, level, member, property, and measure has an identifier.

Each MDX identifier has one of the following forms:

• A regular identifier:

  \( \text{name} \)

  You can use this form if \text{name} follows these rules:
  
  – The first character must be either an alphabetic character (Latin-1) or the underscore character (\_).
    
    In DeepSee MDX, the first character can be numeric if the rest of the characters are also numeric. This DeepSee extension to MDX means that you can conveniently refer to members that have all-numeric names (such as postal codes).
  
  – The other characters must be either alphabetic characters, the underscore character, or numeric characters.
  
  – The name must not be an MDX reserved keyword. Reserved keywords are not case-sensitive in MDX.

• A delimited identifier:

  \([\text{name}]\)

  You must use this form if \text{name} does not follow the preceding rules. If \text{name} includes a right square bracket (\]), you must escape that by using two right square brackets together (\[][])

• (Only for members) A member key:

  \&[\text{key value}]

  For information on key values, see “Key Values.”

Note: \text{name} is not case-sensitive, but \text{key value} is.

Important: Note that \text{name} and \text{key value} are not expressions. That is, you cannot replace them with string-valued expressions.
Comments

You can include a comment in any MDX query or expression, at any place where white space is acceptable.

```/* comment here */```

You can also use the preceding two forms at the end of a line that contains MDX.

Examples

The following examples show valid MDX queries:

```select /* change this later */ birthd.decade.members on 1
from patients
--comment
select birthd.decade.members on 1 --comment
from patients // comment
//another comment```
Expression Types

This reference section provides information on the MDX expression types supported in InterSystems DeepSee.
Numeric Expressions

This section describes how to create and use numeric expressions in DeepSee MDX.

Details

In DeepSee MDX, a numeric expression can have any of the following forms:

- A numeric literal. For example: 37
  The literal cannot start with a decimal point; that is, you must include a leading 0 with any fractional values. For example, 0.1 is valid but .1 is not valid.

- A percentage literal. For example: 10%
  There must be no space between the number and the percent sign.

- An expression that refers to a numeric-valued measure, such as MEASURES.[%COUNT]

- An expression that uses an MDX function that returns a numeric value, for example:
  ```mdx
  AVG(aged.age, MEASURES.[test score])
  ```
  Many MDX functions return numeric values, including AVG, MAX, COUNT, and others. Also, the IIF function can return numeric values; this function evaluates a condition and returns one of two values, depending on the condition.

- An expression that uses mathematical operators to combine numeric expressions. For example:
  ```mdx
  MEASURES.[%COUNT] / 100
  ```
  DeepSee supports the standard mathematical operators: + (addition), − (subtraction), / (division), and * (multiplication). It also supports the standard unary operators: + (positive) and − (negative).

  You can use parentheses to control precedence.
  In the expression, if any value is null, the expression evaluates to null.

  If you divide a value by 0, DeepSee treats the result as null.

  **Tip:** The MDX function IIF can be useful in such expressions.

Uses

You can use numeric expressions in the following ways:

- As a numeric argument to many MDX functions. For example:
  ```mdx
  AVG(diagd.MEMBERS, MEASURES.[%COUNT])
  ```

- As an element of a set.

- As the definition of a calculated member (in this case, a measure).
String Expressions

This section describes how to create and use string expressions in DeepSee MDX.

Details

In DeepSee MDX, a string expression can have any of the following forms:

- A string literal (a double quote character, followed by any characters, followed by another double quote character). For example: "label" or "my property"
- A numeric literal.
- An expression that uses the PROPERTIES function to return the value of a property of a member. For example:
  homed.city.magnolia.PROPERTIES("Population")
  See the reference section “Intrinsic Properties” for intrinsic properties that you can use.
- An expression that uses the IIF function to return a string value.
- An expression that refers to a string-valued measure. For example, suppose that you define a calculated member as follows:
  CREATE MEMBER patients.measures.stringtest AS 'IIF(MEASURES.[avg test score]<60, "low","high")'
  Then the expression MEASURES.stringtest is a string expression.
- An expression that uses the MDX concatenation operator (+) to combine other string expressions. For example:
  "string 1 " + "string 2"
  For another example:
  "Pop: " + homed.city.magnolia.PROPERTIES("Population")

Uses

You can use string expressions in the following ways:

- As an argument to the ORDER function or to the PROPERTIES function.
- As an element of a set.
- As the definition of a calculated member (in this case, a measure).
Logical Expressions

This section describes how to create and use logical expressions in DeepSee MDX.

Details and Uses

MDX does not include a logical data type. However, it provides the FILTER function and the IIF function, both of which take an argument that is treated as true or false. In these contexts, DeepSee MDX interprets numeric and string values as true or false, as follows:

- A numeric expression that evaluates to 0 is treated as false.
- All other numeric expressions are treated as true.
- All string expressions are treated as false.

In the same contexts, you can combine true and false values by using the following standard tools:

- Logical comparison operators: > (greater than), >= (greater than or equal to), = (equal to), < (less than), and <= (less than or equal to). For example, the following expression returns true or false, depending on the value of MEASURES.[%COUNT]:
  \[
  \text{MEASURES.[%COUNT]}>1100
  \]

  For another example, the following expression returns true or false, depending on the name of the current member:
  \[
  \text{colord.CURRENTMEMBER.PROPERTIES("Name")}="\text{Red}\"
  \]

  When a numeric expression is compared to null, the null value is treated as 0.

- The AND operator, the OR operator, and parentheses to control precedence. For example, the following expression returns true or false, depending on the value of MEASURES.[%COUNT]:
  \[
  \text{(MEASURES.[%COUNT]>1100) AND (MEASURES.[%COUNT]<1500)}
  \]

When you compare scalar values of different types, DeepSee compares them as follows:

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both expressions are numeric</td>
<td>Perform a numeric comparison</td>
</tr>
<tr>
<td>Both expressions are strings</td>
<td>Perform a string comparison</td>
</tr>
<tr>
<td>One expression is numeric and the other is a string</td>
<td>The numeric expression is less than the string expression</td>
</tr>
</tbody>
</table>

You cannot use member expressions in a logical comparison expression. For example, the following is not a valid logical expression:

\[
\text{homed.CURRENTMEMBER = homed.hl.city.magnolia}
\]

Instead, you should use the Name property or some other suitable property, all of which are strings. For example, the following is a valid logical expression:

\[
\text{homed.CURRENTMEMBER.Properties("Name") = "magnolia"}
\]

See the reference section “Intrinsic Properties” for intrinsic properties that you can use.

Also see the section “Measure Search Expressions.”
Tuple Expressions

This section describes how to create and use tuple expressions in DeepSee MDX.

Details

In DeepSee MDX, a tuple literal is a comma-separated list of member expressions, enclosed by parentheses:

\[(member\_expr1, member\_expr2, member\_expr3, \ldots)\]

Where \(member\_expr1, member\_expr2, member\_expr3\), and so on are member expressions. The parentheses are required.

In other implementations of MDX, each of these member expressions must be associated with a different dimension. This means that a tuple cannot include more than one member from the same dimension.

In DeepSee MDX, a tuple expression can include more than one member expression from the same dimension. In most cases, the result is null. However, in DeepSee, a level can be based on a list value, which means that a given record can belong to multiple members. For example, the tuple \(\text{allerd.soy}, \text{allerd.wheat}\) represents all patients who are allergic to both soy and wheat, and this tuple could potentially have a non-null value.

If this list includes a member expression for each dimension in the cube, the tuple is fully qualified. Otherwise, it is partially qualified. The following shows an example of a partially qualified tuple from the Patients cube:

\(\text{allerd.[dairy products]}, \text{colord.red}, \text{aged.35}\)

Another example is as follows:

\(\text{diagd.asthma}, \text{aged.[age group].[30 to 59]}, \text{MEASURES.[%COUNT]}\)

Each data cell in the result set returned by a query is a tuple.

The CROSSJOIN function returns a set of tuples, as does the NONEMPTYCROSSJOIN function.

Uses

You can use tuple expressions in the following ways:

- As an element of a set.
- As the argument to the WHERE clause.
Member Expressions

This section describes how to create and use member expressions in DeepSee MDX.

Details

In DeepSee MDX, a member expression can have any of the following forms:

- An member literal, which is an explicit reference to a single member by its name:

  \[\text{[dimension	extunderscore name]}.\text{[hierarchy	extunderscore name]}.\text{[level	extunderscore name]}.\text{[member	extunderscore name]}\]

  Where:
  - \(\text{[dimension	extunderscore name]}\) is an MDX identifier that names a dimension.
  - \(\text{[hierarchy	extunderscore name]}\) is an MDX identifier that names a hierarchy within that dimension. You can omit the hierarchy name. If you do, the query uses the first level with the given name, as defined in this dimension.
  - \(\text{[level	extunderscore name]}\) is an MDX identifier that names a level within that hierarchy. You can omit the level name. If you do, the query uses the first member with the given name, as defined within this dimension.

  For example:

  \([\text{gend}].\text{[h1]}.\text{[gender]}.\text{female}\)

- For a measure, a member literal has the following variant form, because any measure is a member of a dimension called Measures:

  \([\text{MEASURES}].\text{[measure	extunderscore name]}\)

  For example:

  \([\text{MEASURES}].\text{[\$COUNT]}\)

  Note that although a measure is a member, an expression like the preceding returns a scalar value and cannot be used with all member functions.

- An explicit reference to a hierarchy as follows:

  \([\text{dimension	extunderscore name}].\text{[hierarchy	extunderscore name]}\)

  This expression returns the All member of the dimension to which the hierarchy belongs.

  For example:

  \(\text{aged.h1}\)

  The preceding expression returns the All member of the AgeD dimension; in the sample Patients cube, the name of this member is All Patients.

- An expression that uses an MDX function to return a single member. For example:

  \([\text{gend}].\text{[h1]}.\text{[gender]}.\text{female}.\text{NEXTMEMBER}\)

  Many MDX functions return members, including LAG, NEXTMEMBER, PARENT, and others.
• An expression that uses the internal key of a member to return that member, via the following syntax:

\[[\text{dimension\_name}].[\text{hierarchy\_name}].[\text{level\_name}].&[\text{member\_key}]\]

Where \text{member\_key} is the value of the \text{KEY} property of the member. For example:

\text{birthqd.h1.quarter.}&[2]

Note that \text{member\_key} is a case-sensitive literal value rather than an expression. That is, you cannot replace it with a string-valued expression.

For information on how \text{KEY} values are created, see the reference “\text{Intrinsic Properties}.” Also note that you can use the \text{PROPERTIES} function to find the value of the \text{KEY} property or any other property of a member.

• An expression that uses a DeepSee MDX extension to refer to a member in another cube, via the following syntax:

\[[\text{relationship\_name}].\text{member\_expression}\]

Where \text{relationship\_name} is an MDX identifier that names a relationship in the cube used by the query and \text{member\_expression} is a member expression suitable for that cube.

For example, the following member expressions are all equivalent:

\text{[gend].[h1].[gender].Female} \\
\text{[gend].female} \\
\text{gend.H1.gender.female} \\
\text{gend.h1.FEMALE} \\
\text{gend.female}

**Calculated Members**

Most members are defined within the cube definition, as part of the definition of a level. You can also create \textit{calculated members}, which are typically based on other members. You can define calculated members in two ways:

• Within the \text{WITH} clause of a query. The member is available within the rest of the query, but is not available in other queries.

• Within the \text{CREATE MEMBER} statement. The member is available within the rest of the session (for example, within the rest of the session in the MDX shell).

**Uses**

You can use member expressions in the following ways:

• As a member argument to many MDX functions.

• As the argument to the \text{WHERE} clause.

• As an element of a \textit{set}. 
Level Expressions

This section describes how to create and use level expressions in DeepSee MDX.

Details

In DeepSee MDX, a level expression has one of the following forms:

- A level literal, which is a direct reference to the level as follows:
  
  \[[\text{dimension}\_name].[\text{hierarchy}\_name].[\text{level}\_name]\]

  Where:
  
  - \text{\textit{dimension}\_name} is an MDX identifier that names a dimension.
  - \text{\textit{hierarchy}\_name} is an MDX identifier that names a hierarchy within that dimension. You can omit the hierarchy name. If you do, the query uses the first level with the given name, as defined in this dimension.
  - \text{\textit{level}\_name} is an MDX identifier that names a level within that hierarchy.

  For example:
  
  \[[\text{gend}].[\text{h1}].[\text{gender}]\]

- An expression that uses a DeepSee MDX extension to refer to a level in another cube, via the following syntax:
  
  \[[\text{relationship}\_name}.\text{level}\_expression\]

  Where \text{\textit{relationship}\_name} is the name of a relationship in the cube used by the query and \text{\textit{level}\_expression} is a level expression suitable for that cube.

Uses

You can use level expressions in the following way:

- As an argument to the ALLMEMBERS function.
- As an argument to the MEMBERS function.
Hierarchy Expressions

This section describes how to create and use hierarchy expressions in DeepSee MDX.

Details

In DeepSee MDX, a hierarchy expression has one of the following forms:

- A hierarchy literal, which is a direct reference to the hierarchy as follows:

  \[\text{[dimension\_name].[hierarchy\_name]}\]

  Where:

  - \[\text{[dimension\_name]}\] is an MDX identifier that names a dimension.
  - \[\text{[hierarchy\_name]}\] is an MDX identifier that names a hierarchy within that dimension.

  For example:

  \[\text{[gend].[h1]}\]

- A reference to a dimension:

  \[\text{[dimension\_name]}\]

  For example:

  \[\text{[gend]}\]

  DeepSee interprets this as a reference to the first visible hierarchy of that dimension.

- An expression that uses a DeepSee MDX extension to refer to a hierarchy in another cube, via the following syntax:

  \[\text{[relationship\_name].hierarchy\_expression}\]

  Where \text{relationship\_name} is the name of a relationship in the cube used by the query and \text{hierarchy\_expression} is a hierarchy expression suitable for that cube.

Note: You can use a bare hierarchy expression like this only the dimension defines an All member.

Uses

You can use a hierarchy expression as an argument to any of the following functions:

- \text{ALMEMBERS}
- \text{CURRENTMEMBER}
- \text{MEMBERS}
**Set Expressions**

This section describes how to create and use set expressions in DeepSee MDX.

**Details**

The general syntax for a *set expression* is as follows:

```
{expression1, expression2, ...}
```

This list can include any number of items. In DeepSee MDX, if the list includes only one item, you can omit the curly braces.

In this list, `expression1`, `expression2`, and so on can have any of the following forms:

- A member expression.
- A numeric expression or string expression.
- A range of members, specified as follows:

```
member1:member2
```

This expression returns a set that consists of the two given members and all members between them, given the order of members in the level that contains them. The members must belong to the same level.

For example:

```
birthd.year.1960:birthd.year.1980
```

For `member2`, you can omit the dimension, hierarchy, and level identifiers. For example:

```
birthd.year.1960:1980
```

- An expression that uses an MDX function that returns a set, for example:

```
homed.zip.MEMBERS
```

Many MDX functions return sets, including `MEMBERS`, `NONEMPTYCROSSJOIN`, `ORDER`, and others.

- Another set expression.
- The name of a named set. See the following section.
- A tuple expression.

(Note that in other implementations of MDX, for any tuple in a set, you must use the dimensions in the same order as in the other tuples in the set. For example, if the first tuple uses dimension A in its first list item, all the other tuples must do so as well. DeepSee MDX does not have this restriction. Similarly, in other implementations of MDX, a set cannot include a combination of tuples and other types of set elements. DeepSee MDX does not have this restriction either.)

You can precede any set expression with the keyword phrase `NON EMPTY`, for example:

```
NON EMPTY {birthd.year.1960:1980}
NON EMPTY birthd.year.1960:1980
NON EMPTY {homed.zip.MEMBERS}
NON EMPTY homed.zip.MEMBERS
```

The `NON EMPTY` keyword phrase suppresses empty elements of the set; the set is evaluated and then empty elements are removed. This keyword is particularly useful with `CROSSJOIN` and in scenarios where a filter can potentially cause elements to be null.
**Named Sets**

A *named set* consists of two elements: a set name and a set expression. You can define named sets in two ways:

- Within the **WITH** clause of a query. The set name is available within the rest of the query, but is not available in other queries.
- Within the **CREATE SET** statement. The set name is available within the rest of the session (for example, within the rest of the session in the MDX shell).

**Uses**

You can use set expressions in the following ways:

- As a set argument of many MDX functions. Note that for some functions, the set must consist only of members. For other functions, the set must consist only of members or tuples. This book notes these requirements where needed.
- As an axis in the **SELECT** statement.
- As the definition of a named set, as described in the previous subsection.
Measure Search Expressions

This section describes how to create and use measure search expressions, which enable you to access rows from the fact table based on the value of a measure for the facts themselves (that is, at the lowest level rather than at an aggregate level). These expressions are a DeepSee extension to MDX.

Details

A measure search expression has the following syntax, which refers to a special dimension in DeepSee called %SEARCH:

\%SEARCH.&{\text{comparison expression}}

Where comparison expression is a logical expression like the following example:

\{MEASURES\}.\{test score\}>60

Note: In this case, the square brackets are required.

For example, the following query selects all patients with a test score higher than 60:

\text{SELECT FROM patients WHERE %SEARCH.&\{MEASURES\}.\{Test Score\}>60}\
\text{Result: 6,191}

More generally, comparison expression can be a combination of logical expressions. To combine logical expressions, you can use the following standard tools:

• Logical comparison operators: \&gt; (greater than), \&gt;= (greater than or equal to), = (equal to), \&lt; (less than), and \&lt;= (less than or equal to).

• The AND operator, the OR operator, and parentheses to control precedence.

You can also use IS NULL. For example:

\text{SELECT FROM HOLEFOODS WHERE %Search.&\{Measures\}.\{Units Sold\} IS NULL}

Uses

You can use measure search expressions in all the following contexts:

• As the argument for the %FILTER clause
• As the argument for the WHERE clause
• As an argument for the FILTER function.

Additional Notes

DeepSee parses a measure search expression as follows:

1. %Search is treated as a dimension.
2. Because the comparison expression is enclosed inside &\{\}, DeepSee treats it as a KEY value, which permits it to contain arbitrary syntax.
3. The comparison expression is converted to an SQL statement against the fact table.

The preceding means that comparison expression can include SQL syntax.

Also, it may be possible to use a measure in a measure search expression even if it is not marked as searchable="true" in the cube definition. This attribute value causes DeepSee to do two things:
• Display this measure as an option in advanced filters.
• Add additional index, if needed, to enable the measure to be searchable.
Quality Measure Expressions

This section describes how to create and use quality measure expressions, which provide access to the values of quality measures. These expressions are a DeepSee extension to MDX.

Details

A quality measure expression has the following syntax, which refers to a special dimension in DeepSee called %QualityMeasure:

```
[%QualityMeasure].&[catalog/set/qm name]
```

Where catalog is the catalog to which the quality measure belongs, set is a set in that catalog, and qm name is the short name of the quality measure. (The full name of the quality measure is catalog/set/qm name.)

Uses

You can use quality measure expressions in the same way that you use other measure expressions.
MDX Statements and Clauses

This reference section provides information on the MDX statements and clauses supported in InterSystems DeepSee.
%FILTER Clause

Applies a filter to a SELECT statement; describes how to slice the results of a SELECT statement. This clause is similar to WHERE except that you can include multiple %FILTER clauses in a statement. %FILTER is a DeepSee extension to MDX.

Syntax and Details

\[ \text{select\_statement} \ %\text{FILTER} \ \text{set\_expression} \]

Where:

- \text{select\_statement} is a statement that uses SELECT.
- \text{set\_expression} is an expression that returns a set of members or tuples.

Instead of \text{set\_expression}, you can use a measure search expression; see the example to see the behavior of %FILTER in this case.

Because DeepSee MDX automatically converts types where appropriate, you can also use a single member expression or tuple expression in place of the set expression.

You can include as many %FILTER clauses as needed. This clause is particularly useful when you run queries programmatically, because it enables you to filter the results further by simply appending to the SELECT statement. (In contrast, if you use the WHERE clause and you need to add another filter item, it is necessary to rewrite the WHERE clause, because only one WHERE clause is permitted.)

Important: Each set element is used as a separate slicer axis, and the results of all the slicer axes (of all %FILTER clauses) are aggregated together. Each measure is aggregated according to its aggregation method, as specified in the cube definition. (In the examples here, %COUNT is added.)

Example

If you use the %FILTER clause with a measure search expression, the clause uses the rows of the fact table that do meet the given criteria. (A measure search expression is a DeepSee extension to MDX that considers the measure values in the fact table itself.)

\[
\text{SELECT MEASURES.}[%\text{COUNT}] \ ON \ 0 \ FROM \ \text{patients} \ %\text{FILTER} \ %\text{SEARCH.}[[\text{MEASURES}].[\text{age}]<10]
\]

Patient Count
1,370

See Also

See the WHERE clause.
CREATE MEMBER Statement

Creates a calculated member that can be used within the current session.

Syntax and Details

```sql
CREATE SESSION MEMBER calc_mem_details, FORMAT_STRING='format_details',
SOLVE_ORDER=integer
```

Where `calc_mem_details` is as follows:

```
cube_name.[dimension_name].[new_member_name] AS 'value_expression'
```

And:

- `cube_name` is the name of the cube to which you are adding this member.
- `dimension_name` is the name of the dimension to which you are adding this member.
- `new_member_name` is the name of a member; the member may or may not be already defined in the cube. If it is, the definition given here takes precedence.
- `value_expression` is an MDX expression that defines the calculated member, typically in terms of references to other members. For details, see WITH Clause.
- `FORMAT_STRING='format_details'` is an optional clause that specifies how to display the values. This clause is applicable only for numeric values. See FORMAT_STRING Clause.
- `SOLVE_ORDER=integer` is an optional clause that specifies the relative order in which to evaluate this calculated member. This clause is relevant only if the query contains calculated members on more than one axis. See SOLVE_ORDER Clause.

Also see "Identifiers."

When you use the MDX shell, a session is started; the session ends when you exit the shell. During this session, if you use the CREATE MEMBER statement, the member that you create is available until the session ends or until you use the DROP MEMBER statement.

Example

First, in the MDX shell, we define a new member in the Patients cube:

```sql
>>CREATE SESSION MEMBER patients.MEASURES.scoresquared AS 'MEASURES.[test score]*MEASURES.[test score]'
```

```
Elapsed time:       .013701s
```

Then we use the new measure in a query:

```sql
>>SELECT MEASURES.scoresquared ON 0, aged.[age group].MEMBERS ON 1 FROM patients
```

```
1 0 to 29                    66,801,054,681
2 30 to 59                   61,070,271,376
3 60+                         9,120,632,004
```

```
Elapsed time:       .016856s
```

See Also

- WITH Clause
- FORMAT_STRING Clause
• SOLVE_ORDER Clause
• DROP MEMBER Statement
CREATE SET Statement

Creates a named set that can be used within the current session.

Syntax and Details

```
CREATE SESSION SET cube_name.set_name AS 'set_expression'
```

• `cube_name` is the name of the cube to which you are adding this set.

• `set_name` is an unquoted string that names the set. Later you can use this set name in the place of a set expression in any MDX query within the same session.

• `set_expression` is an expression that refers to a set.

When you use the MDX shell, a session is started; the session ends when you exit the shell. During this session, if you use the CREATE SET statement, the member that you create is available until the session ends or until you use the DROP SET statement.

Example

First, in the MDX shell, we define a new named set in the Patients cube:

```
>>CREATE SESSION SET patients.testset AS 'birthd.decade.MEMBERS'
```

```
Elapsed time: .014451s
```

Then we use the named set in a query:

```
>>SELECT MEASURES.[%COUNT] ON 0, testset ON 1 FROM patients
```

```
Patient Count
1 1910s   71
2 1920s   223
3 1930s   572
4 1940s   683
5 1950s   1,030
6 1960s   1,500
7 1970s   1,520
8 1980s   1,400
9 1990s   1,413
10 2000s  1,433
11 2010s  155
```

```
Elapsed time: .018745s
```

See Also

• WITH Clause

• DROP SET Statement
**DRILLFACTS Statement**

Displays the lowest-level data associated with the first cell of results of a given SELECT statement, using the fact and dimension tables defined by the cube.

**Syntax and Details**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DRILLFACTS select_statement</strong></td>
<td></td>
</tr>
<tr>
<td>Or:</td>
<td><strong>DRILLFACTS select_statement RETURN fieldname1, fieldname2, ...</strong></td>
</tr>
<tr>
<td>Or:</td>
<td><strong>DRILLFACTS select_statement RETURN fieldname1, ... %ORDER BY fieldname3, ...</strong></td>
</tr>
</tbody>
</table>

Where:

- `select_statement` is a statement that uses SELECT.
- `fieldname1, fieldname2, fieldname3, fieldname4`, and so on are names of fields in the fact table class defined by the cube.

If you do not specify the RETURN clause, the query returns the IDs of the records.

The `%ORDER BY` clause is an InterSystems extension to MDX. This clause specifies how to sort the displayed records.

For additional details on RETURN and `%ORDER BY`, see **DRILLTHROUGH Statement**.

Internally, DeepSee builds and uses an SQL query.

**Important:** If the SELECT statement returns more than one cell of data, the listing shows only the fields associated with the first cell.

**Example**

The first example does not use RETURN, so it uses the default listing as defined in the cube:

```
DRILLFACTS SELECT diagd.osteoporosis ON 0 FROM patients
```

```
#  ID
1: 7
2: 13
3: 42
4: 123
5: 140
...
```

The next example uses the RETURN clause:

```
drillfacts select diagd.osteoporosis on 0 from patients return dxcolor->dxcolor, dxage, dxpatgroup->dxpatgroup
```

```
#  DxColor  DxAge  DxPatGroup
1: Orange   7      Group A
2: Red      11     Group B
3: <null>   30     Group A
4: Purple   58     Group A
5: Purple   62     None
...
```
See Also

- DRILLTHROUGH Statement
DRILLTHROUGH Statement

Displays the lowest-level data associated with the first cell of results of a given SELECT statement.

Syntax and Details

- **DRILLTHROUGH** select_statement

Or:

- **DRILLTHROUGH** select_statement RETURN fieldname1, fieldname2, ...

Or:

- **DRILLTHROUGH** select_statement RETURN fieldname1, ... %ORDER BY fieldname3, ...

Where:

- select_statement is a statement that uses SELECT.
- fieldname1, fieldname2, fieldname3, fieldname4, and so on are names of fields in the base class used by the cube.

If you do not specify the RETURN clause, the query returns the default listing defined in the DeepSee cube.

The %ORDER BY clause is an InterSystems extension to MDX. This clause specifies how to sort the displayed records. Internally, DeepSee builds and uses an SQL query.

**Important:** If the SELECT statement returns more than one cell of data, the listing shows only the fields associated with the first cell.

Additional Options for RETURN and ORDER BY

In the RETURN and %ORDER BY clauses, note the following points:

- You can use Caché arrow syntax to refer to a property in another table. See “Special Features” in *Using Caché SQL*.
- You can include aliases.
- You can use standard SQL and Caché functions. To use a standard SQL function, enclose it within parentheses so that the function name is not interpreted as a field name; this is not necessary for Caché SQL functions, which start with the percent character (%).
- You can use more advanced SQL features if you use source.field_name rather than field_name.

For example:

```sql
... RETURN %ID,%EXTERNAL(Field1) F1,'$'||source.Sales Sales
```

The first line of any listing is a heading that indicates the field names or their aliases. Below the heading, the listing has a column of data below each heading. In this case, the columns would be as follows:

- %ID — this column displays the %ID field
- F1 — this column uses the Caché SQL %EXTERNAL function to return the value of the Field1 field in DISPLAY format
- Sales — this column displays the Sales field, preceded by a dollar sign ($)
Example

The first example does not use RETURN, so it uses the default listing as defined in the cube:

```
DRILLTHROUGH SELECT homed.Magnolia ON 1 FROM patients
#  PatientID    Age    Gender    TestScore    HomeCity    DoctorGroup
1: SUBJ_10161 0    F    76        3          I
2: SUBJ_10330 0    F    30        3          II
3: SUBJ_10554 0    F    68        3          II
4: SUBJ_10555 0    F    78        3          II
5: SUBJ_10686 0    F    91        3          I
...
```

The next example uses the RETURN clause:

```
DRILLTHROUGH SELECT homed.Magnolia ON 1 FROM patients RETURN Gender, HomeCity->PostalCode
#  Gender    PostalCode
1: F        34577
2: F        34577
3: F        34577
4: F        34577
5: F        34577
...
```

The next example also uses the %ORDER BY clause:

```
DRILLTHROUGH SELECT homed.Magnolia ON 1 FROM patients RETURN PatientID, Age, Gender %ORDER BY Age
#  PatientID    Age    Gender
1: SUBJ_10161 0    F
2: SUBJ_102705 0    M
3: SUBJ_103210 0    M
4: SUBJ_103300 0    F
5: SUBJ_103972 0    M
...
```

The last example uses the %ORDER BY clause with two field names to specify the order:

```
DRILLTHROUGH SELECT homed.Magnolia ON 1 FROM patients RETURN PatientID, Age, Gender %ORDER BY Gender, Age
#  PatientID    Age    Gender
1: SUBJ_10161 0    F
2: SUBJ_103300 0    F
3: SUBJ_105548 0    F
4: SUBJ_105556 0    F
5: SUBJ_106865 0    F
...
```

In this case, the records are sorted first by gender. Within gender, they are sorted by age.
DROP MEMBER Statement

Removes a calculated member that defined earlier in the current session.

Syntax and Details

DROP MEMBER cube_name.calculated_member_expression

• `cube_name` is the name of the cube to which you are adding this member.
• `calculated_member_expression` is an expression that refers to a member. Typically, `calculated_member_expression` has the form `MEASURES.new_measure_name`.

When you use the MDX shell, a session is started; the session ends when you exit the shell. During this session, if you use the CREATE MEMBER statement, the member that you create is available until the session ends or until you use the DROP MEMBER statement.

Example

```plaintext
>>DROP MEMBER patients.MEASURES.avgscore
```

Elapsed time: .011952s

See Also

• CREATE MEMBER Statement
DROP SET Statement

Removes a named set that defined earlier in the current session.

Syntax and Details

<table>
<thead>
<tr>
<th>DROP SET cube_name.set_name</th>
</tr>
</thead>
</table>

• `cube_name` is the name of the cube to which you are adding this member.
• `set_name` is an unquoted string that names the set.

When you use the MDX shell, a session is started; the session ends when you exit the shell. During this session, if you use the CREATE SET statement, the set that you create is available until the session ends or until you use the DROP SET statement.

Example

```mdx
>>DROP SET patients.testset

Elapsed time: .011825s
```

See Also

• CREATE SET Statement
FORMAT_STRING Clause

Used with a definition of a calculated member, this clause specifies the display format for the data.

Syntax and Details

You can use this clause when you define a calculated member with the CREATE MEMBER Statement or with the WITH Clause.

FORMAT_STRING = 'positive_piece;negative_piece;zero_piece;missing_piece;'

Where:

- **positive_piece** controls how a positive value is displayed.
- **negative_piece** controls how a negative value is displayed.
- **zero_piece** controls how zero is displayed.
- **missing_piece** controls how a missing value is displayed; this is not currently used.

Each piece is a literal and consists of one or more characters that include one of the following base units:

<table>
<thead>
<tr>
<th>Base Unit</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Display the value without the thousands separator. Do not include any decimal places.</td>
<td>12345</td>
</tr>
<tr>
<td>#, #</td>
<td>Display the value with the thousands separator. Do not include any decimal places. This is the default display format for positive numbers.</td>
<td>12,345</td>
</tr>
<tr>
<td>#.##</td>
<td>Display the value without the thousands separator. Include two decimal places (or one decimal place for each pound sign after the period). Specify as many pound places after the period as you need.</td>
<td>12345.67</td>
</tr>
<tr>
<td>#, #.##</td>
<td>Display the value with the thousands separator. Include two decimal places (or one decimal place for each pound sign after the period). Specify as many pound places after the period as you need.</td>
<td>12,345.67</td>
</tr>
</tbody>
</table>

You can include additional characters before or after the base unit, as follows:

- If you include a percent sign (%), DeepSee displays the value as a percentage. That is, it multiplies the value by 100 and it displays the percent sign (%) in the position you specify.
- Any other characters are displayed as given, in the position you specify.

If a query includes multiple calculated members with different format strings, the SOLVE_ORDER clause controls which format string is used.

Examples

The following table shows some examples:
FORMAT_STRING Clause

<table>
<thead>
<tr>
<th>Example</th>
<th>Logical Value</th>
<th>Display Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT_STRING='#,##; (#,##);'</td>
<td>6608.9431</td>
<td>6,609</td>
</tr>
<tr>
<td>Note that this corresponds to the default way in which numbers are displayed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORMAT_STRING='#,##.###;'</td>
<td>6608.9431</td>
<td>6,608.943</td>
</tr>
<tr>
<td>FORMAT_STRING='##%;'</td>
<td>6</td>
<td>600%</td>
</tr>
<tr>
<td>FORMAT_STRING='$#,##; ($#,##);'</td>
<td>2195765</td>
<td>$2,195,765</td>
</tr>
<tr>
<td></td>
<td>-3407228</td>
<td>($3,407,228)</td>
</tr>
</tbody>
</table>

See Also

- CREATE MEMBER Statement
- SOLVE_ORDER Clause
- WITH Clause
### SELECT Statement

Executes a query and returns the results. This section describes the basic syntax.

#### Syntax and Details

```plaintext
SELECT set_expression ON 0, set_expression ON 1, ... FROM cube_name
```

Where:

- The ON clause is an optional axis specification. It has the following form:
  ```plaintext
  set_expression ON axis_name_or_number
  ```
  - `set_expression` is an expression that evaluates to a set.
  - `axis_name_or_number` is `COLUMNS` or 0 (these are equivalent), `ROWS` or 1 (these are equivalent), or a higher integer.

The axis specifications describe axes of the query. You can specify any number of axes, from zero to seven. You can specify the axes in any order.

In other implementations of MDX, if you specify axis `n`, you must also specify all lower-numbered axes. In DeepSee MDX, if you do not specify a lower-numbered axis, DeepSee automatically generates the axis for you; on this axis, the query uses the default member specified by your cube definition.

Note that if you use more than two axes, the MDX shell does not display the results in a usable form.

- `cube_name` is the name of the cube to use. The set expressions must make sense within that cube.

If you omit the axes specifications, MDX returns the count of records in the cube. If this is a compound cube, this is the sum of the counts of all cubes combined in that compound cube.

#### Example

The following simple example shows patient counts by ZIP code.

```plaintext
SELECT MEASURES.[%COUNT] ON 0, homed.zip.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,272</td>
<td>32006</td>
</tr>
<tr>
<td>1,111</td>
<td>32007</td>
</tr>
<tr>
<td>3,399</td>
<td>34577</td>
</tr>
<tr>
<td>1,069</td>
<td>36711</td>
</tr>
<tr>
<td>2,149</td>
<td>38928</td>
</tr>
</tbody>
</table>

In the following example, the `patients2` cube does not include the Home Zip level. Instead, this cube has a relationship called Home City that points to another cube, called cities. The query uses this relationship:

```plaintext
SELECT MEASURES.[%COUNT] ON 0, city.cityd.city.members ON 1 FROM patients2
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,097</td>
<td>Cedar Falls</td>
</tr>
<tr>
<td>1,136</td>
<td>Centerville</td>
</tr>
<tr>
<td>1,124</td>
<td>Cypress</td>
</tr>
<tr>
<td>1,089</td>
<td>Elm Heights</td>
</tr>
<tr>
<td>1,133</td>
<td>Juniper</td>
</tr>
<tr>
<td>1,063</td>
<td>Magnolia</td>
</tr>
<tr>
<td>1,124</td>
<td>Pine</td>
</tr>
<tr>
<td>1,083</td>
<td>Redwood</td>
</tr>
<tr>
<td>1,151</td>
<td>Spruce</td>
</tr>
</tbody>
</table>

#### See Also

- DRILLTHROUGH Statement
• WHERE Clause
• WITH Clause
SOLVE_ORDER Clause

Used with a definition of a calculated member, this clause specifies the relative order in which to evaluate this calculated member (relative to other calculated members). This clause lets you control the evaluation order when you use calculated members that have conflicting definitions (or conflicting format strings) on different axes.

Syntax and Details

You can use this clause when you define a calculated member with the CREATE MEMBER Statement or with the WITH Clause.

```plaintext
SOLVE_ORDER=integer
```

The SOLVE_ORDER keyword is not case-sensitive. For integer, specify a literal integer. The default is 0.

This clause specifies the relative order in which to evaluate this calculated member (relative to other calculated members). It also affects which value of FORMAT_STRING is used. A calculated member with a higher value for SOLVE_ORDER is evaluated after other calculated members.

If you use calculated members on more than one axis, and you do not specify SOLVE_ORDER, the system chooses the order in which to evaluate the members.

For calculated members that depend on other calculated members, the system recognizes the dependencies and evaluates the members in the appropriate order; you do not need to use SOLVE_ORDER for these members.

Example

For example, compare the following two queries and their results. In the first example, the calculated member MEASURES.net has SOLVE_ORDER=1. The other calculated member, dateofsale.diff, has a higher value for SOLVE_ORDER and is calculated later.

```plaintext
WITH
MEMBER MEASURES.net AS '(MEASURES.[amount sold] - MEASURES.target)/MEASURES.target',
SOLVE_ORDER=1,FORMAT_STRING='#.##%'
MEMBER dateofsale.diff AS 'dateofsale.2010 - dateofsale.2009',
SOLVE_ORDER=2,FORMAT_STRING='#,#.##'
SELECT
{ MEASURES.[amount sold], MEASURES.target, MEASURES.net } ON 0,
{ dateofsale.2009, dateofsale.2010, dateofsale.diff } ON 1
FROM holefoods
```

<table>
<thead>
<tr>
<th></th>
<th>Amount Sold</th>
<th>Target</th>
<th>net</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2009</td>
<td>2,151.98</td>
<td>2,096.48</td>
<td>2.65%</td>
</tr>
<tr>
<td>2 2010</td>
<td>2,521.69</td>
<td>2,497.92</td>
<td>0.95%</td>
</tr>
<tr>
<td>3 diff</td>
<td>369.71</td>
<td>401.44</td>
<td>-0.02</td>
</tr>
</tbody>
</table>

In the second example, the calculated member MEASURES.net has SOLVE_ORDER=2, which means that it is evaluated after the other calculated member.

```plaintext
WITH
MEMBER MEASURES.net AS '(MEASURES.[amount sold] - MEASURES.target)/MEASURES.target',
SOLVE_ORDER=2,FORMAT_STRING='#.##%'
MEMBER dateofsale.diff AS 'dateofsale.2010 - dateofsale.2009',
SOLVE_ORDER=1,FORMAT_STRING='#,#.##'
SELECT
{ MEASURES.[amount sold], MEASURES.target, MEASURES.net } ON 0,
{ dateofsale.2009, dateofsale.2010, dateofsale.diff } ON 1
FROM holefoods
```

<table>
<thead>
<tr>
<th></th>
<th>Amount Sold</th>
<th>Target</th>
<th>net</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2009</td>
<td>2,151.98</td>
<td>2,096.48</td>
<td>2.65%</td>
</tr>
<tr>
<td>2 2010</td>
<td>2,521.69</td>
<td>2,497.92</td>
<td>0.95%</td>
</tr>
<tr>
<td>3 diff</td>
<td>369.71</td>
<td>401.44</td>
<td>-7.90%</td>
</tr>
</tbody>
</table>
See Also

- CREATE MEMBER Statement
- FORMAT_STRING Clause
- WITH Clause
WHERE Clause

Applies a filter to a SELECT statement; describes how to slice the results of a SELECT statement.

Syntax and Details

```
select_statement WHERE set_expression
```

Where:

- `select_statement` is a statement that uses SELECT.
- `set_expression` is an expression that returns a set of members or tuples.

Instead of `set_expression`, you can use a measure search expression; see the example to see the behavior of WHERE in this case.

Because DeepSee automatically converts types where appropriate, you can also use a single member expression or tuple expression in place of the set expression.

**Important:** Each set element is used as a separate slicer axis, and the results of all the slicer axes are aggregated together. Each measure is aggregated according to its aggregation method, as specified in the cube definition. (In the examples here, %COUNT is added.)

Example

Compare the following two example SELECT statements, one with a WHERE clause and one without a WHERE clause.

```plaintext
SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients
```

Patient Count
---
1 Cedar Falls 1,039
2 Centerville 1,107
3 Cypress 1,096
4 Elm Heights 1,093
5 Juniper 1,150
6 Magnolia 1,092
7 Pine 1,157
8 Redwood 1,125
9 Spruce 1,141

The previous query shows the count of patients in each city. In contrast, consider the following query, which shows the count of male patients in each city:

```plaintext
SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients WHERE gend.male
```

Patient Count
---
1 Cedar Falls 509
2 Centerville 569
3 Cypress 517
4 Elm Heights 531
5 Juniper 574
6 Magnolia 527
7 Pine 569
8 Redwood 553
9 Spruce 557

To demonstrate the effect of multiple items in the WHERE clause, first consider the following query:
WHERE Clause

SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients WHERE colord.green

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cedar Falls</td>
</tr>
<tr>
<td>2 Centerville</td>
</tr>
<tr>
<td>3 Cypress</td>
</tr>
<tr>
<td>4 Elm Heights</td>
</tr>
<tr>
<td>5 Juniper</td>
</tr>
<tr>
<td>6 Magnolia</td>
</tr>
<tr>
<td>7 Pine</td>
</tr>
<tr>
<td>8 Redwood</td>
</tr>
<tr>
<td>9 Spruce</td>
</tr>
</tbody>
</table>

Now consider the following query, which uses both gend.male and colord.green as set elements in the WHERE clause:

SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients WHERE {gend.male,colord.green}

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cedar Falls</td>
</tr>
<tr>
<td>2 Centerville</td>
</tr>
<tr>
<td>3 Cypress</td>
</tr>
<tr>
<td>4 Elm Heights</td>
</tr>
<tr>
<td>5 Juniper</td>
</tr>
<tr>
<td>6 Magnolia</td>
</tr>
<tr>
<td>7 Pine</td>
</tr>
<tr>
<td>8 Redwood</td>
</tr>
<tr>
<td>9 Spruce</td>
</tr>
</tbody>
</table>

By comparing the results for Cedar Falls, for example, you can see that this query adds the results for male patients and the results for patients whose favorite color is green. If you instead wanted to see the results for male patients whose favorite color is green, you would use either a CROSSJOIN or a tuple expression in the WHERE clause, as follows:

SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients WHERE CROSSJOIN(gend.male,colord.green)

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cedar Falls</td>
</tr>
<tr>
<td>2 Centerville</td>
</tr>
<tr>
<td>3 Cypress</td>
</tr>
<tr>
<td>4 Elm Heights</td>
</tr>
<tr>
<td>5 Juniper</td>
</tr>
<tr>
<td>6 Magnolia</td>
</tr>
<tr>
<td>7 Pine</td>
</tr>
<tr>
<td>8 Redwood</td>
</tr>
<tr>
<td>9 Spruce</td>
</tr>
</tbody>
</table>

The following example uses a tuple expression in the WHERE clause:

SELECT MEASURES.[%COUNT] ON 0, homed.city.MEMBERS ON 1 FROM patients WHERE (gend.male,aged.60)

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cedar Falls</td>
</tr>
<tr>
<td>2 Centerville</td>
</tr>
<tr>
<td>3 Cypress</td>
</tr>
<tr>
<td>4 Elm Heights</td>
</tr>
<tr>
<td>5 Juniper</td>
</tr>
<tr>
<td>6 Magnolia</td>
</tr>
<tr>
<td>7 Pine</td>
</tr>
<tr>
<td>8 Redwood</td>
</tr>
<tr>
<td>9 Spruce</td>
</tr>
</tbody>
</table>

You can also use the WHERE clause as a way to display a specific measure:

SELECT gend.gender.MEMBERS ON 0 FROM patients WHERE MEASURES.[avg test score]

<table>
<thead>
<tr>
<th></th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Patients</td>
<td>74.78</td>
<td>74.46</td>
</tr>
</tbody>
</table>

Notice that the measure name is not shown, however.

If you use the WHERE clause with a measure search expression, the clause uses only rows of the fact table that do meet the given criteria. (A measure search expression is a DeepSee extension to MDX that considers the measure values in the fact table itself.)
SELECT MEASURES.[%COUNT] ON 0 FROM patients WHERE %SEARCH.&([MEASURES].[age]<10)
Patient Count
1,370

See Also
See the %FILTER clause.
WITH Clause

Defines one or more elements for use in the SELECT statement.

Syntax and Details

WITH with_details1 with_details2 ... select_statement

Where:

• `select_statement` is a statement that uses SELECT
• `with_details1`, `with_details2`, and so on can have one of the following syntaxes:

  MEMBER calc_mem_definition
  
  Or:

  SET named_set_definition
  
  Or:

  %PARM named_parameter_definition

You can mix these subclauses in a single WITH clause.

Tip: Notice that there is no comma to separate the WITH subclauses from each other. Nor is there a comma between the WITH clause and the SELECT statement.

The following sections provide the details for the MEMBER, SET, and %PARM subclauses.

WITH MEMBER

In a WITH clause, MEMBER defines a calculated member for use in the query. The MEMBER subclause has the following syntax:

MEMBER calc_mem_details, FORMAT_STRING='format_details', SOLVE_ORDER=integer

Where `calc_mem_details` is as follows:

`cube_name.[dimension_name].[new_member_name] AS 'value_expression'

And:

• `cube_name` is the name of a cube.
• `dimension_name` is the name of a dimension.
• `new_member_name` is the name of a member; the member may or may not be already defined in the cube. If it is, the definition given here takes precedence.
• `value_expression` is an MDX expression that defines the calculated member, typically in terms of references to other members.

For example:

MEASURES.[test score]/MEASURES.[%COUNT]
In any context where you use this calculated member, DeepSee first evaluates the Test Score and %COUNT measures in that context and then performs the division.

For another example:

AGGREGATE({colord.red, colord.blue, colord.yellow})

This new member refers to all the records of the fact table that correspond to the red, yellow, or blue members of the colord dimension.

For other variations, see “Defining Calculated Members and Named Sets” in Defining DeepSee Models.

- FORMAT_STRING='format_details' is an optional clause that specifies how to display the values. This clause is applicable only for numeric values. See FORMAT_STRING Clause.
- SOLVE_ORDER=integer is an optional clause that specifies the relative order in which to evaluate this calculated member. This clause is relevant only if the query contains calculated members on more than one axis. See SOLVE_ORDER Clause.

The first example shows a calculated member defined within the WITH clause:

WITH MEMBER MEASURES.avgage AS 'MEASURES.age/MEASURES.%COUNT'
SELECT MEASURES.avgage ON 0, diagd.MEMBERS ON 1 FROM patients

1 None                                33.24
2 asthma                              34.79
3 CHD                                 67.49
4 diabetes                            57.24
5 osteoporosis                        79.46

DeepSee first evaluates the Age and %COUNT measures and then performs the division for the avgage measure.

**WITH SET**

In a WITH clause, SET defines a named set for use in the query. The SET subclause has the following syntax:

```
SET set_name AS 'set_expression'
```

- set_name is an unquoted string that names the set.
- set_expression is an expression that refers to a set.

The following example shows a named set defined within the WITH clause:

WITH SET testset AS '{homed.city.members}' SELECT MEASURES.[%COUNT] ON 0, testset ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cedar Falls</td>
<td>1,045</td>
</tr>
<tr>
<td>Centerville</td>
<td>1,069</td>
</tr>
<tr>
<td>Cypress</td>
<td>1,150</td>
</tr>
<tr>
<td>Elm Heights</td>
<td>1,104</td>
</tr>
<tr>
<td>Juniper</td>
<td>1,155</td>
</tr>
<tr>
<td>Magnolia</td>
<td>1,111</td>
</tr>
<tr>
<td>Pine</td>
<td>1,138</td>
</tr>
<tr>
<td>Redwood</td>
<td>1,111</td>
</tr>
<tr>
<td>Spruce</td>
<td>1,117</td>
</tr>
</tbody>
</table>
WITH %PARM

In a WITH clause, %PARM defines a named parameter for use in the query. The %PARM subclause has the following syntax:

```
%PARM parameter_name AS 'value:default_value'
```

Or:

```
%PARM parameter_name AS 'value:default_value,caption:label'
```

- `parameter_name` is the name of the parameter.
- `default_value` is the default value of the parameter.
- `label` is the caption to use when prompting for a value of this parameter.

When you run a query within the MDX shell, the shell prompts you for values of any named parameters.

Then, to refer to the parameter within the query itself, use `@parameter_name`

For example:

```
>>WITH %PARM c as 'value:Pine' select homed.[city].@c ON 0 FROM patients
Please supply parameter value(s) for this query:
C [Pine]:

Pine
1,073
```

```
Elapsed time: 2.136337s
```

```
>>WITH %PARM c as 'value:Pine' select homed.[city].@c ON 0 FROM patients
Please supply parameter value(s) for this query:
C [Pine]:Magnolia

Magnolia
1,113
```

```
Elapsed time: 2.627897s
```

```
>>WITH %PARM c as 'value:Pine,caption:city' select homed.[city].@c ON 0 FROM patients
Please supply parameter value(s) for this query:
city [Pine]:

Pine
1,073
```

```
Elapsed time: 2.235228s
```

```
>>WITH %PARM c AS 'value:5,caption:count' SELECT TOPCOUNT(birthd.decade.MEMBERS, @c) ON 1 FROM patients
Please supply parameter value(s) for this query:
count [5]:3

1 1970s 1,593
2 1960s 1,505
3 2000s 1,442
```

```
Elapsed time: 1.207581s
```

See Also

The WITH clause defines elements that are available only during the query that defines them.

To define calculated members and named sets for use during the entire session, use the following statements:

- `CREATE MEMBER Statement`
- `CREATE SET Statement`
For information on defining calculated members and named sets as part of the cube definition (available in all sessions), see *Defining DeepSee Models*. 
MDX Functions

This reference section provides information on the MDX functions supported in InterSystems DeepSee.
%ALL

 Enables you to use a member while ignoring any ROW and COLUMN context that uses the hierarchy to which this member belongs. This function is a DeepSee extension to MDX.

**Returned Type**

This function returns a member.

**Syntax and Details**

```
member_expression.%ALL
```

Where:

- `member_expression` is an expression that evaluates to a member.

This function enables you to create calculated members that compare one member of a hierarchy to another member of the hierarchy (for example, comparing one product to all products).

**Example**

Sometimes it is necessary to compute values like the following:

- Percentage of one product compared to all products
- Percentage of one product compared to another product

For example, the following query uses a calculated member that equals the patient count for each age group, as a percentage of patients in all the age groups:

```sql
WITH MEMBER MEASURES.[pct age grps] AS 'aged.CURRENTMEMBER/aged.[all patients].%ALL', FORMAT_STRING='#.##'
SELECT {MEASURES.[%COUNT],MEASURES.[pct age grps]} ON 0,
aged.h1.[age group].MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>pct age grps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>4,216</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>4,212</td>
</tr>
<tr>
<td>3 60+</td>
<td>1,572</td>
</tr>
</tbody>
</table>

The calculated member is defined as the current member of the AgeD dimension, divided by the All member of that dimension:

`aged.CURRENTMEMBER/aged.[all patients].%ALL`

In contrast, consider the following query in which the calculated member does not use the %ALL function:

```sql
WITH MEMBER MEASURES.[BADpct age grps] AS 'aged.CURRENTMEMBER/aged.[all patients]'.
SELECT {MEASURES.[%COUNT],MEASURES.[BADpct age grps]} ON 0,
aged.h1.[age group].MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>BADpct age grps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>4,216</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>4,212</td>
</tr>
<tr>
<td>3 60+</td>
<td>1,572</td>
</tr>
</tbody>
</table>

In this case, the value of `aged.[all patients]` in each row is the same as the value of `aged.CURRENTMEMBER`, because the row members belong to the same hierarchy as `aged.[all patients]`.

Note that the %ALL function does consider the context given by members of other hierarchies. (It ignores only the hierarchy associated with the member that you use with the function.) For example:
WITH MEMBER MEASURES.[pct age grps] AS 'aged.CURRENTMEMBER/aged.[all patients].%ALL', FORMAT_STRING='#.##' 

SELECT CROSSJOIN(gend.MEMBERS,{MEASURES.[%COUNT],MEASURES.[pct age grps]}) ON 0, aged.hl.[age group].MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>pct age grps</th>
<th>Patient Count</th>
<th>pct age grps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>1,985</td>
<td>0.39</td>
<td>2,231</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>2,123</td>
<td>0.42</td>
<td>2,089</td>
</tr>
<tr>
<td>3 60+</td>
<td>926</td>
<td>0.18</td>
<td>646</td>
</tr>
</tbody>
</table>

Here, the first two Patient Count and pct age grps columns correspond to female patients and the second two correspond to male patients. Each pct age grps column indicates the patient count for that gender, as a percentage of all age groups for that gender.

Also note that the %ALL function does not ignore members of its hierarchy if those are used in a WHERE or FILTER clause; that is, the %ALL function fully respects all filtering applied to the query. For example:

WITH MEMBER MEASURES.[pct of all ages] AS 'aged.CURRENTMEMBER/aged.[all patients].%ALL', FORMAT_STRING='#.##' 

SELECT {MEASURES.[%COUNT],MEASURES.[pct of all ages]} ON 0, aged.hl.[age group].MEMBERS ON 1 FROM patients 
WHERE aged.hl.[age group].[0 to 29]

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>pct of all ages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>4,216</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>*</td>
</tr>
<tr>
<td>3 60+</td>
<td>*</td>
</tr>
</tbody>
</table>

See Also

- %MDX
%CELL

Returns the value of another cell in a pivot table, by position. This function is a DeepSee extension to MDX.

**Returned Type**

This function returns a number or a string.

**Syntax and Details**

%CELL(relative_column_position,relative_row_position)

Where:

- **relative_column_position** is an integer. Use 0 for the current column, −1 for the previous column (the column to the left), 1 for the next column (the column to the right), and so on.

- **relative_row_position** is an integer. Use 0 for the current row, −1 for the previous row (the column above), 1 for the next row (the column below), and so on.

DeepSee evaluates %CELL after resolving the rest of the query; this means that you cannot use this function within an expression used by another function.

**Example**

The following example displays rainfall data and cumulative rainfall data for a given span of time:

```
SELECT {MEASURES.[Rainfall Inches],%CELL(-1,0)+%CELL(0,-1)} ON 0, {dated.year.1960:1970} ON 1 FROM cityrainfall
```

<table>
<thead>
<tr>
<th>Rainfall Inches</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1960</td>
<td>177.83</td>
</tr>
<tr>
<td>2 1961</td>
<td>173.42</td>
</tr>
<tr>
<td>3 1962</td>
<td>168.11</td>
</tr>
<tr>
<td>4 1963</td>
<td>188.30</td>
</tr>
<tr>
<td>5 1964</td>
<td>167.58</td>
</tr>
<tr>
<td>6 1965</td>
<td>175.23</td>
</tr>
<tr>
<td>7 1966</td>
<td>182.50</td>
</tr>
<tr>
<td>8 1967</td>
<td>154.44</td>
</tr>
<tr>
<td>9 1968</td>
<td>163.97</td>
</tr>
<tr>
<td>10 1969</td>
<td>184.84</td>
</tr>
<tr>
<td>11 1970</td>
<td>178.31</td>
</tr>
</tbody>
</table>

Notice that the default label here is Expression. You can use %LABEL to provide a more suitable label. For example:

```
SELECT {MEASURES.[Rainfall Inches],%LABEL((%CELL(-1,0)+%CELL(0,-1)),"Cumulative Inches")} ON 0, {dated.year.1960:1970} ON 1 FROM cityrainfall
```

<table>
<thead>
<tr>
<th>Rainfall Inches</th>
<th>Cumulative Inches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1960</td>
<td>177.83</td>
</tr>
<tr>
<td>2 1961</td>
<td>351.25</td>
</tr>
<tr>
<td>3 1962</td>
<td>519.36</td>
</tr>
<tr>
<td>4 1963</td>
<td>707.66</td>
</tr>
<tr>
<td>5 1964</td>
<td>875.24</td>
</tr>
<tr>
<td>6 1965</td>
<td>1,050.47</td>
</tr>
<tr>
<td>7 1966</td>
<td>1,232.97</td>
</tr>
<tr>
<td>8 1967</td>
<td>1,387.41</td>
</tr>
<tr>
<td>9 1968</td>
<td>1,551.38</td>
</tr>
<tr>
<td>10 1969</td>
<td>1,736.22</td>
</tr>
<tr>
<td>11 1970</td>
<td>1,914.53</td>
</tr>
</tbody>
</table>
%FIRST

Returns the first non-empty member of a set. This function is a DeepSee extension to MDX.

Returned Type
This function returns a set element (that is, any type of MDX expression).

Syntax and Details

%FIRST(set_expression, optional_numeric_expression)

Where:

- set_expression is an expression that evaluates to a set, typically a set of members or tuples.
- optional_numeric_expression is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form [MEASURES].[measure_name]

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The %FIRST function returns the first non-missing value evaluated for each member of the given set.

See Also

- %LAST
%KPI

%KPI

Returns a value from a KPI, plugin, or worksheet. This function is a DeepSee extension to MDX.

Returned Type

This function returns a number.

Syntax and Details

%KPI(kpiName,propName,series)

Or:

%KPI(kpiName,propName,series,parmName1,parmValue1,parmName2,parmValue2)

Where:

• kpiName is the name of a KPI, plugin, or worksheet.
• For a KPI or plugin, propName is the quoted name of a <property> element of the KPI or plugin.
  For a worksheet, propName is the name of a column in the worksheet.
• For a KPI or plugin, series is the optional number or the quoted name of a series (row) in the KPI or plugin. The default is 1.
  For a worksheet, series is the name of a row of the worksheet.
• parmName1, parmName2, and so on are optional quoted names of parameters of the KPI or plugin; in most cases, these are filters). Note that parameter names are case-sensitive.
  The order in which you list the filters does not affect the KPI.
• parmValue1, parmValue2, and so on are the corresponding values of the named filters.

%KPI uses all provided parameter values and returns the value of the given propName for the given series. For KPIs and plugins, the caption for a value is the normalized and localized property name.

For MDX-based KPIs and plugins, you can use the special %CONTEXT parameter to cause the KPI to consider the context of query, which is otherwise ignored. For its value, specify a combination of the following flags:

• "rows" specifies that the context of the current pivot row should be used
• "columns" specifies that the context of the current pivot column should be used
• "filters" specifies that the context of the filters of current pivot should be used
• "all" specifies that all the preceding should be used (this is the default)

Use a pipe character (|) to combine flags, for example: "rows|columns". The value "all" is equivalent to "rows|columns|filters".

Example

The following example gets the value of the PatCount property for the first row of the DemoMDX KPI:

SELECT %KPI("demomdx","PatCount") ON 0 FROM patients

Patient Count
115
The following example defines a calculated measure that uses the %DeepSee.Plugin.Median sample plugin:

WITH MEMBER [MEASURES].[Median Amount Sold] AS
  'KPI("%DeepSee.Median","MEDIAN",1,"measure","Amount Sold","%CONTEXT")'
SELECT NON EMPTY {[Measures].[Amount Sold],[MEASURES].[MEDIAN AMOUNT SOLD]} ON 0,
  NON EMPTY [Product].[P1].[Product Name].Members ON 1
FROM [HoleFoods]

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount Sold</th>
<th>Median Amount Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Bagels (dozen)</td>
<td>38.96</td>
<td>2.95</td>
</tr>
<tr>
<td>2 Bundt Cake</td>
<td>1,632.01</td>
<td>19.95</td>
</tr>
<tr>
<td>3 Calamari (frozen)</td>
<td>566.90</td>
<td>22.95</td>
</tr>
<tr>
<td>4 Cheerios (box)</td>
<td>600.11</td>
<td>3.95</td>
</tr>
<tr>
<td>5 Donuts (dozen)</td>
<td>429.36</td>
<td>2.95</td>
</tr>
<tr>
<td>6 Free-range Donut</td>
<td>1,310.64</td>
<td>12.95</td>
</tr>
<tr>
<td>7 Fruit Loops (box)</td>
<td>772.83</td>
<td>4.95</td>
</tr>
<tr>
<td>8 Lifesavers (roll)</td>
<td>248.96</td>
<td>1.15</td>
</tr>
<tr>
<td>9 Onion ring</td>
<td>377.25</td>
<td>4.95</td>
</tr>
<tr>
<td>10 Onion ring</td>
<td>28.57</td>
<td>5.95</td>
</tr>
<tr>
<td>11 Penne (box)</td>
<td>176.72</td>
<td>1.95</td>
</tr>
<tr>
<td>12 Pineapple Rings</td>
<td>512.00</td>
<td>8.95</td>
</tr>
<tr>
<td>13 Pretzels (bag)</td>
<td>88.12</td>
<td>3.95</td>
</tr>
<tr>
<td>14 Swiss Cheese (sl)</td>
<td>445.10</td>
<td>5.95</td>
</tr>
<tr>
<td>15 Tortellini (froz)</td>
<td>1,000.89</td>
<td>6.95</td>
</tr>
<tr>
<td>16 Unsalted Pretzel</td>
<td>316.70</td>
<td>4.25</td>
</tr>
<tr>
<td>17 Ziti (box)</td>
<td>979.43</td>
<td>4.81</td>
</tr>
</tbody>
</table>

See Also

- %MDX
Given an MDX expression, returns the same expression with a different label for use as a row or column header. %LABEL can also specify formatting for the row or column. This function is a DeepSee extension to MDX.

**Returned Type**

This function returns an expression of the same type that you use with the function.

**Syntax and Details**

```
%LABEL(MDX_expression,label,format_string,solve_order,cell_style,heading_style)
```

Where:

- `MDX_expression` is an MDX expression of any type.
- `label` is a string to use as the new label as a row or column header.
- `format_string` is an optional literal (such as "#.##") that specifies how to display the values. See FORMAT_STRING Clause.
- `solve_order` is an optional integer that specifies the order in which to apply labels. See SOLVE_ORDER Clause.
- `cell_style` is an optional literal that specifies a Cascading Style Sheet (CSS) style definition to apply to the data cells.
- `heading_style` is an optional literal that specifies a Cascading Style Sheet (CSS) style definition to apply to the heading.

**Example**

```
SELECT %LABEL(MEASURES.[avg allergy count],"my label") ON 0, colord.MEMBERS ON 1 FROM patients
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
</tr>
<tr>
<td>4</td>
<td>Orange</td>
</tr>
<tr>
<td>5</td>
<td>Purple</td>
</tr>
<tr>
<td>6</td>
<td>Red</td>
</tr>
<tr>
<td>7</td>
<td>Yellow</td>
</tr>
</tbody>
</table>
```

In contrast:

```
SELECT MEASURES.[avg allergy count] ON 0, colord.MEMBERS ON 1 FROM patients
```

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
</tr>
<tr>
<td>4</td>
<td>Orange</td>
</tr>
<tr>
<td>5</td>
<td>Purple</td>
</tr>
<tr>
<td>6</td>
<td>Red</td>
</tr>
<tr>
<td>7</td>
<td>Yellow</td>
</tr>
</tbody>
</table>
```

For more examples, see %CELL, CURRENTMEMBER, IIF, and PROPERTIES.

**See Also**

- IIF
- ISNULL
%LAST

Returns the last non-empty member of a set. This function is a DeepSee extension to MDX.

Returned Type

This function returns a set element (that is, any type of MDX expression).

Syntax and Details

%LAST(set_expression, optional_numeric_expression)

Where:

- *set_expression* is an expression that evaluates to a set, typically a set of members or tuples.
- *optional_numeric_expression* is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form `[MEASURES] . [measure_name]`

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The %LAST function returns the last non-missing value evaluated for each member of the given set.

See Also

- %FIRST
%LIST

Returns a comma-separated list of values, given a set of values. This function is a DeepSee extension to MDX and is intended for use in KPIs.

**Returned Type**

This function returns a string that consists of a comma-separated list of values.

**Syntax and Details**

%LIST(set_expression)

Where:

- set_expression is an expression that evaluates to a set, typically a set of members or tuples.

**Example**

The following example shows the measure %COUNT for each diagnosis, followed by a string that contains a comma-separated list of those values:

WITH SET temp AS 'diagd.MEMBERS'
SELECT MEASURES.[%COUNT] ON 0, {temp,%LIST(temp)} ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None</td>
<td>8,603</td>
</tr>
<tr>
<td>2 asthma</td>
<td>676</td>
</tr>
<tr>
<td>3 CHD</td>
<td>345</td>
</tr>
<tr>
<td>4 diabetes</td>
<td>546</td>
</tr>
<tr>
<td>5 osteoporosis</td>
<td>212</td>
</tr>
<tr>
<td>6 LIST</td>
<td>8603,676,345,546,212</td>
</tr>
</tbody>
</table>

In the following example, the LIST column contains, for each city, a comma-separated list of the birth counts by decade for that city. This is the basic query used in the scorecard on the Demo Trend Lines dashboard.

SELECT {MEASURES.[%COUNT],%LIST(birthd.decade.MEMBERS)} ON 0, homed.city.MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cedar Falls</td>
<td>1,079 11,22,42,92,115,159,176,135,179,21</td>
</tr>
<tr>
<td>2 Centerville</td>
<td>1,058 3,24,51,57,114,157,167,152,161,145,27</td>
</tr>
<tr>
<td>3 Cypress</td>
<td>1,095 12,14,57,84,108,160,159,153,166,39</td>
</tr>
<tr>
<td>4 Elm Heights</td>
<td>1,086 8,22,54,72,121,154,168,135,176,143,33</td>
</tr>
<tr>
<td>5 Juniper</td>
<td>1,152 9,21,56,79,124,165,200,152,169,155,22</td>
</tr>
<tr>
<td>6 Magnolia</td>
<td>1,152 6,18,55,83,116,181,190,159,172,149,27</td>
</tr>
<tr>
<td>7 Pine</td>
<td>1,120 8,22,47,72,108,160,166,173,161,170,33</td>
</tr>
<tr>
<td>8 Redwood</td>
<td>1,132 6,17,61,90,109,174,189,129,171,154,32</td>
</tr>
<tr>
<td>9 Spruce</td>
<td>1,126 11,20,70,74,107,167,171,161,164,156,25</td>
</tr>
</tbody>
</table>

**Note:** This function is intended for use in KPIs displayed in scorecard widgets. Specifically, you use this as the source value for a scorecard property that displays a trend line graphic.
%LOOKUP

Returns one value from a term list. This function is a DeepSee extension to MDX (added in 2012.2).

Returned Type

This function returns a number.

Syntax and Details

%LOOKUP(termlist, key, field, default)

Where:

- `termlist` is the name of a term list.
- `key` is a key value in that term list.
- `field` is an optional field (column) name to use to get the value. By default the value column is returned.
- `default` is the value to return if the term list, key, or field cannot be found.

Unlike `%%TERMLIST`, `%LOOKUP` returns a single value.

If a cell in a term list defines a valid member reference, such as `[Outlet].[h1].[city].[boston]`, then `%LOOKUP` resolves this reference and does not return the member reference as a literal value. This is mainly to be compatible with term lists created for use with `%%TERMLIST`.

Example

Consider a term list called `VALUES` with one key/value pair:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CutOff</td>
<td>10000000</td>
</tr>
</tbody>
</table>

In this case, you can use `%LOOKUP` as follows:

```
SELECT %LOOKUP("Values","CutOff") ON ROWS FROM HOLEFOODS
== 1000000
```

For another example, the following query returns the list of cities whose population is greater than the cut off value in the term list:

```
ON ROWS,Outlet.H1.City.CurrentMember.Properties("Population") ON COLUMNS FROM HOLEFOODS
```

See Also

- `%%TERMLIST`
- `LOOKUP`
%MDX

Executes an MDX query outside of the context of the current query and then returns a single result. This function is a
DeepSee extension to MDX.

Returned Type

This function returns a number or a string.

Syntax and Details

%MDX(mdx_query, parmName1, parmValue1, parmName2, parmValue2)

Where:

• mdx_query is a quoted MDX query. It should return a single value; only the upper left cell is used.
  The query can include named parameters, calculated members, and named sets.

• parmName1, parmName2, and so on are optional named parameters in the query.
  The order in which you list parameters does not affect the query.

• parmValue1, parmValue2, and so on are the corresponding values of the named parameters.

This function executes the given query and returns a single value; if the query returns multiple rows or columns, the function
returns only the upper left cell. You use this to include a subquery within another query.

DeepSee provides the special %CONTEXT parameter which you can use within %MDX. For details, see %KPI, which also
accepts this parameter.

Example

You use %MDX to obtain a value that you want to include in a query but that would otherwise be affected by the row and
column definition of the query. For example, you can use it if you need to access the total record count:

WITH MEMBER A.FRACTION AS 'MEASURES.[%COUNT]/%MDX("SELECT FROM patients")'
SELECT { MEASURES.[%COUNT], A.FRACTION } ON 0, diagd.MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>FRACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,428</td>
<td>0.84</td>
</tr>
<tr>
<td>712</td>
<td>0.07</td>
</tr>
<tr>
<td>343</td>
<td>0.03</td>
</tr>
<tr>
<td>485</td>
<td>0.05</td>
</tr>
<tr>
<td>212</td>
<td>0.02</td>
</tr>
</tbody>
</table>

The following example uses %MDX with a named parameter (City):

SELECT %MDX("WITH %PARM City AS 'value:[All Cities]' SELECT FROM HOLEFOODS WHERE Outlet.@City","City",Outlet.CurrentMember.Properties("NAME")) ON 0, Outlet.City.Members on 1 FROM HOLEFOODS

<table>
<thead>
<tr>
<th>NAME</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>1,633</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Antwerp</td>
<td>421</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atlanta</td>
<td>3,331</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bangalore</td>
<td>3,786</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case, the subquery is as follows:

WITH %PARM City AS 'value:[All Cities]' 
SELECT FROM HOLEFOODS WHERE Outlet.@City","City",Outlet.CurrentMember.Properties("NAME"))
See Also

• %KPI
%NOT

Enables you to exclude a single member of a given level. This function is a DeepSee extension to MDX (added in 2012.1).

**Returned Type**

This function returns a member.

**Syntax and Details**

```
member_expression.%NOT
```

Where:

- `member_expression` is a member identifier. (Note that you cannot use a general member expression.)

This function enables you to exclude the given member.

**Example**

Often it is necessary for the WHERE clause to exclude a single member. For example, you might want to exclude the patients who have no diagnosis:

```
SELECT aged.[age group].MEMBERS ON 1 FROM patients WHERE EXCEPT(diagd.members, diagd.none)
```

<table>
<thead>
<tr>
<th>Age Group</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>44</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>65</td>
</tr>
<tr>
<td>3 60+</td>
<td>72</td>
</tr>
</tbody>
</table>

You can use the %NOT function to rewrite the previous query as follows:

```
SELECT aged.[age group].MEMBERS ON 1 FROM patients WHERE diagd.none.%NOT
```

<table>
<thead>
<tr>
<th>Age Group</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 to 29</td>
<td>44</td>
</tr>
<tr>
<td>2 30 to 59</td>
<td>62</td>
</tr>
<tr>
<td>3 60+</td>
<td>61</td>
</tr>
</tbody>
</table>

If you use this function on the column or row axis, you can see that it returns a member:

```
SELECT aged.[60+].%NOT ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Age Group</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not 60+</td>
<td>845</td>
</tr>
</tbody>
</table>

As you can see the name of the member is NOT followed by the name of the excluded member.

The %NOT function provides several advantages:

- DeepSee does not need to materialize all the members of the level.
- The negation occurs in an earlier part of the processing for greater efficiency.
- %NOT returns a single member which can be combined (internally) with other filters to form simple tuple expressions.

**See Also**

- `EXCEPT`
%OR

Enables you to combine multiple members of a given level into a single member, for efficiency. This function is a DeepSee extension to MDX (added in 2012.1).

**Returned Type**

This function returns a member. The name of the member is the name of the member, followed by +Others.

**Syntax and Details**

\[
\%OR(set\_expression)
\]

Where:

- \(set\_expression\) is an expression that evaluates to a set of members that belong to the same level. This expression must be enclosed in curly braces.

  The members must belong to the same level.

This function enables you to combine the given members into a single unit.

**Example**

Often it is necessary for the WHERE clause to contain a set of multiple members. For example:

```
SELECT gend.MEMBERS ON 1 FROM patients WHERE {allerd.[ant bites],allerd.soy,allerd.wheat}
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>Male</td>
<td>59</td>
</tr>
</tbody>
</table>

This query construction, however, means that DeepSee evaluates the query results multiple times (once for each item in the WHERE clause) and then combines them. This can be undesirably slow and can double-count items. (In this example, a given patient can be counted as many as three times, once for each allergy in the WHERE clause.)

With the %OR function, you can rewrite the query as follows:

```
SELECT gend.MEMBERS ON 1 FROM patients WHERE %OR({allerd.[ant bites],allerd.soy,allerd.wheat})
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Male</td>
<td>57</td>
</tr>
</tbody>
</table>

Note the numbers are lower, because this query does not double-count any patients.

If you use this function on the column or row axis, you can see that it returns a member:

```
SELECT %OR({allerd.[ant bites],allerd.soy,allerd.wheat}) ON 1 FROM patients
```

ant bites+Others 112

The %OR function provides several advantages:

- The members of the set are treated as one unit.
- The combination of members occurs in an earlier part of the processing for greater efficiency.
- %OR returns a single member which can be combined (internally) with other filters to form simple tuple expressions.
%SEARCH

Returns a measure search expression that you can use with the WHERE and %FILTER clauses.

**Returned Type**

See the section “Measure Search Expressions” in “Expression Types.”
%SPACE

Inserts a blank row or column with no label. This function is a DeepSee extension to MDX.

**Returned Type**

This function returns an empty string.

**Syntax and Details**

%SPACE()

**Example**

```sql
SELECT {allerd.MEMBERS,%SPACE(),allersevd.MEMBERS} ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>No.</th>
<th>Allergy</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No Data Available</td>
<td>3,962</td>
</tr>
<tr>
<td>2</td>
<td>additive/coloring agent</td>
<td>423</td>
</tr>
<tr>
<td>3</td>
<td>animal dander</td>
<td>428</td>
</tr>
<tr>
<td>4</td>
<td>ant bites</td>
<td>457</td>
</tr>
<tr>
<td>5</td>
<td>bee stings</td>
<td>407</td>
</tr>
<tr>
<td>6</td>
<td>dairy products</td>
<td>460</td>
</tr>
<tr>
<td>7</td>
<td>dust mites</td>
<td>422</td>
</tr>
<tr>
<td>8</td>
<td>eggs</td>
<td>419</td>
</tr>
<tr>
<td>9</td>
<td>fish</td>
<td>429</td>
</tr>
<tr>
<td>10</td>
<td>mold</td>
<td>438</td>
</tr>
<tr>
<td>11</td>
<td>Nil known allergies</td>
<td>1,382</td>
</tr>
<tr>
<td>12</td>
<td>peanut</td>
<td>441</td>
</tr>
<tr>
<td>13</td>
<td>pollen</td>
<td>424</td>
</tr>
<tr>
<td>14</td>
<td>shellfish</td>
<td>431</td>
</tr>
<tr>
<td>15</td>
<td>soy</td>
<td>455</td>
</tr>
<tr>
<td>16</td>
<td>tree nuts</td>
<td>452</td>
</tr>
<tr>
<td>17</td>
<td>wheat</td>
<td>419</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Nil known allergies</td>
<td>1,382</td>
</tr>
<tr>
<td>20</td>
<td>Minor</td>
<td>1,229</td>
</tr>
<tr>
<td>21</td>
<td>Moderate</td>
<td>1,205</td>
</tr>
<tr>
<td>22</td>
<td>Life-threatening</td>
<td>1,202</td>
</tr>
<tr>
<td>23</td>
<td>Inactive</td>
<td>1,184</td>
</tr>
<tr>
<td>24</td>
<td>Unable to determine</td>
<td>1,141</td>
</tr>
</tbody>
</table>
%TERMLIST

Enables you to create a set of members based on a term list. When used with the %OR function, %TERMLIST is particularly useful for filtering. This function is a DeepSee extension to MDX (added in 2012.2).

Returned Type
This function returns a set.

Syntax and Details

%TERMLIST(term_list_name, flag)

Where:

- term_list_name is a string expression that evaluates to the name of a term list.
- flag, which is optional, is either "EXCLUDE" or "INCLUDE" (the default).

This function returns a set that, by default, consists of members that are identified by the key values in the term list, in combination with the term list pattern. The term list pattern indicates the level to which the members belong, and indicates how to create the full identifiers for the members.

If you specify flag as "EXCLUDE", the set instead consists of all members of the given level except for the ones identified in the term list.

For information on defining term lists, see “Defining Term Lists” in the DeepSee Implementation Guide.

Example
For example, suppose that for HoleFoods, we have a term list named MyCities that is defined as follows:

<table>
<thead>
<tr>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
</tr>
<tr>
<td>Atlanta</td>
</tr>
<tr>
<td>Boston</td>
</tr>
<tr>
<td>New York</td>
</tr>
</tbody>
</table>

Suppose that this term list has the following Pattern expression:

[Outlet].[H1].[City].[*]

Then for this term list, the %TERMLIST function returns a set that consists of the Atlanta, Boston, and New York members of the City level. That is, the following two expressions are equivalent:

%TERMLIST("MyCities")

And:

{[Outlet].[H1].[City].[Atlanta],[Outlet].[H1].[City].[Boston],[Outlet].[H1].[City].[New York]}

When used with the %OR function, %TERMLIST is particularly useful for filtering. For example:

SELECT FROM holefoods %FILTER %OR(%TERMLIST("MyCities"))
See Also

- %LOOKUP
- %OR
**%TIMERANGE**

Enables you to define a range of time members, possibly open-ended. This function is a DeepSee extension to MDX (added in 2012.2).

**Returned Type**

This function returns a member.

**Syntax and Details**

```
%TIMERANGE(start_member,end_member,keyword)
```

Where:

- **start_member** is an optional expression that evaluates to a member of a time level. If you omit this, DeepSee uses the earliest member of this level.
- **end_member** is an optional expression that evaluates to a member of a time level. If you omit this, DeepSee uses the latest member of this level.
- **keyword** is optional and is either **INCLUSIVE** or **EXCLUSIVE**
  
  The default is **INCLUSIVE**.

You must specify `start_member`, `end_member`, or both.

The time level must be in the same cube that you are querying. That is, this function does not support relationships.

**Example**

The following example uses both `start_member` and `end_member`:

```
SELECT NON EMPTY DateOfSale.YearSold.MEMBERS ON 1 FROM holefoods
WHERE %TIMERANGE(DateOfSale.YearSold.&[2009],DateOfSale.YearSold.&[2011])
```

```
1 2009                                  179
2 2010                                  203
3 2011                                  224
```

The next example shows an open-ended range:

```
SELECT NON EMPTY DateOfSale.YearSold.MEMBERS ON 1 FROM holefoods
WHERE %TIMERANGE(DateOfSale.YearSold.&[2009])
```

```
1 2009                                  179
2 2010                                  203
3 2011                                  224
4 2012                                  114
```

The next example shows another open-ended range, this time using the **EXCLUSIVE** keyword:

```
SELECT NON EMPTY DateOfSale.YearSold.MEMBERS ON 1 FROM holefoods
WHERE %TIMERANGE(,DateOfSale.YearSold.&[2009],EXCLUSIVE)
```

```
1 2007                                  124
2 2008                                  156
```
%TIMEWINDOW

Returns a set of members of a time dimension that match the given range template.

Returned Type

This function returns a set of members.

Syntax and Details

%TIMEWINDOW(periodSet,rangeTemplateStart)

Or:

%TIMEWINDOW(periodSet,rangeTemplateStart,rangeTemplateEnd)

Where:

- **periodSet** is a set of members of a time level.
- **rangeTemplateStart** is a member of a time level within the same hierarchy, at a lower level than **periodSet**.
- **rangeTemplateEnd** is another member of a time level within the same hierarchy, at a lower level than **periodSet**. If specified, **rangeTemplateEnd** must fall within the same period as **rangeTemplateStart** (for example, these two members must belong to the same year or to the same month).

   The default for **rangeTemplateEnd** is **rangeTemplateStart**.

DeepSee generates the set of members from **rangeTemplateStart** to **rangeTemplateEnd** and then uses that as a template to specify a time window. For example, if **rangeTemplateStart** is January 2000, and **rangeTemplateEnd** is June 2000, the time window consists of the dates from 1 January to 30 June of any given year.

Then the function examines each member of the given **periodSet** and, for each, returns the child members that fall within the given time window.

This function is intended for use within the WHERE clause or the %FILTER clause. It includes optimizations to return members of higher time levels where possible, so that large numbers of members are not returned.

Example

First, the following query uses %TIMEWINDOW as rows. This query examines birth years and for each one, selects only the patients born between 1 January and 5 January, inclusive:

```
SELECT NON EMPTY %TIMEWINDOW(birthd.year.MEMBERS,birthd.[jan 01 1924],birthd.[jan 05 1924]) ON 1
FROM patients
```

```
1 Jan 4 1918 1
2 Jan 3 1934 1
3 Jan 3 1937 1
4 Jan 4 1937 1
5 Jan 2 1938 1
6 Jan 1 1940 1
7 Jan 1 1941 1
8 Jan 4 1947 1
9 Jan 5 1947 1
10 Jan 2 1949 1
11 Jan 1 1953 1
...
```

In this example, the range template arbitrarily refers to dates in the year 1924; any year could be used instead.

As noted earlier, this function is primarily meant for use in filtering. The following query simply selects all patients born between 1 January and 5 January of any given year:

```
SELECT NON EMPTY %TIMEWINDOW(birthd.year.MEMBERS,birthd.[jan 01],birthd.[jul 05]) ON 1
FROM patients
```
SELECT MEASURES.[%COUNT] ON 0 FROM patients
WHERE %TIMEWINDOW(birthd.year.MEMBERS,birthd.[jan 01 1924],birthd.[jan05 1924])

Patient Count
806

The following query uses the same filter but displays patients grouped by birth years:

SELECT MEASURES.[%COUNT] ON 0, NON EMPTY birthd.year.MEMBERS on 1
FROM patients
WHERE %TIMEWINDOW(birthd.year.MEMBERS,birthd.[jan 01 1924],birthd.[jan05 1924])

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1918</td>
</tr>
<tr>
<td>1</td>
<td>1934</td>
</tr>
<tr>
<td>2</td>
<td>1937</td>
</tr>
<tr>
<td>1</td>
<td>1938</td>
</tr>
<tr>
<td>1</td>
<td>1940</td>
</tr>
<tr>
<td>1</td>
<td>1941</td>
</tr>
<tr>
<td>2</td>
<td>1947</td>
</tr>
<tr>
<td>1</td>
<td>1949</td>
</tr>
<tr>
<td>1</td>
<td>1953</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

To make the result more understandable, the following query uses %LABEL to apply a better caption:

SELECT %LABEL(MEASURES.[%COUNT],"Born Jan 1-5") ON 0, NON EMPTY birthd.year.MEMBERS on 1
FROM patients
WHERE %TIMEWINDOW(birthd.year.MEMBERS,birthd.[jan 01 1924],birthd.[jan05 1924])

<table>
<thead>
<tr>
<th>Born Jan 1-5</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1918</td>
</tr>
<tr>
<td>1</td>
<td>1934</td>
</tr>
<tr>
<td>2</td>
<td>1937</td>
</tr>
<tr>
<td>1</td>
<td>1938</td>
</tr>
<tr>
<td>1</td>
<td>1940</td>
</tr>
<tr>
<td>1</td>
<td>1941</td>
</tr>
<tr>
<td>2</td>
<td>1947</td>
</tr>
<tr>
<td>1</td>
<td>1949</td>
</tr>
<tr>
<td>1</td>
<td>1953</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
%TOPMEMBERS

Returns a set of all members of the first level in the given hierarchy. Or, given a level, it returns a set of all the members of that level. This function is a DeepSee extension to MDX.

Returned Type

This function returns a set of members.

Syntax and Details

level_expression.%TOPMEMBERS

Or:

hierarchy_expression.%TOPMEMBERS

Or:

dimension_expression.%TOPMEMBERS

Where:

• \textit{level_expression} is an \textit{expression that returns a level}. For example:

\[\text{[dimension\_name].[hierarchy\_name].[level\_name]}\]

• \textit{hierarchy\_expression} is an \textit{expression that returns a hierarchy}. For example:

\[\text{[dimension\_name].[hierarchy\_name]}\]

• \textit{dimension\_expression} is a dimension name, included within square brackets if needed (see Identifiers). For example:

\[\text{[dimension\_name]}\]

DeepSee interprets this as a reference to the first visible hierarchy within the given dimension.

Given a level name, this function is equivalent to the \textit{MEMBERS} function.

Given a hierarchy name, this function returns a set that consists of the members of the first level defined in that hierarchy.

Given a dimension name, this function returns a set that consists of the members of the first level defined in the first visible hierarchy of this dimension.

The DeepSee Analyzer uses this function when you drag and drop a dimension into the \textit{Rows} or \textit{Columns}. Specifically, when you drag and drop a dimension, the Analyzer uses the expression \[\text{[dimension\_name].[hierarchy\_name].%TOPMEMBERS, where hierarchy\_name is the first hierarchy defined in the dimension.}\]

Example

For example, consider the following cube contents:

\texttt{BirthD}
\texttt{H1}
\texttt{Decade}
\texttt{Year}
\texttt{Period}
\texttt{Date}
The following query uses the `%TOPMEMBERS` function with `H1` hierarchy (the only hierarchy in this case), so it retrieves all members of the Decade level:

```
SELECT birthd.%TOPMEMBERS ON 1 FROM patients
All Patients
1 1910s                                      71
2 1920s                                      223
3 1930s                                      572
4 1940s                                      683
5 1950s                                      1,030
6 1960s                                      1,500
7 1970s                                      1,520
8 1980s                                      1,400
9 1990s                                      1,413
10 2000s                                     1,433
11 2010s                                     155
```

**See Also**

- MEMBERS
AGGREGATE

Returns the aggregate value for a given measure (or of the current measure), across all elements of a set, according to the aggregation logic of the measure.

Returned Type

This function returns a number.

Syntax and Details

AGGREGATE(set_expression, optional_numeric_expression)

Where:

- set_expression is an expression that evaluates to a set, typically a set of members or tuples.
- optional_numeric_expression is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form [MEASURES].[measure_name]

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the aggregate value of those values.

Example

First, the following query shows values of three measures for the members of the aged.decade level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0, 
birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>5,359</td>
<td>75.17</td>
</tr>
<tr>
<td>227</td>
<td>12,910</td>
<td>74.20</td>
</tr>
<tr>
<td>567</td>
<td>33,211</td>
<td>74.67</td>
</tr>
<tr>
<td>724</td>
<td>38,420</td>
<td>74.67</td>
</tr>
<tr>
<td>1,079</td>
<td>46,883</td>
<td>74.72</td>
</tr>
<tr>
<td>1,475</td>
<td>57,814</td>
<td>74.16</td>
</tr>
<tr>
<td>1,549</td>
<td>49,784</td>
<td>74.35</td>
</tr>
<tr>
<td>1,333</td>
<td>35,919</td>
<td>74.13</td>
</tr>
<tr>
<td>1,426</td>
<td>29,219</td>
<td>74.79</td>
</tr>
<tr>
<td>1,406</td>
<td>20,072</td>
<td>74.95</td>
</tr>
<tr>
<td>134</td>
<td>1,346</td>
<td>73.55</td>
</tr>
</tbody>
</table>

Next, the following query uses AGGREGATE to find the aggregated values for these measures, across this set of members:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0, 
AGGREGATE(birthd.decade.MEMBERS) ON 1 FROM patients
```

AGGREGATE                    10,000            330,947              74.28

The following query uses the second argument of the AGGREGATE function:

```
SELECT AGGREGATE(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

AGGREGATE 10,000

For additional, similar examples, see AVG.

See Also

- AVG
MDX Functions

- MAX
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
ALLMEMBERS

Returns a set of all members of the given level or hierarchy. Or returns a set of all members of the first hierarchy of a dimension. In either case, any calculated members are also returned.

Returned Type

This function returns a set of members.

Syntax and Details

- `level_expression.ALLMEMBERS`
  - Or:
    - `hierarchy_expression.ALLMEMBERS`
  - Or:
    - `dimension_expression.ALLMEMBERS`

Where:

- `level_expression` is an expression that returns a level. For example:
  - `[dimension_name].[hierarchy_name].[level_name]`
- `hierarchy_expression` is an expression that returns a hierarchy. For example:
  - `[dimension_name].[hierarchy_name]`
- `dimension_expression` is a dimension name, included within square brackets if needed (see Identifiers). For example:
  - `[dimension_name]`

DeepSee interprets this as a reference to the first visible hierarchy within the given dimension.

Given a level expression, this function returns a set that consists of the members of that level. The members are in the order specified in the level definition in the cube.

Given a hierarchy expression, this function returns a set that consists of the members of all levels in that hierarchy, including the All member, if defined. The members are returned in hierarchical order.

Given a dimension name, this function returns a set that consists of the members of all levels in the first visible hierarchy of that dimension.

In any case, any calculated members are also returned (in contrast to the MEMBERS function).

For information on hierarchical order, see the HIERARCHIZE function.

Example

The following query displays all members of the Home Zip level as rows:

```sql
SELECT MEASURES.[%COUNT] ON 0, homed.zip.ALLMEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 32006</td>
<td>2,272</td>
</tr>
<tr>
<td>2 32007</td>
<td>1,111</td>
</tr>
<tr>
<td>3 34577</td>
<td>3,399</td>
</tr>
<tr>
<td>4 36711</td>
<td>1,069</td>
</tr>
<tr>
<td>5 38928</td>
<td>2,149</td>
</tr>
</tbody>
</table>
The following query displays all members of all levels in the Home.H1 hierarchy as rows:

```
SELECT MEASURES.[%COUNT] ON 0, homed.h1.ALLMEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>1 32006</th>
<th>2,272</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Juniper</td>
<td></td>
<td>1,155</td>
</tr>
<tr>
<td>3 Spruce</td>
<td></td>
<td>1,117</td>
</tr>
<tr>
<td>4 32007</td>
<td></td>
<td>1,111</td>
</tr>
<tr>
<td>5 Redwood</td>
<td></td>
<td>1,111</td>
</tr>
<tr>
<td>6 34577</td>
<td></td>
<td>3,399</td>
</tr>
<tr>
<td>7 Cypress</td>
<td></td>
<td>1,150</td>
</tr>
<tr>
<td>8 Magnolia</td>
<td></td>
<td>1,111</td>
</tr>
<tr>
<td>9 Pine</td>
<td></td>
<td>1,138</td>
</tr>
<tr>
<td>10 36711</td>
<td></td>
<td>1,069</td>
</tr>
<tr>
<td>11 Centerville</td>
<td></td>
<td>1,069</td>
</tr>
<tr>
<td>12 38928</td>
<td></td>
<td>2,149</td>
</tr>
<tr>
<td>13 Cedar Falls</td>
<td></td>
<td>1,045</td>
</tr>
<tr>
<td>14 Elm Heights</td>
<td></td>
<td>1,104</td>
</tr>
</tbody>
</table>

The following query shows all measures, each aggregated across the cube:

```
SELECT MEASURES.ALLMEMBERS ON 1, gend.gender.MEMBERS on 0 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Male</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Age</td>
<td>187,139</td>
<td>170,117</td>
</tr>
<tr>
<td>3 Avg Age</td>
<td>36.93</td>
<td>34.49</td>
</tr>
<tr>
<td>4 Allergy Count</td>
<td>3,067</td>
<td>3,131</td>
</tr>
<tr>
<td>5 Avg Allergy Count</td>
<td>1.02</td>
<td>1.04</td>
</tr>
<tr>
<td>6 Encounter Count</td>
<td>169,164</td>
<td>158,183</td>
</tr>
<tr>
<td>7 Avg Encounter Cou</td>
<td>33.39</td>
<td>32.07</td>
</tr>
<tr>
<td>8 Test Score</td>
<td>302,267</td>
<td>298,818</td>
</tr>
<tr>
<td>9 Avg Test Score</td>
<td>74.78</td>
<td>74.46</td>
</tr>
</tbody>
</table>

See Also

- MEMBERS
ANCESTOR

Returns the ancestor of the given member, within the given level.

**Returned Type**

This function returns a member.

**Syntax and Details**

\[
\text{ANCESTOR}(\text{member\_expression}, \text{ancestor\_level})
\]

Where:

- \textit{member\_expression} is an expression that returns a member.
  This expression cannot refer to a measure.
- \textit{ancestor\_level} is an expression that returns a level. For example:
  \[
  \text{[dimension\_name].[hierarchy\_name].[level\_name]}
  \]
  This level must be the parent level of \textit{member\_expression} or an ancestor of that member.

This function returns the ancestor of the given member, within the given level.

**Example**

The following query displays the year that is the ancestor of March 24, 1943:

\[
\text{SELECT MEASURES.}[^{\text{COUNT}}] \text{ ON 0, ANCESTOR(birthd.\{Mar 24 1943\},birthd.year) ON 1 FROM patients}
\]

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1943</td>
</tr>
<tr>
<td>76</td>
</tr>
</tbody>
</table>

In contrast, the following query displays the period that is the ancestor of March 24, 1943:

\[
\text{SELECT MEASURES.}[^{\text{COUNT}}] \text{ ON 0, ANCESTOR(birthd.\{Mar 24 1943\},birthd.period) ON 1 FROM patients}
\]

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar-1943</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

**See Also**

- CHILDREN
- CLOSINGPERIOD
- COUSIN
- OPENINGPERIOD
- PARENT
- PERIODSTODATE
AVG

Returns the average value of a given expression (or of the current measure), across all elements of a set that have a non-null value for that expression.

Returned Type

This function returns a number.

Syntax and Details

\[
\text{AVG(set_expression, optional_numeric_expression)}
\]

Where:

- \text{set_expression} is an expression that evaluates to a set, typically a set of members or tuples.
- \text{optional_numeric_expression} is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form \([\text{MEASURES}].[\text{measure_name}]\)

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses \%COUNT, which counts records in the fact table.

The function evaluates the numeric value for each element of the set, ignores any elements for which this value is null, and computes the average value for the remaining elements.

If you want to include the null elements in the average, use an expression for \text{optional_numeric_expression} that replaces null values with zero values.

If the numeric value is null for all elements, the function returns null.

Example

First, the following query shows values of three measures for the members of the aged.decade level:

```sql
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
birthd.decade.Members ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>5,359</td>
<td>75.17</td>
</tr>
<tr>
<td>227</td>
<td>12,910</td>
<td>74.20</td>
</tr>
<tr>
<td>567</td>
<td>33,211</td>
<td>74.67</td>
</tr>
<tr>
<td>724</td>
<td>38,420</td>
<td>73.39</td>
</tr>
<tr>
<td>1,079</td>
<td>46,883</td>
<td>73.72</td>
</tr>
<tr>
<td>1,475</td>
<td>57,814</td>
<td>74.16</td>
</tr>
<tr>
<td>1,549</td>
<td>49,794</td>
<td>74.35</td>
</tr>
<tr>
<td>1,333</td>
<td>35,919</td>
<td>73.93</td>
</tr>
<tr>
<td>1,426</td>
<td>29,219</td>
<td>74.79</td>
</tr>
<tr>
<td>1,406</td>
<td>20,072</td>
<td>74.95</td>
</tr>
<tr>
<td>134</td>
<td>1,346</td>
<td>73.55</td>
</tr>
</tbody>
</table>

Next, the following query shows the average values for these measures for the members of this level:

```sql
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
AVG(birthd.decade.Members) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>909.09</td>
<td>30,866.09</td>
<td>74.28</td>
</tr>
</tbody>
</table>

Here, each value is the average of the values in a column in the preceding query. For example, the Patient Count value is the average of the Patient Count values in the preceding query.

For another example, we use the second argument for AVG:
The following example uses AVG in a query that does not specify a measure:

```
SELECT AVG(birthd.decade.MEMBERS) ON 0 FROM patients
```

```
AVG 909.09
```

In this case, the function uses %COUNT, which counts records in the fact table.

Finally, the following example uses AVG in a query that specifies a measure in the WHERE clause:

```
SELECT AVG(birthd.decade.MEMBERS) ON 0 FROM patients WHERE MEASURES.[encounter count]
```

```
AVG 30,086.09
```

In this case, the function uses the measure specified in the WHERE clause.

**See Also**

- AGGREGATE
- MAX
- MEDIAN
- MIN
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
**BOTTOMCOUNT**

Sorts a set and returns a subset from its lower-valued end, given a desired element count.

**Returned Type**

This function returns a set of members or tuples, depending on the set used.

**Syntax and Details**

```
BOTTOMCOUNT(set_expression, element_count, optional_ordering_expression)
```

Where:

- **set_expression** is an expression that evaluates to a set of members or tuples.
- **element_count** is an integer literal.
  - The function uses this argument to determine the number of elements to return in the subset. If this argument is greater than the number of elements, all elements are returned.
- **optional_ordering_expression** is a numeric-valued expression that determines the order of the set elements.
  - Typically, this expression has the form `[MEASURES].[measure_name]`
  - The function evaluates this expression for each element of the set and sorts the elements of the set in ascending order according to this value. Any hierarchies are ignored.
  - If this argument is omitted, the function uses the current order of the set elements (and this function behaves like the TAIL function).

**Example**

First consider the following query and the results it returns:

```
SELECT MEASURES.[%COUNT] ON 0,
BOTTOMCOUNT(birthd.decade.MEMBERS, 100, MEASURES.[%COUNT]) ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>71</td>
</tr>
<tr>
<td>2 2010s</td>
<td>155</td>
</tr>
<tr>
<td>3 1920s</td>
<td>223</td>
</tr>
<tr>
<td>4 1930s</td>
<td>572</td>
</tr>
<tr>
<td>5 1940s</td>
<td>683</td>
</tr>
<tr>
<td>6 1950s</td>
<td>1,030</td>
</tr>
<tr>
<td>7 1960s</td>
<td>1,400</td>
</tr>
<tr>
<td>8 1990s</td>
<td>1,413</td>
</tr>
<tr>
<td>9 2000s</td>
<td>1,433</td>
</tr>
<tr>
<td>10 1960s</td>
<td>1,500</td>
</tr>
<tr>
<td>11 1970s</td>
<td>1,520</td>
</tr>
</tbody>
</table>

Because **count_expression** is greater than the number of members, all members are returned. The members are sorted in ascending order according to the value of the %COUNT measure.

Next, consider a similar query, using **count_expression** equal to 3:

```
SELECT MEASURES.[%COUNT] ON 0,
BOTTOMCOUNT(birthd.decade.MEMBERS, 3, MEASURES.[%COUNT]) ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>71</td>
</tr>
<tr>
<td>2 2010s</td>
<td>155</td>
</tr>
<tr>
<td>3 1920s</td>
<td>223</td>
</tr>
</tbody>
</table>

This query selects three members from the lower-valued end of the set.
See Also

- TOPCOUNT
**BOTTOMPERCENT**

Sorts a set and returns a subset from its lower-valued end, given a cutoff percentage that is applied to a total across members.

**Returned Type**

This function returns a set of members or tuples, depending on the set used.

**Syntax and Details**

```
BOTTOMPERCENT(set_expression, percentage, ordering_expression)
```

- **set_expression** is an expression that evaluates to a set of members or tuples.
- **percentage** is a numeric literal equal to or less than 100. For example, 15 represents 15 percent.
  
  The function uses this argument to determine the cutoff point for elements to return in the subset.
  
  There is usually a member that straddles the cutoff point; this member is assigned to the upper set, rather than the lower set. As a result, in the returned subset, the cumulative total for `ordering_expression` could be less than `percentage`, as a percentage of the entire set.
- **ordering_expression** is a numeric-valued expression that determines the order of the set members.

  The function evaluates this expression for each element of the set and sorts the elements of the set in ascending order according to this value. Any hierarchies are ignored.

**Example**

First consider the following query and the results it returns:

```
SELECT MEASURES.[%COUNT] ON 0,
BOTTOMPERCENT(birthd.decade.MEMBERS, 100, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>6</td>
</tr>
<tr>
<td>2 1920s</td>
<td>13</td>
</tr>
<tr>
<td>3 2010s</td>
<td>44</td>
</tr>
<tr>
<td>4 1940s</td>
<td>54</td>
</tr>
<tr>
<td>5 1930s</td>
<td>56</td>
</tr>
<tr>
<td>6 1950s</td>
<td>107</td>
</tr>
<tr>
<td>7 1970s</td>
<td>128</td>
</tr>
<tr>
<td>8 1960s</td>
<td>136</td>
</tr>
<tr>
<td>9 1990s</td>
<td>144</td>
</tr>
<tr>
<td>10 1980s</td>
<td>155</td>
</tr>
<tr>
<td>11 2000s</td>
<td>157</td>
</tr>
</tbody>
</table>

Because `percentage` is 100, all members are returned.

Now consider a variation of the preceding, in which `percentage` is 50, so that we see the bottom 50 percent:

```
SELECT MEASURES.[%COUNT] ON 0, BOTTOMPERCENT(birthd.decade.MEMBERS, 50, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>6</td>
</tr>
<tr>
<td>2 1920s</td>
<td>13</td>
</tr>
<tr>
<td>3 2010s</td>
<td>44</td>
</tr>
<tr>
<td>4 1940s</td>
<td>54</td>
</tr>
<tr>
<td>5 1930s</td>
<td>56</td>
</tr>
<tr>
<td>6 1950s</td>
<td>107</td>
</tr>
<tr>
<td>7 1970s</td>
<td>128</td>
</tr>
</tbody>
</table>

The total for the `%COUNT` measure for these members is slightly less than the specified threshold (50% of the total).
See Also

- TOPPERCENT
**BOTTOMSUM**

Sorts a set and returns a subset from its lower-valued end, given a cutoff value that is applied to a total across elements.

**Returned Type**

This function returns a set of members or tuples, depending on the set used.

**Syntax and Details**

```
BOTTOMSUM(set_expression, cutoff_value, ordering_expression)
```

- **set_expression** is an expression that evaluates to a set of members or tuples.
- **cutoff_value** is a numeric literal.
  
The function uses this argument to determine the cutoff value for elements to return in the subset.
  
  For all elements in the returned subset, the sum of the values of **ordering_expression** will be less than or equal to **cutoff_value**.
  
- **ordering_expression** is a numeric-valued expression that determines the order of the set elements.
  
  The function evaluates this expression for each element of the set and sorts the elements of the set in ascending order according to this value. Any hierarchies are ignored.

**Example**

First consider an example in which the cutoff value is high enough to include all members:

```
SELECT MEASURES.[%COUNT] ON 0,
BOTTOMSUM(birthd.decade.MEMBERS, 10000, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1910s</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010s</td>
<td></td>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1920s</td>
<td></td>
<td></td>
<td>223</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1930s</td>
<td></td>
<td></td>
<td></td>
<td>572</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1940s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1950s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1960s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1970s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1,520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now consider a variation in which the cutoff value is set to 2500:

```
SELECT MEASURES.[%COUNT] ON 0,
BOTTOMSUM(birthd.decade.MEMBERS, 2500, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1910s</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010s</td>
<td></td>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1920s</td>
<td></td>
<td></td>
<td>223</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1930s</td>
<td></td>
<td></td>
<td></td>
<td>572</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1940s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>683</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

- TOPSUM
CHILDREN

Returns a set that contains the children, if any, of a specified member.

Returned Type
This function returns a set of members.

Syntax and Details

```
member_expression.CHILDREN
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

If the specified member has no children, this function returns an empty set.

Example

For example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.[1960s].CHILDREN ON 1 FROM patients

Patient Count
1 1960 105
2 1961 153
3 1962 144
4 1963 153
5 1964 136
6 1965 149
7 1966 187
8 1967 159
9 1968 169
10 1969 145
```

See Also

- ANCESTOR
- COUSIN
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- LASTSIBLING
- PARENT
- SIBLINGS
CLOSINGPERIOD

Returns the last descendent member of the given level, at the same level as the given member. This function is intended primarily for use with time levels.

Returned Type

This function returns a member.

Syntax and Details

\[
\text{CLOSINGPERIOD(ancestor\_level,member\_expression)}
\]

Where:

- **ancestor\_level** is an expression that returns a level. For example:
  \[
  [\text{dimension\_name}].[\text{hierarchy\_name}].[\text{level\_name}]
  \]
  This level must be the parent level of **member\_expression** or an ancestor of that member.
- **member\_expression** is an expression that returns a member.
  This expression cannot refer to a measure.

Given a level and a member, this function returns the last member that is a descendent of the given level and that is at the same level as member.

Example

The following query displays the closing quarter for the year that includes Q3 2003:

```sql
SELECT MEASURES.[%COUNT] ON 0, CLOSINGPERIOD (birthd.year,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4 2003</td>
</tr>
<tr>
<td>40</td>
</tr>
</tbody>
</table>

In contrast, the following query displays the closing quarter for the decade that includes Q3 2003:

```sql
SELECT MEASURES.[%COUNT] ON 0, CLOSINGPERIOD (birthd.decade,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4 2010</td>
</tr>
<tr>
<td>36</td>
</tr>
</tbody>
</table>

See Also

- **ANCESTOR**
- **COUSIN**
- **OPENINGPERIOD**
- **PERIODSTODATE**
COUNT

Returns the count of elements in the given set.

Returned Type
This function returns a number.

Syntax and Details

COUNT(set_expression)

Or:

COUNT(set_expression,EXCLUDEEMPTY)

• set_expression is an expression that evaluates to a set.

By default, COUNT considers any empty elements and counts them along with the non-empty elements. If you use the EXCLUDEEMPTY keyword, this function returns the number of non-empty elements.

Example

For example, the following query counts the members of the Home City level:

```sql
SELECT COUNT(homed.city.MEMBERS) ON 0 FROM patients
```

COUNT

Results                                  9

The next examples demonstrate the EXCLUDEEMPTY keyword. First, consider the following query:

```sql
SELECT aged.[age group].MEMBERS ON 0, diagd.MEMBERS ON 1 FROM patients WHERE MEASURES.[%COUNT]
```

<table>
<thead>
<tr>
<th></th>
<th>0 to 29</th>
<th>30 to 59</th>
<th>60+</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None</td>
<td>3,839</td>
<td>3,615</td>
<td>971</td>
</tr>
<tr>
<td>2 asthma</td>
<td>308</td>
<td>282</td>
<td>113</td>
</tr>
<tr>
<td>3 CHD</td>
<td>1</td>
<td>93</td>
<td>229</td>
</tr>
<tr>
<td>4 diabetes</td>
<td>30</td>
<td>246</td>
<td>228</td>
</tr>
<tr>
<td>5 osteoporosis</td>
<td>*</td>
<td>*</td>
<td>200</td>
</tr>
</tbody>
</table>

The following query counts the number of members of the Diagnoses level:

```sql
WITH SET myset AS 'diagd.MEMBERS'
SELECT COUNT(myset) ON 0 FROM patients
```

COUNT

All Patients                              5

The following query counts the number of members of the Diagnoses level and uses the WHERE clause to get only patients in the age group 0 to 29:

```sql
WITH SET myset AS 'diagd.MEMBERS' SELECT COUNT(myset) ON 0 FROM patients WHERE MEASURES.[%COUNT]
```

As you can see, although the query uses the WHERE clause, the COUNT function returns the same value as before; this is because COUNT considers empty elements by default.

The next query is a variation of the preceding but uses EXCLUDEEMPTY:
WITH SET myset AS 'diagd.MEMBERS' SELECT COUNT(myset,EXCLUDEEMPTY) ON 0 FROM patients WHERE aged.[0 to 29]  
COUNT  
4

For another example, you can use COUNT with a set of scalar items, rather than the more common set of members:

WITH SET test AS '"item 1","item 2",23'  
SELECT COUNT(test) ON 0 FROM patients  
COUNT All Patients  
3
COUSIN

Given a reference member and a member of a higher level in the same hierarchy, this function finds the ancestor of the reference member at that higher level, determines the relative position of the reference member to that ancestor, and then returns the descendent of the higher member that has the same relative position. This function is intended primarily for use with time levels.

**Returned Type**

This function returns a member.

**Syntax and Details**

```
COUSIN(member_expression, higher_member_expression)
```

Where:

- `member_expression` is an expression that returns a member.
- `higher_member_expression` is an expression that returns a member that is a member of a higher level in the hierarchy that contains `member_expression`.

These expressions cannot refer to measures.

This function finds the ancestor of the reference member at the higher level, determines the relative position of the reference member to that ancestor, and then returns the descendent of the higher member that has the same relative position. If no such member is found, this function returns an empty set.

**Example**

For example, the following query finds the cousin of March 24, 1943, within the year 1990:

```
SELECT MEASURES.[%COUNT] ON 0, COUSIN(birthd.[Mar 24 1943],birthd.1990) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar 24 1990</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

In contrast, the following query finds the cousin of March 24, 1943, within January 1990:

```
SELECT MEASURES.[%COUNT] ON 0, COUSIN(birthd.[Mar 24 1943],birthd.[jan-1990]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 24 1990</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

**See Also**

- ANCESTOR
- CHILDREN
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- LASTSIBLING
- PARALLELPERIOD
- PARENT
- SIBLINGS
CROSSJOIN

Returns a set of tuples formed by the cross-product of the specified sets.

 Returned Type

This function returns a set of tuples.

 Syntax and Details

CROSSJOIN(set_expression1, set_expression2)

Where:

• set_expression1 and set_expression2 are expressions that evaluate to sets of members.

The function identifies all the members of each set and then generates a set of tuples that combine each member of the first set with each member of the second set.

 Tip: The keyword phrase NON EMPTY is particularly useful with this function. You can use this keyword phrase immediately before any set expression.

 Example

For example:

SELECT MEASURES.[%COUNT] ON 0, CROSSJOIN(diagd.MEMBERS, aged.[age group].MEMBERS) ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None-&gt;0 to 29</td>
<td>3,839</td>
</tr>
<tr>
<td>2 None-&gt;30 to 59</td>
<td>3,615</td>
</tr>
<tr>
<td>3 None-&gt;60+</td>
<td>971</td>
</tr>
<tr>
<td>4 asthma-&gt;0 to 29</td>
<td>308</td>
</tr>
<tr>
<td>5 asthma-&gt;30 to 59</td>
<td>282</td>
</tr>
<tr>
<td>6 asthma-&gt;60+</td>
<td>113</td>
</tr>
<tr>
<td>7 CHD-&gt;0 to 29</td>
<td>1</td>
</tr>
<tr>
<td>8 CHD-&gt;30 to 59</td>
<td>93</td>
</tr>
<tr>
<td>9 CHD-&gt;60+</td>
<td>229</td>
</tr>
<tr>
<td>10 diabetes-&gt;0 to 29</td>
<td>30</td>
</tr>
<tr>
<td>11 diabetes-&gt;30 to 59</td>
<td>246</td>
</tr>
<tr>
<td>12 diabetes-&gt;60+</td>
<td>228</td>
</tr>
<tr>
<td>13 osteoporosis-&gt;0 to 29</td>
<td>*</td>
</tr>
<tr>
<td>14 osteoporosis-&gt;30 to 59</td>
<td>*</td>
</tr>
<tr>
<td>15 osteoporosis-&gt;60+</td>
<td>200</td>
</tr>
</tbody>
</table>

In contrast, suppose that we add the NON EMPTY keyword phrase:

SELECT MEASURES.[%COUNT] ON 0, NON EMPTY CROSSJOIN(diagd.MEMBERS, aged.[age group].MEMBERS) ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None-&gt;0 to 29</td>
<td>3,839</td>
</tr>
<tr>
<td>2 None-&gt;30 to 59</td>
<td>3,615</td>
</tr>
<tr>
<td>3 None-&gt;60+</td>
<td>971</td>
</tr>
<tr>
<td>4 asthma-&gt;0 to 29</td>
<td>308</td>
</tr>
<tr>
<td>5 asthma-&gt;30 to 59</td>
<td>282</td>
</tr>
<tr>
<td>6 asthma-&gt;60+</td>
<td>113</td>
</tr>
<tr>
<td>7 CHD-&gt;0 to 29</td>
<td>1</td>
</tr>
<tr>
<td>8 CHD-&gt;30 to 59</td>
<td>93</td>
</tr>
<tr>
<td>9 CHD-&gt;60+</td>
<td>229</td>
</tr>
<tr>
<td>10 diabetes-&gt;0 to 29</td>
<td>30</td>
</tr>
<tr>
<td>11 diabetes-&gt;30 to 59</td>
<td>246</td>
</tr>
<tr>
<td>12 diabetes-&gt;60+</td>
<td>228</td>
</tr>
<tr>
<td>13 osteoporosis-&gt;0 to 29</td>
<td>*</td>
</tr>
</tbody>
</table>

For another example:
SELECT CROSSJOIN(MEASURES.[%COUNT], gend.gender.members) ON 0, diagd.members ON 1 FROM patients

<table>
<thead>
<tr>
<th></th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None</td>
<td>4,228</td>
<td>4,197</td>
</tr>
<tr>
<td>2 asthma</td>
<td>406</td>
<td>297</td>
</tr>
<tr>
<td>3 CHD</td>
<td>134</td>
<td>189</td>
</tr>
<tr>
<td>4 diabetes</td>
<td>283</td>
<td>221</td>
</tr>
<tr>
<td>5 osteoporosis</td>
<td>180</td>
<td>20</td>
</tr>
</tbody>
</table>

**See Also**

- NONEMPTYCROSSJOIN
CURRENTMEMBER

Enables you to refer to a member programmatically within an iteration through the members of a hierarchy.

Returned Type

This function returns a member.

Syntax and Details

- `hierarchy_expression.CURRENTMEMBER`
- `dimension_expression.CURRENTMEMBER`

• `hierarchy_expression` is an expression that evaluates to a hierarchy.

• `dimension_expression` is a dimension name, included within square brackets if needed (see Identifiers). For example:

  ```
  [dimension_name]
  ```

  DeepSee interprets this as a reference to the first visible hierarchy within the given dimension.

You use this function in a context that iterates through a hierarchy. The CURRENTMEMBER function returns the given member, in that context.

In abstract, this function has the same purpose as does `$this` in Caché ObjectScript.

Note: The CURRENTMEMBER function is not supported in an MDX query that has more than three axes. It is expected that most queries have one or two axes, as shown in this book. It is also not supported for use with the MEASURES dimension. That is, `hierarchy_expression` or `dimension_expression` cannot be MEASURES.

Example

In the following example, cities are used as rows. The data shown in the column is the Principal Export property for each city, retrieved via the PROPERTIES function.

```
SELECT homed.CURRENTMEMBER.PROPERTIES("Principal Export") ON 0, homed.city.MEMBERS ON 1 FROM patients
```

```
1 Cedar Falls                          iron
2 Centerville                   video games
3 Cypress                            gravel
4 Elm Heights                       lettuce
5 Juniper wheat                     wheat
6 Magnolia                       bundt cake
7 Pine                            spaghetti
8 Redwood                           peaches
9 Spruce                                mud
```

The following variation uses the `%LABEL` function to provide a better caption for the data column:

```
SELECT %LABEL(homed.CURRENTMEMBER.PROPERTIES("Principal Export"),"Exports") ON 0, homed.city.MEMBERS ON 1 FROM patients
```

```
1 Cedar Falls                          iron
2 Centerville                   video games
3 Cypress                            gravel
4 Elm Heights                       lettuce
5 Juniper wheat                     wheat
6 Magnolia                       bundt cake
7 Pine                            spaghetti
8 Redwood                           peaches
9 Spruce                                mud
```
The following query shows both the Principal Export and Population properties of the City level:

```sql
SELECT %LABEL(homed.CURRENTMEMBER.PROPERTIES("Principal Export"), "Export"),
%LABEL(homed.CURRENTMEMBER.PROPERTIES("Population"), "Population")) ON 0,
homed.city.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th></th>
<th>Export</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cedar Falls 1</td>
<td>iron</td>
</tr>
<tr>
<td>2</td>
<td>Centerville</td>
<td>video games</td>
</tr>
<tr>
<td>3</td>
<td>Cypress</td>
<td>gravel</td>
</tr>
<tr>
<td>4</td>
<td>Elm Heights</td>
<td>lettuce</td>
</tr>
<tr>
<td>5</td>
<td>Juniper</td>
<td>wheat</td>
</tr>
<tr>
<td>6</td>
<td>Magnolia</td>
<td>bundt cake</td>
</tr>
<tr>
<td>7</td>
<td>Pine</td>
<td>spaghetti</td>
</tr>
<tr>
<td>8</td>
<td>Redwood</td>
<td>peaches</td>
</tr>
<tr>
<td>9</td>
<td>Spruce</td>
<td>mud</td>
</tr>
</tbody>
</table>
**DISTINCT**

Examines a set, removes duplicate elements, and returns a set of the remaining elements.

**Returned Type**

This function returns a set.

**Syntax and Details**

```
DISTINCT(set_expression)
```

- `set_expression` is an expression that evaluates to a set.

**Example**

For example, suppose that the query must return a specific city as reference, which is needed for comparison to the other cities. Consider the following query, which displays a reference city, followed by a set of cities with a given patient count:

```
WITH SET refcity AS '{homed.juniper}'
SELECT MEASURES.[%COUNT] ON 0,
{refcity,FILTER(homed.city.MEMBERS,MEASURES.[%COUNT]>1100)} ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Juniper</td>
<td>1,197</td>
</tr>
<tr>
<td>2 Cedar Falls</td>
<td>1,188</td>
</tr>
<tr>
<td>3 Centerville</td>
<td>1,155</td>
</tr>
<tr>
<td>4 Cypress</td>
<td>1,221</td>
</tr>
<tr>
<td>5 Elm Heights</td>
<td>1,266</td>
</tr>
<tr>
<td>6 Juniper</td>
<td>1,197</td>
</tr>
<tr>
<td>7 Magnolia</td>
<td>1,156</td>
</tr>
<tr>
<td>8 Pine</td>
<td>1,139</td>
</tr>
<tr>
<td>9 Redwood</td>
<td>1,144</td>
</tr>
<tr>
<td>10 Spruce</td>
<td>1,135</td>
</tr>
</tbody>
</table>

Compare to the following query, which removes the duplicate city:

```
WITH SET refcity AS '{homed.juniper}'
SELECT MEASURES.[%COUNT] ON 0,
DISTINCT({refcity,FILTER(homed.city.MEMBERS,MEASURES.[%COUNT]>1100)}) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Juniper</td>
<td>1,197</td>
</tr>
<tr>
<td>2 Cedar Falls</td>
<td>1,188</td>
</tr>
<tr>
<td>3 Centerville</td>
<td>1,155</td>
</tr>
<tr>
<td>4 Cypress</td>
<td>1,221</td>
</tr>
<tr>
<td>5 Elm Heights</td>
<td>1,266</td>
</tr>
<tr>
<td>6 Juniper</td>
<td>1,197</td>
</tr>
<tr>
<td>7 Magnolia</td>
<td>1,156</td>
</tr>
<tr>
<td>8 Pine</td>
<td>1,139</td>
</tr>
<tr>
<td>9 Redwood</td>
<td>1,144</td>
</tr>
<tr>
<td>9 Spruce</td>
<td>1,135</td>
</tr>
</tbody>
</table>
EXCEPT

Examines two sets and returns a set that consists of the elements of the first set, except for any elements that are also in the second set. This function optionally eliminates duplicates in that set.

Returned Type

This function returns a set.

Syntax and Details

| EXCEPT(set_expression1, set_expression2, ALL) |
| EXCEPT(set_expression1, set_expression2) |

- `set_expression1` and `set_expression2` are expressions that evaluate to sets.
- The optional keyword ALL, if included, specifies that all duplicates should be retained. By default, if the first set includes any duplicate elements, only the first of those is included.

If `set_expression2` includes elements that are not in `set_expression1`, those elements are ignored. The returned set includes only elements from `set_expression1`.

Example

Consider the following query which defines two named sets:

```
WITH SET set1 AS '{allerd.eggs,allerd.eggs,allerd.soy,allerd.wheat}'
SET set2 AS '{allerd.[diary products],allerd.pollen,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, EXCEPT(set1,set2) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 eggs</td>
</tr>
<tr>
<td>2 soy</td>
</tr>
</tbody>
</table>

This query shows the members of `set1` that are not also in `set2`. Notice that the member `allerd.eggs` is listed twice within `set1`, but is shown only once in the result.

In contrast, the following variation uses the ALL keyword:

```
WITH SET set1 AS '{allerd.eggs,allerd.eggs,allerd.soy,allerd.wheat}'
SET set2 AS '{allerd.[diary products],allerd.pollen,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, EXCEPT(set1,set2,ALL) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 eggs</td>
</tr>
<tr>
<td>2 eggs</td>
</tr>
<tr>
<td>3 soy</td>
</tr>
</tbody>
</table>

See Also

- INTERSECT
- UNION
**FILTER**

Examines a set and returns the subset in which the given expression is true for each element. The set order is unchanged.

**Returned Type**

This function returns a set.

**Syntax and Details**

```
FILTER(set_expression, logical_expression)
```

- `set_expression` is an expression that evaluates to a set.
- `logical_expression` is a logical expression, typically that examines a measure value or a property value.

Instead of `logical_expression`, you can use a measure search expression; see the example to see the behavior of `FILTER` in this case.

**Example**

For example, consider the following query, which returns only cities with more than 1150 patients.

```
SELECT MEASURES.[%COUNT] ON 0,
FILTER(homed.city.MEMBERS, MEASURES.[%COUNT]>1150) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,188</td>
<td>Cedar Falls</td>
</tr>
<tr>
<td>1,155</td>
<td>Centerville</td>
</tr>
<tr>
<td>1,221</td>
<td>Cypress</td>
</tr>
<tr>
<td>1,266</td>
<td>Elm Heights</td>
</tr>
<tr>
<td>1,197</td>
<td>Juniper</td>
</tr>
<tr>
<td>1,156</td>
<td>Magnolia</td>
</tr>
</tbody>
</table>

In comparison, consider the following query, which returns cities with any patient count:

```
SELECT MEASURES.[%COUNT] ON 0,
FILTER(homed.city.MEMBERS, MEASURES.[%COUNT]>=0) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,188</td>
<td>Cedar Falls</td>
</tr>
<tr>
<td>1,155</td>
<td>Centerville</td>
</tr>
<tr>
<td>1,221</td>
<td>Cypress</td>
</tr>
<tr>
<td>1,266</td>
<td>Elm Heights</td>
</tr>
<tr>
<td>1,197</td>
<td>Juniper</td>
</tr>
<tr>
<td>1,156</td>
<td>Magnolia</td>
</tr>
<tr>
<td>1,139</td>
<td>Pine</td>
</tr>
<tr>
<td>1,144</td>
<td>Redwood</td>
</tr>
<tr>
<td>1,135</td>
<td>Spruce</td>
</tr>
</tbody>
</table>

For another example, the following query uses a more complex filter expression:

```
SELECT MEASURES.[%COUNT] ON 0,
FILTER(diagd.members, (MEASURES.[%COUNT]>500 and MEASURES.[%COUNT]<1000)) ON 1 FROM patients
```

```
<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>746</td>
<td>asthma</td>
</tr>
<tr>
<td>555</td>
<td>diabetes</td>
</tr>
</tbody>
</table>
```

The next example uses a filter expression that evaluates a property:

```
SELECT homed.CURRENTMEMBER.PROPERTIES("Population") ON 0,
FILTER(homed.city.MEMBERS,homed.CURRENTMEMBER.PROPERTIES("Population")>20000) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>ZIP</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>90,000</td>
<td>Cedar Falls</td>
</tr>
<tr>
<td>49,000</td>
<td>Centerville</td>
</tr>
<tr>
<td>33,194</td>
<td>Elm Heights</td>
</tr>
<tr>
<td>29,192</td>
<td>Redwood</td>
</tr>
</tbody>
</table>
If you use `FILTER` with a `measure search expression`, the function returns only those members that are based on at least one fact that meets the given criteria. (A `measure search expression` is a DeepSee extension to MDX that considers the measure values in the fact table itself.)

```
SELECT {MEASURES.[%COUNT]} ON 0, FILTER(diagd.members, %SEARCH.&[{MEASURES].[age]<10}) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>8,425</td>
</tr>
<tr>
<td>asthma</td>
<td>703</td>
</tr>
</tbody>
</table>

This query shows the diagnoses that have at least one patient under ten years old. For other diagnoses such as diabetes and osteoporosis, there are no such young patients.

**See Also**

- `IIF`
FIRSTCHILD

Returns the first child of the given member.

Returned Type

This function returns a member.

Syntax and Details

```
member_expression.FIRSTCHILD
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

The function returns the first child of this member. To determine which child is first, the function considers the default order of the set of children.

Example

For example, consider the following hierarchy:

```
<table>
<thead>
<tr>
<th>Members of the ZIP level</th>
<th>Members of the City level</th>
</tr>
</thead>
<tbody>
<tr>
<td>32006</td>
<td>Juniper</td>
</tr>
<tr>
<td>32007</td>
<td>Spruce</td>
</tr>
<tr>
<td>34577</td>
<td>Redwood</td>
</tr>
<tr>
<td>36711</td>
<td>Cypress</td>
</tr>
<tr>
<td>38928</td>
<td>Magnolia</td>
</tr>
<tr>
<td></td>
<td>Pine</td>
</tr>
<tr>
<td></td>
<td>Centerville</td>
</tr>
<tr>
<td></td>
<td>Cedar Falls</td>
</tr>
<tr>
<td></td>
<td>Elm Heights</td>
</tr>
</tbody>
</table>
```

Now consider the following query:

```
SELECT MEASURES.[%COUNT] ON 0, homed.zip.[34577].FIRSTCHILD ON 1 FROM patients
```

```
Patient Count
Cypress                              1,089
```

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1960.FIRSTCHILD ON 1 FROM patients
```

```
Patient Count
Jan-1960                                  5
```

See Also

- CHILDREN
- COUSIN
- FIRSTSIBLING
- LASTCHILD
• LASTSIBLING
• PARENT
• SIBLINGS
**FIRSTSIBLING**

Returns the first sibling of the given member.

**Returned Type**

This function returns a member.

**Syntax and Details**

```
member_expression.FIRSTSIBLING
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

This function examines all the children of the parent of the given member, and returns the first member of that set (considering the default order of that set).

This function can return the same member that you specify as an argument (if that member is the first sibling).

**Example**

For example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.[Mar 2003].FIRSTSIBLING ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Jan-2003</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

**See Also**

- CHILDREN
- COUSIN
- FIRSTCHILD
- LASTCHILD
- LASTSIBLING
- PARENT
- SIBLINGS
HEAD

Returns a subset from the start of a set, using the current order of the set.

Returned Type

This function returns a set.

Syntax and Details

HEAD(set_expression, optional_integer_expression)

- set_expression is an expression that evaluates to a set.
- optional_integer_expression is an integer literal.

The default value for this argument is 1.

The function uses this argument to determine the number of elements to return in the subset.

This function returns a set that consists of the specified number of elements from the start of the given set (considering the current order of the set). If integer_expression is less than 1, the function returns the empty set. If integer_expression is greater than the number of elements of the set, the function returns the original set.

The elements of the subset are returned in the same order specified by the original set.

Example

SELECT MEASURES.[%COUNT] ON 0, HEAD(birthd.decade.MEMBERS, 3) ON 1
FROM patients

<table>
<thead>
<tr>
<th></th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>71</td>
</tr>
<tr>
<td>2 1920s</td>
<td>223</td>
</tr>
<tr>
<td>3 1930s</td>
<td>572</td>
</tr>
</tbody>
</table>

See Also

- TAIL
HIERARCHISE

Synonym for HIERARCHIZE.

See Also

- HIERARCHIZE
HIERARCHIZE

Given a set, returns a set that is in hierarchical order (the order specified by the hierarchy).

**Returned Type**

This function returns a set of members.

**Syntax and Details**

```
HIERARCHIZE(set_expression)
```

Or:

```
HIERARCHIZE(set_expression, POST)
```

Where:

- `set_expression` is an expression that evaluates to a set of members.
- If `POST` is specified, child members precede their parents. This is called *post-natural order*.

If the set members are in different hierarchies, the order of the hierarchies themselves is indeterminate. That is, if some members are from hierarchy A and the others are from hierarchy B, the A members will be listed consecutively in hierarchical order and the B members will be listed consecutively in hierarchical order, but there is no rule governing whether the A members or the B members are first overall.

**Example**

Within a hierarchy, the hierarchical order is determined as follows:

- The All member of the dimension, if present, is first.
- The next member is the first member of the highest level of that hierarchy.
- The next member is the first child of that member.

And so on. For example, consider the following hierarchy:

<table>
<thead>
<tr>
<th>Members of the ZIP level</th>
<th>Members of the City level</th>
</tr>
</thead>
<tbody>
<tr>
<td>32006</td>
<td>Juniper</td>
</tr>
<tr>
<td>32007</td>
<td>Spruce</td>
</tr>
<tr>
<td>34577</td>
<td>Redwood</td>
</tr>
<tr>
<td>36711</td>
<td>Cypress</td>
</tr>
<tr>
<td>38928</td>
<td>Magnolia</td>
</tr>
<tr>
<td></td>
<td>Pine</td>
</tr>
<tr>
<td></td>
<td>Centerville</td>
</tr>
<tr>
<td></td>
<td>Cedar Falls</td>
</tr>
<tr>
<td></td>
<td>Elm Heights</td>
</tr>
</tbody>
</table>

To see the overall hierarchical order of these members, we use the following query, which uses a set consisting of all members of dimension to which these members belong:
The following example creates a set of several members of the Home City and Home ZIP levels and then uses the HIERARCHIZE function to place these members into hierarchical order:

```
SELECT MEASURES.[%COUNT] ON 0, HIERARCHIZE({homed.36711, homed.38928, homed.[elm heights], homed.Spruce}) ON 1 FROM patients
```

```
<table>
<thead>
<tr>
<th></th>
<th>City</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36711</td>
<td>1,069</td>
</tr>
<tr>
<td>2</td>
<td>Spruce</td>
<td>1,117</td>
</tr>
<tr>
<td>3</td>
<td>38928</td>
<td>2,149</td>
</tr>
<tr>
<td>4</td>
<td>Elm Heights</td>
<td>1,104</td>
</tr>
</tbody>
</table>
```

In contrast, the next example uses the POST keyword:

```
SELECT MEASURES.[%COUNT] ON 0, HIERARCHIZE({homed.36711, homed.38928, homed.[elm heights], homed.Spruce}, POST) ON 1 FROM patients
```

```
<table>
<thead>
<tr>
<th></th>
<th>City</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36711</td>
<td>1,069</td>
</tr>
<tr>
<td>2</td>
<td>Spruce</td>
<td>1,117</td>
</tr>
<tr>
<td>3</td>
<td>Elm Heights</td>
<td>1,104</td>
</tr>
<tr>
<td>4</td>
<td>38928</td>
<td>2,149</td>
</tr>
</tbody>
</table>
```

**See Also**

- ORDER
IIF

Returns one of two values, depending on the value of a given logical expression.

**Returned Type**

This function returns a **number** or a **string**, depending on the arguments used and the value of the logical expression.

**Syntax and Details**

IIF(check_expression, expression1, expression2)

- **expression1** and **expression2** are numeric or **string** expressions. They do not have to be the same type.
  
  DeepSee MDX does not support other types of arguments.

  To compare to a null value, use the **ISNULL** function instead.

- **check_expression** is a **logical expression**, typically that compares a measure or a property to a constant.

  If **check_expression** is true, the function returns the value given by **expression1**. Otherwise, it returns the value given by **expression2**.

**Example**

For example:

```sql
SELECT IIF(MEASURES.[%COUNT]<500, "fewer than 500", "500 or more") ON 0, diagd.MEMBERS ON 1 FROM patients
```

```
IIF
1 None                          500 or more
2 asthma                        500 or more
3 CHD                        fewer than 500
4 diabetes                      500 or more
5 osteoporosis               fewer than 500
```

As a variation, the following query uses **%LABEL** to apply a suitable caption to the data column:

```sql
SELECT %LABEL(IIF(MEASURES.[%COUNT]<500, "fewer than 500", "500 or more"),"Patient Count") ON 0, diagd.MEMBERS ON 1 FROM patients
```

```
Patient Count
1 None                          500 or more
2 asthma                        500 or more
3 CHD                        fewer than 500
4 diabetes                      500 or more
5 osteoporosis               fewer than 500
```

For another example, the following query uses the value of a property:

```sql
SELECT %LABEL(IIF(homed.h1.CURRENTMEMBER.PROPERTIES("Population")>20000,"big","small"),"Town Size") ON 0, homed.city.MEMBERS ON 1 FROM patients
```

```
Town Size
1 Cedar Falls                           big
2 Centerville                           big
3 Cypress                                small
4 Elm Heights                           big
5 Juniper                               small
6 Magnolia                              small
7 Pine                                  small
8 Redwood                              big
9 Spruce                                small
```
WITH MEMBER MEASURES.iif AS 'IIF(homed.CURRENTMEMBER.PROPERTIES("name")="Pine", " ", MEASURES.[%COUNT])'
SELECT MEASURES.iif ON 0, homed.city.MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>City</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cedar Falls</td>
<td>1,120</td>
</tr>
<tr>
<td>Centerville</td>
<td>1,106</td>
</tr>
<tr>
<td>Cypress</td>
<td>1,139</td>
</tr>
<tr>
<td>Elm Heights</td>
<td>1,078</td>
</tr>
<tr>
<td>Juniper</td>
<td>1,109</td>
</tr>
<tr>
<td>Magnolia</td>
<td>1,122</td>
</tr>
<tr>
<td>Pine</td>
<td>1,122</td>
</tr>
<tr>
<td>Redwood</td>
<td>1,128</td>
</tr>
<tr>
<td>Spruce</td>
<td>1,108</td>
</tr>
</tbody>
</table>

See Also

- ISNULL
- FILTER
INTERSECT

Returns a set that consists of the elements that occur in both of the two given sets, optionally eliminating duplicates in that set.

Returned Type

This function returns a set.

Syntax and Details

\[
\text{INTERSECT(set_expression1, set_expression2, ALL)}
\]

Or:

\[
\text{INTERSECT(set_expression1, set_expression2)}
\]

- \text{set_expression1} and \text{set_expression2} are expressions that evaluate to sets.
- The optional keyword ALL, if included, specifies that all duplicates in the second set should be retained. By default, if the returned set includes any duplicate elements, only the first of those is included. This keyword does not affect duplicates in the first set.

Example

Consider the following query which defines two named sets:

```mdx
WITH SET set1 AS '{allerd.eggs,allerd.soy,allerd.wheat,allerd.wheat}'
SET set2 AS '{allerd.[dairy products],allerd.pollen,allerd.soy,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, INTERSECT(set1,set2) ON 1 FROM patients
```

Patient Count
- soy: 462
- wheat: 479

In contrast, consider the following variation, which uses the ALL keyword:

```mdx
WITH SET set1 AS '{allerd.eggs,allerd.soy,allerd.wheat,allerd.wheat}'
SET set2 AS '{allerd.[dairy products],allerd.pollen,allerd.soy,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, INTERSECT(set1,set2,ALL) ON 1 FROM patients
```

Patient Count
- soy: 462
- wheat: 479
- wheat: 479

Finally, you can of course use more interesting sets as arguments. For example:

```mdx
WITH SET set1 AS 'TOPCOUNT(homed.city.members,5,MEASURES.[avg allergy count])'
SET set2 AS 'TOPCOUNT(homed.city.members,5,MEASURES.[avg age])'
SELECT MEASURES.[%COUNT] ON 0, INTERSECT(set1,set2) ON 1 FROM patients
```

Patient Count
- Centerville: 1,155
- Magnolia: 1,156

See Also

- EXCEPT
- UNION
ISNULL

Evaluates a scalar MDX expression and returns either its value or an alternative value (if the value of the expression is null). This function is a DeepSee extension to MDX.

Returned Type

This function returns an expression of the same type that you use with the function.

Syntax and Details

```
ISNULL(MDX_expression,value_if_null)
```

Where:

- `MDX_expression` is a scalar MDX value (a numeric, string, or logical expression).
- `value_if_null` is the value to return if `MDX_expression` evaluates to null.

Example

The following shows a simple example:

```
SELECT ISNULL(MEASURES.[%COUNT],"None") ON 0, birthd.decade.MEMBERS ON 1 FROM patients where diagd.chd
```

```
<table>
<thead>
<tr>
<th>ISNULL</th>
<th>birthd.decade.MEMBERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1910s</td>
</tr>
<tr>
<td>1</td>
<td>1920s</td>
</tr>
<tr>
<td>12</td>
<td>1930s</td>
</tr>
<tr>
<td>3</td>
<td>1940s</td>
</tr>
<tr>
<td>10</td>
<td>1950s</td>
</tr>
<tr>
<td>3</td>
<td>1960s</td>
</tr>
<tr>
<td>None</td>
<td>1970s</td>
</tr>
<tr>
<td>1</td>
<td>1980s</td>
</tr>
<tr>
<td>None</td>
<td>1990s</td>
</tr>
<tr>
<td>None</td>
<td>2000s</td>
</tr>
<tr>
<td>None</td>
<td>2010s</td>
</tr>
</tbody>
</table>
```

See Also

- IIF
- %LABEL
Given a level member and a nonnegative integer, this function counts backward in the level and returns a previous member. The details are different for time dimensions and data dimensions.

**Returned Type**

This function returns a member.

**Syntax and Details**

```
member_expression.LAG(optional_integer_expression)
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

- `optional_integer_expression` is a nonnegative integer literal.
  
  The default value for this argument is 0; in this case, the function returns the member given by `member_expression`.

  If `integer_expression` is 1, this function is equivalent to the PREVMEMBER function.

This function examines the members of the level to which the given member belongs, counts backward from the current member (using `integer_expression`), and returns the member at that position. For time dimensions, this function ignores any parent level. For data dimensions, this function considers the parent level; it counts backward from the current member within the given parent member. (Note that the terms *time dimension* and *data dimension* refer specifically to the dimension type as defined in the cube. See *Defining DeepSee Models*.)

**Example**

The first examples use a time dimension. Consider the following query, shown for reference:

```
SELECT MEASURES.[%COUNT] ON 0, 
{birthd.1948,birthd.1949,birthd.1950,birthd.1951,birthd.1952} ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1948</td>
</tr>
<tr>
<td>2 1949</td>
</tr>
<tr>
<td>3 1950</td>
</tr>
<tr>
<td>4 1951</td>
</tr>
<tr>
<td>5 1952</td>
</tr>
</tbody>
</table>

The following query uses LAG:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.LAG(1) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
</tr>
</tbody>
</table>

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.LAG(2) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
</tr>
</tbody>
</table>

In this sample, the year level is the child of the decade level, which means that the members 1949 and 1950 belong to different parents. As you can see, the LAG function ignores the parent level when you use the function with a time dimension.

The second examples use a data dimension (the HomeD dimension). To see the hierarchy in this dimension, see the examples in the FIRSTCHILD function. The following query uses LAG with this dimension:
SELECT MEASURES.[%COUNT] ON 0, homed.city.Magnolia.LAG(1) ON 1 FROM patients

Cypress

Patient Count
104

Because this is a data dimension, this query retrieves the previous member of the city level within the parent ZIP code. Within this ZIP code, Cypress is the first city, so the following query returns no results:

SELECT MEASURES.[%COUNT] ON 0, homed.city.Cypress.LAG(1) ON 1 FROM patients

Patient Count

See Also

- LEAD
- NEXTMEMBER
- PREVMEMBER
LASTCHILD

Returns the last child of the given member.

Returned Type
This function returns a member.

Syntax and Details

```
member_expression.LASTCHILD
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

The function returns the last child of this member. To determine which child is last, the function considers the default order of the set of children.

Example

For example:

```
SELECT MEASURES.[%COUNT] ON 0, homed.zip.[34577].LASTCHILD ON 1 FROM patients
```

```
<table>
<thead>
<tr>
<th>Pine</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,100</td>
</tr>
</tbody>
</table>
```

To see a picture of the hierarchy used in this example, see the FIRSTCHILD function.

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.[1950].LASTCHILD ON 1 FROM patients
```

```
<table>
<thead>
<tr>
<th>Dec-1950</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
```

See Also

- CHILDREN
- COUSIN
- FIRSTSIBLING
- FIRSTCHILD
- LASTSIBLING
- PARENT
- SIBLINGS
LASTSIBLING

Returns the last sibling of the given member.

Returned Type

This function returns a member.

Syntax and Details

\[
\text{member_expression}.\text{LASTSIBLING}
\]

Where:

- \textit{member_expression} is an expression that returns a member.

This expression cannot refer to a measure.

This function examines all the children of the parent of the given member, and returns the last member of that set (considering the default order of that set).

This function can return the same member that you use as its argument (if that is the last sibling).

Example

For example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.[Mar 2003].LASTSIBLING ON 1 FROM patients
```

Patient Count
Dec-2003 17

See Also

- CHILDREN
- COUSIN
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- PARENT
- SIBLINGS
LEAD

Given a level member and a nonnegative integer, this function counts forward in the level and returns a later member. The details are different for time dimensions and data dimensions.

Returned Type

This function returns a member.

Syntax and Details

```
member_expression.LEAD(optional_integer_expression)
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

- `optional_integer_expression` is a nonnegative integer literal.
  
  The default value for this argument is 0; in this case, the function returns the member given by `member_expression`.
  
  If `integer_expression` is 1, this function is equivalent to the `NEXTMEMBER` function.

This function examines the members of the level to which the given member belongs, counts forward from the current member (using `integer_expression`), and returns the member at that position. For time dimensions, this function ignores any parent level. For data dimensions, this function considers the parent level; it counts forward from the current member within the given parent member. (Note that the terms `time dimension` and `data dimension` refer specifically to the dimension type as defined in the cube. See Defining DeepSee Models.)

Example

The first examples use a time dimension. Consider the following query, shown for reference:

```
SELECT MEASURES.[%COUNT] ON 0, 
{birthd.1948,birthd.1949,birthd.1950,birthd.1951,birthd.1952} ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1948</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>1949</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1950</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>1951</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>1952</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

The following query uses LEAD:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.LEAD(1) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.LEAD(3) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

In this sample, the year level is the child of the decade level, which means that the members 1948 and 1951 belong to different parents. As you can see, the LEAD function ignores the parent level when you use the function with a time dimension.
The second examples use a data dimension (the HomeD dimension). To see the hierarchy in this dimension, see the examples in the FIRSTCHILD function. The following query uses LEAD with this dimension:

```mdx
SELECT MEASURES.[%COUNT] ON 0, homed.city.Magnolia.LEAD(1) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Pine</th>
<th>114</th>
</tr>
</thead>
</table>

Because this is a data dimension, this query retrieves the next member of the city level within the parent ZIP code. Within this ZIP code, Pine is the last city, so the following query returns no results:

```mdx
SELECT MEASURES.[%COUNT] ON 0, homed.city.Pine.LEAD(1) ON 1 FROM patients
```

| Patient Count | *   |

**See Also**

- LAG
- NEXTMEMBER
- PREVMEMBER
LOG

Returns the base-ten logarithm of the given numeric value.

Returned Type
This function returns a number.

Syntax and Details
LOG(numeric_expression)

Where:
• numeric_expression is an expression that evaluates to a number.
  Typically, this expression has the form [MEASURES].[measure_name]

Example
The first example shows the base-ten logarithm for the %COUNT measure:
SELECT LOG(MEASURES.[%COUNT]) ON 0 FROM patients

LOG 4

The next example uses the %LABEL function to apply a more detailed caption:
SELECT %LABEL(LOG(MEASURES.[%COUNT]),"LOG PAT CNT") ON 0 FROM patients

LOG PAT CNT 4

See Also
• POWER
• SQRT
LOOKUP

Enables you to perform string replacements within a query. This function is a DeepSee extension to MDX (added in 2012.2).

Returned Type

This function returns a string.

Syntax and Details

```
LOOKUP(term_list_name, lookup_value,default,alternative_field)
```

Where the arguments are string expressions as follows:

- `term_list_name` evaluates to the name of a term list.
- `lookup_value` evaluates to the string to look up in the term list.
- `default`, which is optional, evaluates to the value to return if `lookup_value` is not found in the term list.
- `alternative_field`, which is optional, is the name of the field to return. The default is "value".

This argument is not case-sensitive.

This function examines the given term list, finds the term whose "key" field equals the string given by `lookup_value` and then returns the value contained in the field identified by `alternative_field`.

All term lists have at least two fields: "key" and "value". You can add additional fields. For information, see “Defining Term Lists” in the DeepSee Implementation Guide.

DeepSee notes the most recent modification time stamp of any term list and invalidates any query cache that uses an outdated term list.

Example

For example, suppose that for HoleFoods, we have a term list named Teams that is defined as follows:

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta</td>
<td>Braves</td>
</tr>
<tr>
<td>Boston</td>
<td>Red Sox</td>
</tr>
<tr>
<td>New York</td>
<td>Yankees</td>
</tr>
</tbody>
</table>

Here is a simple query that uses this term list:

```
SELECT Lookup("Teams",Outlet.Boston.Properties("NAME")) ON ROWS FROM HOLEFOODS
```

Here is a more complex query:
SELECT Lookup("Teams",Outlet.CURRENTMEMBER.Properties("NAME"),"No Team") ON 0,
outlet.city.MEMBERS ON 1 FROM HOLEFOODS

<table>
<thead>
<tr>
<th>Lookup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>2</td>
<td>Antwerp</td>
</tr>
<tr>
<td>3</td>
<td>Atlanta</td>
</tr>
<tr>
<td>4</td>
<td>Bangalore</td>
</tr>
<tr>
<td>5</td>
<td>Barcelona</td>
</tr>
<tr>
<td>6</td>
<td>Beijing</td>
</tr>
<tr>
<td>7</td>
<td>Berlin</td>
</tr>
<tr>
<td>8</td>
<td>Boston</td>
</tr>
<tr>
<td>9</td>
<td>Brasilia</td>
</tr>
</tbody>
</table>

... 

See Also

- %LOOKUP
MAX

Returns the maximum value of a given expression (or of the current measure), across all elements of a set.

Returned Type

This function returns a number.

Syntax and Details

\[
\text{MAX(set_expression, optional\_numeric\_expression)}
\]

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members or tuples.
- `optional\_numeric\_expression` is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form `[MEASURES].[measure\_name]`

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses `%COUNT`, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the largest of those values.

Example

First, the following query shows values of three measures for the members of the `aged.decade` level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1910s</td>
<td>80</td>
<td>5,359</td>
</tr>
<tr>
<td>1920s</td>
<td>227</td>
<td>12,910</td>
</tr>
<tr>
<td>1930s</td>
<td>567</td>
<td>33,211</td>
</tr>
<tr>
<td>1940s</td>
<td>724</td>
<td>38,420</td>
</tr>
<tr>
<td>1950s</td>
<td>1,079</td>
<td>46,883</td>
</tr>
<tr>
<td>1960s</td>
<td>1,475</td>
<td>57,814</td>
</tr>
<tr>
<td>1970s</td>
<td>1,549</td>
<td>49,794</td>
</tr>
<tr>
<td>1980s</td>
<td>1,333</td>
<td>35,919</td>
</tr>
<tr>
<td>1990s</td>
<td>1,426</td>
<td>29,219</td>
</tr>
<tr>
<td>2000s</td>
<td>1,306</td>
<td>20,072</td>
</tr>
<tr>
<td>2010s</td>
<td>134</td>
<td>1,346</td>
</tr>
</tbody>
</table>

Next, the following query shows the maximum values for these measures for the members of this level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
MAX(birthd.decade.MEMBERS) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>1,549</td>
<td>57,814</td>
</tr>
</tbody>
</table>

Here, each value is the maximum of the values in a column in the preceding query. For example, the `Patient Count` value is the maximum of the `Patient Count` values in the preceding query.

Notice that these maximum values do not belong to the same member. For example, the decade with the highest patient count is not the same as the decade with the highest average test score.

For another example, we use the second argument for MAX:

```
SELECT MAX(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

MAX
1,549

For additional, similar examples, see AVG.
See Also

- AGGREGATE
- AVG
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
MEDIAN

Returns the value closest to the median value, for a given expression (or of the current measure), across all elements of a set that have a non-null value for that expression.

**Returned Type**

This function returns a number.

**Syntax and Details**

MEDIAN(set_expression, optional_numeric_expression)

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members or tuples.
- `optional_numeric_expression` is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form `[MEASURES].[measure_name]`

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The function evaluates the numeric value for each element of the set, ignores any set elements for which this value is null, and finds the value that is closest to the median for the remaining elements.

**Example**

First, the following query shows values of three measures for the members of the aged.decade level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>5,359</td>
<td>75.17</td>
</tr>
<tr>
<td>227</td>
<td>12,910</td>
<td>74.20</td>
</tr>
<tr>
<td>567</td>
<td>33,211</td>
<td>74.67</td>
</tr>
<tr>
<td>724</td>
<td>38,420</td>
<td>73.39</td>
</tr>
<tr>
<td>1,079</td>
<td>46,883</td>
<td>73.72</td>
</tr>
<tr>
<td>1,475</td>
<td>57,814</td>
<td>74.16</td>
</tr>
<tr>
<td>1,549</td>
<td>49,794</td>
<td>74.35</td>
</tr>
<tr>
<td>1,333</td>
<td>29,219</td>
<td>74.95</td>
</tr>
<tr>
<td>1,426</td>
<td>20,072</td>
<td>74.13</td>
</tr>
<tr>
<td>1,406</td>
<td>1,346</td>
<td>73.55</td>
</tr>
</tbody>
</table>

Next, the following query shows the average values for these measures for the members of this level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
MEDIAN(birthd.decade.MEMBERS) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIAN</td>
<td>1,079</td>
<td>74.20</td>
</tr>
</tbody>
</table>

Here, each value is the median of the values in a column in the preceding query. For example, the Patient Count value is the median value of the Patient Count values in the preceding query.

For another example, we use the second argument for MEDIAN:

```
SELECT MEDIAN(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

For additional, similar examples, see AVG.
See Also

- AGGREGATE
- AVG
- MAX
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
MEMBERS

Returns a set of all members of the given level or hierarchy, not including any calculated members.

Returned Type
This function returns a set of members.

Syntax and Details

```
level_expression.MEMBERS
```

Or:

```
hierarchy_expression.MEMBERS
```

Or:

```
dimension_expression.MEMBERS
```

Where:

- `level_expression` is an expression that returns a level. For example:

  `[dimension_name].[hierarchy_name].[level_name]`

- `hierarchy_expression` is an expression that returns a hierarchy. For example:

  `[dimension_name].[hierarchy_name]`

- `dimension_expression` is a dimension name, included within square brackets if needed (see Identifiers). For example:

  `[dimension_name]`

  DeepSee interprets this as a reference to the first visible hierarchy within the given dimension.

Given a level name, this function returns a set that consists of the members of that level, not including any calculated members. The members are in the default order specified in the level definition in the cube. This default order is as follows:

- For non-date levels, members are sorted in increasing order alphabetically by name, unless the cube specifies a different sort order. DeepSee provides flexible options for the default sort order of each level.

- For date levels, members are sorted chronologically.

Given a hierarchy name, this function returns a set that consists of the members of all levels in that hierarchy, including the All member, if defined. The members are returned in hierarchical order. For information on hierarchical order, see the HIERARCHIZE function.

Given a dimension name, this function returns a set that consists of the members of all levels of the first visible hierarchy in this dimension.

Example

The following query displays all members of the Home Zip dimension as rows:
SELECT MEASURES.[%COUNT] ON 0, homed.zip.MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,272</td>
</tr>
<tr>
<td>2</td>
<td>1,111</td>
</tr>
<tr>
<td>3</td>
<td>3,399</td>
</tr>
<tr>
<td>4</td>
<td>1,069</td>
</tr>
<tr>
<td>5</td>
<td>2,149</td>
</tr>
</tbody>
</table>

The following query displays all members of all levels in the homed.h1 hierarchy as rows:

<table>
<thead>
<tr>
<th>Member</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 32006</td>
<td>2,272</td>
</tr>
<tr>
<td>2 Juniper</td>
<td>1,155</td>
</tr>
<tr>
<td>3 Spruce</td>
<td>1,117</td>
</tr>
<tr>
<td>4 32007</td>
<td>1,111</td>
</tr>
<tr>
<td>5 Redwood</td>
<td>1,111</td>
</tr>
<tr>
<td>6 34577</td>
<td>3,399</td>
</tr>
<tr>
<td>7 Cypress</td>
<td>1,150</td>
</tr>
<tr>
<td>8 Magnolia</td>
<td>1,111</td>
</tr>
<tr>
<td>9 Pine</td>
<td>1,138</td>
</tr>
<tr>
<td>10 36711</td>
<td>1,069</td>
</tr>
<tr>
<td>11 Centerville</td>
<td>1,069</td>
</tr>
<tr>
<td>12 38928</td>
<td>2,149</td>
</tr>
<tr>
<td>13 Cedar Falls</td>
<td>1,045</td>
</tr>
<tr>
<td>14 Elm Heights</td>
<td>1,104</td>
</tr>
</tbody>
</table>

See Also

- ALLMEMBERS
Returns the minimum non-null value of a given expression (or of the current measure), across all elements of a set.

**Returned Type**
This function returns a number.

**Syntax and Details**

\[
\text{MIN}(\text{set_expression}, \text{optional_numeric_expression})
\]

Where:
- **set_expression** is an expression that evaluates to a set, typically a set of members or tuples.
- **optional_numeric_expression** is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form \([\text{MEASURES}].[\text{measure_name}]\)

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the smallest of those values, ignoring any null values.

**Example**

First, the following query shows values of three measures for the members of the aged.decade level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
       birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>80</td>
<td>5,359</td>
</tr>
<tr>
<td>2 1920s</td>
<td>227</td>
<td>12,910</td>
</tr>
<tr>
<td>3 1930s</td>
<td>567</td>
<td>33,211</td>
</tr>
<tr>
<td>4 1940s</td>
<td>724</td>
<td>38,420</td>
</tr>
<tr>
<td>5 1950s</td>
<td>1,079</td>
<td>46,883</td>
</tr>
<tr>
<td>6 1960s</td>
<td>1,475</td>
<td>57,814</td>
</tr>
<tr>
<td>7 1970s</td>
<td>1,549</td>
<td>49,794</td>
</tr>
<tr>
<td>8 1980s</td>
<td>1,333</td>
<td>35,919</td>
</tr>
<tr>
<td>9 1990s</td>
<td>1,426</td>
<td>29,219</td>
</tr>
<tr>
<td>10 2000s</td>
<td>1,406</td>
<td>20,072</td>
</tr>
<tr>
<td>11 2010s</td>
<td>134</td>
<td>1,346</td>
</tr>
</tbody>
</table>

Next, the following query shows the minimum values for these measures for the members of this level:

```
SELECT MIN(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td>80</td>
<td>1,346</td>
</tr>
</tbody>
</table>

Here, each value is the minimum of the values in a column in the preceding query. For example, the Patient Count value is the minimum of the Patient Count values in the preceding query.

Notice that these minimum values do not belong to the same member. For example, the decade with the lowest patient count is not the same as the decade with the highest lowest test score.

For another example, we use the second argument for MIN:

```
SELECT MIN(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

| MIN | 80 |

DeepSee MDX Reference
For additional, similar examples, see AVG.

**See Also**

- AGGREGATE
- AVG
- MAX
- MEDIAN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
**NEXTMEMBER**

Returns the next member of the level to which the given member belongs. The details are different for time dimensions and data dimensions.

**Returned Type**

This function returns a member.

**Syntax and Details**

```
member_expression.NEXTMEMBER
```

Where:

- `member_expression` is an expression that returns a member.

This expression cannot refer to a measure.

This function examines the members of the level to which the given member belongs, and returns the next member of that set (considering the default order of that set). For data dimensions, this function considers the parent level; it looks for the next member within the given parent member. (Note that the terms time dimension and data dimension refer specifically to the dimension type as defined in the cube. See *Defining DeepSee Models*.)

The NEXTMEMBER function is equivalent to LEAD(1).

**Example**

The first examples use a time dimension. Consider the following query, shown for reference:

```
SELECT MEASURES.[%COUNT] ON 0,
(birthd.1948,birthd.1949,birthd.1950,birthd.1951,birthd.1952) ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1948</td>
</tr>
<tr>
<td>2 1949</td>
</tr>
<tr>
<td>3 1950</td>
</tr>
<tr>
<td>4 1951</td>
</tr>
<tr>
<td>5 1952</td>
</tr>
</tbody>
</table>

10
4
12
8
6

The following query uses NEXTMEMBER:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1948.NEXTMEMBER ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
</tr>
</tbody>
</table>

4

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1949.NEXTMEMBER ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
</tr>
</tbody>
</table>

12

In this sample, the year level is the child of the decade level, which means that the members 1949 and 1950 belong to different parents. As you can see, the NEXTMEMBER function ignores the parent level when you use the function with a time dimension.

The second examples use a data dimension (the HomeD dimension). To see the hierarchy in this dimension, see the examples in the FIRSTCHILD function. The following query uses NEXTMEMBER with this dimension:
SELECT MEASURES.[%COUNT] ON 0, homed.city.Magnolia.NEXTMEMBER ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine</td>
</tr>
<tr>
<td>114</td>
</tr>
</tbody>
</table>

Because this is a data dimension, this query retrieves the next member of the city level within the parent ZIP code. Within this ZIP code, Pine is the last city, so the following query returns no results:

SELECT MEASURES.[%COUNT] ON 0, homed.city.Pine.NEXTMEMBER ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

See Also

- LAG
- LEAD
- PREVMEMBER
NONEMPTYCROSSJOIN

Returns a set that consists of the cross-product of the given sets, excluding any tuples that are null.

**Returned Type**

This function returns a set of tuples.

**Syntax and Details**

```mdx
NONEMPTYCROSSJOIN(set_expression1, set_expression2, ..., Count)
```

Where:

- `set_expression1` and `set_expression2` are expressions that evaluate to sets of members.

The function identifies all the members of each set and then generates a set of tuples that combine each member of the first set with each member of the second set. The returned set does not include empty tuples.

**Note:** If you use this function in a query that refers to a compound cube, it returns the same results as CROSSJOIN. This is necessary to ensure that the subqueries have the same number of rows and can be combined. In this case, to exclude empty tuples, precede the function with the keyword phrase NON EMPTY.

**See Also**

- CROSSJOIN
OPENINGPERIOD

Returns the first descendent member of the given level, at the same level as the given member. This function is intended primarily for use with time levels.

**Returned Type**

This function returns a member.

**Syntax and Details**

OPENINGPERIOD(ancestor_level,member_expression)

Where:

- **ancestor_level** is an expression that returns a level. For example:
  
  `[dimension_name].[hierarchy_name].[level_name]`

  This level must be the parent level of **member_expression** or an ancestor of that member.

- **member_expression** is an expression that returns a member.

  This expression cannot refer to a measure.

Given a level and a member, this function returns the first member that is a descendent of the given level and that is at the same level as member.

**Example**

The following query displays the opening quarter for the year that includes Q3 2003:

```
SELECT MEASURES.[%COUNT] ON 0, OPENINGPERIOD(birthd.year,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 2003</td>
</tr>
<tr>
<td>35</td>
</tr>
</tbody>
</table>

In contrast, the following query displays the opening quarter for the **decade** that includes Q3 2003:

```
SELECT MEASURES.[%COUNT] ON 0, OPENINGPERIOD(birthd.decade,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 2000</td>
</tr>
<tr>
<td>33</td>
</tr>
</tbody>
</table>

**See Also**

- ANCESTOR
- CLOSINGPERIOD
- COUSIN
- PERIODSTODATE
ORDER

Returns a set that is ordered as specified.

Returned Type

This function returns a set.

Syntax and Details

ORDER(set_expression, ordering_expression, optional_keyword)

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members.
- `ordering_expression` is a numeric expression or a string expression that determines the order of the set elements.
  
  For numeric values, typically `ordering_expression` is `[MEASURES].[measure_name]`
  
  The function evaluates this expression for each element of the set and sorts the elements of the set according to this value.

- `optional_keyword` controls how MDX handles any hierarchies in the set. Use one of the following keywords:
  
  - `ASC` — Use this to sort in ascending order (using the value returned by `ordering_expression`), while preserving the hierarchy. For information on hierarchical order, see HIERARCHIZE.
    
    If you omit the keyword, the function sorts in this way.
  
  - `DESC` — Use this to sort in descending order (using the value returned by `ordering_expression`), while preserving the hierarchy.
  
  - `BASC` — Use this to break the hierarchy and sort all members in ascending order (using the value returned by `ordering_expression`).
  
  - `BDESC` — Use this to break the hierarchy and sort all members in descending order (using the value returned by `ordering_expression`).

Example

For example, the following query sorts cities in descending order by average test score, respecting the ZIP codes to which the cities belong:

```
SELECT MEASURES.[avg test score] ON 0,
ORDER(homed.city.MEMBERS, MEASURES.[avg test score], DESC) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2 Magnolia</td>
</tr>
<tr>
<td>3 Cypress</td>
</tr>
<tr>
<td>4 Centerville</td>
</tr>
<tr>
<td>5 Cedar Falls</td>
</tr>
<tr>
<td>6 Elm Heights</td>
</tr>
<tr>
<td>7 Juniper</td>
</tr>
<tr>
<td>8 Spruce</td>
</tr>
<tr>
<td>9 Redwood</td>
</tr>
</tbody>
</table>

To see a picture of the hierarchy used in this example, see the FIRSTCHILD function.

In contrast, the following example uses the BDESC keyword and disregards the hierarchy:
SELECT MEASURES.[avg test score] ON 0,
ORDER(homed.city.MEMBERS, MEASURES.[avg test score], BDESC) ON 1 FROM patients

<table>
<thead>
<tr>
<th>Avg Test Score</th>
<th>stringtest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Pine</td>
<td>75.67</td>
</tr>
<tr>
<td>2 Centerville</td>
<td>74.85</td>
</tr>
<tr>
<td>3 Magnolia</td>
<td>74.65</td>
</tr>
<tr>
<td>4 Cedar Falls</td>
<td>74.62</td>
</tr>
<tr>
<td>5 Cypress</td>
<td>74.61</td>
</tr>
<tr>
<td>6 Juniper</td>
<td>74.52</td>
</tr>
<tr>
<td>7 Elm Heights</td>
<td>74.36</td>
</tr>
<tr>
<td>8 Redwood</td>
<td>74.16</td>
</tr>
<tr>
<td>9 Spruce</td>
<td>74.14</td>
</tr>
</tbody>
</table>

For another example, the following query defines a string measure (as a calculated member) and then uses it with the ORDER function:

WITH MEMBER measures.stringtest AS 'IIF(MEASURES.[avg test score]<75, "low","high")'
SELECT {MEASURES.[avg test score], MEASURES.stringtest} on 0,
ORDER(homed.city.MEMBERS, measures.stringtest, BASC) ON 1 FROM patients

<table>
<thead>
<tr>
<th>Avg Test Score</th>
<th>stringtest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Pine</td>
<td>75.67</td>
</tr>
<tr>
<td>2 Cedar Falls</td>
<td>74.62</td>
</tr>
<tr>
<td>3 Centerville</td>
<td>74.85</td>
</tr>
<tr>
<td>4 Cypress</td>
<td>74.61</td>
</tr>
<tr>
<td>5 Elm Heights</td>
<td>74.36</td>
</tr>
<tr>
<td>6 Juniper</td>
<td>74.52</td>
</tr>
<tr>
<td>7 Magnolia</td>
<td>74.65</td>
</tr>
<tr>
<td>8 Redwood</td>
<td>74.16</td>
</tr>
<tr>
<td>9 Spruce</td>
<td>74.14</td>
</tr>
</tbody>
</table>

See Also

• HIERARCHIZE
PARALLELPERIOD

Given a reference member, a parent level of that member, and an integer, this function counts backward in the parent level, finds a previous member in that level, and then returns its child that has the same position as the reference member.

Returned Type

This function returns a member.

Syntax and Details

PARALLELPERIOD(level_expression,offset,member_expression)

Where:

- *level_expression* is an expression that returns a level. For example:

  [dimension_name].[hierarchy_name].[level_name]

  This level must be a higher level within the hierarchy that contains the reference member.

- *offset* is an integer literal.

  You can use negative integers.

- *member_expression* is an expression that returns a member.

  This expression cannot refer to a measure.

  This is used as the reference member.

For the given member, this function examines the ancestor within the given level, counts backward from that member (using *offset*), finds another member in that level, and returns the child member that has the same position as the reference member.

This function ignores the hierarchy; that is, two members can be considered adjacent even if they have different parents.

Example

For example, the following query finds the quarter that is parallel to Q1 1943, by looking back one year:

```
SELECT MEASURES.[%COUNT] ON 0, PARALLELPERIOD(birthd.year,1,birthd.[Q1 1943]) ON 1 FROM patients
Patient Count
Q1 1942                                  22
```

In contrast, the following query finds the quarter that is parallel to Q1 1943, by looking one decade backward:

```
SELECT MEASURES.[%COUNT] ON 0, PARALLELPERIOD(birthd.decade,1,birthd.[Q1 1943]) ON 1 FROM patients
Patient Count
Q1 1939                                  17
```

As noted previously, you can specify a negative integer for *offset*. The following query finds the quarter that is parallel to Q1 1943, by looking ahead three years:

```
SELECT MEASURES.[%COUNT] ON 0, PARALLELPERIOD(birthd.year,-3,birthd.[Q1 1943]) ON 1 FROM patients
Patient Count
Q1 1946                                  18
```
See Also

- COUSIN
PARENT

Returns the member that is the parent of the given member.

Returned Type

This function returns a member.

Syntax and Details

```
member_expression.PARENT
```

Where:

- `member_expression` is an expression that returns a member, possibly the All member for the dimension.
  
  This expression cannot refer to a measure.
  
  The function returns the parent of this member.

Apart from measures, every member belongs to a level, and every level belongs to a hierarchy, which defines the parent-child relationships among the members. DeepSee creates an All level (with a single All member), at the highest part of each hierarchy (unless the cube disables the All level for the dimension).

Example

For example:

```
SELECT MEASURES.[%COUNT] ON 0, homed.city.[Elm Heights].PARENT ON 1 FROM patients
```

Patient Count

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38928</td>
<td>2,276</td>
</tr>
</tbody>
</table>

To see a picture of the hierarchy used in this example, see the FIRSTCHILD function.

See Also

- ANCESTOR
- CHILDREN
- COUSIN
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- LASTSIBLING
- SIBLINGS
PERCENTILE

Evaluates a given expression (or the current measure), across all elements of a set, and returns the number that is at a given percentile level. This number is greater than the given percentage of the values in the set.

Returned Type

This function returns a number.

Syntax and Details

PERCENTILE(set_expression, optional_numeric_expression, optional_percentile_value)

Where:

- set_expression is an expression that evaluates to a set, typically a set of members or tuples.
- optional_numeric_expression is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form [MEASURES].[measure_name]

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

- optional_percentile_value is a numeric literal that represents the percentile to find. For example, use 30 to find the 30th percentile, which is the value that is greater than 30 percent of the other values.

If you omit this argument, DeepSee computes the 50th percentile.

The function evaluates the numeric value for each element of the set and returns a value that is at the given percentile.

Example

First, the following query shows the Patient Count measure for the members of the aged.year level. The ORDER function sorts these members into order by their value of Patient Count so that we can easily compare the later results to this query:

```sql
SELECT MEASURES.[%COUNT] ON 0, ORDER(birthd.year.MEMBERS,MEASURES.[%COUNT],BASC) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1916</td>
</tr>
<tr>
<td>2 1924</td>
</tr>
<tr>
<td>3 1917</td>
</tr>
<tr>
<td>4 1911</td>
</tr>
<tr>
<td>5 1912</td>
</tr>
<tr>
<td>6 1914</td>
</tr>
<tr>
<td>7 1922</td>
</tr>
<tr>
<td>8 1918</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>93 1964</td>
</tr>
<tr>
<td>94 1999</td>
</tr>
<tr>
<td>95 1994</td>
</tr>
<tr>
<td>96 1966</td>
</tr>
<tr>
<td>97 1973</td>
</tr>
<tr>
<td>98 1968</td>
</tr>
<tr>
<td>99 1970</td>
</tr>
<tr>
<td>100 1978</td>
</tr>
</tbody>
</table>

Next, the following query shows the 5th percentile value for these members:

```sql
SELECT MEASURES.[%COUNT] ON 0, PERCENTILE(birthd.year.MEMBERS,,5) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>5 Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.50</td>
</tr>
</tbody>
</table>

The following query shows the 95th percentile instead:
SELECT MEASURES.[%COUNT] ON 0, PERCENTILE(birthd.year.MEMBERS,,95) ON 1 FROM patients

95 Percentile                           170

For another example, we use the second argument for PERCENTILE:

SELECT PERCENTILE(birthd.year.MEMBERS,MEASURES.[%COUNT],50) ON 1 FROM patients

50 Percentile                           121

For additional, similar examples, see AVG.

See Also

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
PERCENTILERANK

For a given numeric value, this function evaluates a given expression (or the current measure), across all elements of a set, and returns the percentile rank of that expression — the percentage of values that are the same or lower.

Returned Type

This function returns a number.

Syntax and Details

PERCENTILERANK(set_expression, numeric_expression, comparison_value)

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members or tuples.
- `numeric_expression` is a numeric-valued expression that the function evaluates for each set element. Typically, this expression has the form `[MEASURES].[measure_name]`
- `comparison_value` is a numeric literal that represents the value against which DeepSee compares the numeric expression for all set members.

The function evaluates `numeric_expression` for each element of the set, compares that value to `comparison_value`, and returns the percentile rank of the comparison value.

See Also

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILE
- STDDEV
- STDDEVP
- SUM
- VAR
- VARP
PERIODSTODATE

Returns the set of child or descendent members of the given level, up to and including the given member. This function is intended primarily for use with time levels.

Returned Type

This function returns a set of members.

Syntax and Details

```
PERIODSTODATE(ancestor_level,member_expression)
```

Where:

- `ancestor_level` is an expression that returns a level. For example:
  ```
  [dimension_name].[hierarchy_name].[level_name]
  ```
  
  This level must be either an ancestor to the member given by `member_expression`.

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

Given a level and a member, this function returns a set that consists of a range of members, from the first member that is a descendent of the given level, up to and including the given member. The members are in the default order specified in the level definition in the cube.

Example

The following query displays all quarters up to Q3 2003, within the year:

```
SELECT MEASURES.[%COUNT] ON 0, PERIODSTODATE(birthd.year,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 2003</td>
<td>35</td>
</tr>
<tr>
<td>Q2 2003</td>
<td>44</td>
</tr>
<tr>
<td>Q3 2003</td>
<td>43</td>
</tr>
</tbody>
</table>

In contrast, the following query displays all quarters up to Q3 2003, within the decade:

```
SELECT MEASURES.[%COUNT] ON 0, PERIODSTODATE(birthd.decade,birthd.[Q3 2003]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 2000</td>
<td>33</td>
</tr>
<tr>
<td>Q2 2000</td>
<td>33</td>
</tr>
<tr>
<td>Q3 2000</td>
<td>45</td>
</tr>
<tr>
<td>Q4 2000</td>
<td>39</td>
</tr>
<tr>
<td>Q1 2001</td>
<td>37</td>
</tr>
<tr>
<td>Q2 2001</td>
<td>40</td>
</tr>
<tr>
<td>Q3 2001</td>
<td>36</td>
</tr>
<tr>
<td>Q4 2001</td>
<td>37</td>
</tr>
<tr>
<td>Q1 2002</td>
<td>39</td>
</tr>
<tr>
<td>Q2 2002</td>
<td>35</td>
</tr>
<tr>
<td>Q3 2002</td>
<td>40</td>
</tr>
<tr>
<td>Q4 2002</td>
<td>38</td>
</tr>
<tr>
<td>Q1 2003</td>
<td>35</td>
</tr>
<tr>
<td>Q2 2003</td>
<td>44</td>
</tr>
<tr>
<td>Q3 2003</td>
<td>43</td>
</tr>
</tbody>
</table>

This function returns a set expression in the form of a range of members. That is, for example, the following two expressions are equivalent:

```
PERIODSTODATE(birthd.decade,birthd.[Q3 2003])
birthd.[Q1 2000]:birthd.[Q3 2003]
```
See Also

- ANCESTOR
- COUSIN
- CLOSINGPERIOD
- OPENINGPERIOD
POWER

Returns the given numeric value raised to the power of the second argument.

**Returned Type**

This function returns a number.

**Syntax and Details**

```mdx
POWER(numeric_expression, power)
```

Where:

- `numeric_expression` is an expression that evaluates to a number.
  - Typically, this expression has the form `[MEASURES].[measure_name]`
- `power` is a numeric expression.
  - Typically, this expression is an integer.
  - If you use a decimal value less than 1, be sure to start the number with 0. For example: `0.5`

**Example**

The first example shows the `%COUNT` measure raised to the third power:

```mdx
SELECT POWER(MEASURES.[%COUNT], 3) ON 0 FROM patients
```

```
POWER
1,000,000,000,000
```

The next example uses the `%LABEL` function to apply a more detailed caption:

```mdx
SELECT %LABEL(POWER(MEASURES.[%COUNT], 3),"PAT CNT^3") ON 0 FROM patients
```

```
PAT CNT^3
1,000,000,000,000
```

The following example shows a fractional power:

```mdx
SELECT POWER(MEASURES.[%COUNT], 0.5) ON 0 FROM patients
```

```
POWER
100
```

**See Also**

- LOG
- SQRT
PREVMEMBER

Returns the previous member of the level to which the given member belongs. The details are different for time dimensions and data dimensions.

Returned Type

This function returns a member.

Syntax and Details

`member_expression.PREVMEMBER`

Where:

- `member_expression` is an expression that returns a member.
  - This expression cannot refer to a measure.

This function examines the members of the level to which the given member belongs, and returns the previous member of that set (considering the default order of that set). For time dimensions, this function ignores any parent level. For data dimensions, this function considers the parent level; it counts backward from the current member within the given parent member. (Note that the terms `time dimension` and `data dimension` refer specifically to the dimension type as defined in the cube. See *Defining DeepSee Models*.)

The `PREVMEMBER` function is equivalent to `LAG(1)`.

Example

The first examples use a time dimension. Consider the following query, shown for reference:

```
SELECT MEASURES.[%COUNT] ON 0,
  {birthd.1948,birthd.1949,birthd.1950,birthd.1951,birthd.1952} ON 1
FROM patients
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1948</td>
<td>10</td>
</tr>
<tr>
<td>1949</td>
<td>4</td>
</tr>
<tr>
<td>1950</td>
<td>12</td>
</tr>
<tr>
<td>1951</td>
<td>8</td>
</tr>
<tr>
<td>1952</td>
<td>6</td>
</tr>
</tbody>
</table>

The following query uses `PREVMEMBER`:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1951.PREVMEMBER ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>12</td>
</tr>
</tbody>
</table>

For another example:

```
SELECT MEASURES.[%COUNT] ON 0, birthd.1950.PREVMEMBER ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1949</td>
<td>4</td>
</tr>
</tbody>
</table>

In this sample, the year level is the child of the decade level, which means that the members 1949 and 1950 belong to different parents. As you can see, the `PREVMEMBER` function ignores the parent level when you use the function with a time dimension.

The second examples use a data dimension (the HomeD dimension). To see the hierarchy in this dimension, see the examples in the `FIRSTCHILD` function. The following query uses `PREVMEMBER` with this dimension:
SELECT MEASURES.[%COUNT] ON 0, homed.city.Magnolia.PREVMEMBER ON 1 FROM patients

Cypress

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
</tr>
</tbody>
</table>

Because this is a data dimension, this query retrieves the previous member of the city level within the parent ZIP code. Within this ZIP code, Cypress is the first city, so the following query returns no results:

SELECT MEASURES.[%COUNT] ON 0, homed.city.Cypress.PREVMEMBER ON 1 FROM patients

<table>
<thead>
<tr>
<th>Patient Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

**See Also**

- LAG
- LEAD
- NEXTMEMBER
PROPERTIES

Returns the value of the given property, for the given member.

Returned Type
This function returns a string.

Syntax and Details

\[
\text{member_expression.PROPERTIES(property_name)}
\]

Where:

- \text{member_expression} is an expression that returns a member.
  
  This expression cannot refer to a measure.

- \text{property_name} is a string that equals the name of the property.
  
  All members have certain intrinsic properties, listed in the reference “Intrinsic Properties.” A cube definition can include definitions of additional properties.

Names of properties are not case-sensitive.

If the given member does not have the property, the function returns an empty string.

Example

The following example gets the value of the KEY property, which is an intrinsic property:

\[
\text{SELECT docd.h1.CURRENTMEMBER.PROPERTIES("KEY") ON 0, docd.[doctor].MEMBERS ON 1 FROM patients}
\]

<table>
<thead>
<tr>
<th></th>
<th>Doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Adam, Dan 41</td>
</tr>
<tr>
<td>3</td>
<td>Adam, Danielle 391</td>
</tr>
</tbody>
</table>

The following variation uses \text{%LABEL} to provide a better caption:

\[
\text{SELECT %LABEL(docd.h1.CURRENTMEMBER.PROPERTIES("key"),"key") ON 0, docd.doctor.MEMBERS ON 1 FROM patients}
\]

<table>
<thead>
<tr>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 None</td>
</tr>
<tr>
<td>2 Adam, Dan 41</td>
</tr>
<tr>
<td>3 Adam, Danielle 391</td>
</tr>
</tbody>
</table>

The following example uses \text{CURRENTMEMBER} and iterates through the ZIP codes to retrieve the values of two intrinsic properties: ID and LEVEL_NUMBER:

\[
\text{WITH SET test AS '[(homed.h1.CURRENTMEMBER.PROPERTIES("id"), homed.h1.CURRENTMEMBER.PROPERTIES("level_number"))']}
\]

\[
\text{SELECT test ON 0, homed.zip.MEMBERS ON 1 FROM patients}
\]

<table>
<thead>
<tr>
<th>Home ZIP</th>
<th>Home ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 32006</td>
<td>2 1</td>
</tr>
<tr>
<td>2 32007</td>
<td>4 1</td>
</tr>
<tr>
<td>3 34577</td>
<td>1 1</td>
</tr>
<tr>
<td>4 36711</td>
<td>5 1</td>
</tr>
<tr>
<td>5 38928</td>
<td>3 1</td>
</tr>
</tbody>
</table>

As a variation, the following query uses \text{%LABEL} to provide better captions:
WITH SET test AS '{%LABEL(homed.h1.CURRENTMEMBER.PROPERTIES("id"),"id"),
%LABEL(homed.h1.CURRENTMEMBER.PROPERTIES("level_number"),"level_number")}'
SELECT test ON 0, homed.zip.MEMBERS ON 1 FROM patients

<table>
<thead>
<tr>
<th>id</th>
<th>level_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

For more examples, see CURRENTMEMBER.
RANK

Returns an integer that indicates the rank of the given member, within the given set.

Returned Type
This function returns a number.

Syntax and Details
RANK(member_expression, set_expression, optional_numeric_expression)

Where:

• member_expression is an expression that returns a member.
• set_expression is an expression that returns a set.
• optional_numeric_expression is a numeric-valued expression that the function evaluates for each member in the set.

Typically, this expression is [MEASURES].[measure_name]

– If you specify this argument, DeepSee evaluates this expression for the member and for every other member in the set. DeepSee then returns an integer that indicates how the given member ranks compared to these other members. The member with the lowest value is at position 1.
– If you do not specify this argument, DeepSee returns the ordinal position of the member within the given set. The first position is 1.

Example
For example, the following query shows the rank of the member colord.green within the set of members of the colord dimension, when the members are ranked by patient count:

```
SELECT RANK(colord.green, colord.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

```
Green                                  2
```

To see that this is correct, consider the following query, which sorts the members of this dimension by the patient count:

```
SELECT MEASURES.[%COUNT] ON 0,
ORDER(colord.MEMBERS, MEASURES.[%COUNT]) ON 1 FROM patients
```

```
Patient Count
1 None                                1,243
2 Green                               1,304
3 Blue                                2,381
4 Orange                              1,302
5 Purple                              1,276
6 Red                                 1,244
7 Yellow                              1,250
```
ROUND

Evaluates a numeric MDX expression and returns a rounded value. This function is a DeepSee extension to MDX (added in 2012.2).

Returned Type
This function returns a numeric expression.

Syntax and Details

```
ROUND(numeric_expression,decimal_places)
```

Where:

- `numeric_expression` is a numeric expression.
- `decimal_places` is an integer literal that specifies the number of decimal places to use for the returned value. The default is 0.

Example
The following shows a simple example:

```
SELECT ROUND(MEASURES.[avg allergy count],2) ON 0,
patgrpd.[patient group].MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>GROUP</th>
<th>ROUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.97</td>
</tr>
<tr>
<td>B</td>
<td>1.06</td>
</tr>
<tr>
<td>None</td>
<td>0.97</td>
</tr>
</tbody>
</table>

DeepSee MDX Reference
SIBLINGS

Returns a set that contains the specified member and all its siblings.

Returned Type
This function returns a set of members.

Syntax and Details

```
member_expression.SIBLINGS
```

Where:

- `member_expression` is an expression that returns a member.
  
  This expression cannot refer to a measure.

Example

For example:

```
SELECT MEASURES.[%COUNT] ON 0, homed.cypress.SIBLINGS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cypress</td>
<td>1,089</td>
</tr>
<tr>
<td>2 Magnolia</td>
<td>1,073</td>
</tr>
<tr>
<td>3 Pine</td>
<td>1,039</td>
</tr>
</tbody>
</table>

To see a picture of the hierarchy used in this example, see the `FIRSTCHILD` function.

See Also

- CHILDREN
- COUSIN
- FIRSTCHILD
- FIRSTSIBLING
- LASTCHILD
- LASTSIBLING
- PARENT
**SQRT**

Returns the square root of the given numeric value.

**Returned Type**

This function returns a number.

**Syntax and Details**

```
SQRT(numeric_expression)
```

Where:

- `numeric_expression` is an expression that evaluates to a number.

  Typically, this expression has the form `[MEASURES].[measure_name]`

**Example**

The first example shows the square root of the Patient Count measure:

```
SELECT SQRT(MEASURES.[%COUNT]) ON 0 FROM patients
```

```
SQRT 100
```

The next example uses the `%LABEL` function to apply a more detailed caption:

```
SELECT %LABEL(SQRT(MEASURES.[%COUNT]),"SQRT PAT CNT") ON 0 FROM patients
```

```
SQRT PAT CNT 100
```

**See Also**

- **LOG**
- **POWER**
STDDEV

Returns the standard deviation of a given expression (or of the current measure), across all elements of a set.

Returned Type

This function returns a number.

Syntax and Details

STDDEV(set_expression, optional_numeric_expression)

Where:

- set_expression is an expression that evaluates to a set, typically a set of members or tuples.
- optional_numeric_expression is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form [MEASURES].[measure_name]

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses %COUNT, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the standard deviation of those values.

Example

First, the following query shows two measure values for the members of the aged.decade level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
        birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>80</td>
</tr>
<tr>
<td>2 1920s</td>
<td>227</td>
</tr>
<tr>
<td>3 1930s</td>
<td>567</td>
</tr>
<tr>
<td>4 1940s</td>
<td>724</td>
</tr>
<tr>
<td>5 1950s</td>
<td>1,079</td>
</tr>
<tr>
<td>6 1960s</td>
<td>1,475</td>
</tr>
<tr>
<td>7 1970s</td>
<td>1,549</td>
</tr>
<tr>
<td>8 1980s</td>
<td>1,333</td>
</tr>
<tr>
<td>9 1990s</td>
<td>1,426</td>
</tr>
<tr>
<td>10 2000s</td>
<td>1,406</td>
</tr>
<tr>
<td>11 2010s</td>
<td>134</td>
</tr>
</tbody>
</table>

Next, the following query shows the standard deviations for these measures for the members of this level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
        STDDEV(birthd.decade.MEMBERS) ON 1 FROM patients
```

| STDDEV | 579.41 | 18,401.33 |

Here, each value is the standard deviation of the values in a column in the preceding query. For example, the Patient Count value is the standard deviation of the Patient Count values in the preceding query.

For another example, we use the second argument for STDDEV:

```
SELECT STDDEV(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

| STDDEV | 579.41 |

For additional, similar examples, see AVG.
**See Also**

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEVP
- SUM
- VAR
- VARP
# STDDEVP

Returns the population standard deviation of a given expression, across all elements of a set.

## Returned Type

This function returns a number.

## Syntax and Details

```
STDDEVP(set_expression, optional_numeric_expression)
```

Where:

- **set_expression** is an expression that evaluates to a set, typically a set of members or tuples.

- **optional_numeric_expression** is a numeric-valued expression that the function evaluates for each set element.

  Typically, this expression has the form `[MEASURES].[measure_name]`

  If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell.

The function evaluates the numeric value for each element of the set and returns the population standard deviation of those values.

## Example

First, the following query shows two measure values for the members of the `aged.decade` level:

```sql
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
       birthd.decade.MEMBERS ON 1 FROM patients
Patient Count  Encounter Count
  1  1910s                        80                5,359
  2  1920s                        227               12,910
  3  1930s                        567               33,211
  4  1940s                        724               38,420
  5  1950s                       1,079              46,883
  6  1960s                       1,475              57,814
  7  1970s                       1,549              49,794
  8  1980s                       1,333              35,919
  9  1990s                       1,426              29,219
 10  2000s                       1,406              20,072
 11  2010s                        134                1,346
```

Next, the following query shows the population standard deviations for these measures for the members of this level:

```sql
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
       STDDEVP(birthd.decade.MEMBERS) ON 1 FROM patients
Patient Count  Encounter Count
  552.44                17,544.98
```

Here, each value is the population standard deviation of the values in a column in the preceding query. For example, the Patient Count value is the population standard deviation of the Patient Count values in the preceding query.

For another example, we use the second argument for STDDEVP:

```sql
SELECT STDDEVP(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
STDDEVP 552.44
```

For additional, similar examples, see AVG.

## See Also

- [AGGREGATE](#)
• **AVG**
• **MAX**
• **MEDIAN**
• **MIN**
• **PERCENTILE**
• **PERCENTILERANK**
• **STDDEV**
• **SUM**
• **VAR**
• **VARP**
STDEV

Synonym for STDDEV.

See Also

• STDDEV
STDEVP

Synonym for STDDEV.

See Also

- STDDEV
SUBSET

Returns a set of elements from a given set, by position. The first member is at position 0.

Returned Type

This function returns a set.

Syntax and Details

\[
\text{SUBSET}(\text{set_expression}, \text{first_element_position}, \text{optional_element_count})
\]

Where:

- \text{set_expression} is an expression that evaluates to a set.
- \text{first_element_position} is an integer literal that specifies the position of the first element to return.
  
  The position of the first element is 0.
- \text{optional_element_count} is an integer literal that specifies the position of the number of element to return.
  
  If you omit this argument, the function returns the element at \text{first_element_position} and all elements that follow it.

Example

\[
\begin{align*}
\text{SELECT MEASURES.}[\%\text{COUNT}] \text{ ON 0, SUBSET(homed.city.MEMBERS, 0, 3) ON 1 FROM patients} \\
\hline
\text{Patient Count} & \\
1 \text{ Cedar Falls} & 1,188 \\
2 \text{ Centerville} & 1,155 \\
3 \text{ Cypress} & 1,221 \\
\end{align*}
\]

In contrast, consider this example, which shows the complete set:

\[
\begin{align*}
\text{SELECT MEASURES.}[\%\text{COUNT}] \text{ ON 0, homed.city.MEMBERS ON 1 FROM patients} \\
\hline
\text{Patient Count} & \\
1 \text{ Cedar Falls} & 1,188 \\
2 \text{ Centerville} & 1,155 \\
3 \text{ Cypress} & 1,221 \\
4 \text{ Elm Heights} & 1,266 \\
5 \text{ Juniper} & 1,197 \\
6 \text{ Magnolia} & 1,156 \\
7 \text{ Pine} & 1,139 \\
8 \text{ Redwood} & 1,144 \\
9 \text{ Spruce} & 1,135 \\
\end{align*}
\]
SUM

Returns the sum of a given expression (or of the current measure), across all elements of a set.

Returned Type

This function returns a number.

Syntax and Details

```
SUM(set_expression, optional_numeric_expression)
```

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members or tuples.
- `optional_numeric_expression` is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form `[MEASURES].[measure_name]`

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the `WHERE` clause, if any. If the query itself does not specify a measure, DeepSee instead uses `%COUNT`, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the sum of those values.

Example

First, the following query shows values of three measures for the members of the `aged.decade` level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1910s</td>
<td>80</td>
<td>5,359</td>
</tr>
<tr>
<td>1920s</td>
<td>227</td>
<td>12,910</td>
</tr>
<tr>
<td>1930s</td>
<td>567</td>
<td>33,211</td>
</tr>
<tr>
<td>1940s</td>
<td>724</td>
<td>38,420</td>
</tr>
<tr>
<td>1950s</td>
<td>1,079</td>
<td>46,883</td>
</tr>
<tr>
<td>1960s</td>
<td>1,475</td>
<td>57,814</td>
</tr>
<tr>
<td>1970s</td>
<td>1,549</td>
<td>49,794</td>
</tr>
<tr>
<td>1980s</td>
<td>1,333</td>
<td>35,919</td>
</tr>
<tr>
<td>1990s</td>
<td>1,426</td>
<td>29,219</td>
</tr>
<tr>
<td>2000s</td>
<td>1,306</td>
<td>20,072</td>
</tr>
<tr>
<td>2010s</td>
<td>134</td>
<td>1,346</td>
</tr>
</tbody>
</table>

Next, the following query shows the sums for these measures for the members of this level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count],MEASURES.[avg test score]} ON 0,
SUM(birthd.decade.MEMBERS) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Encounter Count</th>
<th>Avg Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>10,000</td>
<td>330,947</td>
</tr>
</tbody>
</table>

Here, each value is the sum of the values in a column in the preceding query. For example, the Patient Count value is the average of the Patient Count values in the preceding query. The Avg Test Score value is the sum of the average test scores and is probably not a useful value.

For another example, we use the second argument for SUM:

```
SELECT SUM(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

```
SUM
10,000
```

For additional, similar examples, see AVG.
See Also

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- VAR
- VARP
TAIL

Returns a subset from the end of a set, using the current order of the set.

Returned Type

This function returns a set.

Syntax and Details

TAIL(set_expression, optional_integer_expression)

- set_expression is an expression that evaluates to a set.
- optional_integer_expression is an integer literal.
  The default value for this argument is 1.
  The function uses this argument to determine the number of elements to return in the subset.

This function returns a set that consists of the specified number of elements from the end of the given set (considering the current order of the set). If integer_expression is less than 1, the function returns the empty set. If integer_expression is greater than the number of elements of the set, the function returns the original set.

The elements of the subset are returned in the same order specified by the original set.

Example

SELECT MEASURES.[%COUNT] ON 0, TAIL(birthd.decade.MEMBERS, 3) ON 1
FROM patients

<table>
<thead>
<tr>
<th>Patient</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1990s</td>
<td>1,413</td>
</tr>
<tr>
<td>2 2000s</td>
<td>1,433</td>
</tr>
<tr>
<td>3 2010s</td>
<td>155</td>
</tr>
</tbody>
</table>

See Also

- HEAD
**TOPCOUNT**

Sorts a set and returns a subset from its higher-valued end, given a desired element count.

**Returned Type**

This function returns a set of members or tuples, depending on the set used.

**Syntax and Details**

\[
\text{TOPCOUNT(set_expression, element_count, optional_ordering_expression)}
\]

Where:

- set_expression is an expression that evaluates to a set of members or tuples.
- element_count is an integer literal.
  
  The function uses this argument to determine the number of elements to return in the subset. If this argument is greater than the number of elements, all elements are returned.
- optional_ordering_expression is a numeric-valued expression that determines the order of the set elements.

  Typically, ordering_expression is \([\text{MEASURES}].\[\text{measure\_name}\] \)

  The function evaluates this expression for each element of the set and sorts the elements of the set in descending order according to this value. Any hierarchies are ignored.

  If this argument is omitted, the function uses the current order of the set elements (and this function behaves like the HEAD function).

**Example**

First consider the following query and the results it returns:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPCOUNT(birthd.decade.MEMBERS, 100, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970s</td>
<td>1,520</td>
</tr>
<tr>
<td>1960s</td>
<td>1,500</td>
</tr>
<tr>
<td>2000s</td>
<td>1,433</td>
</tr>
<tr>
<td>1990s</td>
<td>1,413</td>
</tr>
<tr>
<td>1980s</td>
<td>1,400</td>
</tr>
<tr>
<td>1950s</td>
<td>1,030</td>
</tr>
<tr>
<td>1940s</td>
<td>683</td>
</tr>
<tr>
<td>1930s</td>
<td>572</td>
</tr>
<tr>
<td>1920s</td>
<td>223</td>
</tr>
<tr>
<td>2010s</td>
<td>155</td>
</tr>
<tr>
<td>1910s</td>
<td>71</td>
</tr>
</tbody>
</table>

Because count_expression is greater than the number of members, all members are returned. The members are sorted in ascending order according to the value of the %COUNT measure.

Next, consider a similar query, using count_expression equal to 3:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPCOUNT(birthd.decade.MEMBERS, 3, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970s</td>
<td>1,520</td>
</tr>
<tr>
<td>1960s</td>
<td>1,500</td>
</tr>
<tr>
<td>2000s</td>
<td>1,433</td>
</tr>
</tbody>
</table>

This query selects three members from the higher-valued end of the set.
See Also

- BOTTOMCOUNT
TOPPERCENT

Sorts a set and returns a subset from its higher-valued end, given a cutoff percentage that is applied to a total across set elements.

Returned Type

This function returns a set of members or tuples, depending on the set used.

Syntax and Details

```
TOPPERCENT(set_expression, percentage, ordering_expression)
```

- **set_expression** is an expression that evaluates to a set of members or tuples.
- **percentage** is a numeric literal that is less than or equal to 100. That is, 15 represents 15 percent.

  The function uses this argument to determine the cutoff point for elements to return in the subset.

  There is usually a member that straddles the cutoff point; this member is assigned to the upper set, rather than the lower set. As a result, in the returned subset, the cumulative total for **ordering_expression** could be greater than **percentage**, as a percentage of the entire set.

- **ordering_expression** is a numeric-valued expression that determines the order of the set elements.

  The function evaluates this expression for each element of the set and sorts the elements of the set in descending order according to this value. Any hierarchies are ignored.

Example

First consider the following query and the results it returns:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPPERCENT(birthd.decade.MEMBERS, 100, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2000s</td>
<td>157</td>
</tr>
<tr>
<td>2 1980s</td>
<td>155</td>
</tr>
<tr>
<td>3 1990s</td>
<td>144</td>
</tr>
<tr>
<td>4 1960s</td>
<td>136</td>
</tr>
<tr>
<td>5 1970s</td>
<td>128</td>
</tr>
<tr>
<td>6 1950s</td>
<td>107</td>
</tr>
<tr>
<td>7 1930s</td>
<td>56</td>
</tr>
<tr>
<td>8 1940s</td>
<td>54</td>
</tr>
<tr>
<td>9 2010s</td>
<td>44</td>
</tr>
<tr>
<td>10 1920s</td>
<td>13</td>
</tr>
<tr>
<td>11 1910s</td>
<td>6</td>
</tr>
</tbody>
</table>

Because **percentage** is 100, all members are returned.

Now consider a variation of the preceding, in which **percentage** is 50, so that we see the top 50 percent:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPPERCENT(birthd.decade.MEMBERS, 50, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2000s</td>
<td>157</td>
</tr>
<tr>
<td>2 1980s</td>
<td>155</td>
</tr>
<tr>
<td>3 1990s</td>
<td>144</td>
</tr>
<tr>
<td>4 1960s</td>
<td>136</td>
</tr>
</tbody>
</table>

The total for the `%COUNT` measure for these members is a little more than 50% of the total. (If the 1960s were omitted, the total count would be less than 50%.)
See Also

- BOTTOMPERCENT
TOPSUM

Sorts a set and returns a subset from its higher-valued end, given a cutoff value that is applied to a total across elements.

Returned Type

This function returns a set of members or tuples, depending on the set used.

Syntax and Details

```
TOPSUM(set_expression, cutoff_value, ordering_expression)
```

- **set_expression** is an expression that evaluates to a set of members or tuples.
- **cutoff_value** is a numeric literal.
  
The function uses this argument to determine the cutoff value for elements to return in the subset.
  
  For all elements in the returned subset, the sum of the values of **ordering_expression** will be less than or equal to **cutoff_value**.
  
- **ordering_expression** is a numeric-valued expression that determines the order of the set members.

  The function evaluates this expression for each element of the set and sorts the elements of the set in descending order according to this value. Any hierarchies are ignored.

Example

First consider an example in which the cutoff value is high enough to include all members:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPSUM(birthd.decade.MEMBERS, 10000, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>1970s</th>
<th>1960s</th>
<th>2000s</th>
<th>1990s</th>
<th>1980s</th>
<th>1950s</th>
<th>1940s</th>
<th>1930s</th>
<th>1920s</th>
<th>2010s</th>
<th>1910s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,520</td>
<td>1,500</td>
<td>1,433</td>
<td>1,413</td>
<td>1,400</td>
<td>1,030</td>
<td>683</td>
<td>572</td>
<td>223</td>
<td>155</td>
<td>71</td>
</tr>
</tbody>
</table>

Now consider a variation in which the cutoff value is set to 2500:

```
SELECT MEASURES.[%COUNT] ON 0,
TOPSUM(birthd.decade.MEMBERS, 2500, MEASURES.[%COUNT]) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>1970s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,520</td>
</tr>
</tbody>
</table>

See Also

- **BOTTOMSUM**
UNION

Returns a set that consists of the elements of the two given sets, optionally eliminating duplicates.

Returned Type

This function returns a set.

Syntax and Details

\[
\text{UNION(set_expression1, set_expression2, ALL)}
\]

Or:

\[
\text{UNION(set_expression1, set_expression2)}
\]

- \textit{set_expression1} and \textit{set_expression2} are expressions that evaluate to sets.
- The optional keyword ALL, if included, specifies that all duplicates should be retained. By default, if the returned set includes any duplicate elements, only the first of those is included.

Example

Consider the following query which defines two named sets:

```mdx
WITH SET set1 AS '{allerd.eggs,allerd.soy,allerd.wheat}'
SET set2 AS '{allerd.[dairy products],allerd.pollen,allerd.soy,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, UNION(set1,set2) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 eggs</td>
<td>451</td>
</tr>
<tr>
<td>2 soy</td>
<td>462</td>
</tr>
<tr>
<td>3 wheat</td>
<td>479</td>
</tr>
<tr>
<td>4 dairy products</td>
<td>463</td>
</tr>
<tr>
<td>5 pollen</td>
<td>447</td>
</tr>
</tbody>
</table>

This query shows all the members that are in \textit{set1} and \textit{set2}.

In contrast, consider the following variation, which uses the ALL keyword to keep duplicates:

```mdx
WITH SET set1 AS '{allerd.eggs,allerd.soy,allerd.wheat}'
SET set2 AS '{allerd.[dairy products],allerd.pollen,allerd.soy,allerd.wheat}'
SELECT MEASURES.[%COUNT] ON 0, UNION(set1,set2,ALL) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 eggs</td>
<td>451</td>
</tr>
<tr>
<td>2 soy</td>
<td>462</td>
</tr>
<tr>
<td>3 wheat</td>
<td>479</td>
</tr>
<tr>
<td>4 dairy products</td>
<td>463</td>
</tr>
<tr>
<td>5 pollen</td>
<td>447</td>
</tr>
<tr>
<td>6 soy</td>
<td>462</td>
</tr>
<tr>
<td>7 wheat</td>
<td>479</td>
</tr>
</tbody>
</table>

Finally, you can of course use more interesting sets as arguments. For example:

```mdx
WITH SET set1 AS 'TOPCOUNT(homed.city.members,5,MEASURES.[avg allergy count])'
SET set2 AS 'TOPCOUNT(homed.city.members,5,MEASURES.[avg age])'
SELECT MEASURES.[%COUNT] ON 0, UNION(set1,set2) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Juniper</td>
<td>1,197</td>
</tr>
<tr>
<td>2 Spruce</td>
<td>1,135</td>
</tr>
<tr>
<td>3 Centerville</td>
<td>1,155</td>
</tr>
<tr>
<td>4 Redwood</td>
<td>1,144</td>
</tr>
<tr>
<td>5 Magnolia</td>
<td>1,156</td>
</tr>
<tr>
<td>6 Cedar Falls</td>
<td>1,188</td>
</tr>
<tr>
<td>7 Elm Heights</td>
<td>1,266</td>
</tr>
<tr>
<td>8 Pine</td>
<td>1,139</td>
</tr>
</tbody>
</table>
See Also

- EXCEPT
- INTERSECT
VAR

Returns the variance of a given expression (or of the current measure), across all elements of a set.

Returned Type

This function returns a number.

Syntax and Details

\[
\text{VAR}(\text{set\_expression}, \text{optional\_numeric\_expression})
\]

Or:

\[
\text{VAR}(\text{set\_expression})
\]

Where:

- \textit{set\_expression} is an expression that evaluates to a set, typically a set of members or tuples.
- \textit{optional\_numeric\_expression} is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form \([\text{MEASURES}].[\text{measure\_name}]\)

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses \%(\text{COUNT})\, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the variance of those values.

Example

First, the following query shows two measure values for the members of the \textit{aged.decade} level:

```
SELECT {MEASURES.[\%COUNT],MEASURES.[encounter count]} ON 0,
birthd.decade.MEMBERS ON 1 FROM patients

Patient Count | Encounter Count
1 1910s        | 80                | 5,359
2 1920s        | 227               | 12,910
3 1930s        | 567               | 33,211
4 1940s        | 724               | 38,420
5 1950s        | 1,079             | 46,883
6 1960s        | 1,475             | 57,814
7 1970s        | 1,549             | 49,794
8 1980s        | 1,333             | 35,919
9 1990s        | 1,426             | 29,219
10 2000s       | 1,406             | 20,072
11 2010s       | 134               | 1,346
```

Next, the following query shows the variances for these measures for the members of this level:

```
SELECT {MEASURES.[\%COUNT],MEASURES.[encounter count]} ON 0,
VAR(birthd.decade.MEMBERS) ON 1 FROM patients

VAR
335,710.89
```

Here, each value is the variance of the values in a column in the preceding query. For example, the \textit{Patient Count} value is the variance of the \textit{Patient Count} values in the preceding query.

For another example, we use the second argument for VAR:

```
SELECT VAR(birthd.decade.MEMBERS, MEASURES.[\%COUNT]) ON 0 FROM patients

VAR
335,710.89
```
For additional, similar examples, see AVG.

See Also

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VARP
**VARIANCE**

Synonym for VAR.

**See Also**

- VAR
VARIANCEP

Synonym for VARP.

See Also

- VARP
VARP

Returns the population variance of a given expression, across all elements of a set.

Returned Type
This function returns a number.

Syntax and Details

```
VARP(set_expression, optional_numeric_expression)
```

Where:

- `set_expression` is an expression that evaluates to a set, typically a set of members or tuples.
- `optional_numeric_expression` is a numeric-valued expression that the function evaluates for each set element.

Typically, this expression has the form `[MEASURES].[measure_name]`

If you do not specify a numeric expression, DeepSee uses the measure used by the current result cell. For example, this might be the measure used on the 0 axis or the measure specified in the WHERE clause, if any. If the query itself does not specify a measure, DeepSee instead uses `%COUNT`, which counts records in the fact table.

The function evaluates the numeric value for each element of the set and returns the population variance of those values.

Example
First, the following query shows two measure values for the members of the `aged.decade` level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
       birthd.decade.MEMBERS ON 1 FROM patients
```

<table>
<thead>
<tr>
<th></th>
<th>Patient Count</th>
<th>Encounter Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1910s</td>
<td>80</td>
<td>5,359</td>
</tr>
<tr>
<td>2 1920s</td>
<td>227</td>
<td>12,910</td>
</tr>
<tr>
<td>3 1930s</td>
<td>567</td>
<td>33,211</td>
</tr>
<tr>
<td>4 1940s</td>
<td>724</td>
<td>38,420</td>
</tr>
<tr>
<td>5 1950s</td>
<td>1,079</td>
<td>46,883</td>
</tr>
<tr>
<td>6 1960s</td>
<td>1,475</td>
<td>57,814</td>
</tr>
<tr>
<td>7 1970s</td>
<td>1,549</td>
<td>49,794</td>
</tr>
<tr>
<td>8 1980s</td>
<td>1,333</td>
<td>35,919</td>
</tr>
<tr>
<td>9 1990s</td>
<td>1,426</td>
<td>29,219</td>
</tr>
<tr>
<td>10 2000s</td>
<td>1,406</td>
<td>20,072</td>
</tr>
<tr>
<td>11 2010s</td>
<td>134</td>
<td>1,346</td>
</tr>
</tbody>
</table>

Next, the following query shows the population variances for these measures for the members of this level:

```
SELECT {MEASURES.[%COUNT],MEASURES.[encounter count]} ON 0,
       VARP(birthd.decade.MEMBERS) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th></th>
<th>Patient Count</th>
<th>Encounter Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARP</td>
<td>305,191.72</td>
<td>307,826,410.63</td>
</tr>
</tbody>
</table>

Here, each value is the population variance of the values in a column in the preceding query. For example, the Patient Count value is the population variance of the Patient Count values in the preceding query.

For another example, we use the second argument for VARP:

```
SELECT VARP(birthd.decade.MEMBERS, MEASURES.[%COUNT]) ON 0 FROM patients
```

| VARP    | 305,191.72 |

For additional, similar examples, see AVG.
See Also

- AGGREGATE
- AVG
- MAX
- MEDIAN
- MIN
- PERCENTILE
- PERCENTILERANK
- STDDEV
- STDDEVP
- SUM
- VAR
**VISUALTOTALS**

Given a set of members in hierarchical order, returns that set with its visual totals. In the visual totals, the actual value for any higher-level member is replaced with the sum of the values for the children that are included in the query.

**Returned Type**

This function returns a set of members.

**Syntax and Details**

```
VISUALTOTALS(set_expression, optional_parent_name_pattern)
```

Where:

- `set_expression` is an expression that evaluates to a set of members. This set can include members at different levels within the same dimension but the members should be in hierarchical order.

- `optional_parent_name_pattern` is a string that includes an asterisk (*) in the place where the parent name is to be used. For example: "SUB *" or "* (SUBTOTAL)"

If you omit this, no extra strings are added to the parent names.

**Example**

First consider the following query, which does not use VISUALTOTALS:

```
WITH SET demo AS 'HIERARCHIZE({homed.32006,homed.34577,homed.CYPRESS,homed.PINE,homed.SPRUCE})'
SELECT MEASURES.[%COUNT] ON 0, demo ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>32006</th>
<th>1,117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spruce</td>
<td>1,117</td>
<td></td>
</tr>
<tr>
<td>34577</td>
<td>3,399</td>
<td></td>
</tr>
<tr>
<td>Cypress</td>
<td>1,150</td>
<td></td>
</tr>
<tr>
<td>Pine</td>
<td>1,138</td>
<td></td>
</tr>
</tbody>
</table>

This query shows the patient count for each of the listed ZIP codes and cities. The patient count for each ZIP code is the total patient count for that ZIP code.

Now consider the following variation, which does use VISUALTOTALS:

```
WITH SET demo AS 'HIERARCHIZE({homed.32006,homed.34577,homed.CYPRESS,homed.PINE,homed.SPRUCE})'
SELECT MEASURES.[%COUNT] ON 0, VISUALTOTALS(demo) ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>32006 (included cities)</th>
<th>1,117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spruce</td>
<td>1,117</td>
<td></td>
</tr>
<tr>
<td>34577</td>
<td>2,288</td>
<td></td>
</tr>
<tr>
<td>Cypress</td>
<td>1,150</td>
<td></td>
</tr>
<tr>
<td>Pine</td>
<td>1,138</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the patient count for any higher-level members (the ZIP codes) reflects only the children that are included in the query. For example, the patient count for ZIP code 34577 is the sum of the patient counts for the cities of Pine and Cypress.

For another variation, consider the following query, which is like the preceding except that it also uses the second argument to VISUALTOTALS:

```
VISUALTOTALS */WITH SET demo AS 'HIERARCHIZE({homed.32006,homed.34577,homed.CYPRESS,homed.PINE,homed.SPRUCE})'
SELECT MEASURES.[%COUNT] ON 0, VISUALTOTALS(demo,"* (included cities)") ON 1 FROM patients
```

<table>
<thead>
<tr>
<th>Patient Count</th>
<th>32006 (included cities)</th>
<th>1,117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spruce</td>
<td>1,117</td>
<td></td>
</tr>
<tr>
<td>34577 (included cities)</td>
<td>2,288</td>
<td></td>
</tr>
<tr>
<td>Cypress</td>
<td>1,150</td>
<td></td>
</tr>
<tr>
<td>Pine</td>
<td>1,138</td>
<td></td>
</tr>
</tbody>
</table>
The values shown are the same as in the preceding query, but each ZIP code is shown with the trailing string "(included cities)."
Intrinsic Properties

This reference section provides information on the intrinsic properties for levels in DeepSee cubes.
## Intrinsic Properties

This section lists the intrinsic properties for levels in DeepSee cubes.

### Intrinsic Properties

All members have the following intrinsic properties. The table shows examples using the city of Juniper from the sample.

<table>
<thead>
<tr>
<th>Property</th>
<th>Details</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>Key for the member, for use with the syntax &amp;[key] when referring to the member, as discussed in “Member Expressions.” For details, see “Key Values.”</td>
<td>Juniper</td>
</tr>
<tr>
<td>ID</td>
<td>Internal ID of the member.</td>
<td>6</td>
</tr>
<tr>
<td>NAME</td>
<td>Display name of the member. This is taken from the source value or (if present) from the value of a property marked <code>isName=&quot;true&quot;</code></td>
<td>Juniper</td>
</tr>
<tr>
<td>MEMBER_NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUBE_NAME</td>
<td>Name of the cube to which this member belongs.</td>
<td>PATIENTS</td>
</tr>
<tr>
<td>LEVEL_NUMBER</td>
<td>Number of the level to which this member belongs. Level 0 is the All level.</td>
<td>3</td>
</tr>
<tr>
<td>LEVEL_CAPTION</td>
<td>Localized version of the <code>displayName</code> of the current level. The <code>displayName</code> attribute is specified within the cube definition.</td>
<td>City</td>
</tr>
<tr>
<td>LEVEL</td>
<td>Uppercase version of value returned by <code>LEVEL_CAPTION</code></td>
<td>CITY</td>
</tr>
</tbody>
</table>

Names of properties are not case-sensitive.
## Key Values

This section describes how DeepSee generates KEY values for level members.

### Key Values

DeepSee generates KEY values as follows. Note that these values are case-sensitive, unlike all other items in MDX.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Format of KEY value</th>
<th>Example Member Name*</th>
<th>Corresponding KEY Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null member</td>
<td>&lt;null&gt;</td>
<td>None</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>All member</td>
<td>*</td>
<td>All Patient Addresses</td>
<td>*</td>
</tr>
<tr>
<td>Member of a level that uses the HourNumber time function</td>
<td>Integer that represents the hour number</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Member of a level that uses the DayMonthYear time function</td>
<td>Integer that represents the date in $HOROLOG format</td>
<td>Dec 27 2004</td>
<td>59896</td>
</tr>
<tr>
<td>Member of a level that uses the DayNumber time function</td>
<td>Integer that represents the day number</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Member of a level that uses the WeekNumber time function</td>
<td>Integer that represents the day number</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Member of a level that uses the MonthNumber time function</td>
<td>Integer that represents the month number</td>
<td>July</td>
<td>7</td>
</tr>
<tr>
<td>Member of a level that uses the WeekYear time function</td>
<td>String in the form YYYYMWW</td>
<td>2012W06</td>
<td>2012W06</td>
</tr>
<tr>
<td>Member of a level that uses the MonthYear time function</td>
<td>Integer in the form YYYYMM</td>
<td>July 2010</td>
<td>201007</td>
</tr>
<tr>
<td>Member of a level that uses the QuarterNumber time function</td>
<td>Integer that represents the quarter number</td>
<td>Q3</td>
<td>3</td>
</tr>
<tr>
<td>Member of a level that uses the QuarterYear time function</td>
<td>Integer in the form YYYYQQ</td>
<td>Q4 2010</td>
<td>201004</td>
</tr>
<tr>
<td>Member of a level that uses the Year time function</td>
<td>Four-digit integer that represents the year number</td>
<td>2009</td>
<td>2009</td>
</tr>
<tr>
<td>Member of a level that uses the Decade time function</td>
<td>Four-digit integer that represents the year number, followed by s</td>
<td>1990s</td>
<td>1990s</td>
</tr>
<tr>
<td>Other scenarios</td>
<td>Source value of the member (which is also used as the member name by default)</td>
<td>Juniper</td>
<td>Juniper</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sorenson, Violet</td>
<td>169</td>
</tr>
</tbody>
</table>

*For levels in time dimensions, this column shows their default display names. Depending on how the level is defined, the member names might be different. See Defining DeepSee Models.

**The key for a member is always the source value for the member. In a well-defined cube, that value is unique. See “Defining Member Keys and Names Appropriately” in Defining DeepSee Models.

You can use the PROPERTIES function to find the value of the KEY property or any other property of a member.
NOW Member for Time Levels

This reference section provides information on the NOW member for date/time levels.
NOW Member

This section provides information on the NOW member for date/time levels. This syntax is a DeepSee extension to MDX.

**Basic Syntax**

```
date_time_level.[NOW]
```

Where:
- `date_time_level` is a level expression that refers to a level in a date/time dimension.

This syntax returns the member of the given level that corresponds to the current date and time. The following table shows examples (created on 24 May 2012):

<table>
<thead>
<tr>
<th>Expression</th>
<th>Returned member</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthd.decade.[NOW]</td>
<td>birthd.decade.2010s</td>
</tr>
<tr>
<td>birthd.[quarter year].[NOW]</td>
<td>birthd.[quarter year].[Q2 2012]</td>
</tr>
<tr>
<td>birthqd.[quarter].[NOW]</td>
<td>birthd.[quarter year].Q2</td>
</tr>
<tr>
<td>birthtd.[NOW]</td>
<td>birthtd.4pm</td>
</tr>
</tbody>
</table>

NOW is not case-sensitive. Also, the square brackets around it are optional unless you are using one of the variations discussed below in this topic.

**Dates Relative to Now**

You can also use expressions of the following form:

```
date_time_level.[NOW-integer]
```

Or:

```
date_time_level.[NOW+integer]
```

The following table shows examples (created on 24 May 2012):

<table>
<thead>
<tr>
<th>Expression</th>
<th>Returned member</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthd.[quarter year].[NOW-4]</td>
<td>birthd.[quarter year].[Q2 2011]</td>
</tr>
<tr>
<td>birthqd.quarter.[NOW-4]</td>
<td>birthqd.quarter.Q2</td>
</tr>
</tbody>
</table>

Be careful when using such expressions with a time level that is independent of the overall calendar. In such cases, `[NOW-integer]` refers to an earlier bucket in a cycle. Notice the third example in the table. The `birthqd.quarter` level groups records by quarter, independent of year. For this level, NOW refers only to the current quarter number. This level has four members in a cycle, and for this level, `[NOW-4]` is equivalent to `[NOW]`.

The same logic applies to expressions of the form `[NOW+integer]`.

**Restrictions on Use in Range Expressions**

You cannot use NOW in range expressions for a level that is based on any of the following time functions:
<table>
<thead>
<tr>
<th>Time Function...</th>
<th>Typical Members</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuarterNumber</td>
<td>Q4</td>
<td>These levels are independent of the year</td>
</tr>
<tr>
<td>MonthNumber</td>
<td>November</td>
<td></td>
</tr>
<tr>
<td>DayNumber</td>
<td>24</td>
<td>This level is independent of the year and the part of the year</td>
</tr>
<tr>
<td>HourNumber</td>
<td>1am</td>
<td>These levels are independent of the day</td>
</tr>
<tr>
<td>MinuteNumber</td>
<td>01:24</td>
<td></td>
</tr>
</tbody>
</table>

**Additional Options for Day Levels**

For a level that is based on the **DayMonthYear** time function, DeepSee supports the additional expressions that use a combination of year count, month count, and day count, as follows:

```
daymonthyear_level.[NOW-offset_expression]
```

Or:

```
daymonthyear_level.[NOW+offset_expression]
```

In simple cases, **offset_expression** is **nnynmnd**, as follows:

- **nn** is a one or two digit integer.
- The optional unit **nny** specifies the number of years.
- The optional unit **nmm** specifies the number of months.
- The optional unit **nnd** specifies the number of days.

The following table shows examples (created on 24 May 2012):

<table>
<thead>
<tr>
<th>Expression</th>
<th>Returned member</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthd.date.[NOW-4y3m2d]</td>
<td>birthd.date.[Feb 22 2008]</td>
</tr>
<tr>
<td>birthd.date.[NOW-1m]</td>
<td>birthd.date.[Apr 24 2012]</td>
</tr>
</tbody>
</table>

By default, the units are added together. For example, **4y3m2d** means four years plus three months plus two days.

You can instead include a minus sign between units to subtract. **1y-1d** means one year, minus one day. For example:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Returned member</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateofsale.actual.daysold.[NOW-1y-1d]</td>
<td>dateofsale.actual.daysold.[May 25 2011]</td>
</tr>
</tbody>
</table>

DeepSee automatically accounts for leap years. The internal logic also accounts for the varying lengths of the months.
### Quick Function Reference

The following table summarizes the syntax and return type of each supported MDX function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ALL</td>
<td>member_expression.%ALL</td>
<td>member</td>
</tr>
<tr>
<td>%CELL</td>
<td>%CELL(relative_column_position, relative_row_position)</td>
<td>number or string</td>
</tr>
<tr>
<td>%FIRST</td>
<td>%FIRST(set_expr)</td>
<td>any</td>
</tr>
<tr>
<td>%KPI</td>
<td>%KPI(kpi_name,kpi_prop_name,kpi_series_name)</td>
<td>number</td>
</tr>
<tr>
<td>%LABEL</td>
<td>%LABEL(MDX_expr, label, format_details, sort_order)</td>
<td>same as MDX_expr</td>
</tr>
<tr>
<td>%LAST</td>
<td>%LAST(set_expr)</td>
<td>any</td>
</tr>
<tr>
<td>%LIST</td>
<td>%LIST(set_expr)</td>
<td>string (comma-separated list)</td>
</tr>
<tr>
<td>%MDX</td>
<td>%MDX(&quot;MDX select query&quot;, parm, value, parm, value, parm, value,...)</td>
<td>number or string</td>
</tr>
<tr>
<td>%NOT</td>
<td>member_expression.%NOT</td>
<td>member</td>
</tr>
<tr>
<td>%OR</td>
<td>%OR(set_expr)</td>
<td>member</td>
</tr>
<tr>
<td>%SEARCH</td>
<td>%SEARCH.&amp;[comparison_expression]</td>
<td>measure search expression</td>
</tr>
<tr>
<td>%SPACE</td>
<td>%SPACE()</td>
<td>empty space</td>
</tr>
<tr>
<td>%TERMLIST</td>
<td>%TERMLIST(term_list_name, incl_excl_flag)</td>
<td>set</td>
</tr>
<tr>
<td>%TIMERANGE</td>
<td>%TIMERANGE(start_member, end_member, INCLUSIVE</td>
<td>EXCLUSIVE)</td>
</tr>
<tr>
<td>%TIMEWINDOW</td>
<td>%TIMEWINDOW(set_expr, start_member,optional_end_member)</td>
<td>set of members</td>
</tr>
</tbody>
</table>
| %TOPMEMBERS | level_expr.%TOPMEMBERS
hierarchy_expr.%TOPMEMBERS
dimension_expr.%TOPMEMBERS | set of members |
<p>| AGGREGATE | AGGREGATE(set_expr, optional_numeric_expr) | number |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Syntax</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLMEMBERS</td>
<td><code>level_expr.ALLMEMBERS</code></td>
<td>set of members</td>
</tr>
<tr>
<td></td>
<td><code>hierarchy_expr.ALLMEMBERS</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>dimension_expr.ALLMEMBERS</code></td>
<td></td>
</tr>
<tr>
<td>ANCESTOR</td>
<td><code>ANCESTOR(member_expr, ancestor_level)</code></td>
<td>member</td>
</tr>
<tr>
<td>AVG</td>
<td><code>AVG(set_expr, optional_numeric_expr)</code></td>
<td>number</td>
</tr>
<tr>
<td>BOTTOMCOUNT</td>
<td><code>BOTTOMCOUNT(set_expr, element_count,optional_ordering_expr)</code></td>
<td>set of members or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tuples</td>
</tr>
<tr>
<td>BOTTOMPERCENT</td>
<td><code>BOTTOMPERCENT(set_expr, element_count,optional_ordering_expr)</code></td>
<td>set of members or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tuples</td>
</tr>
<tr>
<td>BOTTOMSUM</td>
<td><code>BOTTOMSUM(set_expr, element_count,optional_ordering_expr)</code></td>
<td>set of members or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tuples</td>
</tr>
<tr>
<td>CHILDREN</td>
<td><code>member_expr.CHILDREN</code></td>
<td>set of members</td>
</tr>
<tr>
<td>CLOSINGPERIOD</td>
<td><code>CLOSINGPERIOD(ancestor_level, member_expr)</code></td>
<td>member</td>
</tr>
<tr>
<td>COUNT</td>
<td><code>COUNT(set_expr)</code></td>
<td>number</td>
</tr>
<tr>
<td></td>
<td><code>COUNT(set_expr, EXCLUDEEMPTY)</code></td>
<td></td>
</tr>
<tr>
<td>COUSIN</td>
<td><code>COUSIN(member_expr, higher_member_expr)</code></td>
<td>member</td>
</tr>
<tr>
<td>CROSSJOIN</td>
<td><code>CROSSJOIN(set_expr1, set_expr2)</code></td>
<td>set of tuples</td>
</tr>
<tr>
<td></td>
<td><code>NON EMPTY CROSSJOIN(set_expr1, set_expr2)</code></td>
<td></td>
</tr>
<tr>
<td>CURRENTMEMBER</td>
<td><code>hierarchy_expr.CURRENTMEMBER</code></td>
<td>member</td>
</tr>
<tr>
<td></td>
<td><code>dimension_expr.CURRENTMEMBER</code></td>
<td></td>
</tr>
<tr>
<td>DISTINCT</td>
<td><code>DISTINCT(set_expr)</code></td>
<td>set</td>
</tr>
<tr>
<td>EXCEPT</td>
<td><code>EXCEPT(set_expr1, set_expr2, ALL)</code></td>
<td>set</td>
</tr>
<tr>
<td></td>
<td><code>EXCEPT(set_expr1, set_expr2)</code></td>
<td></td>
</tr>
<tr>
<td>FILTER</td>
<td><code>FILTER(set_expr, logical_expr)</code></td>
<td>set</td>
</tr>
<tr>
<td>FIRSTCHILD</td>
<td><code>member_expr.FIRSTCHILD</code></td>
<td>member</td>
</tr>
<tr>
<td>FIRSTSIBLING</td>
<td><code>member_expr.FIRSTSIBLING</code></td>
<td>member</td>
</tr>
<tr>
<td>HEAD</td>
<td><code>HEAD(set_expr, optional_integer_expr)</code></td>
<td>set</td>
</tr>
<tr>
<td>HIERARCHIZE,</td>
<td><code>HIERARCHIZE(set_expr)</code></td>
<td>set of members</td>
</tr>
<tr>
<td>HIERARCHISE</td>
<td><code>HIERARCHIZE(set_expr, POST)</code></td>
<td></td>
</tr>
<tr>
<td>IIF</td>
<td><code>IIF(logical_expr, expression1, expression2)</code></td>
<td>number or string</td>
</tr>
<tr>
<td>INTERSECT</td>
<td><code>INTERSECT(set_expr1, set_expr2)</code></td>
<td>set</td>
</tr>
<tr>
<td>ISNULL</td>
<td><code>ISNULL(scalar_expression, scalar_value_if_null)</code></td>
<td>number or string</td>
</tr>
<tr>
<td>LAG</td>
<td><code>member_expr.LAG(optional_nonnegative_integer_expr)</code></td>
<td>member</td>
</tr>
<tr>
<td>LASTCHILD</td>
<td><code>member_expr.LASTCHILD</code></td>
<td>member</td>
</tr>
<tr>
<td>Function</td>
<td>Syntax</td>
<td>Return Type</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>LASTSIBLING</td>
<td>member_expr.LASTSIBLING</td>
<td>member</td>
</tr>
<tr>
<td>LEAD</td>
<td>member_expr.LEAD(optional_nonnegative_integer_expr)</td>
<td>member</td>
</tr>
<tr>
<td>LOG</td>
<td>LOG(numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>LOOKUP(term_list_name, lookup_value, default, alternative_field)</td>
<td>string</td>
</tr>
<tr>
<td>MAX</td>
<td>MAX(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>MEDIAN(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>MEMBERS</td>
<td>level_expr.MEMBERS</td>
<td>set of members</td>
</tr>
<tr>
<td></td>
<td>hierarchy_expr.MEMBERS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dimension_expr.MEMBERS</td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>MIN(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>NEXTMEMBER</td>
<td>member_expr.NEXTMEMBER</td>
<td>member</td>
</tr>
<tr>
<td>OPENINGPERIOD</td>
<td>OPENINGPERIOD(ancestor_level, member_expr)</td>
<td>member</td>
</tr>
<tr>
<td>ORDER</td>
<td>ORDER(set_expr, ordering_expr, ASC/DESC/BASC/BDESC)</td>
<td>set</td>
</tr>
<tr>
<td>PARALLELPERIOD</td>
<td>PARALLELPERIOD(level_expr, offset, member_expr)</td>
<td>member</td>
</tr>
<tr>
<td>PARENT</td>
<td>member_expr.PARENT</td>
<td>member</td>
</tr>
<tr>
<td>PERCENTILE</td>
<td>PERCENTILE(set_expr, numeric_expr, numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>PERCENTILERANK</td>
<td>PERCENTILERANK(set_expr, numeric_expr, comparison_value)</td>
<td>number</td>
</tr>
<tr>
<td>PERIODSTODATE</td>
<td>PERIODSTODATE(ancestor_level, member_expr)</td>
<td>set of members</td>
</tr>
<tr>
<td>POWER</td>
<td>POWER(numeric_expr,numeric_expr_for_power)</td>
<td>number</td>
</tr>
<tr>
<td>PREVMEMBER</td>
<td>member_expr.PREVMEMBER</td>
<td>member</td>
</tr>
<tr>
<td>PROPERTIES</td>
<td>member_expr.PROPERTIES(property_name)</td>
<td>string</td>
</tr>
<tr>
<td>RANK</td>
<td>RANK(tuple_expr, set_expr, numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>ROUND</td>
<td>ROUND(numeric_expr,decimal_places)</td>
<td>number</td>
</tr>
<tr>
<td>SIBLINGS</td>
<td>member_expr.SIBLINGS</td>
<td>set of members</td>
</tr>
<tr>
<td>SQRT</td>
<td>SQRT(numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>STDDEV, STDEV</td>
<td>STDDEV(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>STDDEVP, STDEVVP</td>
<td>STDDEVP(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>SUBSET</td>
<td>SUBSET(set_expr, first_element_expr, optional_element_count)</td>
<td>set</td>
</tr>
<tr>
<td>SUM</td>
<td>SUM(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>TAIL</td>
<td>TAIL(set_expr, optional_integer_expr)</td>
<td>set</td>
</tr>
<tr>
<td>Function</td>
<td>Syntax</td>
<td>Return Type</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>TOPCOUNT</td>
<td>TOPCOUNT(set_expr, element_count, optional_ordering_expr)</td>
<td>set of members or tuples</td>
</tr>
<tr>
<td>TOPPERCENT</td>
<td>TOPPERCENT(set_expr, element_count, optional_ordering_expr)</td>
<td>set of members or tuples</td>
</tr>
<tr>
<td>TOPSUM</td>
<td>TOPSUM(set_expr, element_count, optional_ordering_expr)</td>
<td>set of members or tuples</td>
</tr>
<tr>
<td>UNION</td>
<td>UNION(set_expr1, set_expr2)</td>
<td>set</td>
</tr>
<tr>
<td></td>
<td>UNION(set_expr1, set_expr2, ALL)</td>
<td></td>
</tr>
<tr>
<td>VAR, VARIANCE</td>
<td>VAR(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>VARP, VARIANCEP</td>
<td>VARP(set_expr, optional_numeric_expr)</td>
<td>number</td>
</tr>
<tr>
<td>VISUALTOTALS</td>
<td>VISUALTOTALS(set_expr, optional_parent_name_pattern)</td>
<td>set of members</td>
</tr>
</tbody>
</table>